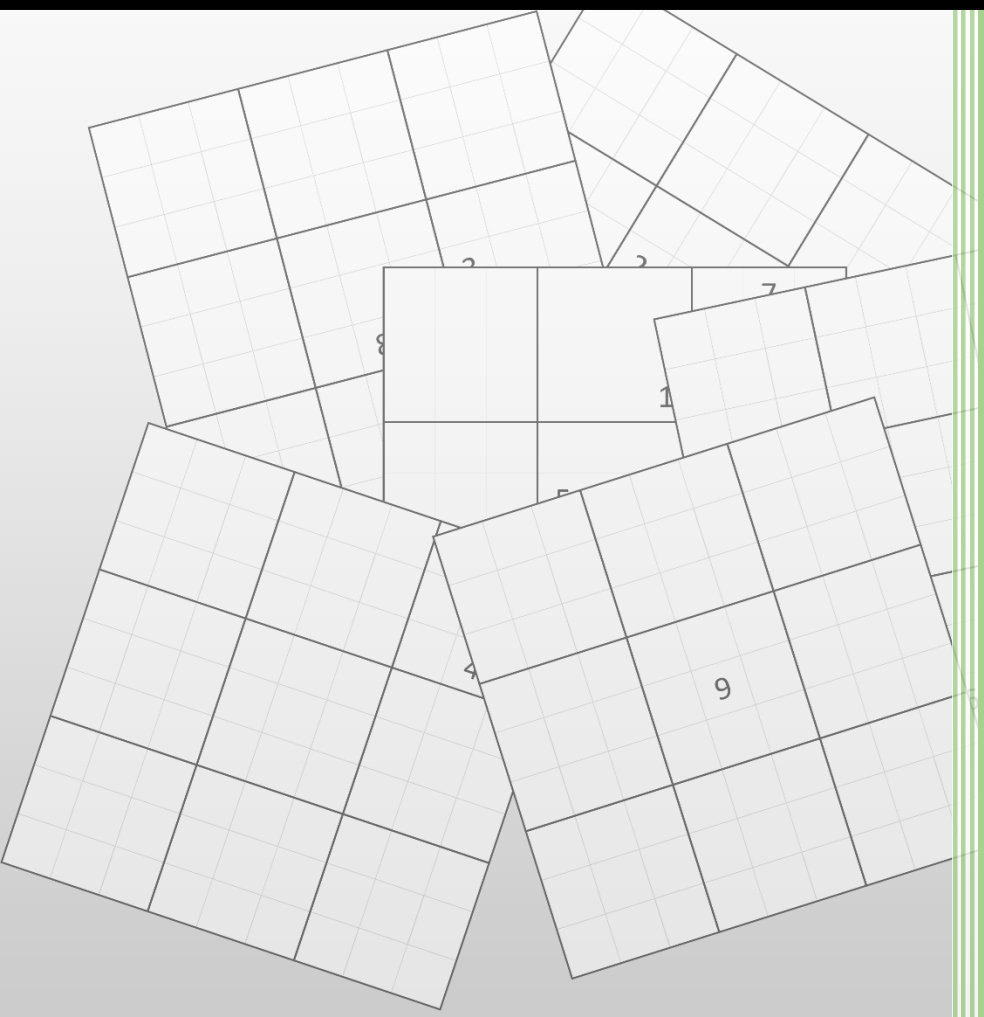
2023

A Program To Generate A Sudoku



Hayden Manton

Gold Crest Award

Abstract

Creating a Sudoku manually is the most common approach for designing the classic Japanese puzzle since its believed creation around half a century ago. For a standard 9x9 grid, this can be a very time consuming task yet, in this report an approach for a program has been constructed to drastically reduce this time. Research was conducted to initially understand the complexities of a Sudoku puzzle and their varying levels of difficulty as a result of differing properties and dimensions. This investigation led to an interactive website accompanying 4 individual high-speed puzzle generation programs utilising 4 different programming languages, reducing the time taken to create a sudoku puzzle significantly.

Contents

Section 5 – Filling the Grid

Section 6 – Emptying the Grid

Section 7 – Solving the Grid

Section 8 – The Survey

Section 9 – The Final Program

Section 10 – More Iterations of the Sudoku Generator

Section 11 – Conclusion

Section 12 – Reflection

Introduction

The inspiration behind this project originates from a curiosity in puzzles and how they are created. I developed an interest in potentially creating a program that generates a puzzle much faster than creating them by hand. After thought, I reduced the pool of puzzles down to three which I deemed interesting and useful enough to create a program around. The three puzzles were: Crosswords; Word Searches and Sudokus yet he most useful program to develop, from these three options, would be a program to create a sudoku puzzle, as, creating one by hand would take much longer than I think I could develop a program to do. Yet, consequently, this means that the development process will be much more difficult due to the complex rules and nature that must be adhered to within sudoku.

The Sudoku is a number-based puzzle where the method to win is by filling a 9x9 grid with numbers(1-9) in such a way that fits within several rules. This report presents the development of a program using Visual Basic that aims to generate Sudoku puzzles of varying difficulty levels, providing players with a quick and easy alternative to creating puzzles by hand.

My goal is to thoroughly investigate the concept of Sudoku, including the various methods used to solve them and the process of creating them. I intend to carefully evaluate multiple approaches for each step while designing the program using flowcharts or pseudocode. The design will be implemented using the programming language Visual Basic and the IDE Visual Studio. I will also use feedback from an anonymous survey to inform my decision-making. Lastly, I will evaluate the effectiveness of the program by comparing it to my initial objectives, research findings, and survey results.

Aims & Objectives

I will have achieved my aims if I…

- Successfully research what a sudoku puzzle is.
- Learn new techniques and methods for completing a Sudoku.
- Create multiple approaches for the program.
- Evaluate different approaches and choose the most appropriate ones.
- Produce an overall plan for the development of the program.
- Conduct an anonymous survey and gather useful results.
- Can alter my plan based on the feedback from the survey.
- Implement the program successfully.
- Make the program faster than creating a puzzle by hand.
- Ensure the program is easy to use.

What is a Sudoku?

Sudoku is a puzzle with squares, grids, and numbers. The puzzle uses the numbers 1-9 inclusive, containing 9 of each number within a grid of 9x9 (81 squares). The grid also contains 3x3 areas, of which there are 9, creating a display where a larger grid overlaps a smaller grid.

Unsolved Puzzle

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

Solved Puzzle

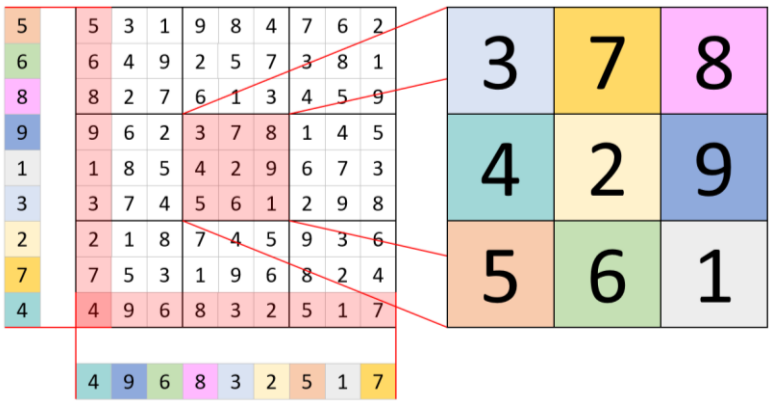
6	5	8	9	3	1	4	2	7
4	3	2	6	7	8	9	1	5
9	1	7	2	4	5	6	8	3
2	9	6	3	5	4	8	7	1
5	8	1	7	6	2	3	4	9
7	4	3	8	1	9	2	5	6
1	2	9	5	8	3	7	6	4
8	6	5	4	9	7	1	3	2
3	7	4	1	2	6	5	9	8

A sudoku puzzle can only ever have one solution yet there are varying difficulties available. Generally, the more numbers that are present in the unsolved puzzle, the easier the puzzle is to solve because less numbers need to be inputted. Also, as the puzzle progresses it tends to become easier as, each square has fewer possible options for inputting numbers. Finally, providing each input is valid, it is not possible to fail a sudoku, only not complete it, yet one invalid move can have a knock-on chain reaction effect where the puzzle then becomes unsolvable.

Fundamental Principles

The fundamental principles of a sudoku puzzle can be divided into two parts:

- All rows/columns must contain the numbers 1-9 only once.
- All designated 3x3 grids, of which there are 9, must contain the numbers 1-9 only once.



The above diagram displays the two fundamental principles of a sudoku puzzle. It is clear to see in this diagram that the leftmost column and the bottom row both contain every number between 1-9 only once, where they share the number 4 in the same position on their intersection. The central 3x3 grid also follows the same rule by which it contains every number between 1-9 only once. The order of the rows/columns/grids is also random, making the number of variations of solved sudoku grids very large, therefore it is very uncommon to see repetition or regular patterns in numbers.

Research

- What Makes a Sudoku Easy/Difficult?

Sudoku Randomness

The fewer rules and guidelines given for a sudoku puzzle initially, the more difficult the puzzle is to solve, for two reasons. The first reason is that with less help at the start, the solver goes into the puzzle with less information, therefore, making them feel less confident. The second reason is, the fewer rules/guidelines that are given, generally, the more random the puzzle is.

Valid Regular Sudoku

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

Invalid Diagonal Sudoku

An example of this is with a Diagonal Sudoku (An X Sudoku), which is a sudoku that also requires two diagonal lines (top left to bottom right and top right to bottom left) to also contain every number between 1-9 only once, as shown in the diagram above. The solved sudoku above would be a valid outcome for a regular Sudoku, yet, not for an X Sudoku. This proves that there are many different solved puzzles that are not valid if a new rule is introduced. The total number of solved sudoku varieties is widely agreed to be 6,670,903,752,021,072,936,960.[1]

Using this value, it is also possible to calculate the number of grids that wouldn't be valid for a diagonal sudoku yet would be valid for a regular sudoku:

Calculating the Probability that a random grid will be valid for a Diagonal Sudoku

$\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\left(1-\frac{3}{n}\right)\ldots\left(1-\frac{k-1}{n}\right)$ *Where* k = *The number of values in the list,* n = *The range of options for a number to be*

When $k = 9, n = 9$:

$$\left(1-\frac{1}{9}\right)\left(1-\frac{2}{9}\right)\left(1-\frac{3}{9}\right)\left(1-\frac{4}{9}\right)\left(1-\frac{5}{9}\right)\left(1-\frac{6}{9}\right)\left(1-\frac{7}{9}\right)\left(1-\frac{8}{9}\right) = 9.36656708\times10^{-4}$$

$\left(9.36656708\times10^{-4}\right)^2 = 8.77325789\times10^{-7}$ (*Squared because there are two diagonal parts*)

Calculating the Number of possible Diagonal Sudoku grids

$$1-8.77325789\times10^{-7} = 0.9999991227$$

$$(0.9999991227)(6670903752021072936960) = 6.6708979\times10^{21}$$

$$6670903752021072936960 - 6.6708979\times10^{21} = 5.85202107\times10^{15}$$

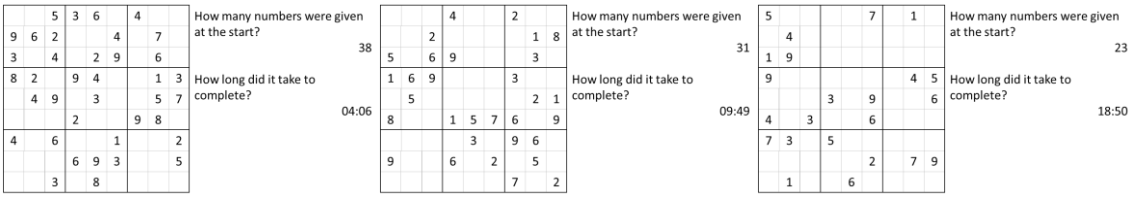
These calculations show the number of possible Diagonal Sudoku grids is much less compared to the number of possible Regular Sudoku grids, reducing the randomness and variety of puzzles greatly, therefore, I will not be considering the 'Diagonal Sudoku' for the program.

Research

- Does the Quantity of Starting Numbers Impact the Sudoku Puzzle?

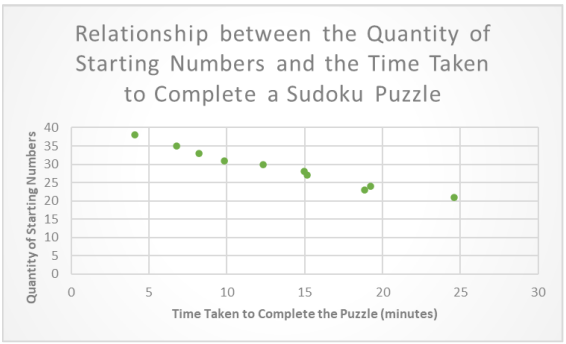
The views on this question are very mixed. Some believe that *"the difficulty can be viewed as a manner of observing how many givens are in each row, column, or box, and how these can help you discover the correct place for each number missing*," and suggest *"Easy Sudoku puzzles come with at least three in each row, column, box, and number, along with 30 givens,"* whereas *"Hard Sudokus might come in the upper 20s, along with entire boxes or numbers unaccounted for."*[2] Others think that *"there is no connection between the difficulty of a Sudoku puzzle and the number of clues it contains. This is a myth. There are very hard puzzles with many clues and easy puzzles with very few clues."*[3]

I believe the answer lies somewhere in the middle, as I think there probably is some correlation between the difficulty of sudoku, and the amount of numbers initially provided. Due to the inconclusive answer, I decided to carry out my own test. I first gathered 10 random sudoku puzzles, with varying levels of difficulty, and attempted to prove or disprove a correlation between the quantity of numbers initially provided and the difficulty of the puzzle.



The above three diagrams show 3/10 of the puzzles that I attempted, varying in difficulty and I quantified the difficulty using the time it took to complete the puzzle.

	Time Taken to Complete a Sudoku Puzzle	Quantity of Starting Numbers
1	04.10 minutes	38
2	06.77 minutes	35
3	08.23 minutes	33
4	09.82 minutes	31
5	12.32 minutes	30
6	14.95 minutes	28
7	15.15 minutes	27
8	18.83 minutes	23
9	19.23 minutes	24
10	24.62 minutes	21



What do these results show?

This is a graphical representation of the data gathered, with the table of data to the left and a scatter graph to the right. It is clear to see that there is a fairly strong correlation between the 'Quantity of Starting Numbers' and the 'Time Taken to Complete a Sudoku Puzzle' however, this doesn't mean to say that the second research opinion was incorrect as, the sample size (10 puzzles) isn't a very large number and so may not be an accurate reflection of the true correlation. Furthermore, sudoku puzzles can have many starting numbers without the puzzle also being easy, as other aspects of the setup must be considered, such as, whether a number is missing from the puzzle or not, an example being in the third unsolved sudoku puzzle displayed higher on this page.

Research

- Pencil Markings

When solving a sudoku, some prefer to use pencil markings as a method to solve the puzzle.

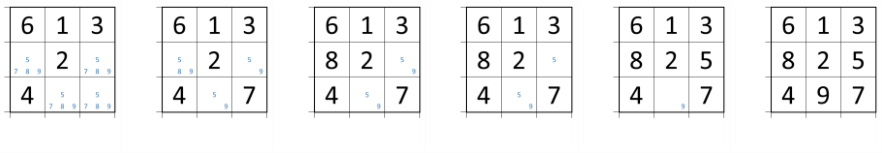
Advantages

- This method of solving a sudoku helps with memory and allows someone to attempt to solve a problem and come back to another part of the puzzle at a later point.
- It also gives an overall perspective of the state of a puzzle and guarantees, if done correctly, that the puzzle will be solved without much mental problem solving.

Disadvantages

- Although there is a certainty of a completed puzzle, using this methodology, the amount of time it takes to initially setup and overall is much more.
- With up to 9 nine numbers in each empty square, a puzzle, using this method, can look much more complex which may put many off using pencil markings.

"In a nutshell, pencil marking is writing little numbers as a way of keeping track of the remaining possibilities, or candidates, for all the cells that are still unsolved. The idea is that you would then logically and methodically remove those marks, or candidates, one by one." *[4]*



How To Use Pencil Markings Efficiently[5]

- One efficient way to use pencil markings is to work through the numbers from 1-9.
- For each number work through each designated 3x3 grid.
- Within each square, if the number is unrestricted then pencil mark it in that position.
- Once the 9x9 puzzle is full with pencil marks, it can now be solved.
- If there is only one pencil mark at a location, then that number must go there.
- If there is a row where one number is only marked once, that number goes there.
- If a column has one number that is only marked once, that number must goes there.
- If a square has a number with only one marking, the number can be inputted there.
- Every time a new number is inputted, markings must be updated.
- Update the markings in the square, row & column of the number if necessary.

Note: Pencil Marking by hand is more difficult and would take more time.

Research

- Phistomefel Ring

What is the Phistomefel Ring?

The 'Phistomefel Ring' is *"the observation of a particular pattern in completed Sudoku puzzles that can aid with solving uncompleted puzzles. The observed pattern is that the 16 digits in the four 2×2 corner regions will match the digits in the 16-cell ring circling the central 3×3 region."*[6] It is important to note that using the 'Phistomefel Ring' theory does not, by default, indicate where select numbers should be located, yet it does determine the regions in which they can be present.

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

[1,1,2,2,3,4,5,5,5,7,7,7,8,9,9,9]

[1,1,2,2,3,4,5,5,5,7,7,7,8,9,9,9]

	6					3		
1	4					5		
			9		3	6		8
		1				5		
		4				8		
		5						
		6			8	7		
	1							9
		8					6	3

[1,1,3,3,4,5,6,6,9]

[1,3,4,5,5,6,6,7,8,8,9]

8	6						3	
1	4						5	
			9		3	6		8
		1				5		
		4				8		
		5						
		6			8	7		
	1						8	9
		8					6	3

[1,1,3,3,4,5,6,6,9] + [8,8]

[1,3,4,5,5,6,6,7,8,8,9]

An Example of How the Phistomefel Ring is Used to Help Solve a Sudoku Puzzle

The two diagrams above display how this theory can be used in practice. It isn't initially visible that it is possible to add numbers to the puzzle on the left. Without the use of the 'Phistomefel Ring', this puzzle would be unsolvable from this point and no more numbers would be able to be added without more hints. Yet using this methodology, it can be concluded that at least two more '8's should be added to the 16 digits in the four 2x2 corner regions since there are at least two '8's in the 16-cell ring circling the central 3x3 region that are not present in the corner regions. There are '8's present in the top right and bottom left 3x3 designated grids, and in both cases, the '8's are not within the 2x2 corner regions within those grids. This means that the '8's must be added to the corner regions located within the top left and bottom right designated 3x3 grids within their 2x2 corner regions. Luckily, there is only one available space within each of the determined regions therefore the '8's can be inputted into those locations validly.

A	A						A	A
A	A						A	A
		C	B	B	B	C		
		B				B		
		B				B		
		B				B		
		C	B	B	B	C		
A	A						A	A
A	A						A	A

- Not All A's are equal.
- Not all B/C's are equal.
- The contents of A equal the contents of B+C.
- The number in C can never equal any number in A within the same 3x3 designated grid.
- The maximum repetitions for a number in the 16-digit list is four, and in this case, none of them will be at a C.
- The maximum repetitions for a number in the 16-digit list, if all appearances of the number don't exist at B, is two.

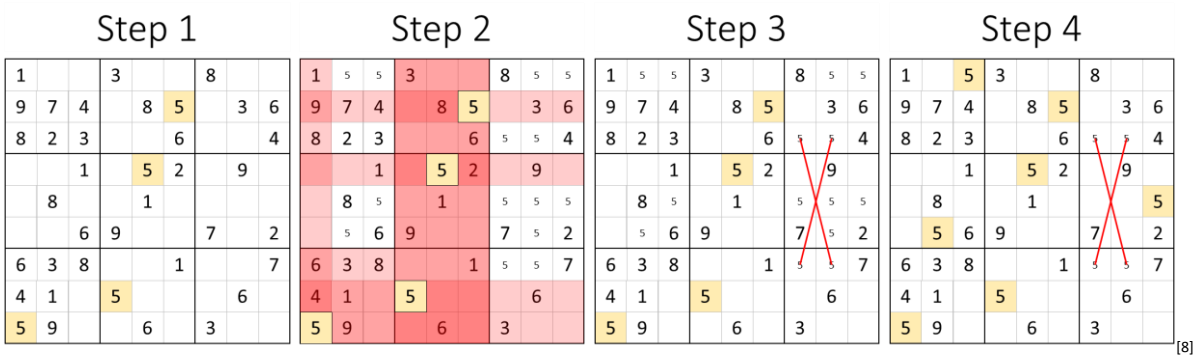
[7]

Research

- X-Wings

What is an X-Wing in a Sudoku Puzzle?

Simply put, X-wings are used as a technique to elegantly eliminate pencil marks from a partially solved sudoku puzzle, but the best way to show what X-Wings are in sudoku puzzles, and how they are used, is to use an example of where an X-Wing can be used to confirm numbers in a sudoku.

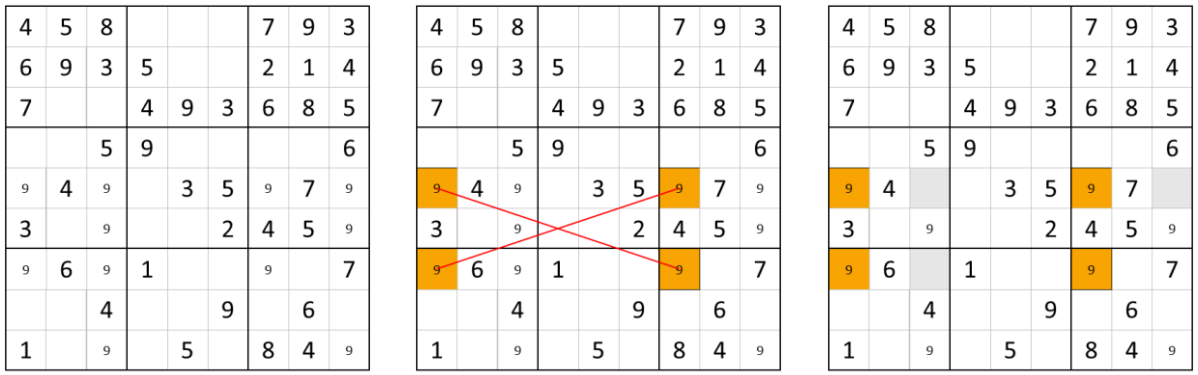


[8]

Steps:

1. Here every position of the number ‘5’ is highlighted in yellow.
2. Then every row, column, and square in which the number ‘5’ is located, is highlighted in red. Then every empty square that isn’t highlighted in either yellow or red is then pencil marked with the number ‘5’
3. There then must be a search for an X-Wing. To find an X-Wing look for a setup where two a duo of alike numbers on the same row, are replicated either directly above or below on another row.
4. Finally, it may be possible to remove some of the pencil marks of ‘5’. If there are different pencil marks of ‘5’ along the same columns of the four ‘5’s that make up the X-Wing, these can now be removed as, it would not be possible for a ‘5’ to be in those locations. In this example, this creates only one position available for the number ‘5’ in the right-middle 3x3 designated grid, which then has a chain reaction on the rest of the puzzle, where every other 5, that isn’t already filled in, or a part of the X-Wing can now be allocated a final position.

Note: There is not always an X-Wing present in every stage of a partially solved sudoku, yet there can also be multiple.



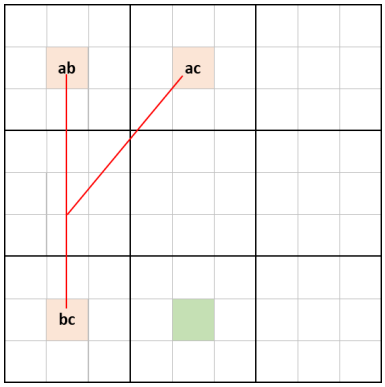
Research

- Y-Wings

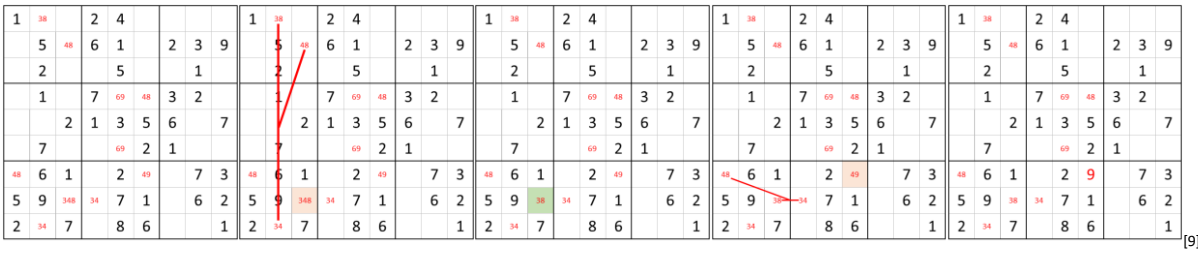
What is a Y-Wing in a Sudoku Puzzle?

"Y-Wings are a fairly advanced Sudoku-solving technique that allows you to eliminate candidates in particular rows or columns of the grid. This is why it's known as a candidate eliminator strategy. Similar to the X Wing strategy, Y Wings involves identifying a particular pattern of candidates in the grid but unlike X Wing, it only involves three different cells, not four. While this pattern is not always the easiest to spot, it is sometimes necessary to solve harder Sudoku puzzles."[9]

The example below shows how Y-Wings can be used to eliminate possibilities elsewhere in a sudoku puzzle. 'ab' cannot contain a 'c' yet it must contain either an 'a' or a 'b', meaning either 'ac' or 'bc' must contain a 'c'. Using this logic, it would not be possible for a 'c' to be in the location shaded in green, therefore, the pencil marked 'c' could be removed in that position.



The images below shoe how this idea can be used in practice



[9]

First, some pencil marks are added to the partially complete sudoku puzzle. A Y-Wing is found and the location that this Y-Wing effects is highlighted. The pencil marked '4' is then removed from the highlighted square. This forms another Y-Wing, yet horizontal, and the location at which this Y-Wing impacts the puzzle is then highlighted. The pencil marked '4' at the new highlighted position can then be eliminated, leaving only a 9, pencil marked there, which can now be confirmed as a final position.

[10]

Research

- What is the Most Difficult Sudoku Puzzle?

The puzzle displayed below was created by a Finnish mathematician (Arto Inkala) in 2012 and is said to be the most challenging Sudoku Puzzle according to many sources, containing only 21 numbers at the start. “Based on the number of deductions that need to be made to fill in a single cell, this puzzle achieves a difficulty rating of 11 stars, compared with five stars for the average newspaper sudoku.”[11]

Puzzle

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

Solution

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

Another sudoku which is referred to as the ‘most difficult known sudoku puzzle’, was also created in 2012 and contains 1 fewer number to start, with only 20 numbers initially.

Puzzle

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

Solution

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

What these two puzzles demonstrate is that very difficult sudoku puzzles start with very few clues (numbers), I will take this into account when creating the program, as I do not wish for the puzzles to be overly difficult to solve.

Research

- Different Sizes of Sudoku Puzzles

What are the Different-Sized Sudoku Puzzles?

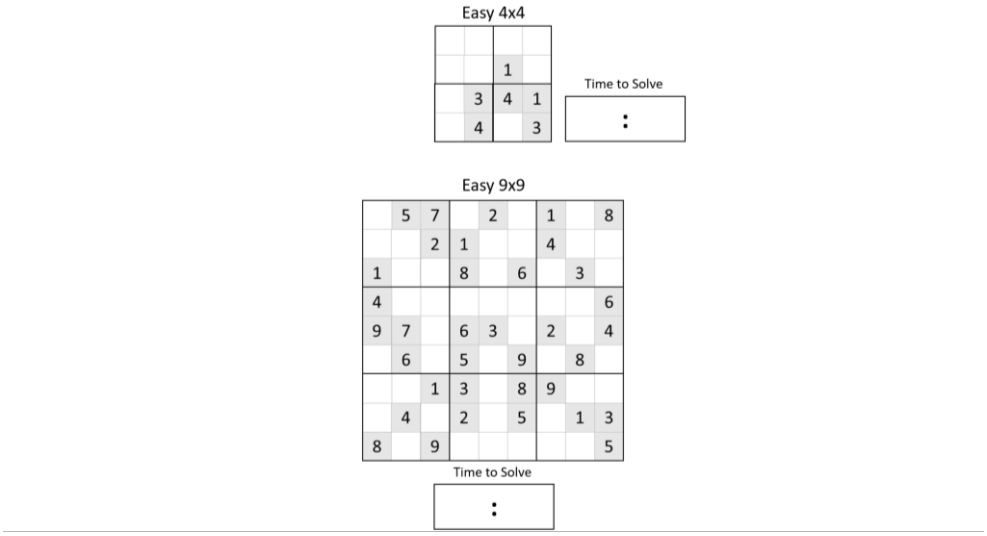
There are many different types of sudoku puzzles that follow the regular rules and patterns yet exist in different shapes and sizes. The short answer to this question is that a Sudoku can be as big as you like yet some are not enjoyable to solve as they are either too small/simple or too large/complex. So there is a limit to where the amount of time it would take someone to solve wouldn't be feasible. Some common puzzles include:

- 4x4 (4 regions)
- 6x4 (4 regions)
- 6x6 (6 regions)
- 6x6 (9 regions)
- 9x6 (6 regions)
- 9x6 (9 regions)
- 9x9 (Most Common)
- 16x16 (16 regions)
- 25x25 (25 regions)

What are the Effects of Changing the Size of a Sudoku Puzzle?

I wasn't able to find any sources online discussing this topic or giving any definitive answer, thus I decided to carry out a test to see if there is an impact on the difficulty of a sudoku puzzle when you change the size.

For the test, I created 2 sudoku puzzles: one relatively easy 9x9 puzzle and one easy 4x4 puzzle. I controlled the difficulty by manipulating the number of clues given at the start, in both cases, being relatively high in comparison to the overall area of the puzzles.



I handed out these two puzzles (displayed above) to 3 anonymous people. The results were clear. For the 4x4 puzzle, the average time to complete was 54 seconds whereas the time taken to complete the 9x9 puzzle was 387 seconds. There were no anomalies in the results, showing an obvious pattern and correlation between the difficulty of a sudoku puzzle and its size. Due to this, I will only be concerning 9x9 standard-sized sudoku puzzles in my project, as they are the most common and take the most reasonable amount of time to complete.

Results: **4x4** – *[01:07] [00:35] [01:00]* **9x9** – *[05:28] [06:57] [06:55]*

Scenarios

- Basic

Determining the Missing Number in an Incomplete Square

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

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

Determining the Missing Number in an Incomplete Column

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

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

Determining the Two Missing Numbers in an Incomplete Row

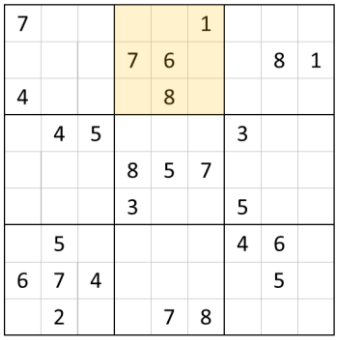
9	6	4		1	7		2	5
		5					1	6
3	7	1	5	6	2	4	9	8
1		9	2	7		6		4
		2	1		6			
8	3	6	4			2	7	1
5	1	3	7	8	4	9	6	2
6		8	9	5	3	1	4	
4		7	6	2	1	5	8	

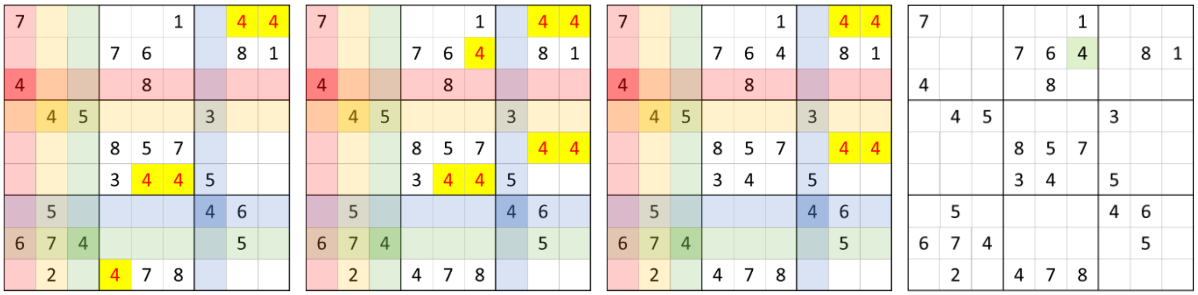
9	6	4		1	7		2	5
		5					1	6
3	7	1	5	6	2	4	9	8
1		9	2	7		6		4
		2	1		6			
8	3	6	4	9	5	2	7	1
5	1	3	7	8	4	9	6	2
6		8	9	5	3	1	4	
4		7	6	2	1	5	8	

Scenarios

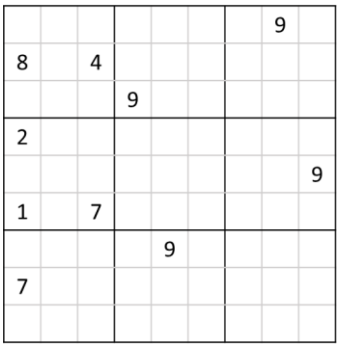
- Intermediate

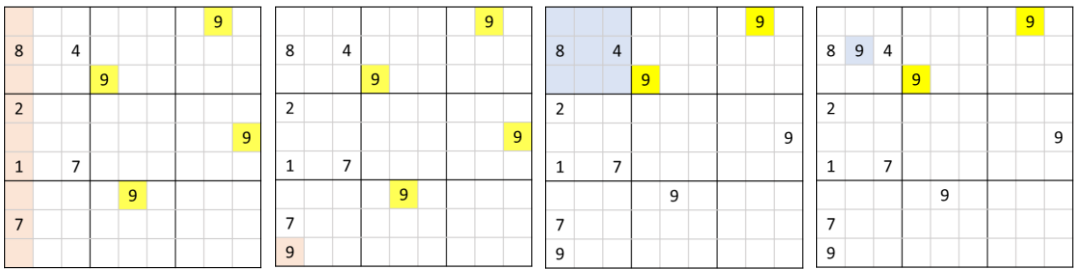
Determining the Location for the Number ‘4’ in the Area Shaded Orange

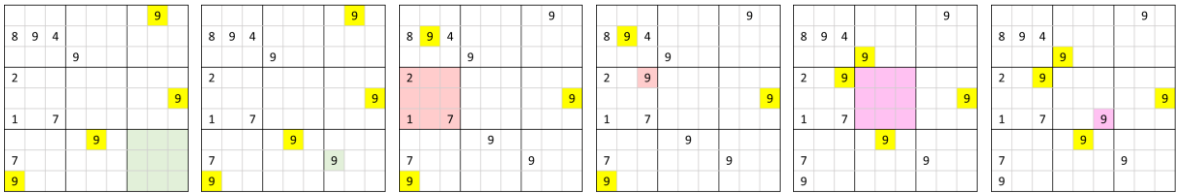




Determining the Location for Every Number ‘9’ in the Sparsely Completed Grid







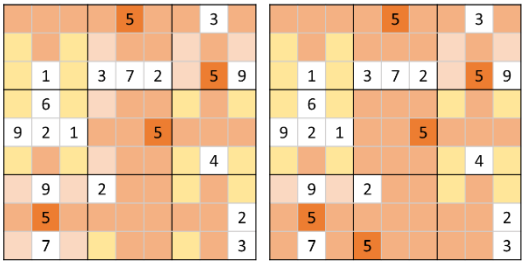
Scenarios

- Advanced

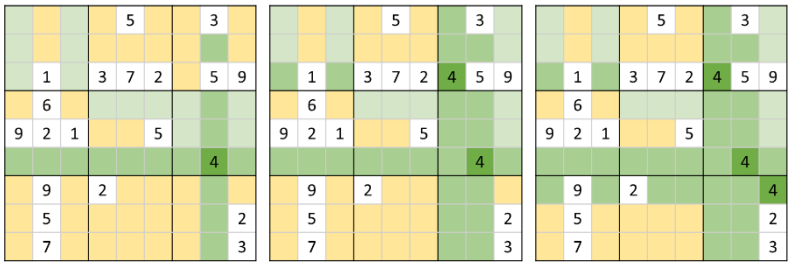
Determining a Location for the Number ‘5’

				5			3	
	1		3	7	2		5	9
	6							
9	2	1			5			
							4	
	9		2					
	5							2
	7							3

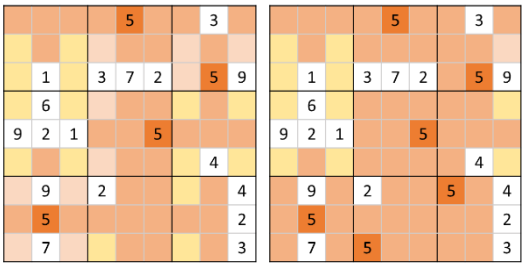
The images below highlight the positions where the number ‘5’ can/cannot be located.
It is clear to see here that there are two possible locations for ‘5’ in the bottom right square.



The images below highlight the positions where the number ‘4’ can/cannot be located.
Using the X-Wing rule and basic step by step methods, ‘4’ can be located in the bottom right square.



The two images below highlight the new positions where the number ‘5’ can/cannot be located.
Since the additions of the number ‘4’ there is now only one location, for the number ‘5’, possible in the grid.



The image below displays the solution to this scenario with the position of the ‘5’ in the bottom right square.

				5			3	
	1		3	7	2		5	9
	6							
9	2	1			5			
							4	
	9		2			5		4
	5							2
	7		5					3

Difficulties with Making a Sudoku

- Filling the Grid (Pages 28-29)

Filling the sudoku grid is essential for creating puzzles that follow the game's rules. This step involves placing numbers in a 9x9 grid, ensuring that each row, column, and 3x3 subgrid has numbers 1 to 9 without repetition. However, this straightforward task presents challenges due to intricate rules and the goal of puzzle diversity.

What is Required?

Filling the grid requires crafting a configuration that adheres to sudoku rules. Each cell must hold a number respecting row, column, and subgrid constraints. This demands a systematic approach to balance predictability and randomness, generating diverse puzzles.

Why/How Does this Create a Difficulty?

The challenge lies in intricate cell interdependencies and preventing rule violations. Placing a number affects subsequent placements, making it tough to balance constraints while maintaining coherence. Ensuring a unique solution further complicates matters, as configurations must prevent duplicates or unsolvable puzzles.

Balancing predictability and randomness adds another layer of complexity. Crafting puzzles with consistent difficulty and multiple solving paths requires strategic initial placements.

Strategies like backtracking and constraint propagation are common solutions. Backtracking explores options while recording choices for contradiction resolution. Constraint propagation, exemplified by the "naked singles" rule, narrows potential numbers for cells based on neighbouring values.

Conclusion

In conclusion, filling the sudoku grid involves navigating intricate rule interactions and meticulous considerations for solvability, uniqueness, and diversity. Overcoming these challenges is pivotal in creating puzzles that captivate and challenge enthusiasts, setting the stage for subsequent puzzle generation. This step exemplifies the elegance and complexity of the classic sudoku game.

Difficulties with Making a Sudoku

- Emptying the Grid (Pages 30-41)

Emptying the sudoku grid is a pivotal stage in puzzle creation, transforming a complete grid into an engaging puzzle. This process involves selectively removing numbers while ensuring the resulting puzzle remains solvable and possesses a unique solution. However, the task of emptying the grid presents its own set of challenges, as maintaining solvability and difficulty requires careful decision-making.

What is Required?

Emptying the grid involves removing numbers from a completed sudoku grid while preserving solvability and a single solution. The goal is to strategically choose cells for removal, creating a puzzle that provides a satisfying challenge. This necessitates an understanding of how different numbers' removal impacts puzzle difficulty and the potential for multiple solving paths.

Why/How Does this Create a Difficulty?

The challenge in emptying the grid emerges from maintaining a delicate balance between retaining solvability and crafting an enjoyable puzzle. Indiscriminate removal of cells can result in a puzzle with multiple solutions or one that's too simple to solve. Ensuring that the puzzle remains uniquely solvable requires a keen understanding of the intricacies of sudoku rules and the effects of cell removal.

The difficulty also lies in maintaining the puzzle's desired level of challenge. Some numbers contribute more to puzzle difficulty than others. Deciding which numbers to remove and in what order to maintain the puzzle's solvability and desired difficulty requires careful consideration. This decision-making process is a blend of mathematical analysis and intuitive puzzle design.

Strategies for achieving this balance include removing numbers from different areas of the grid, considering cells with overlapping influence, and accounting for possible solving paths that arise from unique combinations of remaining numbers.

Conclusion

In conclusion, emptying the sudoku grid is an intricate process that involves strategic removal of numbers to craft puzzles with desired solvability and challenge. This stage highlights the art of puzzle design, as it requires a nuanced understanding of sudoku rules, skillful decision-making, and a consideration of how players approach puzzle-solving. Overcoming the challenges of emptying the grid contributes significantly to the overall puzzle quality and enjoyment.

Difficulties with Making a Sudoku

- Solving the Grid (Pages 42-57)

Solving the sudoku grid is a crucial step in puzzle creation, ensuring that the puzzle's solvability and uniqueness are maintained. This process involves developing algorithms that can navigate incomplete grids to find valid solutions, all while adhering to the game's rules. However, this seemingly routine step is far from simple, given the intricate nature of sudoku and the potential for exponential solution paths.

What is Required?

Solving the grid requires implementing algorithms that can fill in the missing numbers within a partially completed sudoku grid. The algorithms must satisfy the same constraints that apply to filling the grid initially, ensuring that each row, column, and 3x3 subgrid contains all numbers from 1 to 9 without repetition. This necessitates an efficient exploration of possible solution paths while avoiding contradictions.

Why/How Does this Create a Difficulty?

The challenge in solving the sudoku grid arises from the need to navigate through a vast solution space while respecting the game's rules. The exponential growth in possibilities as cells are filled makes the process computationally demanding. The intricacy is further compounded by the incomplete information available in partially filled grids, introducing ambiguity and requiring algorithms to make informed guesses and backtrack when necessary.

The requirement to find a unique solution adds another layer of complexity. Determining whether a particular placement will lead to multiple solutions can be intricate, requiring careful analysis of possible consequences. This uniqueness constraint is essential in confirming the puzzle's integrity.

Efficiently solving the grid requires the utilization of techniques like constraint propagation and backtracking. Constraint propagation narrows down potential values for cells based on existing placements, reducing the number of possibilities to explore. Backtracking, on the other hand, allows for intelligent trial and error, facilitating the process of correction in case of rule violations.

Conclusion

In conclusion, solving the sudoku grid is a non-trivial task that demands algorithmic sophistication and a deep understanding of the game's intricacies. This stage's significance lies in confirming the puzzle's solvability, ensuring that players can approach the puzzle with the assurance of a valid solution. Overcoming the challenges of solving the grid showcases the complexity and elegance inherent in the design of sudoku puzzles, setting the stage for players to engage in the rewarding process of puzzle-solving.

Filling the Grid

- Method 1 (Brute Force)

How Does this Method Work?

This method is very simple to understand as, simply put, the grid will be filled via a method that continuously attempts random patterns of numbers, until the grid is valid. There are three ways to do this, and all share the same likelihood of success. The first is to randomize the numbers 1-9, for each 3x3 designated area of the sudoku puzzle, which would be repeated 9 times to complete the whole grid. The other two ways to do this would be to, instead of randomizing the numbers 1-9 for the 3x3 regions, do this for the rows/columns. The second part of this method requires a validation of the grid to make sure that it can be used to create a playable sudoku puzzle. This can be achieved by every time there is a new randomly generated grid, every row/column/3x3 square is checked to make sure the numbers 1-9 only appear once in each scenario, otherwise the grid is deemed to be invalid. If it is determined that the grid is invalid, then the grid must be randomized and checked again repeatedly. If the grid is checked and the grid is valid, then this method is over, and the filling of the grid is complete.

The easiest version of this method to write an algorithm for would be to randomize the rows as a few simple 'For Loops' could achieve this relatively quickly without much manipulation of the position of the numbers in the grid.

Advantages

- Once the algorithm has been run, there is a guarantee of a valid grid.
- Within the algorithm, the loop is very simple and time efficient.
- The grid that is outputted is always unpredictable/random making the solution to the puzzle unpredictable.

Disadvantages

- The algorithm, on average, will not be complete in a suitable timeframe.
- There is no stopping condition meaning, if the algorithm takes too long to run, there is no way of exiting the loop.
- It isn't time predictable.

Pseudocode

```
Sub Main
        Dim validGrid As Boolean = False
        Dim Grid(8,8) As Integer
        While validGrid = False
                fillGrid(Grid(,))
                If checkValidGrid(Grid(,)) = True
                        validGrid = True
                End If
        End While
        DisplayGrid(Grid(,))
End Sub
```

```
Sub fillGrid(Gird(,))
        Dim orderedNums(8) As Integer = [1,2,3,4,5,6,7,8,9]
        Dim randomNums(8) As Integer
        Dim tempStore As Integer
        Dim rndNumber As Integer
        For count = 0 To 8
                randomNums(count) = orderedNums(count)
        End For
        For count = 0 To 8
                rndNumber = (RANDOM NUMBER BETWEEN 1-9)
                tempStore = randomNums(count)
                randomNums(count) = randomNums(rndNumber)
                randomNums(rndNumber) = tempStore
        End For
...
```

Filling the Grid

- Method 2 (Using Grids & Regions)

How Does this Method Work?

The way this method works is slightly more complex than the previous method, yet still isn't overall difficult to understand with the use of diagrams and examples. Additionally, it does share some similarities, with the second half of this method. The first step for this method is to randomize a 3x3 region containing the numbers 1-9, which will be in the centre of the puzzle. These 9 positions of numbers can be inputted into the grid there and then as these will be a part of the valid grid at the end. The next step is to use a loop to again, randomize a grid of size 3x3, to be located underneath the first randomized region, and then check to see if, in that location, the 9 numbers would be valid if they were positioned there. If this is not valid, then the loop should repeat until it is. Step three is the same but for the 3x3 region above the previous two. These methods should then be used to create a 'Cross' shape made up of five 3x3 regions leaving the corner regions empty for now. Once there is a 'Cross' shape, then a random number input approach, similar to 'Method 1', could be used to fill in the rest of the grid.

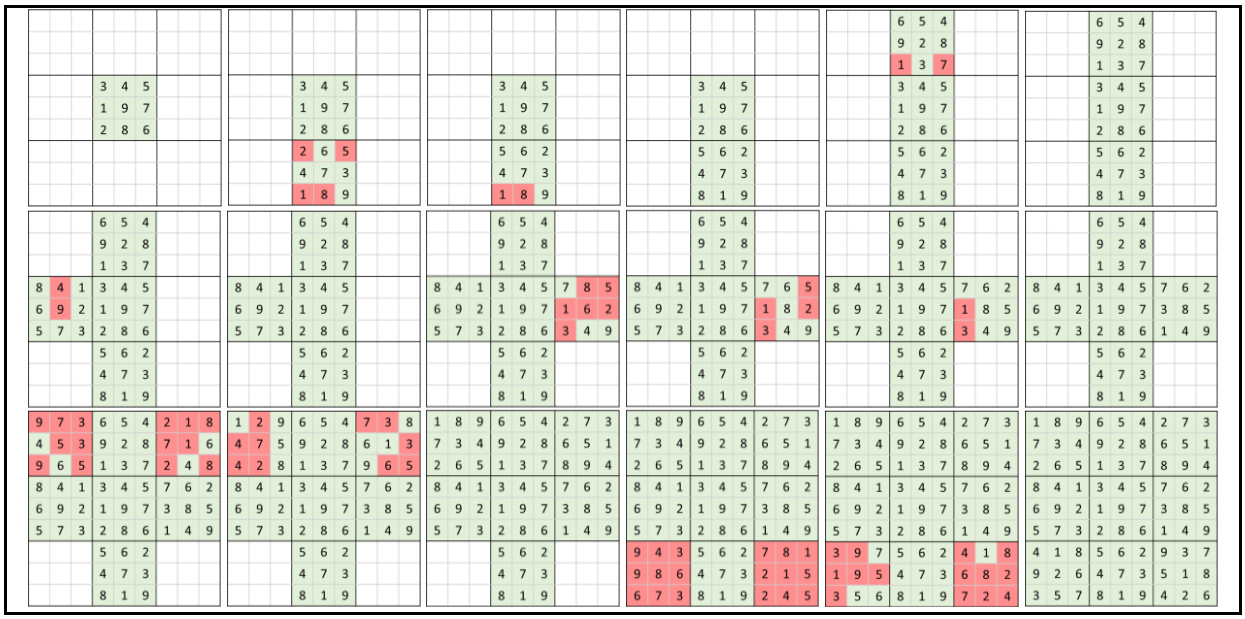
Advantages

- The first half of the algorithm (forming the 'cross') is reasonably time efficient.
- If the forming of the puzzle was displayed in time with the algorithm, it would be visually appealing to view.

Disadvantages

- Because there is no 'undo' part of the algorithm, it may take a while to fill the corner regions of the puzzle.
- The algorithm doesn't work in a predictable timeframe, so the method isn't very reliable.

How The Algorithm Would Look



Filling the Grid

- Method 3 (Backtracking with Stacks)

How Does this Method Work?

Method 3 would use a system where the algorithm would move linearly through the grid, along the rows, and would then input a random yet valid number. Every number inputted must be added to the top of a stack with the capacity to hold 80 numbers. However, if the algorithm ever reaches a point where there are no valid numbers to input, the algorithm will then start to backtrack using the stack with increasing depths backward, and attempts forward, until the process progresses from the location, at which no number could be inputted, all whilst the stack and status of the grid are updated accordingly. This algorithm should repeat until there is a valid 9x9 grid that can be used to generate a Sudoku puzzle.

Advantages

- This method is a very efficient.
- Easy to code, as method replicates how this is done by hand.

Disadvantages

- It is impossible to predict how long the method would take to complete.
- It would take time to write the code.

Pseudocode for the Main Subroutine

```
Sub FillingTheGrid
        Dim CurrentPos As Integer = 1
        While CurrentPos <> 81
            If InputIntoGrid(CurrentPos) = Valid
                CurrentPos += 1
            Else
                For Bckwrds = 1 To CurrentPos
                    Dim BackTrack As Boolean = True
                    BackTrack(Bckwrds,CurrentPos)
                    For Frwrds = 1 To Bckwrds
                        Dim NumOfValid As Integer = 1
                        If InputIntoGrid(CurrentPos) <> Valid
                            Exit For
                        Else
                            NumOfValid += 1
                        End If
                    End For
                    If NumOfValid = Bckwrds
                        Exit For
                    End If
                End For
            End If
        End While
End Sub
```

How Could the Algorithm be Improved?

The above image displays a concept for the main subroutine if this method was to be used in the final program. To be optimised, the addition of a type of abstract data type (A Stack), could be used to minimise the amount of code that would need to be written. This improvement would also allow for easy add-ons to the program in future, if required.

Filling the Grid

- Method 4 (Randomising Patterns)

How Does this Method Work?

This method works by randomising elements of the grid consecutively in a way which doesn't change the validity of the grid at any point. First, larger rows and columns are randomised, next smaller rows and columns are randomised and then each individual number is randomised, yet this algorithm is much easier to understand if displayed visually.

Advantages

- The algorithm is time predictable.
- It has a random outcome.
- The process is visually appealing.

Disadvantages

- It could take more time than other methods.
- It's possible for patterns to form.

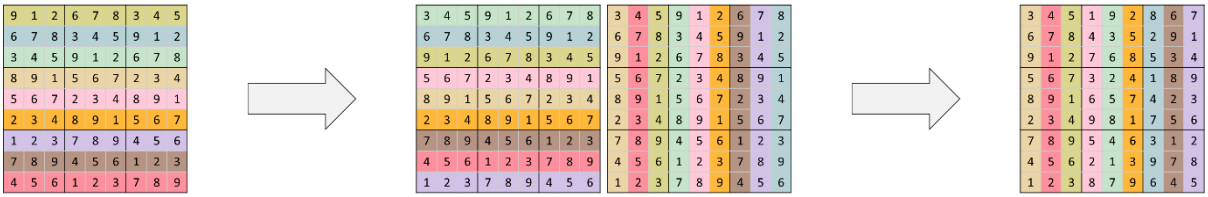
How The Algorithm Would Look

First, the 'Larger' Rows & Columns are Rearranged in a New Random Order

(Larger rows & columns are of size 3x9 containing three 3x3 grid regions each, displayed in varying colours below)

Second, the 'Smaller' Rows & Columns are Rearranged in a New Random Order

(The smaller rows & columns are shuffled in a way where they still remain as a part of their larger rows & columns)



Third, all Numbers are Randomised, Creating a New Random 9x9 Grid to be Used

(The individual numbers below are assigned different values, and the overall change is also displayed below)

How could the outcome be made more random?

This method could be modified to produce a more random outcome by changing the starting grid to one than doesn't display any repeating characteristics or patterns. As, currently, it is possible, that when the rows and columns are randomised, no rows or columns change order and so the only aspect of the puzzle that is randomised is when the numbers are changed in the last step. Yet this too, on a rare occasion, could produce an outcome which is the same as the input. The solution to all of this is to change the initial grid to a more unique grid containing no patterns.

Filling the Grid

- Method 5 (Pencil Mark Possibilities)

How Does this Method Work?

This method requires a 2-dimentional array that contains the contents of every square (9x9 Grid). Each value should initially be assigned the value '0' to signify the fact that originally the grid is empty at every square location. A 3-dimensional array should also be assigned before the algorithm is run, to pencil mark the grid. The first two dimensions of the array should represent the coordinates of the grid and the 3rd should represent the number of which the pencil mark refers to therefore, the array should store Boolean values, for whether the number is valid in that position. Another 2-dimentional array, storing strings, should hold which 3x3 regions each square (coordinate) is within, in the grid. This would be useful to be able to pencil mark the grid.

Advantages

- The algorithm would be time predictable.
- No patterns would show in the grid.

Disadvantages

- Complex to understand and code if expansions to the code are ever required.

Pseudocode for the Main Subroutine

```
Sub Main
        For count = 1 To 81
                ValidSqaureFound = False
                While ValidSquareFound = False
                        RandomRow = RandomNum(1-9)
                        RandomCol = RandomNum(1-9)
                        If Grid(RandomRow, RandomCol) <> 0
                                ValidNumberFound = False
                                While ValidNumberFound = False
                                        InputNum = RandomNum(1-9)
                                        If ValidInSquare(RandomRow, RandomCol, InputNum) = False
                                                Exit If
                                        Else If ValidInColumn(RandomRow, RandomCol, InputNum) = False
                                                Exit If
                                        Else If ValidInRow(RandomRow, RandomCol, InputNum) = False
                                                Exit If
                                        Else
                                                ValidNumberFound = True
                                        End If
                                End While
                                Grid(RandomRow, RandomCol) = InputNum
                                PencilMarkTheGrid()
                                ValidSquareFound = True
                        End If
                End While
        End For
End Sub
```

How Could the Algorithm be Improved?

This method could be modified for the better by, instead of randomising coordinates to deal with, instead it could linearly move through the grid finding the position with the least number of pencil marks at one location. If a location only has one pencil mark, then input that number, otherwise input a random option from the pencil marks. This modified method, using the same techniques should also pencil mark the grid after every input yet, here, the outcome would be much more time predictable and overall, a faster version of the method to fill the grid.

Filling the Grid

- Method 6 (Consecutive Pencil Markings)

How Does this Method Work?

Method 3 would select a random location in the grid and input a random number between 1-9 into a 2-dimensional array representing the grid display. Then that location should be updated in another 2D array (used to represent whether the location is empty/pencil marked/filled in). Following this point, the 3x3 region which the inputted number is in, then is all pencil marked, along with the row and column. From the locations in the grid that have been pencil marked, a new random location is chosen for a possible, random number to be inputted. Then the pencil marked grid should be updated in the 3x3 region, row, and column. This process should then repeat until the grid has been filled.

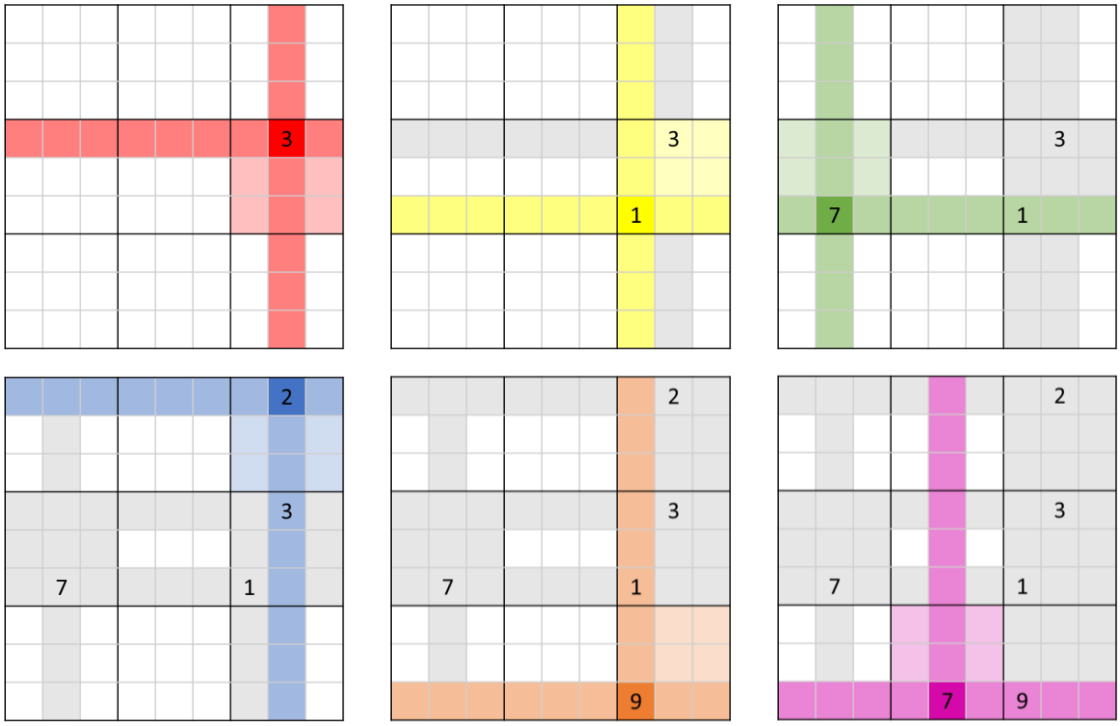
Advantages

- Will produce a random valid grid each time the algorithm is run.
- It won't create patterns.

Disadvantages

- More difficult to program than other methods.
- Not very time efficient.

How The Algorithm Would Look



The six images above display how this algorithm would work using an example of 6 steps, which would, in practice be repeated another 75 times before completion. The coloured areas represent the currently inputted number and the locations which need to be updated with pencil marks as a result. The grey/shaded positions represent the locations in the grid which have been pencil marked but are not currently required to be updated as a result of the inputted number. The co-ordinates highlighted in white have not been pencil marked or had any number inputted, therefore, using this method, a number cannot be inputted into those locations.

Filling the Grid

- Chosen Approach: **Method 4** (Randomising Patterns)

Method	Why was the method not chosen?
1.	Although this method is very easily understood and written in code, there is no stopping condition, and it isn't time predictable.
2.	It is an unreliable method due to the algorithm not working in a predictable timeframe and, is likely to take an inefficient amount of time.
3.	This method is easy to understand but would take an inefficient amount of time to code and this method is also impossible to predict how long the method would take.
5.	Even though the algorithm would be time predictable, and no patterns would show in the grid, the code would be difficult to expand because the code would be complex.
6.	This method would produce a valid completely random grid without patterns however, other methods are easier to code.

Why Was Method 4 Chosen?

Method 4 begins with a standard, pre-determined, valid sudoku grid, with all numbers inputted, and randomises the components of the grid whilst keeping the validity and randomness. This is achieved by taking the rows, columns and regions and randomising the numbers within them at the same time as following all basic rules of a sudoku puzzle. Therefore, the method is predictable and is guaranteed to have a successful result every time the algorithm is run, and this reliability is what makes this way of filling the grid, the most efficient and the obvious choice of method above others. A negative to this approach would be that patterns are able to form, but the probability of these forming is very unlikely. If an enhancement to the method was to be made to include a way of reducing the likelihood of patterns forming, this would create a definite longer average timescale for the algorithm to run, and it could no longer be claimed that this method is time efficient and consequently, the efficacy would reduce. Moreover, the idea that patterns can form, increases the randomness due to no bias towards the layout of any grid, thus method 4, without any further enhancements, would be truly random.

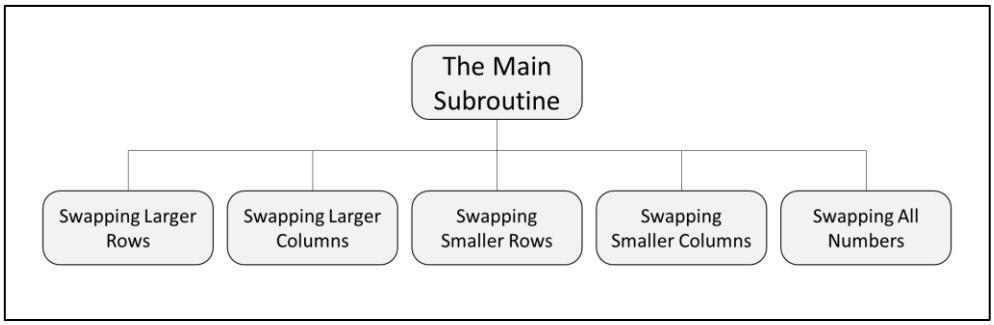
Pseudocode for the Main Function

```
Function FillGridMethod4(grid As Integer(,)) As Integer(,)
    Dim standardGrid As Integer(,) = GenerateStandardGrid()
    Dim shuffledNumbers As List(Of Integer) = ShuffleNumbers()
    For Each row In grid
        RandomizeCells(row, shuffledNumbers)
    For i = 0 To 8
        Dim column(grid.GetUpperBound(0)) As Integer
        For j = 0 To grid.GetUpperBound(0)
            column(j) = grid(j, i)
        RandomizeCells(column, shuffledNumbers)
        For j = 0 To grid.GetUpperBound(0)
            grid(j, i) = column(j)
    For r = 0 To 6 Step 3
        For c = 0 To 6 Step 3
            Dim region(8) As Integer
            For i = 0 To 2
                For j = 0 To 2
                    region(i * 3 + j) = grid(r + i, c + j)
                RandomizeCells(region, shuffledNumbers)
            For i = 0 To 2
                For j = 0 To 2
                    grid(r + i, c + j) = region(i * 3 + j)
    Return grid
End Function
' Additional functions as mentioned before...
```

Filling the Grid – Proof of Concept Program

- The Code

Structure Chart



Declaring Global Variables

```
Dim FillingGrid(8, 8) As Integer
Dim FlngRwOne() As Integer = {3, 8, 6, 5, 4, 1, 2, 9, 7}
Dim FlngRwTwo() As Integer = {2, 4, 9, 7, 3, 8, 6, 1, 5}
Dim FlngRwThree() As Integer = {5, 1, 7, 6, 9, 2, 4, 3, 8}
Dim FlngRwFour() As Integer = {6, 3, 4, 2, 7, 9, 5, 8, 1}
Dim FlngRwFive() As Integer = {1, 7, 5, 8, 6, 4, 9, 2, 3}
Dim FlngRwSix() As Integer = {9, 2, 8, 3, 1, 5, 7, 4, 6}
Dim FlngRwSeven() As Integer = {7, 5, 2, 9, 8, 3, 1, 6, 4}
Dim FlngRwEight() As Integer = {4, 9, 3, 1, 5, 6, 8, 7, 2}
Dim FlngRwNine() As Integer = {8, 6, 1, 4, 2, 7, 3, 5, 9}
```

The code above is to declare the starting grid using nine one-dimensional arrays representing the nine rows in the grid. A single two-dimensional array is also declared here as a global variable representing the 9x9 grid which will need to be accessed throughout the program.

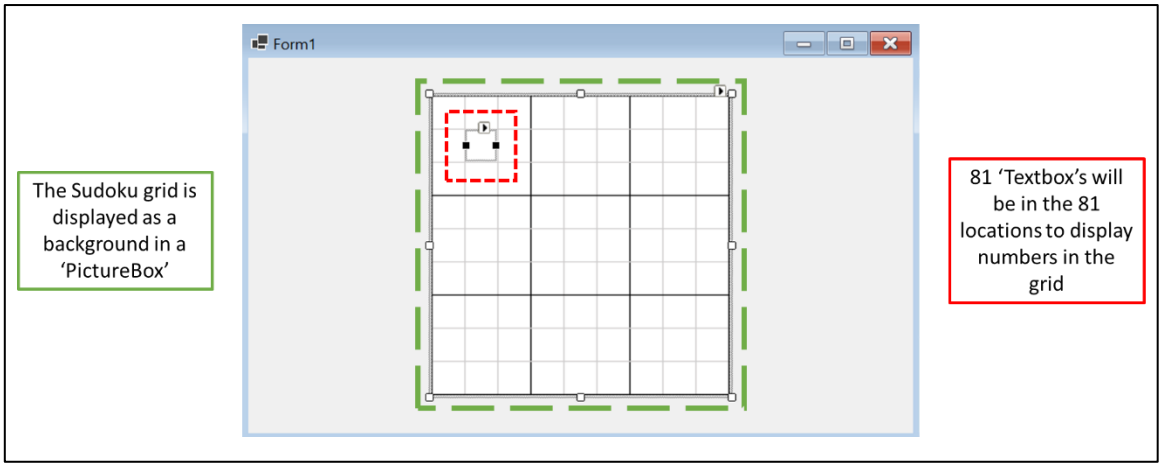
The Main Subroutine

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        For count = 0 To 8
            FillingGrid(0, count) = FlngRwOne(count)
            FillingGrid(1, count) = FlngRwTwo(count)
            FillingGrid(2, count) = FlngRwThree(count)
            FillingGrid(3, count) = FlngRwFour(count)
            FillingGrid(4, count) = FlngRwFive(count)
            FillingGrid(5, count) = FlngRwSix(count)
            FillingGrid(6, count) = FlngRwSeven(count)
            FillingGrid(7, count) = FlngRwEight(count)
            FillingGrid(8, count) = FlngRwNine(count)
        Next
        SwapLargerRows()
        SwapLargerColumns()
        SwapSmallerRows()
        SwapSmallerColumns()
        SwapAllNums()
        'Write code here to display the puzzle on the form
End Sub
```

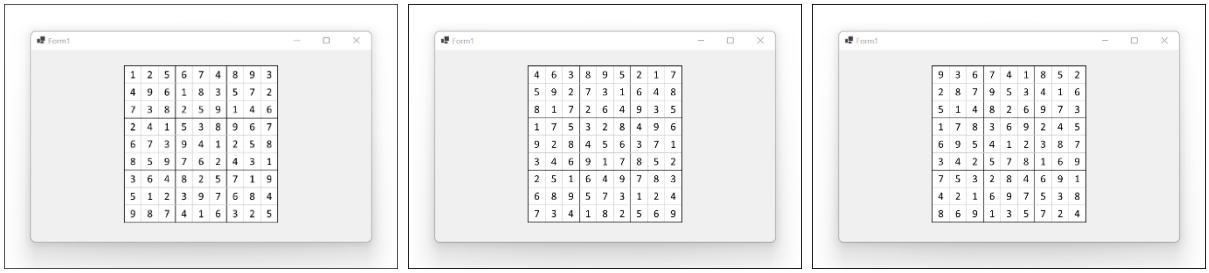
The main subroutine for this concept program, displayed above, initially inputs the nine rows into the starting grid. Following this, the larger rows (areas of 3x9 composing of three vertical 3x3 squared regions) will be shuffled into a random order in the subroutine *'SwapLargerRows'*. Next, larger columns (areas of 9x3 composing of three horizontal 3x3 squared regions) will be re-ordered randomly within the subroutine *'SwapLargerColumns'*. Then the smaller rows and columns (areas of 1x9 and 9x1 throughout the grid) will be mixed into a random order within their confined larger rows and columns, in the subroutines *'SwapSmallerRows'* and *'SwapSmallerColumns'*. Finally, a list of the numbers 1-9 will be randomised to relocate and swap the positions of all numbers in the subroutine *'SwapAllNums'*.

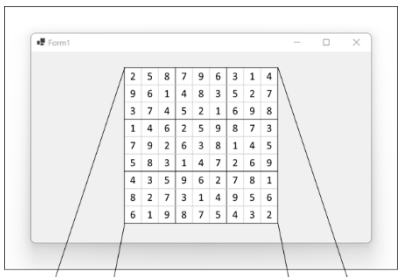
Filling the Grid – Proof of Concept Program

- The User Interface



Proof the Program Works





The four examples of randomised grids here, display what the program can produce efficiently. The form requires no buttons or any other components for the user to interact with, as this is just a concept program, and all algorithms are run in the background directly from the main subroutine linked to the main form. This means when the program is run and 'Form1' is displayed the randomised grid has already been formulated and therefore is displayed almost instantly and simultaneously with the rest of the user interface.

As is visible to the left, these grids are all completely valid and can be incorporated into the completion of the final program. This is due to there being only one of every number from 1-9 in every row, column, and 3x3 designated regions outlined by the darker black lines.

The design for this concept program only required a basic interface as it has not been made for any user but is purely proof that the method works in a way which can be easily merged into the final program.

This concept program works as expected, and therefore will be used as a support for the final program.

Emptying the Grid

- Method 1 (Removing 1 Number from Each Group)

How Does this Method Work?

The first method for emptying the grid works in a very logical way. Initially the algorithm will cycle through each row removing a random number from those rows. Then the algorithm will check for any full columns and remove a number from any that are present. Following this, the 3x3 regions would be checked for being full or not and if some are then a number will also be removed from each full square. A possible addition to this method could be to add a further step in the event of a rare scenario using the 'Phistomefel Ring'. This would work by determining if the four 2x2 areas in all corners of the grids are full and if they are and the 'Phistomefel Ring' is also full of numbers then a random co-ordinate would be selected from the four areas and the ring, and the number removed from that location.

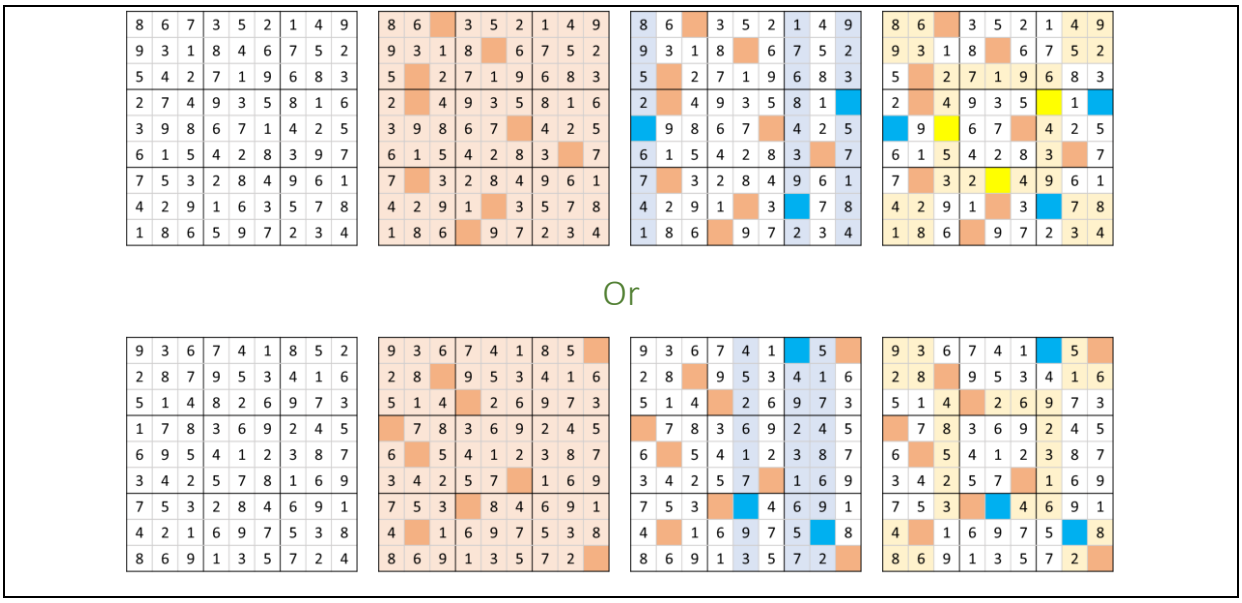
Advantages

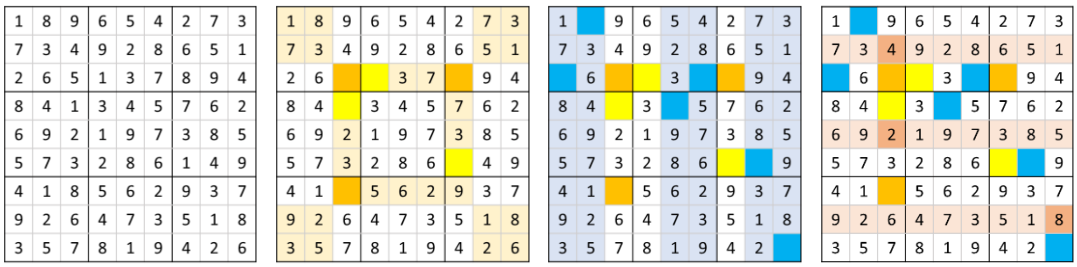
- Will produce a random valid grid.
- It won't create patterns.

Disadvantages

- The puzzle would be too easy to solve.
- Not very time efficient.

How The Algorithm Would Look





The above 4 images show an alternative order of operations, reversing the originally planned order. Both can produce a puzzle with a similar number of empty locations to be solved, yet, starting with the 'Phistomefel Ring' could produce a slightly more challenging puzzle as there will be more steps to solve parts of the puzzle towards the end of solving it.

Emptying the Grid

- Method 2 (Removing Random Numbers)

How Does this Method Work?

This method is very simple in theory. A quantity of numbers to be removed must first be decided depending on the required difficulty of the puzzle. Then of the quantity, coordinates must be randomised for numbers to be removed in those locations. The algorithm should then check to see if the puzzle is then possible and if it isn't then repeat from the start until a solvable puzzle is the outcome.

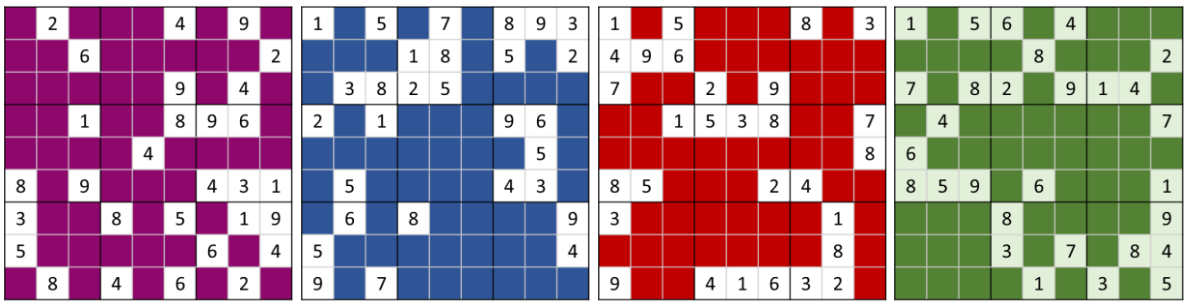
Advantages

- The difficulty can be selected.
- The puzzle will be unique.

Disadvantages

- The algorithm wouldn't be time efficient to run.

How The Algorithm Would Look



Unsolvable	Unsolvable	Unsolvable	**Solvable**

Pseudocode for the Main Subroutine

```
Sub EmptyGrid()
        Dim TempIntStore As Integer = 0
        Dim ValidPuzzle As Boolean = False
        While ValidPuzzle = False
                For count = 0 To 80
                        RandomLocation = Cint(Rnd()*80)
                        TempIntStore = ListOfLocations(count)
                        ListOfLocations(count) = ListOfLocations(RandomLocation)
                        ListOfLocations(RandomLocation) = TempIntStore
                Next
                For count = 1 To QuantOfNumbersToRemove
                        Grid(ListOfLocations(count)) = 0
                Next
                If CheckValidGrid(Grid()) = True
                        ValidPuzzle = True
                End If
        End While
End Sub
```

Emptying the Grid

- Method 3 (Linearly Removing Numbers)

How Does this Method Work?

Method 3 works in a way which every row and column will be linearly cycled through and, one-by-one, each number is checked to see if it can be removed from the grid whilst upholding the validity of the grid and ability to solve the puzzle.

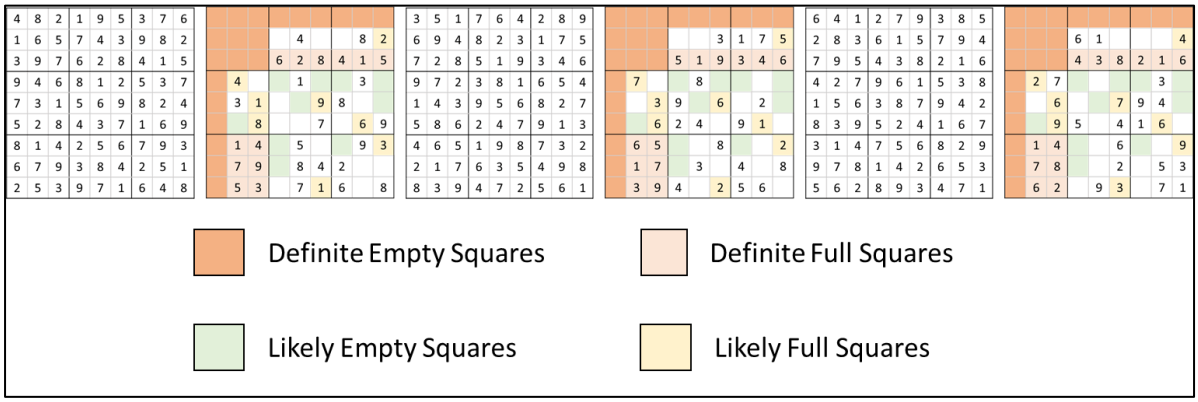
Advantages

- This method is time efficient.
- The puzzle will be unique.

Disadvantages

- Patterns will be formed with this algorithm.

How The Algorithm Would Look



From viewing how this algorithm would look and the patterns that would form using this method, there is a way that this algorithm can be optimised. The improved version should, from the start, empty all locations which are definite empty squares and confirm the locations of all numbers in the definite full squares. The algorithm should then cycle through the other locations linearly following the original approach of removing numbers.

Pseudocode for the Main Subroutine

```
Sub EmptyGrid()
        Dim ConfirmedLocations(8,8) As Boolean = False
        For a = 0 To 2
                For b = 0 To 2
                        Grid(a,b) = 0
                        ConfirmedLocations(a,b) = True
                Next
        Next
        For a = 3 To 8
                Grid(0,a) = 0
                ConfirmedLocations(0,a) = True
                ConfirmedLocations(2,a) = True
                Grid(a,0) = 0
                ConfirmedLocations(a,0) = True
        Next
        For a = 6 To 8
                For b = 1 To 2
                        ConfirmedLocations(a,b)
                Next
        Next
        For a = 0 To 8
                For b = 0 To 8
                        If CanRemove(a,b,Grid) = True
                                Grid(a,b) = 0
                        End If
                        ConfirmedLocations(a,b) = True
                Next
        Next
End Sub
```

Emptying the Grid

- Method 4 (Removing Translated Groups)

How Does this Method Work?

Method 4 groups together random collections of numbers and their translated counterparts throughout the grid and determines whether they can be removed from the grid or not to create the puzzle.

Advantages

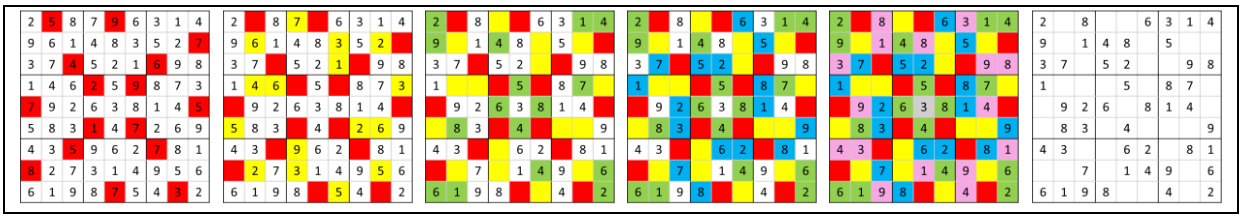
- The likely difficulty can be predicted by the size of the groups of numbers.
- The algorithm is time predictable.

Disadvantages

- The more difficult the puzzle, the longer the algorithm will take to run
- Patterns will be formed.

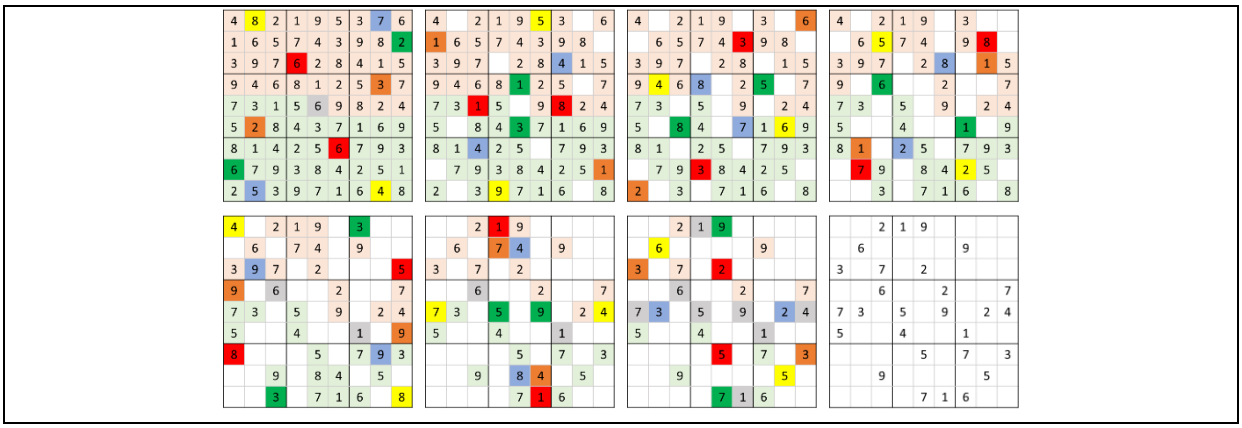
How The Algorithm Would Look

Version 1



Version 1 uses an area of 4x5 (20 locations) to choose 4 randomly selected numbers from. 12 more numbers are then selected which are the translated counterparts to the original 4 numbers in the 3 other 4x5 regions. This larger group of 16 numbers is then tested to see whether they can be removed and if so, they are. This is then repeated 4 further times identically, where the random numbers were previously unconsidered to be removed.

Version 2



Version 2 works in a very similar way to version 1 yet, instead of 4 areas of 4x5, this version uses 2 larger areas of 40 locations. This method also doesn't group multiple numbers in each region and instead pairs only 2 numbers (1 in each area) and checks to see if each pair can be removed or not. Furthermore, the central number, in both versions, acts alone and can be removed/confirmed at any stage of this algorithm yet, the earlier, the likelier the number is removed.

Emptying the Grid

- Method 5 (Removing Individual Random Numbers)

How Does this Method Work?

This method randomises a list of the 81 coordinates and checks the locations, in the randomised order, to see whether it is possible to remove them from the grid whilst keeping a sudoku puzzle that is both valid and solvable.

Advantages.

- The algorithm is easy to code and expand/understand.

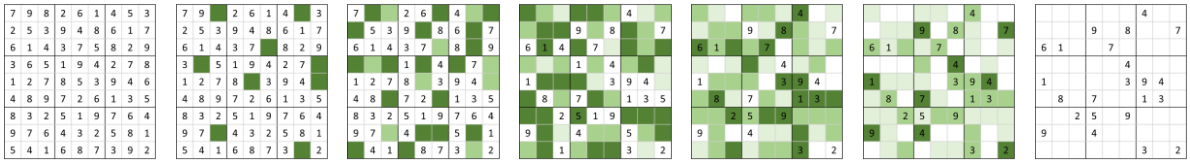
Disadvantages

- Other methods are more time efficient & predictable.

Pseudocode for the Main Subroutine

```
Sub EmptyGrid()
        Dim ListOfRows(80) As Integer
        Dim ListOfColumns(80) As Integer
        For a = 1 To 81
                ListOfRows(a - 1) = (a MOD 9) + 1
                ListOfColumns(a-1) = (a MOD 9) + 1
        Next
        For count = 0 To 80
                RandomNumber = Cint(Rnd()*80)
                TempStoreInt = ListOfRows(count)
                ListOfRows(count) = ListOfRows(RandomNumber)
                ListOfRows(RandomNumber) = TempStoreInt
                RandomNumber = Cint(Rnd()*80)
                TempStoreInt = ListOfColumns(count)
                ListOfColumns(count) = ListOfColumns(RandomNumber)
                ListOfColumns(RandomNumber) = TempStoreInt
        Next
        For a = 0 To 80
                        CurrentRow = ListOfRows(a)
                        CurrentCol = ListOfColumns(a)
                        If CheckValidRemoval(CurrentRow,CurrentCol) = True
                                Grid(CurrentRow,CurrentCol) = 0
                        End If
        Next
End Sub
```

The pseudocode displayed above is a representation of what the algorithm could be structured like in the main subroutine. First, 2 arrays of size 81, are filled with the 81 x-coordinates and 81 y-coordinates (Rows and Columns) using a 'For Loop' and the 'MOD' function. Then the two arrays are shuffled and the order within them is randomised independently of each other. Finally, these two, randomised arrays are now used in their new order together, to cycle though 81 co-ordinates and either confirm or remove the numbers in those positions. This will create a unique, random sudoku puzzle that is possible to solve.



Emptying the Grid

- Method 6 (Method 1 & 5 Together)

How Does this Method Work?

Method 6 combines the algorithms of method 1 & 5, to create an improved version of the 2. Method 1 can be used to initially remove numbers from the three main groups (Rows, Columns & 3x3 Regions) which won't require the need to be able to solve the grid, therefore saving time and, in turn, improving efficiency of the algorithm. Following this, Method 5 can be used to randomly remove numbers by randomising a list of coordinates and testing to see if, 1 by 1, the puzzle will be solvable without them in those locations. If the puzzle is possible to solve without the number, then it will be removed if the difficulty is required to be at maximum. This method aims to optimise 2 methods into one and create the most efficient approach to the problem of 'Emptying the Grid'. The combination of the two methods also provides a result of a puzzle that gets progressively easier, as a guarantee, as the puzzle progresses towards the completion.

Advantages.

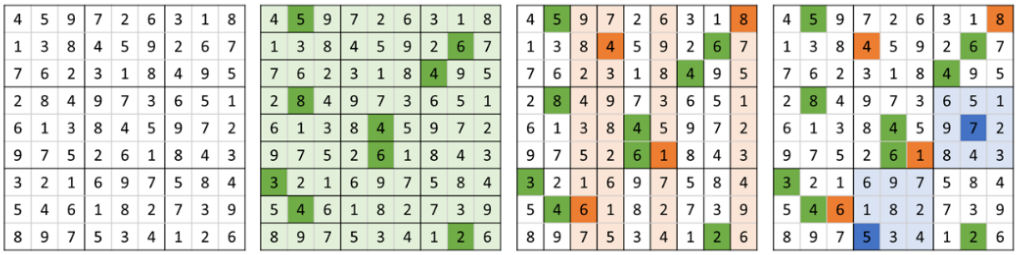
- This method provides a puzzle which is progressively easier.
- Puzzles are random with no patterns.

Disadvantages

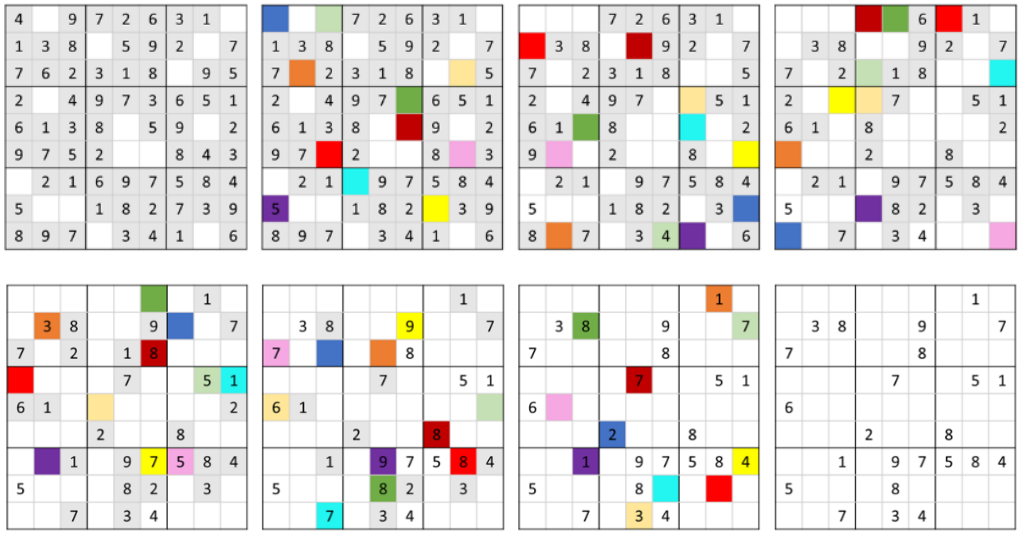
- This method is not the quickest approach to emptying a Sudoku grid.
- Other methods are easier to code.

How The Algorithm Would Look

Method 1



Method 5



Emptying the Grid

- Chosen Approach: **Method 6** (Methods 1 & 5)

Method	Why was the method not chosen?
1.	Although this method would create a valid puzzle, the puzzle wouldn't be very difficult to solve and, considering the result, the algorithm is very time inefficient.
2.	This method would also not be time efficient and so wouldn't be suitable to use for the final program.
3.	Even though, each puzzle will be created with a time efficient algorithm with this method and each puzzle will be unique, patterns are certain to form.
4.	A puzzle of any level of difficulty can be formed with yet, the method isn't time predictable and so isn't suitable for this program.
5.	Compared to other methods, this method would be simple to code, however other methods work in a more time efficient way.

Why Was Method 6 Chosen?

Method 6 is a combination of both methods 1 & 5. First all full rows, columns and 3x3 squared regions have one number removed from them. Once this is complete, and at least nine numbers have been removed from the grid efficiently, the second part of the algorithm is run. This part shuffles a list of the coordinates that still have numbers located at them, and then this list is followed linearly and the number in the processed location is removed if the puzzle can still be solved without it. If the number cannot be removed, then it will remain and will make up a part of the puzzle. The puzzles that are created using this method won't show any patterns because every selection that is made in the algorithm uses a random number generator. This is suitable for the program as; a fair level of difficulty should remain in the puzzles which can be achieved when there are no patterns present in the sudoku puzzles. The algorithm also produces puzzles that will get progressively easier which is ideal in a sudoku puzzle, to start off, having to use the more technical skills and the more effort that is inputted into the puzzle, the easier it should become. Nevertheless, this approach will not be as easy to code when compared to other methods yet, this method being the most efficient will expectantly be worth the time to code. Furthermore, method 6 doesn't provide the puzzle in the quickest of ways, yet, due to the same reasons as have just been mentioned, there should be a pay off to this in the product that the program will produce.

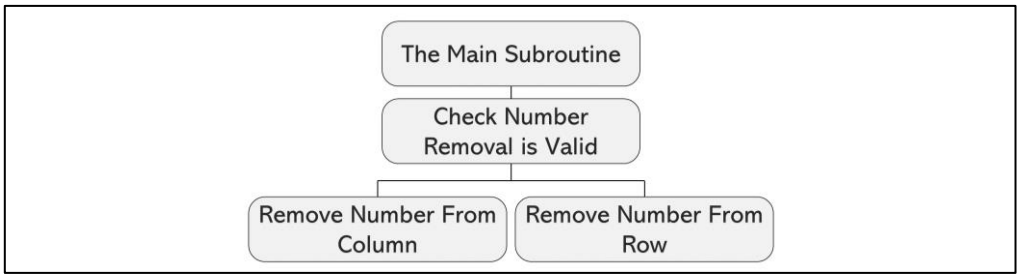
Pseudocode for the Main Function

```
Function CreatePuzzleMethod6(initialGrid As Integer(,)) As Integer(,)
    Dim grid As Integer(,) = CopyGrid(initialGrid)
    Dim remainingCoords As List(Of Tuple(Of Integer, Integer)) =
GetFilledCellCoordinates(grid)

    ' Part 1: Removing numbers from full rows, columns, and regions
    For Each coord In remainingCoords
        RemoveNumberFromRowColumnRegion(grid, coord.Item1,
coord.Item2)
    Next

    Dim numbersRemoved = 0
    Dim shuffledCoords As List(Of Tuple(Of Integer, Integer)) =
ShuffleCoordinates(remainingCoords)

    ' Part 2: Iteratively removing numbers and checking solvability
    For Each coord In shuffledCoords
        If CanRemoveNumber(grid, coord.Item1, coord.Item2) Then
            grid(coord.Item1, coord.Item2) = 0
            numbersRemoved += 1
            If numbersRemoved >= 9 Then Exit For
        End If
    Next

    Return grid
End Function

' Additional functions as mentioned before...
```

Emptying the Grid – Proof of Concept Program

- The Code

Structure Chart



Declaring Global Variables

```
Dim ROne() As Integer = {1, 2, 5, 6, 7, 4, 8, 9, 3}
Dim RTwo() As Integer = {4, 9, 6, 1, 8, 3, 5, 7, 2}
Dim RThree() As Integer = {7, 3, 8, 2, 5, 9, 1, 4, 6}
Dim RFour() As Integer = {2, 4, 1, 5, 3, 8, 9, 6, 7}
Dim RFive() As Integer = {6, 7, 3, 9, 4, 1, 2, 5, 8}
Dim RSix() As Integer = {8, 5, 9, 7, 6, 2, 4, 3, 1}
Dim RSeven() As Integer = {3, 6, 4, 8, 2, 5, 7, 1, 9}
Dim REight() As Integer = {5, 1, 2, 3, 9, 7, 6, 8, 4}
Dim RNine() As Integer = {9, 8, 7, 4, 1, 6, 3, 2, 5}
Dim PMgrid(8, 8, 8) As Boolean
Dim Grid(8, 8) As Integer
```

The Main Subroutine

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Randomize()
        For a = 0 To 8
            For b = 0 To 8
                If a = 0 Then
                    Grid(a, b) = ROne(b)
                ElseIf a = 1 Then
                    Grid(a, b) = RTwo(b)
                ElseIf a = 2 Then
                    Grid(a, b) = RThree(b)
                ElseIf a = 3 Then
                    Grid(a, b) = RFour(b)
                ElseIf a = 4 Then
                    Grid(a, b) = RFive(b)
                ElseIf a = 5 Then
                    Grid(a, b) = RSix(b)
                ElseIf a = 6 Then
                    Grid(a, b) = RSeven(b)
                ElseIf a = 7 Then
                    Grid(a, b) = REight(b)
                ElseIf a = 8 Then
                    Grid(a, b) = RNine(b)
                End If
            Next
        Next
```

The code for the Main Subroutine is continued on the following 3 pages…

Emptying the Grid – Proof of Concept Program

- The Code

Continued code from the Main Subroutine

```
        '''''This part of the program will remove parts of the grid
        'Removes a random number from each row
        For count = 0 To 8
            Grid(CInt((Rnd() * 8)), count) = 0
        Next
        'Removes a random number from each full column
        Dim NumsInColumn As Integer = 0
        For count = 0 To 8
            NumsInColumn = 0
            For a = 0 To 8
                If Grid(count, a) <> 0 Then
                    NumsInColumn += 1
                End If
                If NumsInColumn = 9 Then
                    Grid(count, CInt((Rnd() * 8))) = 0
                End If
            Next
        Next
        'Selects a random order to cycle through the grid and if there are 4
points locatimng that one position then it will move on
        Dim Columns() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
        Dim Rows() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
        Dim TempIntStr As Integer = 0
        Dim RandomNumber As Integer = 0
        For countC = 0 To 8
            RandomNumber = CInt((Rnd() * 8))
            TempIntStr = Columns(countC)
            Columns(countC) = Columns(RandomNumber)
            Columns(RandomNumber) = TempIntStr
        Next
        For countR = 0 To 8
            RandomNumber = CInt((Rnd() * 8))
            TempIntStr = Columns(countR)
            Columns(countR) = Columns(RandomNumber)
            Columns(RandomNumber) = TempIntStr
        Next
        Dim CurrentRow As Integer = 0
        Dim CurrentCol As Integer = 0
        Dim CurrentNum As Integer = 0
        For a = 0 To 8
            For b = 0 To 8
                CurrentRow = Rows(a) - 1
                CurrentCol = Columns(b) - 1
                CurrentNum = Grid(CurrentRow, CurrentCol)
                If CurrentNum <> 0 Then
                    If CheckRemovalIsValid(CurrentCol, CurrentRow, CurrentNum) =
True Then
                        Grid(CurrentRow, CurrentCol) = 0
                    End If
                End If
            Next
        Next
```

Emptying the Grid – Proof of Concept Program

- The Code

Continued code from the Main Subroutine

```
        'Checks to see if there is a full 3x3 region and if there is, remove a
random number
        Dim NumOfEmpty As Integer = 0
        For r = 0 To 2
            For c = 0 To 2
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2), CInt(Rnd() * 2)) = 0
        End If
        NumOfEmpty = 0
        For r = 0 To 2
            For c = 3 To 5
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2), CInt(Rnd() * 2) + 3) = 0
        End If
        NumOfEmpty = 0
        For r = 0 To 2
            For c = 6 To 8
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2), CInt(Rnd() * 2) + 6) = 0
        End If
        NumOfEmpty = 0
        For r = 3 To 5
            For c = 0 To 2
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2) + 3, CInt(Rnd() * 2)) = 0
        End If
        NumOfEmpty = 0
        For r = 3 To 5
            For c = 3 To 5
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
```

Emptying the Grid – Proof of Concept Program

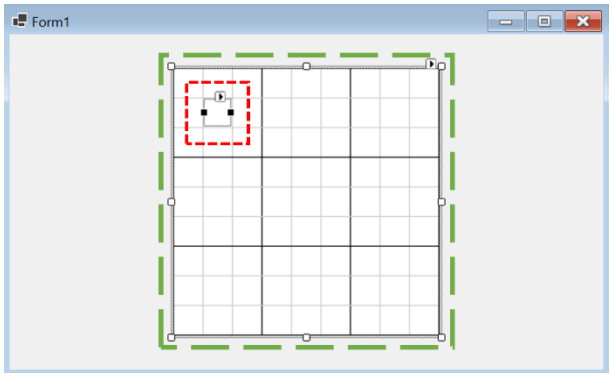
- The Code

Continued code from the Main Subroutine

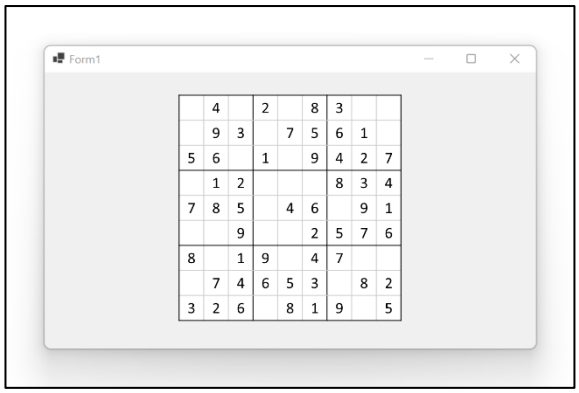
```
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2) + 3, CInt(Rnd() * 2) + 3) = 0
        End If
        NumOfEmpty = 0
        For r = 3 To 5
            For c = 6 To 8
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2) + 3, CInt(Rnd() * 2) + 6) = 0
        End If
        NumOfEmpty = 0
        For r = 6 To 8
            For c = 0 To 2
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2) + 6, CInt(Rnd() * 2)) = 0
        End If
        NumOfEmpty = 0
        For r = 6 To 8
            For c = 3 To 5
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2) + 6, CInt(Rnd() * 2) + 3) = 0
        End If
        NumOfEmpty = 0
        For r = 6 To 8
            For c = 6 To 8
                If Grid(r, c) = 0 Then
                    NumOfEmpty += 1
                End If
            Next
        Next
        If NumOfEmpty = 0 Then
            Grid(CInt(Rnd() * 2) + 6, CInt(Rnd() * 2) + 6) = 0
        End If
        NumOfEmpty = 0
End Sub
```

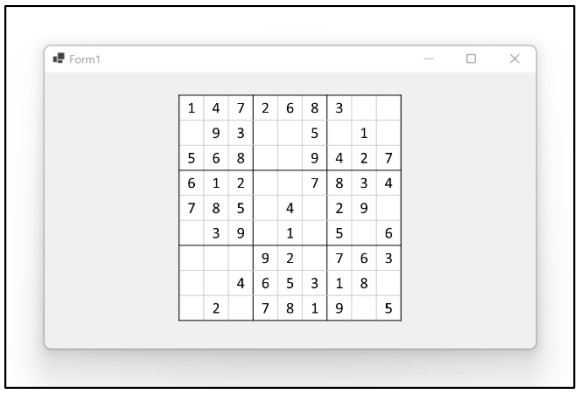
Emptying the Grid – Proof of Concept Program

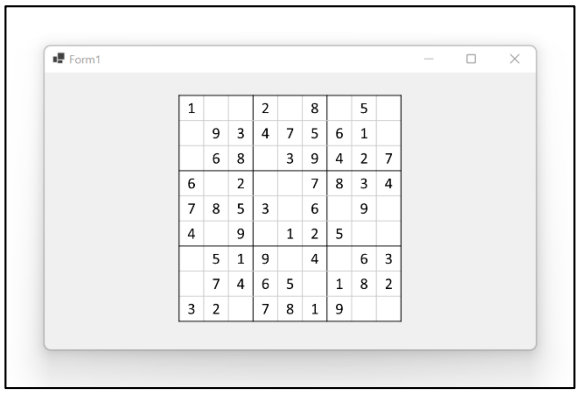
- The User Interface

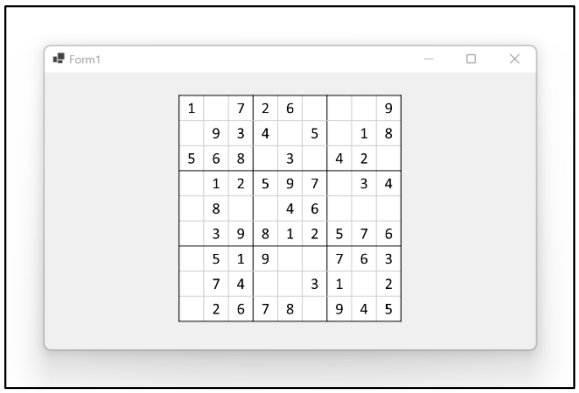


This is the design for the UI which is identical to the UI used for the concept program to fill the grid. More annotations to this alike user interface can be found on page 29

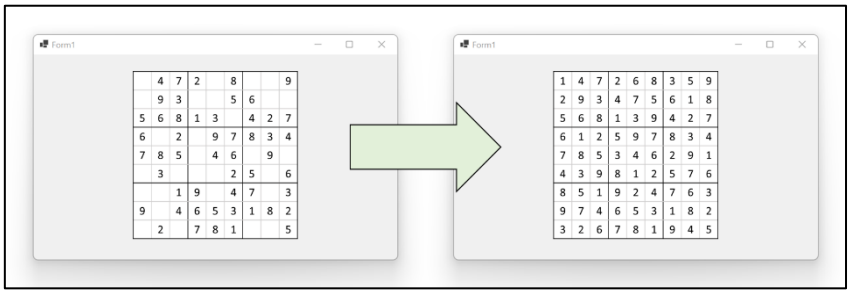








The 4 images above display different and independent outcomes produced by the program. All of these puzzles are: completely possible; random in the result and unique to the point that two duplicate puzzles would be very unlikely. Below is proof that the puzzles created with this new algorithm are possible.



Solving the Grid

- Method 1 (Random Input Attempts)

How Does this Method Work?

This Method will linearly move throughout the grid, along the rows, from top to bottom, and input a random number into each empty location until the grid is full of numbers. This will then be checked to be valid or not using the fundamental rules of a Sudoku Puzzle. If the puzzle is deemed to be valid then the sudoku is solved yet, if the puzzle is invalid, the cycle will repeat until the puzzle is valid. There is also an addition to this algorithm that needs to be made which is the part that is required to determine if the puzzle inputted into the algorithm is solvable or not. This could be achieved by having a limit to the number of cycles possible before ending and determining that the puzzle isn't solvable, this limit should be high enough so that a solvable puzzle is very likely to be recognized but not too high as to take an unreasonable time to compute in the event of an unsolvable possible, which would be a common occurrence.

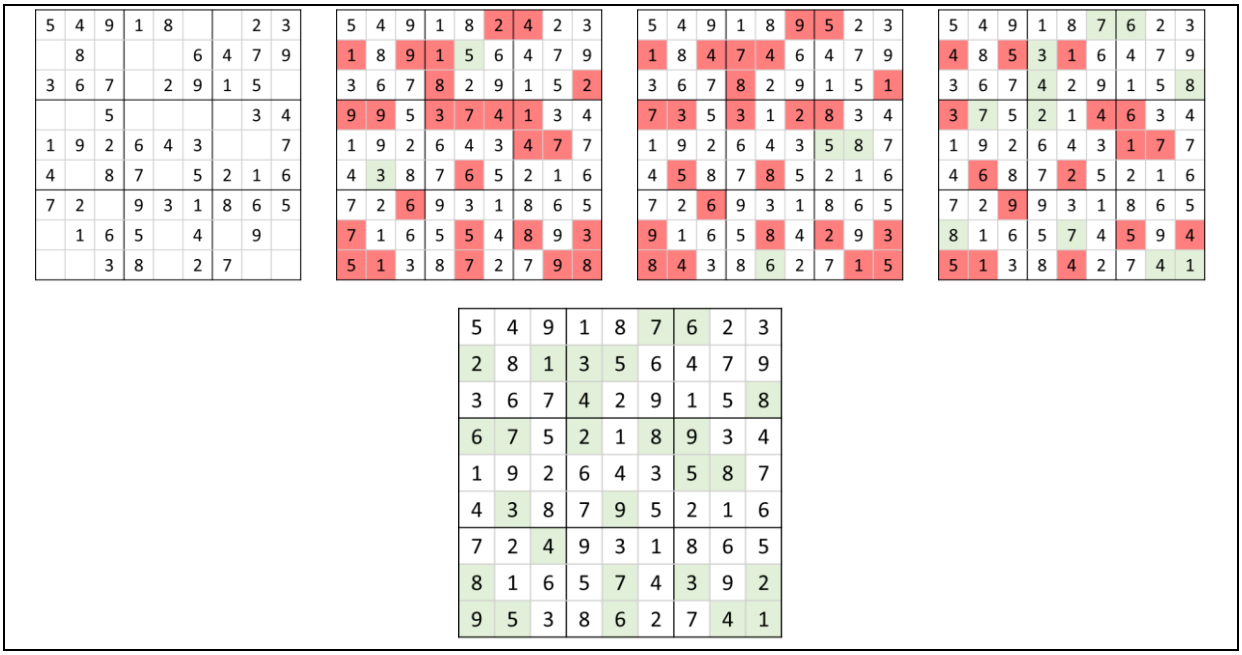
Advantages.

- This algorithm will return a guaranteed solved grid when called.

Disadvantages

- This method is not time predictable
- Other methods are likely quicker.

How The Algorithm Would Look



The above 5 diagrams display how Method 1 would run. The first image, located in the top left is the grid that is imported into the algorithm, being the grid that is required to be solved. The 2nd, 3rd, 4th and 5th diagrams are then examples of how the algorithm would look step by step with different attempts showing different levels of success until eventually, in the final 5th diagram, there is a successful attempt at solving the grid, and the algorithm would terminate.

Solving the Grid

- Method 2 (Undo if no Inputs)

How Does this Method Work?

The flow of this algorithm will move across the rows, left to right. For each empty location in the grid, a valid number will be inputted based on the fundamental rules of a Sudoku Puzzle. If there is no valid number possible to input, then there will be a backtrack starting at a depth of 1 and increasing until the algorithm moves past the location at which no number can be inputted that is valid or, another alike coordinate is located elsewhere in the grid. This will loop until the grid is full and valid or has concluded that the inputted puzzle isn't solvable therefore, there could be a limited number of backtracks possible before the puzzle is deemed unsolvable.

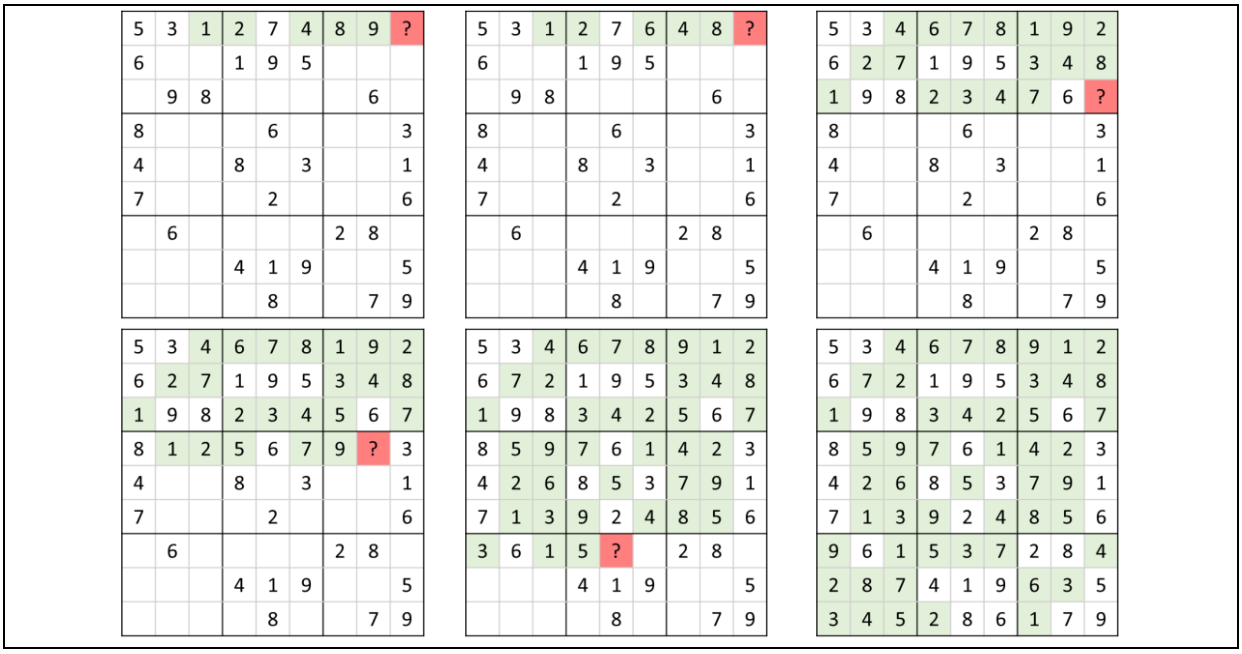
Advantages.

- The amount of time that this algorithm would take to run could be much quicker than the others.

Disadvantages

- This method is not time predictable.
- This method is much more difficult to code than the others.

How The Algorithm Would Look



The above 6 diagrams show how the second method works with the use of visual aid. Each image shows the progression of the algorithm, and, at each question mark, the process is backtracked, until possible again, and then re-attempted until either: there is another empty location at which no number can be inputted or, the gird is completely solved and there are no more empty locations left in the grid. The latter is displayed in the 6th image.

Solving the Grid

- Method 3 (Attempting Different Approaches)

How Does this Method Work?

Method 3 uses a loop to cycle through three approaches of calculating the number of pencil marks in different groups and altering the grid accordingly, if necessary. The first approach checks each 3x3 region to find a number that only appears once as a pencil mark. Every time this is the case, that number is inputted in that location. The second approach does the same but for every row instead of regions, and the third approach uses all 9 columns. After every approach, the grid is refreshed with pencil marks. This entire loop needs to be repeated enough times to guarantee a solved grid if the puzzle is valid.

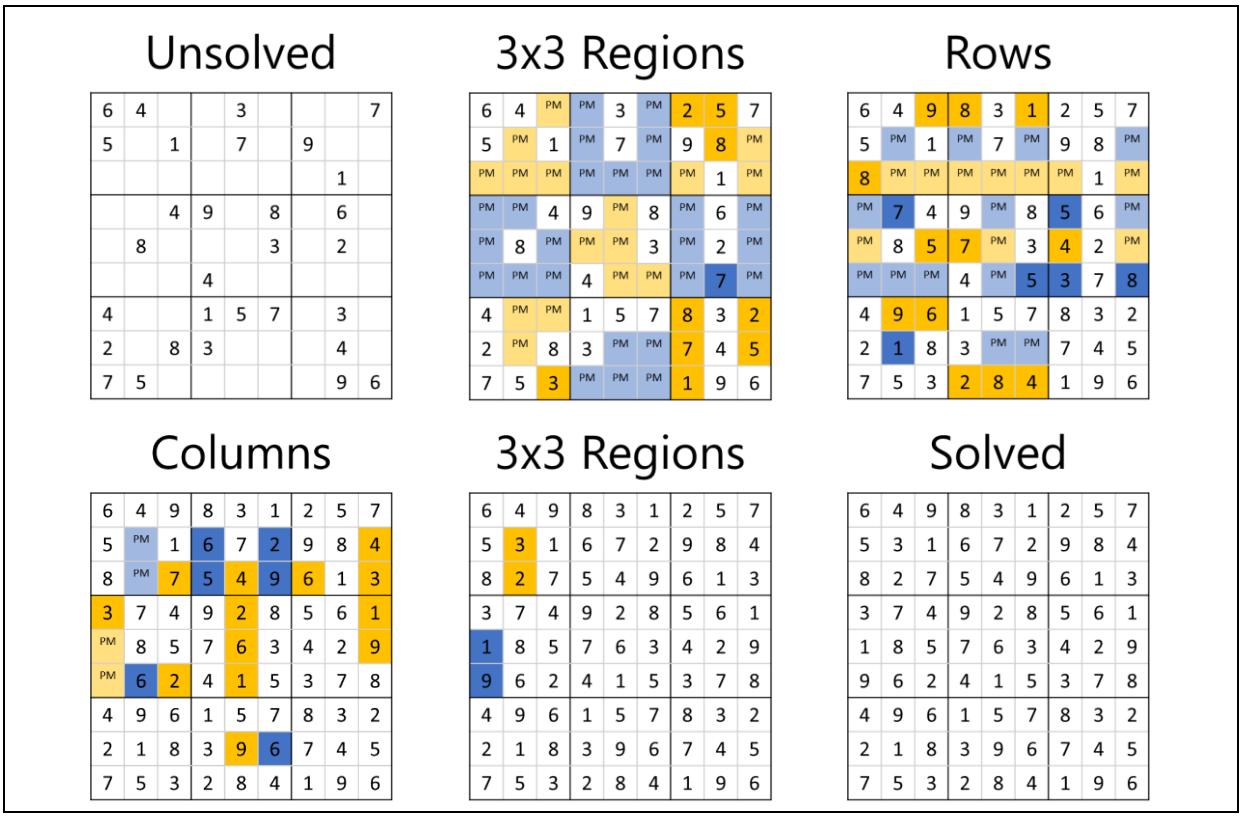
Advantages.

- On average, this algorithm is the fastest, of the three methods, to run.
- This method is predictable.

Disadvantages

- This method is very complex to code because of the quantity of code that would need to be written.

How The Algorithm Would Look



The first diagram above shows the inputted grid into method 3. The loop of 6 steps is then begun. The next 3 images, display the first loop by the algorithm. The 2nd image first pencil marks the grid, and then moves through each, separated 3x3 regions, and if there are any obvious numbers to input, then they are. After the grid is pencil marked again, in the 3rd image, the grid is then split into 9 rows and the same process of inputting valid numbers is done. Before the same is repeated with columns instead. This whole process is then repeated until the algorithm outputs a solved grid, as displayed in image 6.

Solving the Grid

- Chosen Approach: **Method 3** (Attempting Different Approaches)

Method	Why was the method not chosen?
1.	Although this algorithm will work and produce a solved puzzle eventually when run, it isn't time predictable and, is not unlikely to take an unreasonable amount of time, in comparison to the chosen method, which is why method 1 wasn't chosen.
2.	This method has the potential to be much quicker than the other 2, yet this algorithm is not time predictable and would take much more time to code than the other two.

Why Was Method 3 Chosen?

In the pursuit of an optimal Sudoku puzzle-solving strategy, Method 3, 'Attempting Different Approaches,' emerges as the chosen path due to its balanced blend of efficiency, predictability, and a strategic approach that minimizes complexity while maintaining effectiveness.

In evaluating different methods, Method 1, 'Random Input Attempts,' stands as an intuitive yet unpredictable choice. While guaranteeing eventual grid resolution, its non-deterministic runtime raises efficiency concerns. The reliance on random inputs introduces variability in the time taken to find a solution, making it less favourablee for consistent performance. Meanwhile, Method 2, 'Undo if no Inputs,' offers potential speed gains. However, this advantage comes at the cost of complexity. Implementing a robust backtracking mechanism necessitates intricate coding, presenting challenges in clarity and reliability. This complexity might offset speed gains and hinder development. Enter Method 3, a middle ground optimizing algorithmic principles and pragmatic execution. By sequentially cycling through distinct approaches—analyzing 3x3 regions, rows, and columns—Method 3 narrows possible numbers for each cell. This systematic process relies on logical deduction, reducing computational dead ends.

Method 3's strength lies in its calculated, strategic nature. It aligns with Sudoku's rules while avoiding exhaustive methods. The approach's iterative loop enhances runtime predictability, crucial for resource allocation. Dependable time estimates aid planning and efficient resource utilization. Method 3 demands extensive coding due to multifaceted execution. Encompassing three techniques requires thoughtful coding. Yet, this investment translates into maintainable solutions. Method 3's modularity streamlines debugging, optimization, and future enhancements. Visualizing Method 3 underscores its step-by-step nature. Sequentially targeting regions, rows, and columns highlights its methodical approach. Images showcase each cycle's journey towards a solution. This visualization captures Method 3's elegance, systematically converging to a solution.

In conclusion, Method 3 embodies an efficient, predictable, strategic Sudoku-solving approach. It avoids other methods' drawbacks by embracing logical deduction, ensuring consistent runtime predictability, and promoting maintainable code. Method 3 strikes a balance aligned with crafting an effective Sudoku puzzle-solving program.

Solving the Grid – Proof of Concept Program

- The Code

Declaring Global Variables

```
Dim RowOne() As Integer = {0, 5, 8, 0, 3, 0, 0, 2, 0}
    Dim RowTwo() As Integer = {4, 0, 2, 0, 0, 0, 9, 0, 5}
    Dim RowThree() As Integer = {0, 0, 7, 0, 0, 0, 6, 8, 0}
    Dim RowFour() As Integer = {2, 9, 0, 0, 5, 4, 0, 7, 0}
    Dim RowFive() As Integer = {5, 0, 0, 0, 6, 2, 0, 0, 0}
    Dim RowSix() As Integer = {0, 0, 3, 8, 1, 0, 2, 5, 0}
    Dim RowSeven() As Integer = {1, 0, 9, 0, 0, 3, 0, 6, 4}
    Dim RowEight() As Integer = {8, 6, 5, 4, 9, 0, 1, 3, 0}
    Dim RowNine() As Integer = {0, 7, 0, 0, 0, 6, 0, 0, 0}
    Dim SlvGrid(8, 8) As Integer
    Dim PencilMarks(8, 8, 8) As Boolean
    Dim ChangesMade As Boolean = False
```

The Main Subroutine

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        For count = 0 To 8
            SlvGrid(0, count) = RowOne(count)
            SlvGrid(1, count) = RowTwo(count)
            SlvGrid(2, count) = RowThree(count)
            SlvGrid(3, count) = RowFour(count)
            SlvGrid(4, count) = RowFive(count)
            SlvGrid(5, count) = RowSix(count)
            SlvGrid(6, count) = RowSeven(count)
            SlvGrid(7, count) = RowEight(count)
            SlvGrid(8, count) = RowNine(count)
        Next
        PencilMarkTheGrid()
        For count = 0 To 80
            PencilMarkTheGrid()
            CheckMultiAppearInRegion()
            PencilMarkTheGrid()
            CheckMultiAppearInRow()
            PencilMarkTheGrid()
            CheckMultiAppearInCol()
        Next
    End Sub
```

The provided code constitutes a proof-of-concept program aimed at solving Sudoku puzzles. It commences by declaring global variables to represent the puzzle grid and to track changes. The main subroutine, executed upon the form's loading, initializes the puzzle grid by populating it with given row values. The program then employs an iterative approach to iteratively refine possibilities. The "PencilMarkTheGrid" function is utilized to mark possible numbers for each cell. Subsequent checks for multiple appearances of numbers in regions, rows, and columns aid in updating pencil marks. Through this systematic process of deduction and analysis, the program endeavors to successfully solve the Sudoku puzzle.

Solving the Grid – Proof of Concept Program

- The Code

Checking for Multiple Appearances of Numbers in Columns

```
Sub CheckMultiAppearInCol()
        Dim QuantNumInCol As Integer = 0
        Dim R As Integer = 0
        Dim C As Integer = 0
        For num = 0 To 8
            For b = 0 To 8
                For a = 0 To 8
                    PencilMarkTheGrid()
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInCol += 1
                        R = a
                        C = b
                    End If
                Next
                If QuantNumInCol = 1 Then
                    SlvGrid(R, C) = num + 1
                    PencilMarkTheGrid()
                End If
                    QuantNumInCol = 0
            Next
            PencilMarkTheGrid()
        Next
        PencilMarkTheGrid()
        '''''Checking for only one pencil mark in a square
        Dim NumberToInput As Integer = 0
        For a = 0 To 8
            For b = 0 To 8
                QuantNumInCol = 0
                For num = 0 To 8
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInCol += 1
                        NumberToInput = num
                    End If
                Next
                If QuantNumInCol = 1 Then
                    SlvGrid(a, b) = NumberToInput + 1
                    PencilMarkTheGrid()
                End If
            Next
        Next


    End Sub
```

The provided code establishes a proof-of-concept program with the goal of solving Sudoku puzzles. It begins by declaring global variables to represent the puzzle grid and to monitor changes. Executed upon the form's loading, the main subroutine initializes the puzzle grid by populating it with predefined row values. Employing an iterative approach, the program systematically refines potential solutions. The "PencilMarkTheGrid" function is employed to designate possible numbers for each cell. Subsequent checks for number repetition within regions, rows, and columns facilitate updates to these pencil marks. This structured process of deduction and analysis drives the program's pursuit of a successful Sudoku puzzle resolution.

Solving the Grid – Proof of Concept Program

- The Code

Checking for Multiple Appearances of Numbers in Rows

```
Sub CheckMultiAppearInRow()
        Dim QuantNumInRow As Integer = 0
        Dim R As Integer = 0
        Dim C As Integer = 0
        For num = 0 To 8
            For a = 0 To 8
                For b = 0 To 8
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRow += 1
                        R = a
                        C = b
                    End If
                Next
                If QuantNumInRow = 1 Then
                    SlvGrid(R, C) = num + 1
                    PencilMarkTheGrid()
                End If
                QuantNumInRow = 0
            Next
            PencilMarkTheGrid()
        Next
        PencilMarkTheGrid()
        '''''Checking for only one pencil mark in a square
        Dim NumberToInput As Integer = 0
        For a = 0 To 8
            For b = 0 To 8
                QuantNumInRow = 0
                For num = 0 To 8
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRow += 1
                        NumberToInput = num
                    End If
                Next
                If QuantNumInRow = 1 Then
                    SlvGrid(a, b) = NumberToInput + 1
                    PencilMarkTheGrid()
                End If
            Next
        Next
    End Sub
```

The provided code segment, "CheckMultiAppearInRow()," scans rows in the Sudoku grid for multiple appearances of numbers. It detects instances where a number appears only once, marking it as a potential solution for that cell. By iteratively analyzing each row and coordinating with "PencilMarkTheGrid()," the code aims to systematically reduce possibilities and contribute to solving the Sudoku puzzle.

Solving the Grid – Proof of Concept Program

- The Code

Checking for Multiple Appearances of Numbers in Regions

```
Sub CheckMultiAppearInRegion()
        Dim QuantNumInRegion As Integer = 0
        Dim R As Integer = 0
        Dim C As Integer = 0
        ChangesMade = False
        '''''Checking for only one appearance of a number in a region
        'Check Region 1:
        For num = 0 To 8
            For a = 0 To 2
                For b = 0 To 2
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        R = a
                        C = b
                    End If
                Next
            Next
            If QuantNumInRegion = 1 Then
                SlvGrid(R, C) = num + 1
                PencilMarkTheGrid()
            End If
            QuantNumInRegion = 0
        Next
        QuantNumInRegion = 0
        'Check Region 2:
        For num = 0 To 8
            For a = 0 To 2
                For b = 3 To 5
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        R = a
                        C = b
                    End If
                Next
            Next
            If QuantNumInRegion = 1 Then
                SlvGrid(R, C) = num + 1
                PencilMarkTheGrid()
            End If
            QuantNumInRegion = 0
        Next
        QuantNumInRegion = 0
```

Solving the Grid – Proof of Concept Program

- The Code

Continued code from the Previous Page

```
        'Check Region 3:
         For num = 0 To 8
             For a = 0 To 2
                 For b = 6 To 8
                     If PencilMarks(a, b, num) = True Then
                         QuantNumInRegion += 1
                         R = a
                         C = b
                     End If
                 Next
             Next
             If QuantNumInRegion = 1 Then
                 SlvGrid(R, C) = num + 1
                 PencilMarkTheGrid()
             End If
             QuantNumInRegion = 0
         Next
         QuantNumInRegion = 0
         'Check Region 4:
         For num = 0 To 8
             For a = 3 To 5
                 For b = 0 To 2
                     If PencilMarks(a, b, num) = True Then
                         QuantNumInRegion += 1
                         R = a
                         C = b
                     End If
                 Next
             Next
             If QuantNumInRegion = 1 Then
                 SlvGrid(R, C) = num + 1
                 PencilMarkTheGrid()
             End If
             QuantNumInRegion = 0
         Next
         QuantNumInRegion = 0
         'Check Region 5:
         For num = 0 To 8
             For a = 3 To 5
                 For b = 3 To 5
                     If PencilMarks(a, b, num) = True Then
                         QuantNumInRegion += 1
                         R = a
                         C = b
                     End If
                 Next
             Next
             If QuantNumInRegion = 1 Then
                 SlvGrid(R, C) = num + 1
                 PencilMarkTheGrid()
             End If
             QuantNumInRegion = 0
         Next
         QuantNumInRegion = 0
```

Solving the Grid – Proof of Concept Program

- The Code

Continued code from the Previous Page

```
        'Check Region 6:
        For num = 0 To 8
            For a = 3 To 5
                For b = 6 To 8
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        R = a
                        C = b
                    End If
                Next
            Next
            If QuantNumInRegion = 1 Then
                SlvGrid(R, C) = num + 1
                PencilMarkTheGrid()
            End If
            QuantNumInRegion = 0
        Next
        QuantNumInRegion = 0
        'Check Region 7:
        For num = 0 To 8
            For a = 6 To 8
                For b = 0 To 2
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        R = a
                        C = b
                    End If
                Next
            Next
            If QuantNumInRegion = 1 Then
                SlvGrid(R, C) = num + 1
                PencilMarkTheGrid()
            End If
            QuantNumInRegion = 0
        Next
        QuantNumInRegion = 0
        'Check Region 8:
        For num = 0 To 8
            For a = 6 To 8
                For b = 3 To 5
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        R = a
                        C = b
                    End If
                Next
            Next
            If QuantNumInRegion = 1 Then
                SlvGrid(R, C) = num + 1
                PencilMarkTheGrid()
            End If
            QuantNumInRegion = 0
        Next
        QuantNumInRegion = 0
```

Solving the Grid – Proof of Concept Program

- The Code

Continued code from the Previous Page

```
        'Check Region 9:
        For num = 0 To 8
            For a = 6 To 8
                For b = 6 To 8
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        R = a
                        C = b
                    End If
                Next
            Next
            If QuantNumInRegion = 1 Then
                SlvGrid(R, C) = num + 1
                PencilMarkTheGrid()
            End If
            QuantNumInRegion = 0
        Next
        QuantNumInRegion = 0


        PencilMarkTheGrid()
        '''''Checking for only one pencil mark in a square
        Dim NumberToInput As Integer = 0
        For a = 0 To 8
            For b = 0 To 8
                QuantNumInRegion = 0
                For num = 0 To 8
                    If PencilMarks(a, b, num) = True Then
                        QuantNumInRegion += 1
                        NumberToInput = num
                    End If
                Next
                If QuantNumInRegion = 1 Then
                    SlvGrid(a, b) = NumberToInput + 1
                    PencilMarkTheGrid()
                End If
            Next
        Next
    End Sub
```

The provided code segment, "CheckMultiAppearInRegion()," systematically scans Sudoku regions for instances of multiple number appearances. The subroutine assesses each of the nine regions independently. By analyzing pencil marks within each region's cells, the code identifies numbers that uniquely appear within that region. If a single number exclusively emerges, it's considered a potential solution for the corresponding cell. This process repeats for each region, enabling systematic deduction of solutions. The code then extends this analysis to individual cells, determining if only one pencil mark remains within a square. When such conditions are met, the number is directly inserted into the cell within the solved grid, followed by a call to "PencilMarkTheGrid()" for further updates. This meticulous approach significantly contributes to solving the Sudoku puzzle by reducing uncertainties in cell contents.

Solving the Grid – Proof of Concept Program

- The Code

Pencil Marking the Grid

```
Sub PencilMarkTheGrid()
        For a = 0 To 8
            For b = 0 To 8
                For c = 0 To 8
                    PencilMarks(a, b, c) = False
                Next
            Next
        Next
        For Num = 1 To 9
            For a = 0 To 8
                For b = 0 To 8
                    If SlvGrid(a, b) = 0 Then
                        If CheckRepetitionInCol(Num, a, b) = False Then
                            If CheckRepetitionInRow(Num, a, b) = False Then
                                If CheckRepetitioninSquare(Num, a, b) = True Then
                                    PencilMarks(a, b, Num - 1) = True
                                End If
                            End If
                        End If
                    End If
                Next
            Next
        Next
    End Sub
```

The "PencilMarkTheGrid()" code segment systematically marks potential numbers within the Sudoku grid. Through nested loops, the subroutine initializes pencil marks for all cells. It then iterates through each cell, considering numbers from 1 to 9 for empty cells. If a cell is vacant, it assesses column, row, and square regions for number repetitions. Pencil marks are enabled for numbers that meet the criteria, providing potential solutions for the cell. This methodical process contributes to narrowing down possibilities, enhancing the program's capability to solve the Sudoku puzzle by deducing appropriate numbers for empty cells.

Solving the Grid – Proof of Concept Program

- The Code

Checking for Repetitions in Squares

```
Function CheckRepetitioninSquare(CurrentNumber As Integer, Row As Integer, Column
As Integer)
        Dim ValidInSquare As Boolean = True
        Dim SqrStatus As Integer = 0
        If (Column + 1) Mod 3 = 1 Then
            If (Row + 1) Mod 3 = 1 Then
                SqrStatus = 1
            ElseIf (Row + 1) Mod 3 = 2 Then
                SqrStatus = 4
            Else
                SqrStatus = 7
            End If
        ElseIf (Column + 1) Mod 3 = 2 Then
            If (Row + 1) Mod 3 = 1 Then
                SqrStatus = 2
            ElseIf (Row + 1) Mod 3 = 2 Then
                SqrStatus = 5
            Else
                SqrStatus = 8
            End If
        Else
            If (Row + 1) Mod 3 = 1 Then
                SqrStatus = 3
            ElseIf (Row + 1) Mod 3 = 2 Then
                SqrStatus = 6
            Else
                SqrStatus = 9
            End If
        End If
        Dim QuantInSqr As Integer
        QuantInSqr = 0
        If SqrStatus = 1 Then
            For a = 0 To 2
                For b = 0 To 2
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        ElseIf SqrStatus = 2 Then
            For a = 0 To 2
                For b = -1 To 1
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
```

Solving the Grid – Proof of Concept Program

- The Code

Continued code from the Previous Page

```
        ElseIf SqrStatus = 3 Then
            For a = 0 To 2
                For b = -2 To 0
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        ElseIf SqrStatus = 4 Then
            For a = -1 To 1
                For b = 0 To 2
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        ElseIf SqrStatus = 5 Then
            For a = -1 To 1
                For b = -1 To 1
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        ElseIf SqrStatus = 6 Then
            For a = -1 To 1
                For b = -2 To 0
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        ElseIf SqrStatus = 7 Then
            For a = -2 To 0
                For b = 0 To 2
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        ElseIf SqrStatus = 8 Then
            For a = -2 To 0
                For b = -1 To 1
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
```

Solving the Grid – Proof of Concept Program

- The Code

Continued code from the Previous Page

```
        ElseIf SqrStatus = 9 Then
            For a = -2 To 0
                For b = -2 To 0
                    If SlvGrid(Row + a, Column + b) = CurrentNumber Then
                        QuantInSqr += 1
                    End If
                Next
            Next
        End If
        If QuantInSqr = 1 Then
            ValidInSquare = False
        End If
        Return ValidInSquare
    End Function
```

Checking for Repetitions in Columns

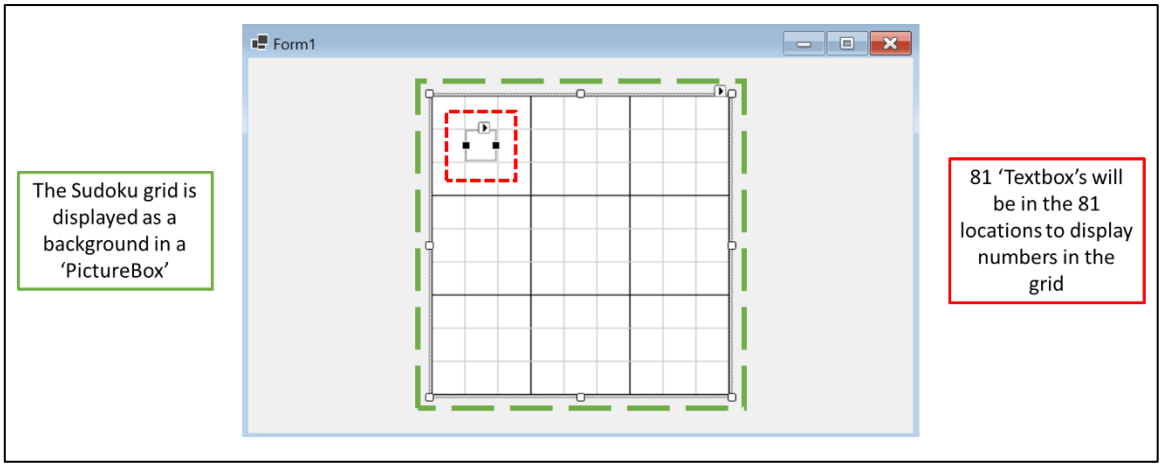
```
Function CheckRepetitionInCol(CurrentNumber As Integer, Row As Integer, Column As
Integer)
        Dim RepInCol As Boolean = False
        Dim NumCNinC As Integer = 0
        For count = 0 To 8
            If SlvGrid(count, Column) = CurrentNumber Then
                NumCNinC += 1
            End If
        Next
        If NumCNinC = 1 Then
            RepInCol = True
        End If
        Return NumCNinC
    End Function
```

Checking for Repetitions in Columns

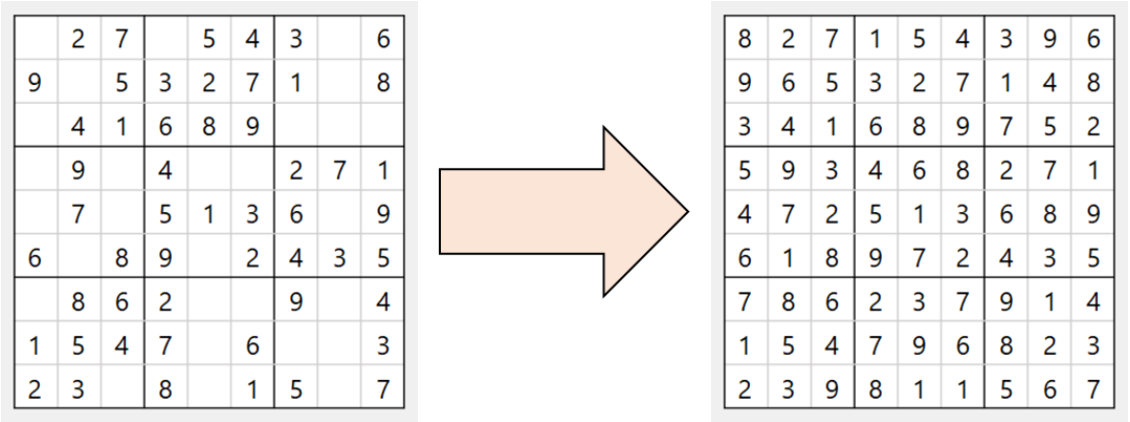
```
Function CheckRepetitionInRow(CurrentNumber As Integer, Row As Integer, Column As
Integer)
        Dim RepInRow As Boolean = False
        Dim NumCNinR As Integer = 0
        For count = 0 To 8
            If SlvGrid(Row, count) = CurrentNumber Then
                NumCNinR += 1
            End If
        Next
        If NumCNinR = 1 Then
            RepInRow = True
        End If
        Return NumCNinR
    End Function
```

Solving the Grid – Proof of Concept Program

- The User Interface



Proof the Program Works



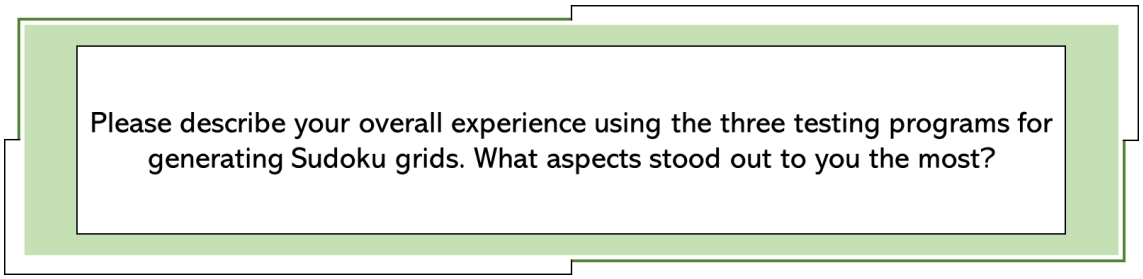
	2	7		5	4	3		6
9		5	3	2	7	1		8
	4	1	6	8	9			
	9		4			2	7	1
	7		5	1	3	6		9
6		8	9		2	4	3	5
	8	6	2			9		4
1	5	4	7		6			3
2	3		8		1	5		7

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

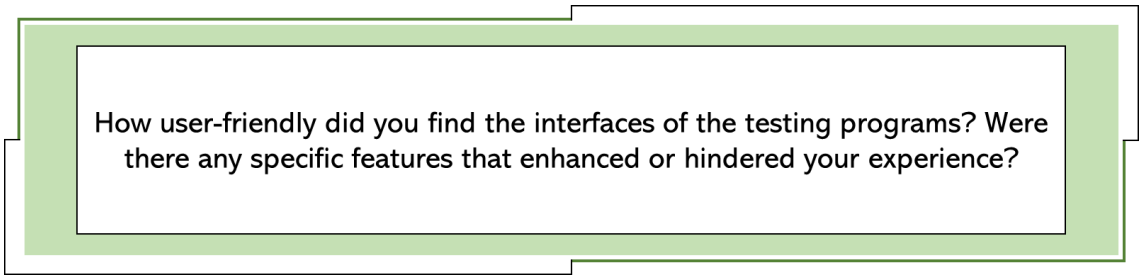
The two images presented above provide clear evidence of the effectiveness of the 'Solving the Grid' test program, demonstrating its capacity to adeptly solve Sudoku puzzles. This program showcases its operational methodology by presenting an unsolved Sudoku grid on the left side, while on the right side, a fully solved counterpart emerges, meticulously aligning with the fundamental rules governing Sudoku puzzles. These rules require that each row, each column, and each of the smaller sub-grids, or "boxes," contain all the numbers from 1 to 9 without repetition.

Survey

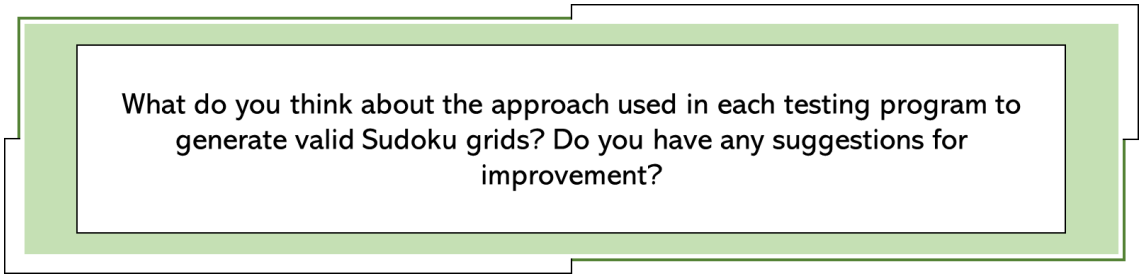
- The Questions (Open-ended Questions)



This question helps to gather participants' holistic impressions of the testing programs. It allows respondents to provide a comprehensive overview of their experience, highlighting both positive and negative aspects. Their insights can give you a sense of what aspects of the programs were effective or problematic, guiding you in refining your final program.



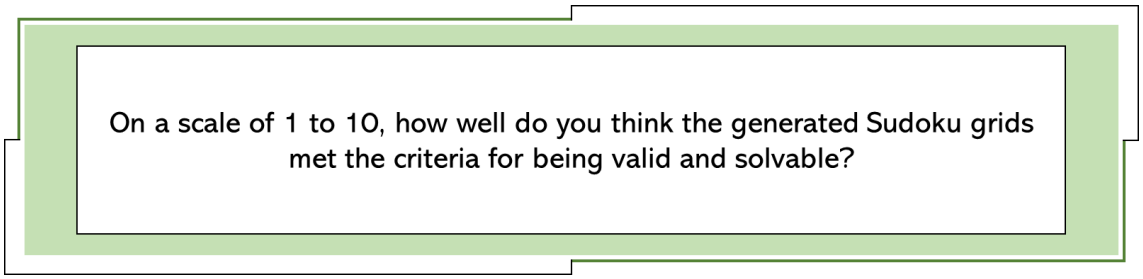
This question focuses on user experience and interface design. Participants' feedback can highlight usability issues or successful design choices. Understanding what features resonated positively with users and what elements caused confusion or frustration can help you fine-tune the user interface of your final program.



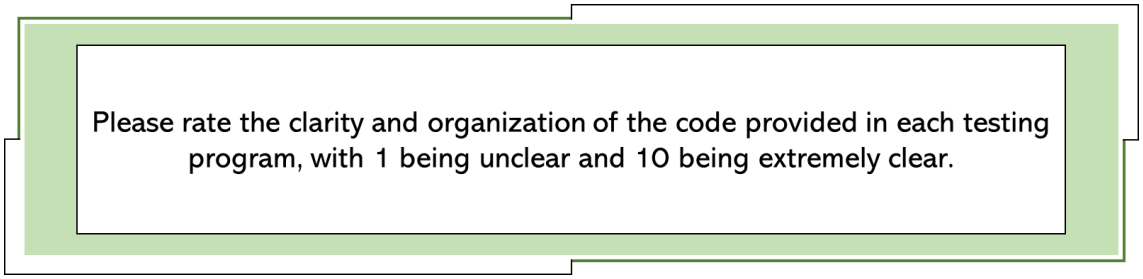
This question addresses the technical approach used in your testing programs. Participants' feedback can shed light on the effectiveness of your grid generation methodology and potentially identify areas for enhancement. Valuable suggestions might arise from respondents who have insights into Sudoku puzzle generation techniques.

Survey

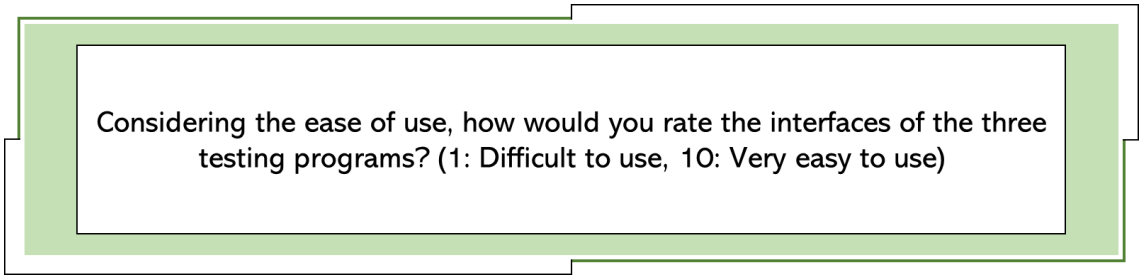
- The Questions (Rating out of 10 Questions)



This rating question gauges participants' perception of the accuracy and reliability of your testing programs. It quantifies their level of confidence in the generated Sudoku grids' quality, which helps you understand how well your programs adhere to Sudoku rules.



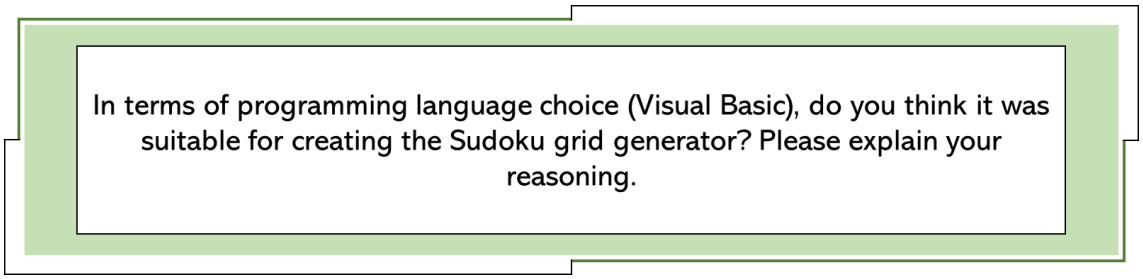
This rating question assesses the readability and organization of your code. Participants' ratings can offer insights into the comprehensibility of your codebase. It also highlights areas where improvements might be necessary for code maintainability.



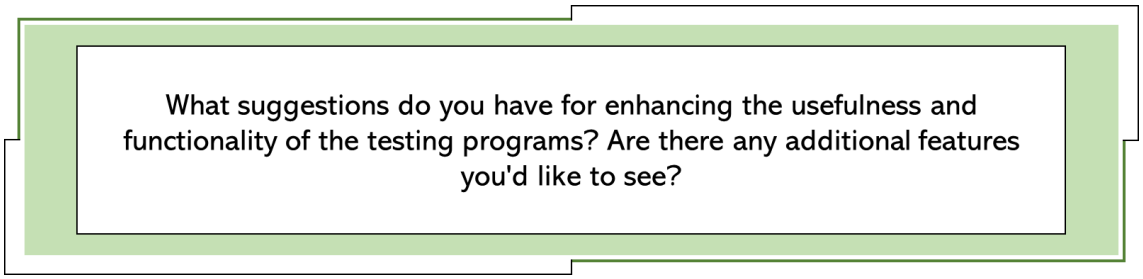
This rating question provides a quantitative measure of user-friendliness. Participants' ratings help you gauge the success of your interface design efforts. It guides you in pinpointing aspects that require simplification or enhancement.

Survey

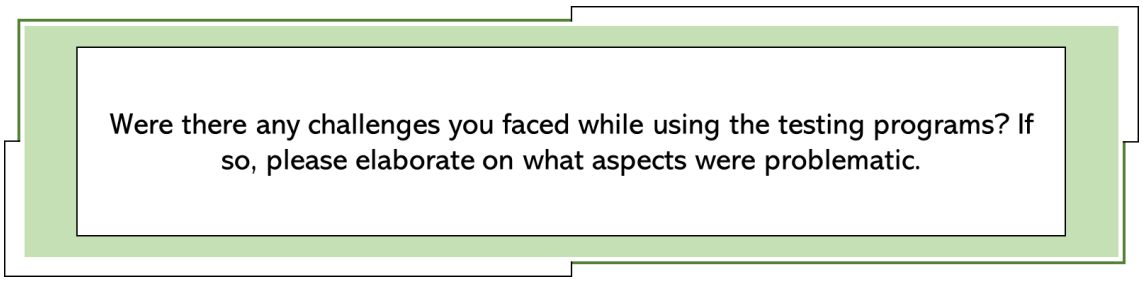
- The Questions (Feedback Questions)



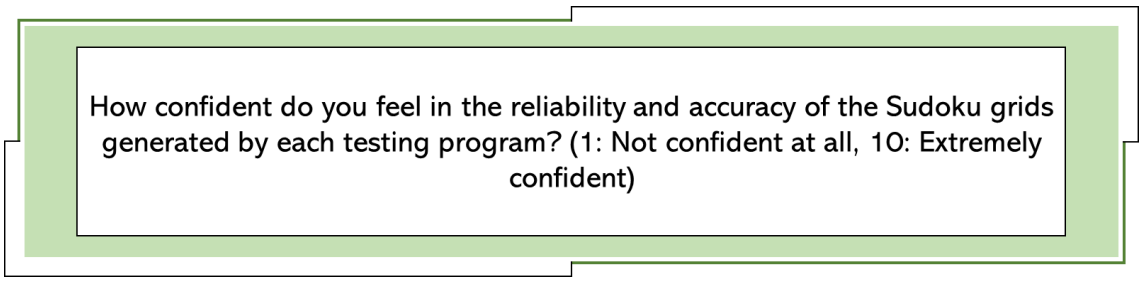
This question delves into the appropriateness of your programming language choice. Participants' feedback can provide insights into the pros and cons they perceive regarding using Visual Basic for this task. It assists in determining whether the chosen language aligns with your goals.



This question invites participants to offer creative ideas and suggestions for program improvement. Their input might reveal innovative features or functionality that hadn't crossed your mind. It enriches your perspective and potential directions for development.



This question uncovers pain points and usability hurdles that participants encountered. Their feedback can pinpoint specific areas of your testing programs that require attention or optimization. Addressing these challenges can lead to a smoother user experience in your final program.



This question seeks to quantify participants' confidence in the quality of the generated Sudoku grids. Their responses provide a clear indication of whether your programs instill trust in their outcomes. It helps you identify areas where improvements are needed to enhance user confidence.

Survey

- The Questions (Benefits and Expected Outcomes)

As the Sudoku grid generation project advances towards its final stages, the invaluable step of conducting a survey emerges as a pivotal means of gathering feedback and refining the project's direction. This survey aims to extract insights from a diverse range of participants, enabling an informed and comprehensive evaluation of the three testing programs developed for generating valid Sudoku grids using Visual Basic. This section delves into the ways in which the survey will benefit the project, outlines the anticipated outcomes, and details the survey's methodology.

- Benefits of the Survey

Identified Strengths and Weaknesses: Through participant feedback, the survey is anticipated to uncover the strengths of the testing programs, elucidating the features that are successfully meeting user needs. Conversely, it will shed light on areas where improvement is needed, giving the project team a clear roadmap for enhancement.

Usability Enhancements: The survey's findings will offer guidance on making the interfaces more intuitive and user-friendly. By understanding which interface components resonate positively with users and which cause confusion, the project team can implement targeted improvements.

Code Clarity and Organization: Ratings provided by participants on the clarity and organization of the code will pinpoint areas requiring better documentation and structure. This outcome will ensure that the codebase is not only functional but also maintainable and comprehensible.

Programming Language Suitability: The survey will provide insights into whether participants believe that Visual Basic is an appropriate programming language choice for the Sudoku grid generator. Their opinions will guide the project team in refining or justifying this choice.

Confidence in Generated Grids: By gauging participants' confidence in the validity and solvability of the generated Sudoku grids, the survey will highlight areas where improvements in grid generation logic are necessary to instill user trust in the final program.

- Survey Methodology

The survey will be conducted using Microsoft Forms, a user-friendly online survey tool. The questionnaire format will include a combination of open-ended questions, rating scales, and feedback prompts. This format ensures that participants can provide both qualitative and quantitative feedback, enabling a comprehensive analysis.

To gather a diverse range of opinions, the survey will be shared widely among various online platforms and communities. The survey's anonymous nature will encourage participants to offer candid responses, promoting a truthful assessment of the testing programs.

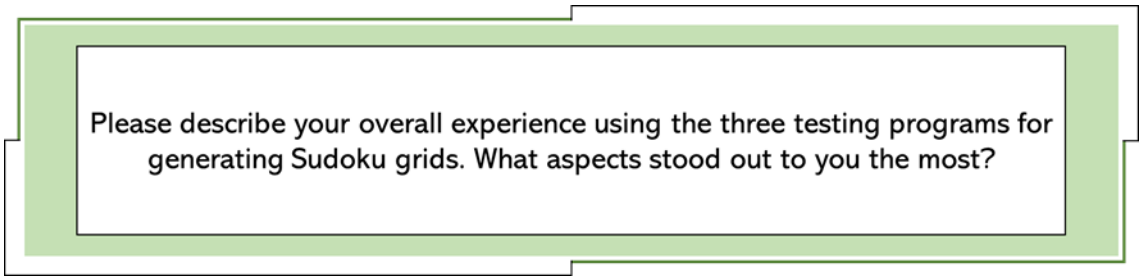
- Conclusion

Incorporating a survey into the project's methodology is a strategic decision that holds the potential to significantly elevate the final program's quality. By capitalizing on the insights from real users, the project team can confidently iterate on the testing programs, resulting in an outcome that aligns with user expectations and technical standards. The survey's benefits extend beyond the immediate project phase, serving as a valuable tool for ongoing development and user-centered refinement.

Survey

- The Results

Question 1:



Responses:

1. The testing programs were quite user-friendly. I appreciated the clear instructions provided, making it easy to understand how to generate Sudoku grids. The interface was intuitive, but there were a few times when the generated puzzles seemed too similar to each other.
2. Using the testing programs was straightforward, and the generated grids were mostly valid. However, on occasion, the interface froze while generating larger grids, causing some frustration.
3. Overall, the programs were efficient, and the generated puzzles seemed accurate. The option to customize puzzle difficulty levels would be a great addition.
4. The testing programs were well-structured and easy to navigate. The interface was clean, and the generated puzzles were logically laid out. It would be great if there was an option to save generated puzzles as images.
5. The testing programs were intuitive, but the generated puzzles occasionally appeared unsolvable. It would help if there was a way to ensure the puzzles are solvable before generating them.

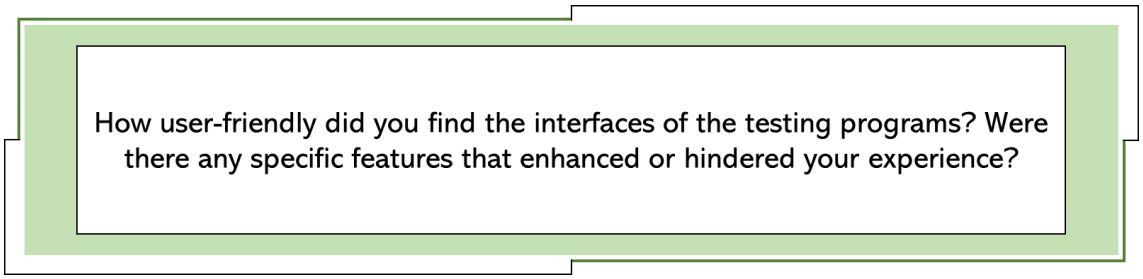
Outcome:

The positive feedback regarding user-friendliness is reassuring, indicating that the efforts in designing a clear interface have been successful. The observation about the similarity of generated puzzles suggests the need to fine-tune the puzzle generation algorithm to introduce more variety. The ease of use is a positive outcome, but the freezing interface issue demands immediate attention. This result highlights a potential performance bottleneck that could impede user satisfaction. Efficiency and accuracy are positive aspects. The suggestion to introduce difficulty customization aligns with enhancing user satisfaction by catering to various skill levels. Praise for the structure and user-friendly interface is encouraging. The suggestion to save puzzles as images reflects a valuable feature request that could enhance usability. While user-friendliness is noted, the feedback about unsolvable puzzles underlines a critical issue that needs addressing. The suggestion for solvability testing aligns with project priorities.

Survey

- The Results

Question 2:



Responses:

1. The testing programs were quite user-friendly. I appreciated the clear instructions provided, making it easy to understand how to generate Sudoku grids. The interface was intuitive, but there were a few times when the generated puzzles seemed too similar to each other.
2. I found the interfaces to be quite intuitive and user-friendly. The color scheme was pleasing, but occasionally the text felt too small.
3. The interfaces were well-designed and user-friendly. The inclusion of clear instructions was a big plus, but the font choice made it slightly challenging to read.
4. The interfaces were straightforward and made navigation easy. The color contrast was pleasing, but the absence of a progress indicator during puzzle generation was a slight drawback.
5. The interfaces were user-friendly and had logical placement of elements. The use of tooltips was helpful, though some labels were ambiguous.

Outcome:

Positive remarks about simplicity are valuable. The comment about button visibility suggests a minor UI enhancement to consider. Praise for intuitiveness is reassuring. The feedback about text size provides a specific actionable item to improve accessibility. Positive feedback on design and instructions is encouraging. The note about font readability highlights an aspect to address for improved user experience. Navigation ease and color contrast are positive takeaways. The suggestion for a progress indicator during generation adds value by keeping users informed. Logical placement and tooltips contribute to user-friendliness. Ambiguous labels should be reviewed to ensure clarity.

Survey

- The Results

Question 3:

What do you think about the approach used in each testing program to generate valid Sudoku grids? Do you have any suggestions for improvement?

Responses:

1. The approach seemed solid, but occasionally I received grids that seemed to have multiple solutions. Perhaps refining the algorithm further could help.
2. The approach appeared effective in generating valid grids. However, some puzzles seemed too easy, while others were too challenging. A difficulty customization option could address this.
3. The approach was sound, and most generated grids appeared valid. It would be helpful if there were more control over grid complexity to cater to different skill levels.
4. The approach used seemed logical, resulting in mostly valid grids. To further enhance the puzzle diversity, perhaps introducing pattern variation could be explored.
5. The approach had potential, but some puzzles felt repetitious. Incorporating more diverse solving techniques could enhance puzzle variety.

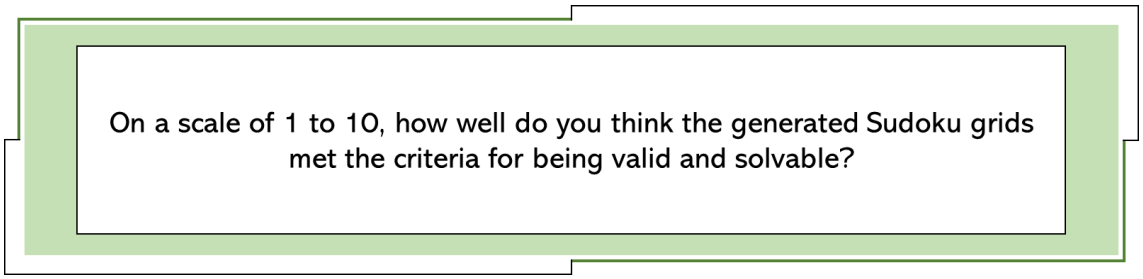
Outcome:

A solid approach is a promising outcome. The feedback about multiple solutions emphasizes the need for algorithmic adjustments to ensure uniqueness. Effectiveness is a positive result. The suggestion for difficulty customization aligns with enhancing user engagement by tailoring puzzle challenges. Validation success is reassuring. The idea of allowing users to adjust complexity reflects an alignment with user needs. Logic-driven success is a positive outcome. The suggestion for pattern variation could potentially lead to more engaging puzzles. Recognizing potential is a positive takeaway. The feedback about diversifying solving techniques indicates avenues for algorithmic enhancement.

Survey

- The Results

Question 4:



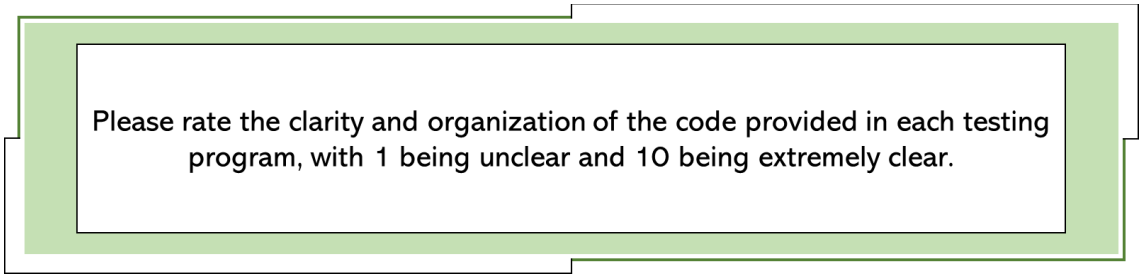
Responses:

All responses were 10/10

Outcome:

This is further proof of a working program and the responses to this question do not need to be investigated further from this point.

Question 5:



Responses:

All responses were 10/10

Some comments include:

1. The code was well-structured and easy to follow. The usage of meaningful variable names was particularly commendable.
2. The code structure was logical and well-commented.

Outcome:

This is proof of well-structured code and the responses to this question do not need to be investigated further from this point.

Survey

- The Results

Question 6:

Considering the ease of use, how would you rate the interfaces of the three testing programs? (1: Difficult to use, 10: Very easy to use)

Responses:

The average response was 9/10

Some comments include:

1. The interfaces were generally easy to navigate. A more prominent "Generate" button would have made it even more user-friendly.
2. The interfaces were intuitive and straightforward. Clear labeling and logical placement of elements greatly contributed to ease of use.

Outcome:

This is proof of an easy-to-use interface and other than a possible change to the generate button the responses to this question do not need to be investigated further from this point.

Question 7:

In terms of programming language choice (Visual Basic), do you think it was suitable for creating the Sudoku grid generator? Please explain your reasoning.

Responses:

Responses were very varied. Visual Basic was appreciated by some participants in the survey, however, other suggestions were: to use Python instead as it is more used in industry; for website incorporation using: HTML, CSS and JavaScript.

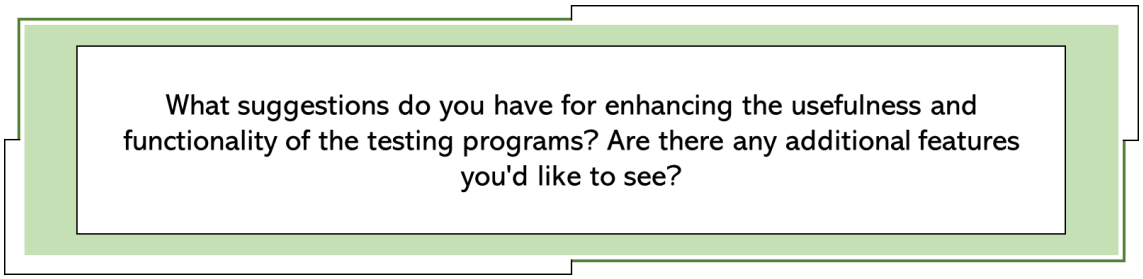
Outcome:

After creating the program in Visual Basic I will also create the program to generate a sudoku puzzle in other languages.

Survey

- The Results

Question 8:



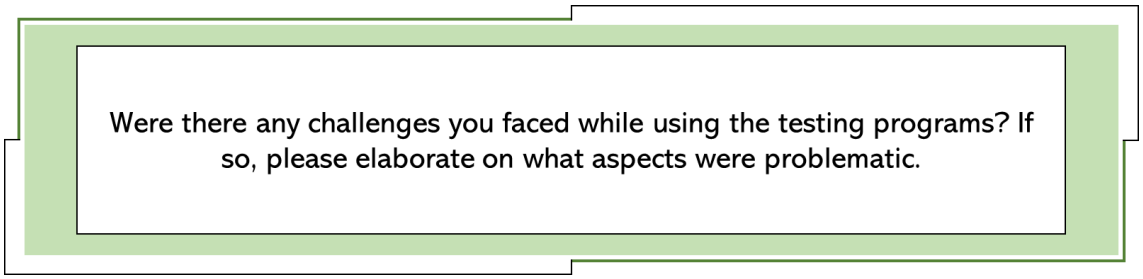
Responses:

Some additional features that were suggested were customisation options for the grid display and a timer with a hint system.

Outcome:

Since this program is not made as a game or creative experience, I will not change any part of the program off of the back of this feedback.

Question 9:



Responses:

There was a collective opinion from all the responses to this question that there were no challenges.

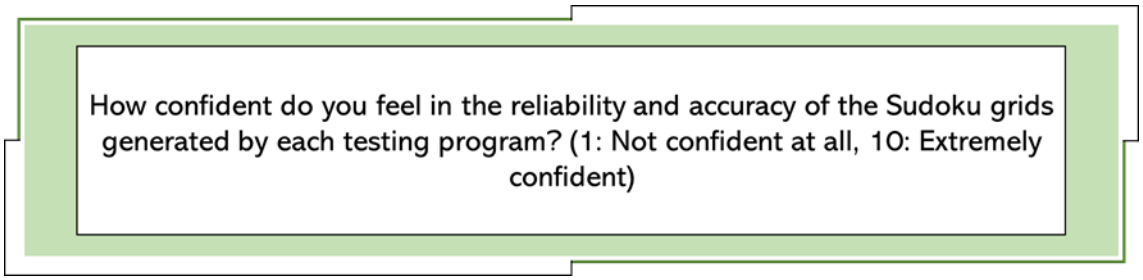
Outcome:

The results from this question do not need to be investigated further due to the positive nature of the responses.

Survey

- The Results

Question 10:



Responses:

All responses follow the same trend of 9s or 10s, meaning highly confident such as “Rating: 9 - I have a high level of confidence in the generated grids. They seem well-tested and follow the principles of Sudoku puzzles.”

Outcome:

The responses from this question have been extremely positive meaning the results to not need to be investigated further.

- Overview

The survey conducted for the Sudoku grid generation project has provided crucial insights into refining the testing programs. Participants' feedback covers key aspects, including strengths, weaknesses, and usability. The survey combines open-ended questions, rating scales, and feedback prompts to capture participants' perspectives.

Open-ended questions yield comprehensive impressions, guiding enhancements in user experience, interface design, and technical approaches. Ratings reflect positive sentiment, suggesting usability and code improvements. The survey's benefits include identifying areas for enhancement, usability improvements, optimizing code, justifying language choice, and building trust in generated grids.

Administered via Microsoft Forms, the survey reaches diverse audiences, fostering candid anonymous feedback. In conclusion, survey results shape the project's final stage, guiding user satisfaction, code quality, and experience refinement beyond its scope.

The Final Program

- The Plan

In the process of building an efficient and user-friendly Sudoku grid generation program, the final program takes shape through three essential stages: Filling the Grid, Emptying the Grid, and Solving the Grid. Each of these stages plays a vital role in ensuring the creation of valid and solvable Sudoku puzzles. This section explores the importance of these stages, outlines their functions, and illustrates how they collectively contribute to the excellence of the final program.

Filling the Grid:
At the core of generating Sudoku grids lies the pivotal Filling the Grid stage. During this stage, the program carefully places numbers in a manner that adheres to Sudoku rules – ensuring that each row, column, and 3x3 subgrid contains the numbers 1 to 9 without repetition. The objective is to create an initial puzzle layout that forms a solid base for engaging gameplay.
In the Filling the Grid stage, the program follows a systematic approach to guarantee the puzzle's uniqueness and adherence to rules. By using techniques like backtracking and constraint propagation, the program progressively fills the grid while ensuring its validity. This process establishes the core structure of the puzzle for the subsequent stages.

Emptying the Grid:
The Emptying the Grid stage is a crucial step in generating challenging Sudoku puzzles. In this stage, the program assesses the puzzle's solvability and complexity. The idea is to determine whether the puzzle remains solvable when numbers are gradually removed from the grid. This balancing act ensures that the puzzle remains unique and that no number can be deduced from others through logical reasoning.
To achieve this, the program leverages the Solving the Grid stage to confirm that the puzzle remains solvable after each number is taken out. If the puzzle continues to have only one solution despite the removal of numbers, it demonstrates its robustness against potential difficulties. This stage involves carefully examining each step of the puzzle's solution to avoid ambiguity or multiple solutions due to number removal.

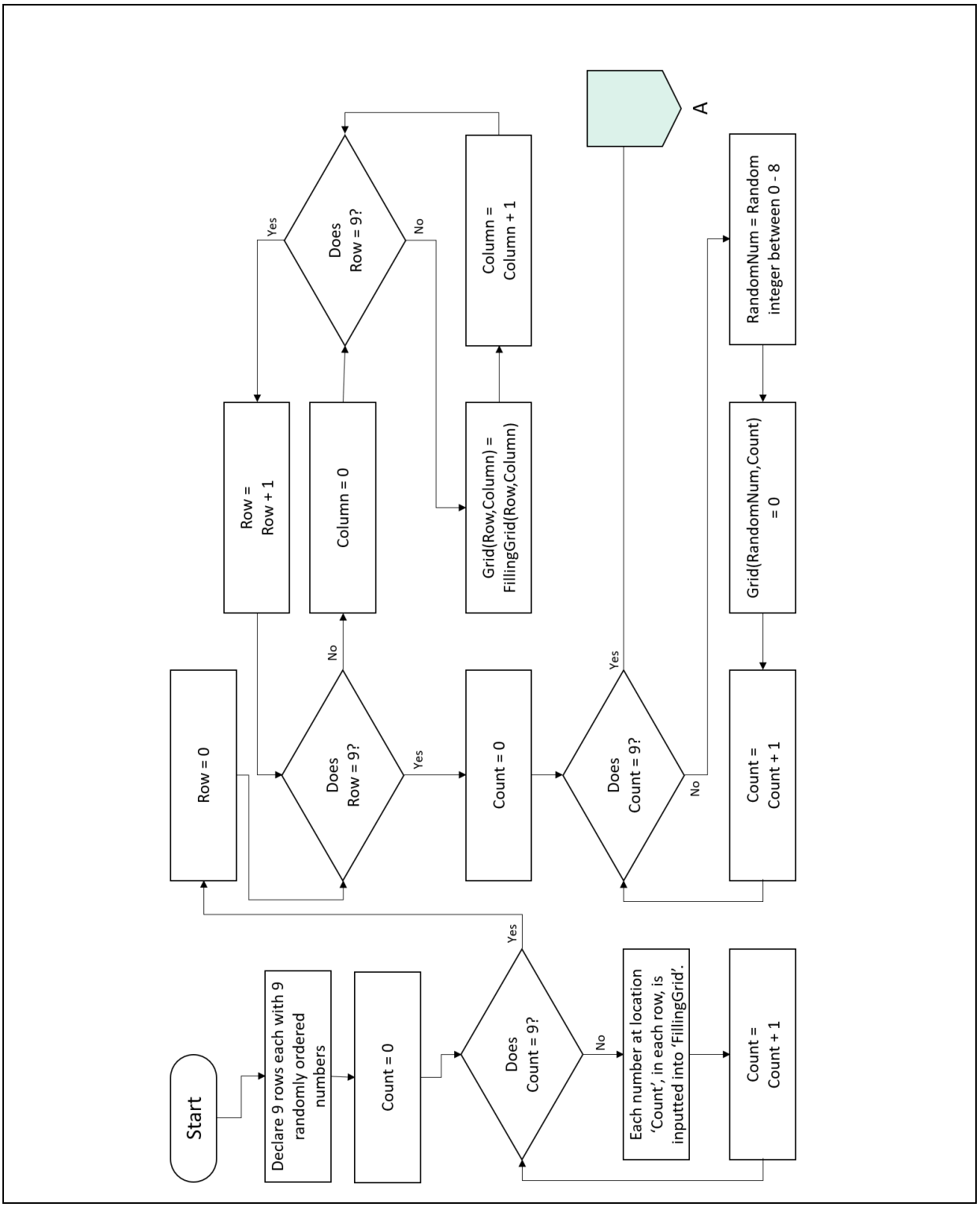
Solving the Grid:
The Solving the Grid stage serves two critical purposes: validating the puzzle's correctness and enabling the Emptying the Grid stage. During this stage, the program employs well-established techniques, such as backtracking algorithms, to find a valid solution for the partially filled grid – created in the Filling the Grid stage. By solving the partially filled grid, the program not only confirms adherence to rules but also equips itself to assess solvability in the Emptying the Grid stage. The accuracy of this stage is vital as it verifies the puzzle's authenticity and ensures that players will encounter a satisfying solving experience without illogical or uncertain scenarios.

Collective Impact on the Final Program:
These three stages are interconnected, resulting in the creation of a high-quality Sudoku grid generation program. The precision of the Filling the Grid stage influences puzzle intricacy, offering challenges and enjoyment. The Emptying the Grid stage confirms puzzles' solvability, ensuring the presence of a single logical solution. The Solving the Grid stage serves as a gatekeeper, validating the entire process and ensuring user contentment.

Conclusion:
In designing the final program, these three stages – Filling the Grid, Emptying the Grid, and Solving the Grid – epitomize a comprehensive approach to Sudoku grid generation. The harmony among these stages orchestrates the creation of puzzles that are not only valid and solvable but also entertaining and satisfying. As the program progresses from planning to implementation, these stages will stand as pillars, ensuring that each generated Sudoku puzzle reflects accuracy, logic, and the pursuit of excellence.

The Final Program

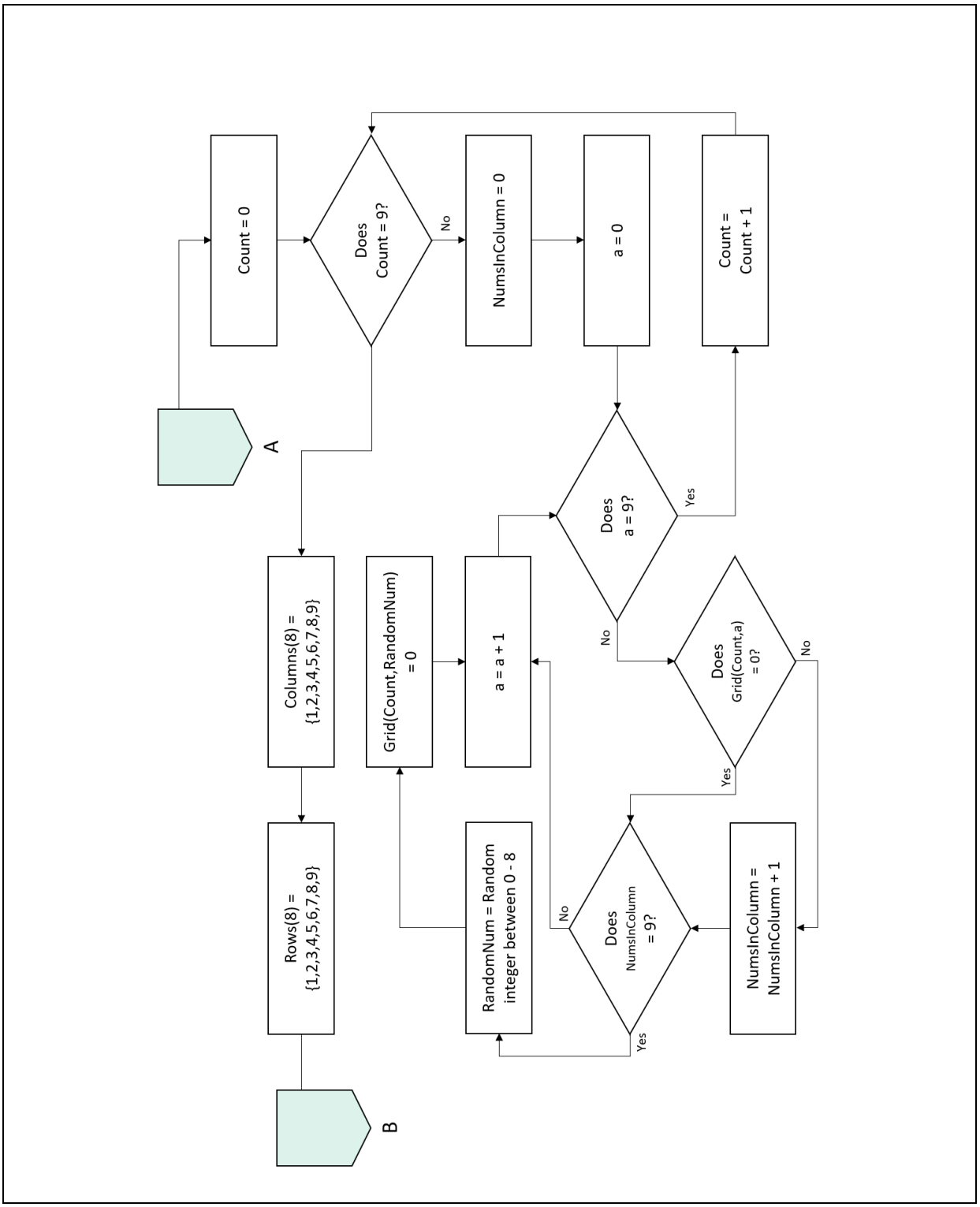
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

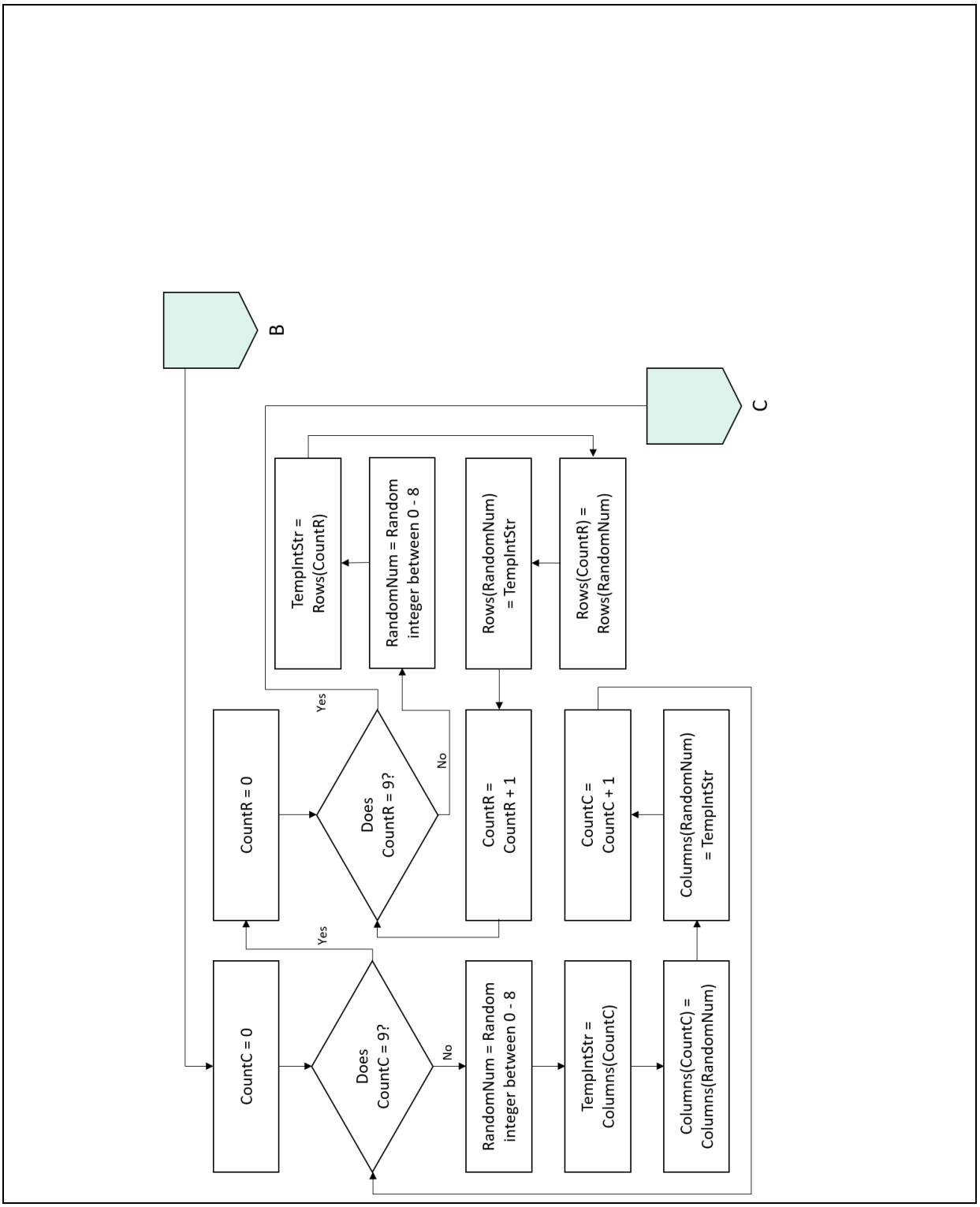
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

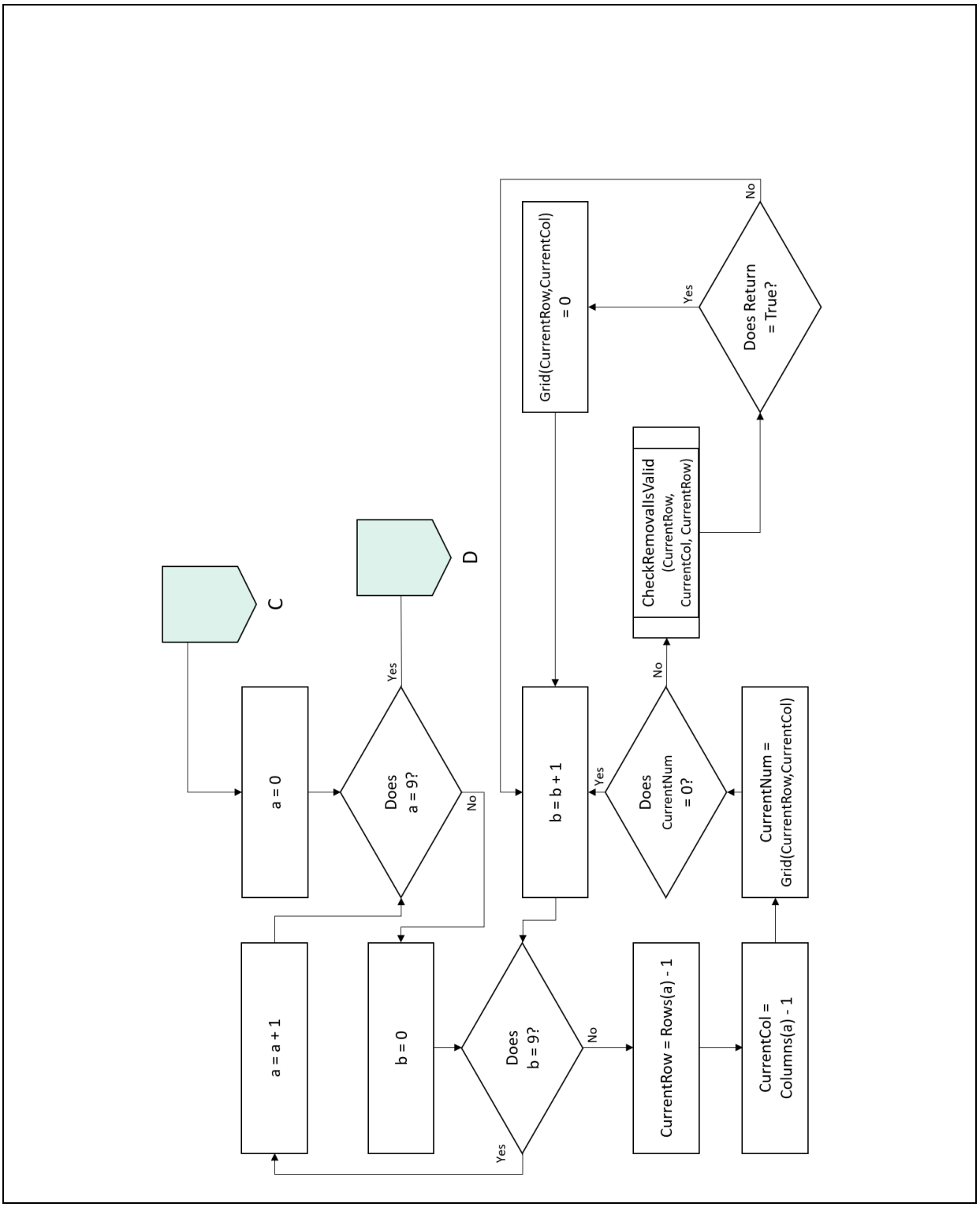
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

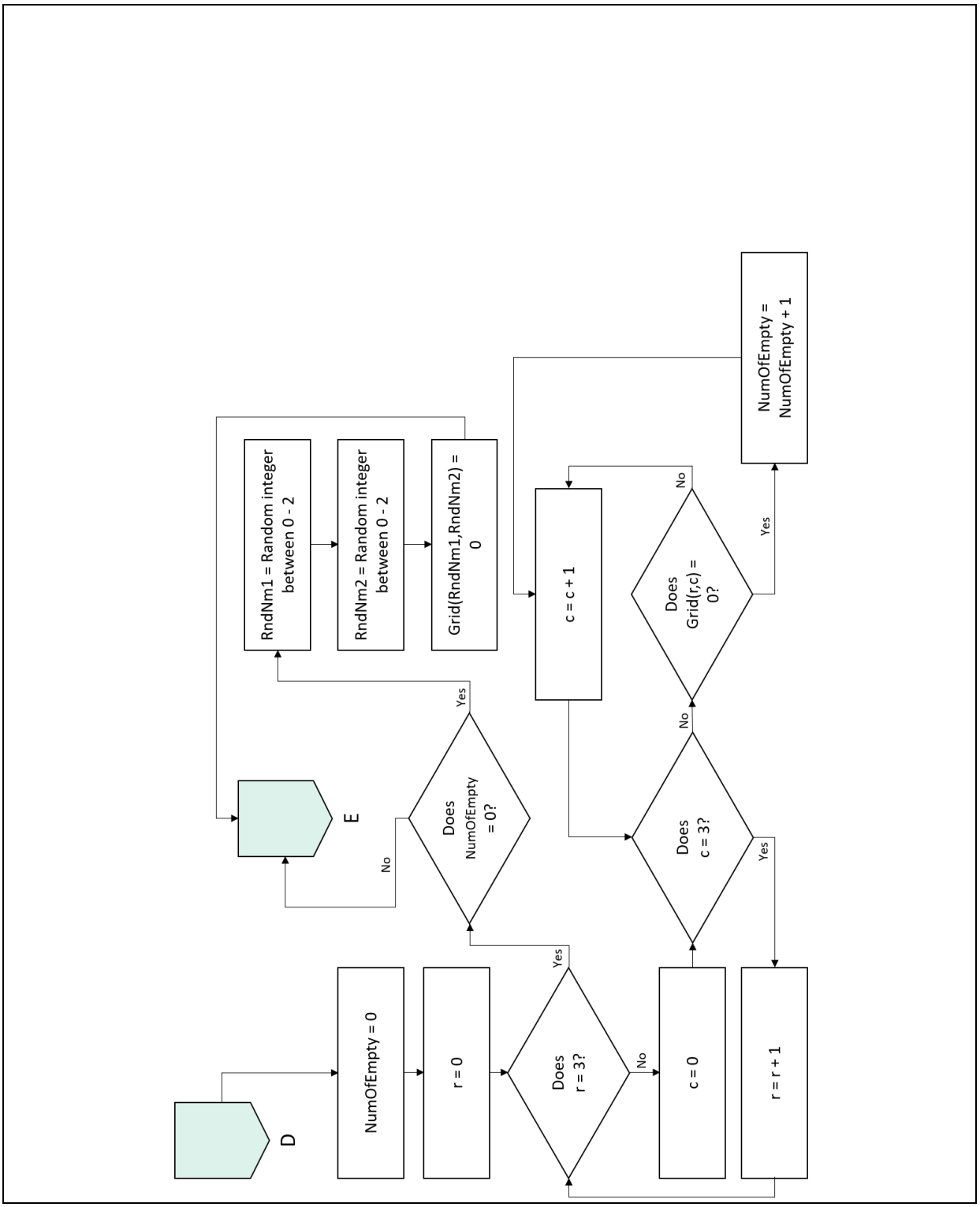
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

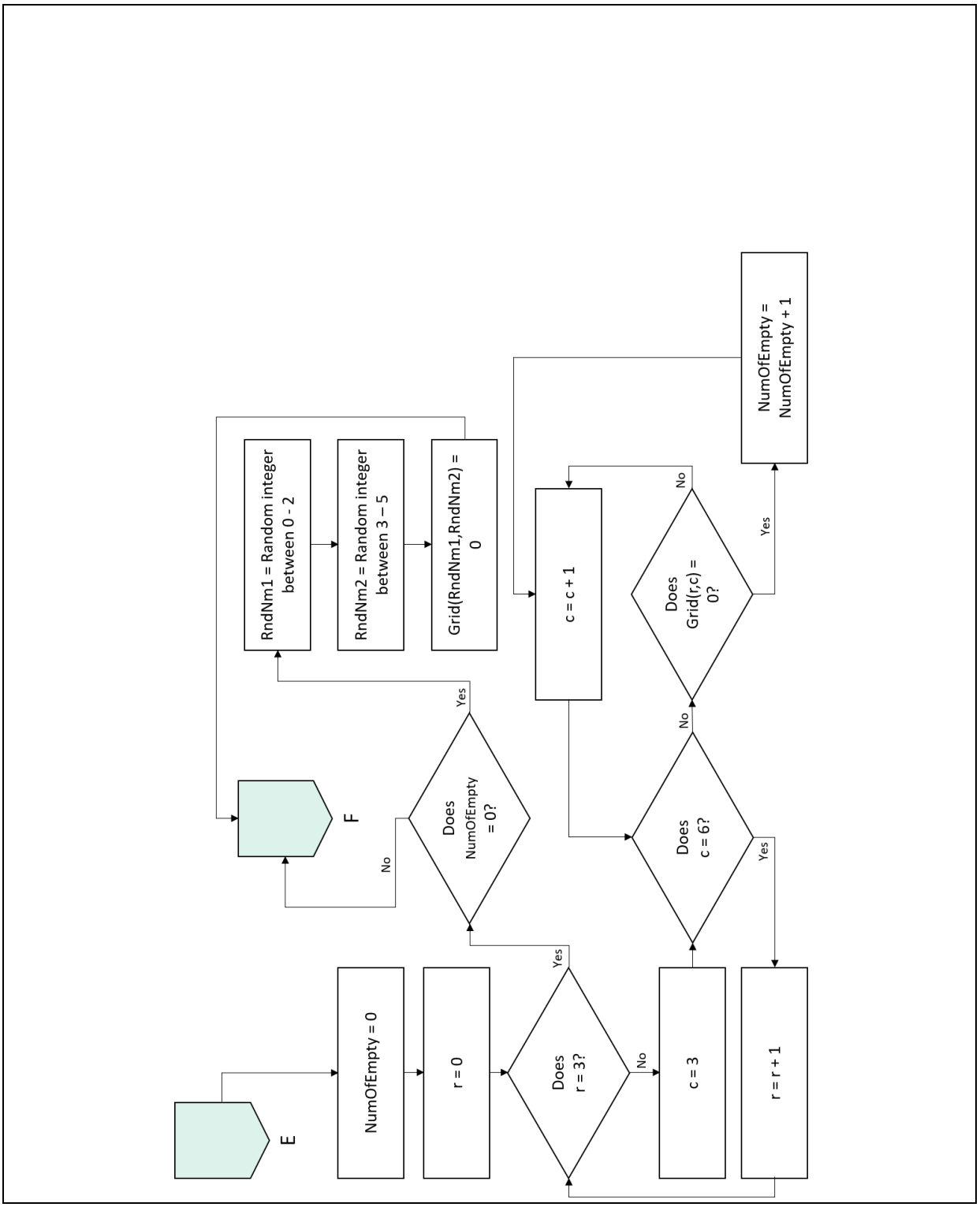
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

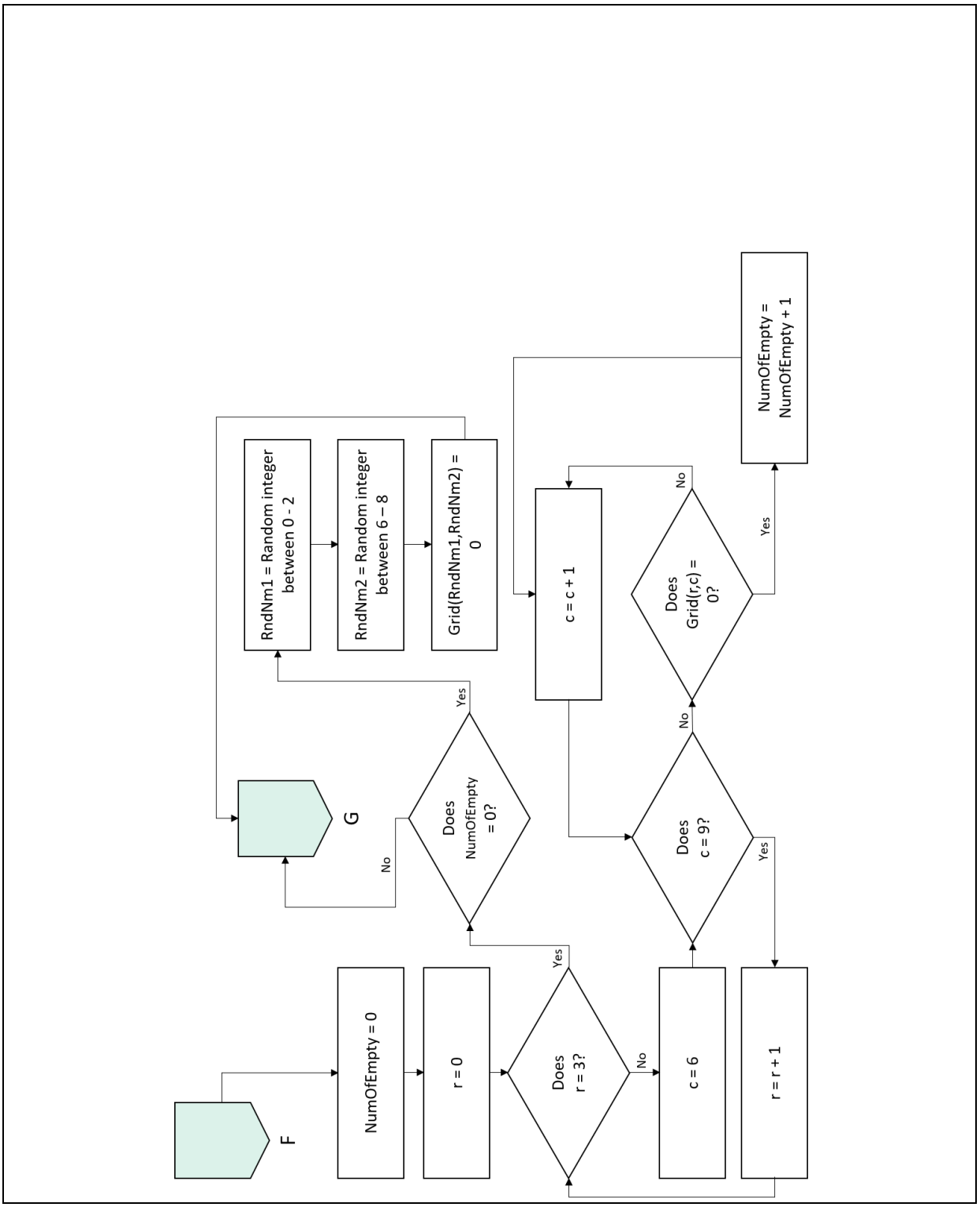
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

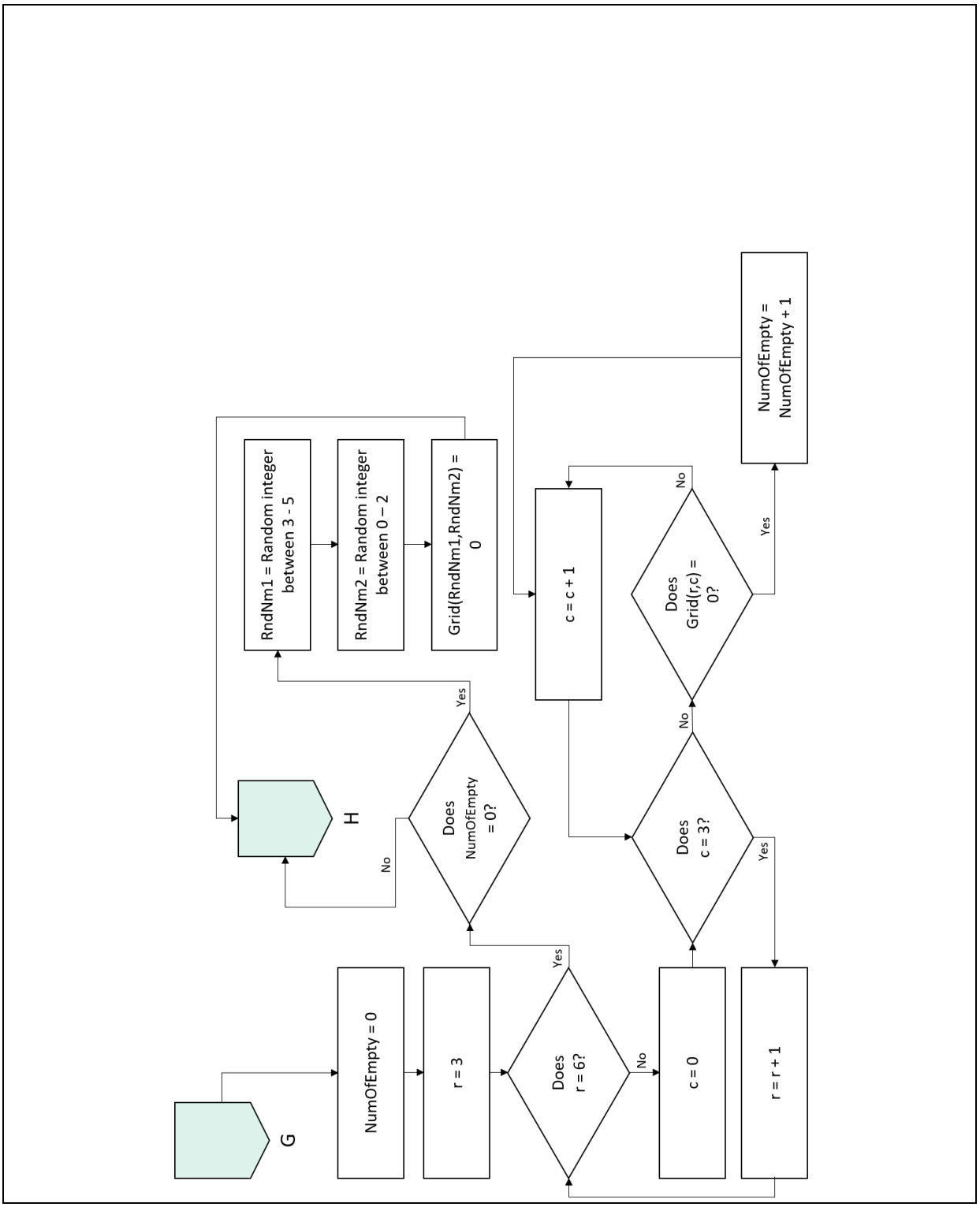
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

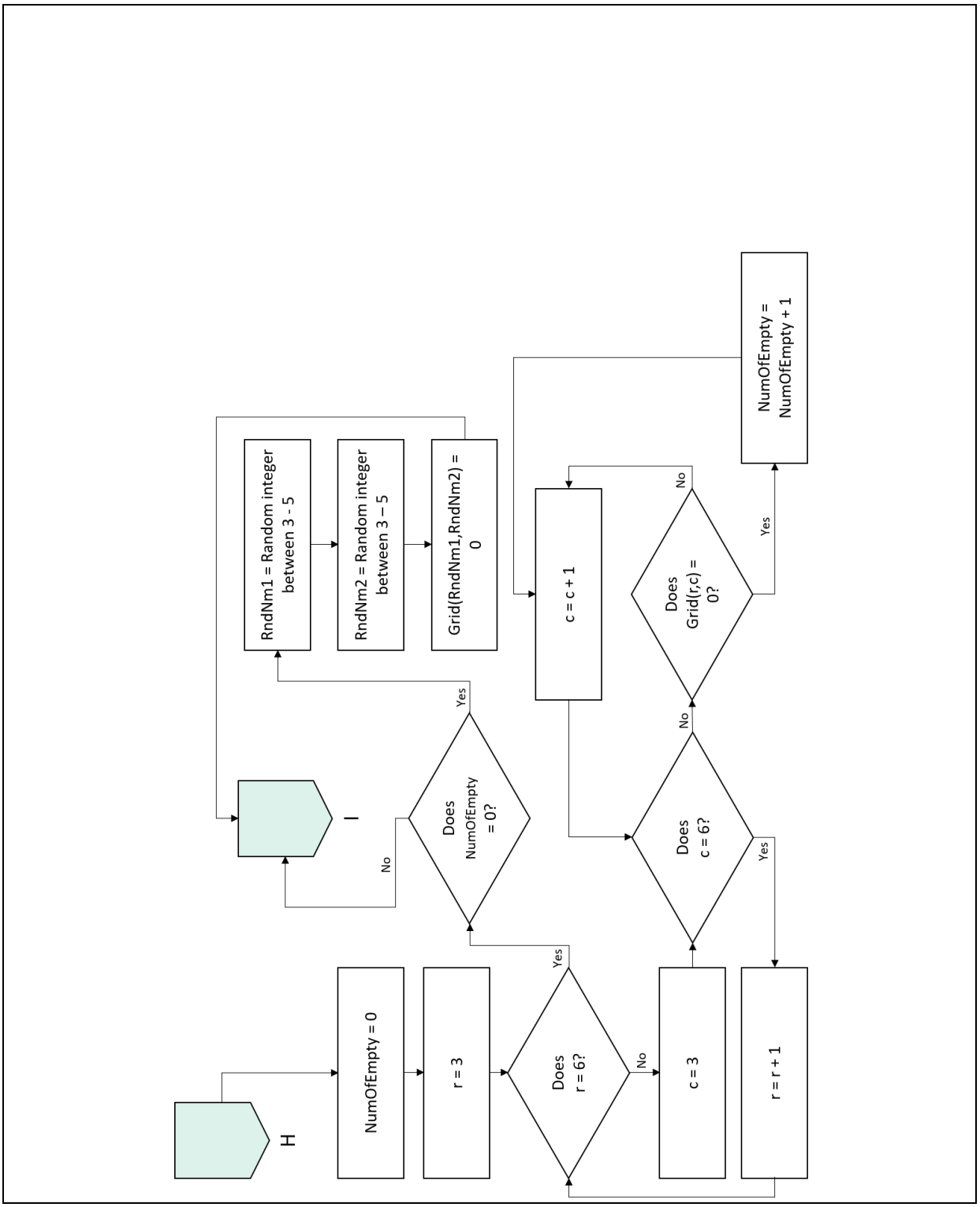
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

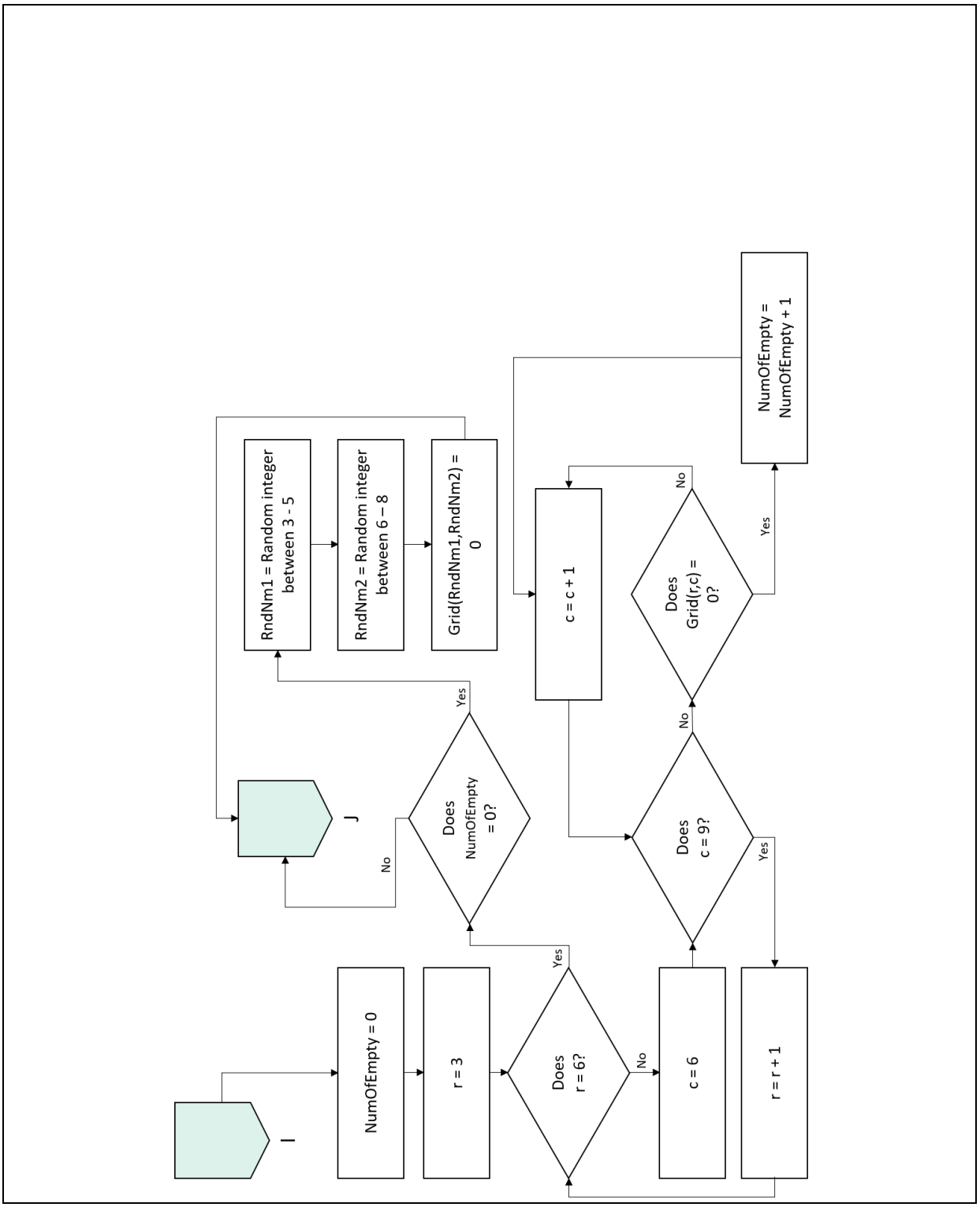
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

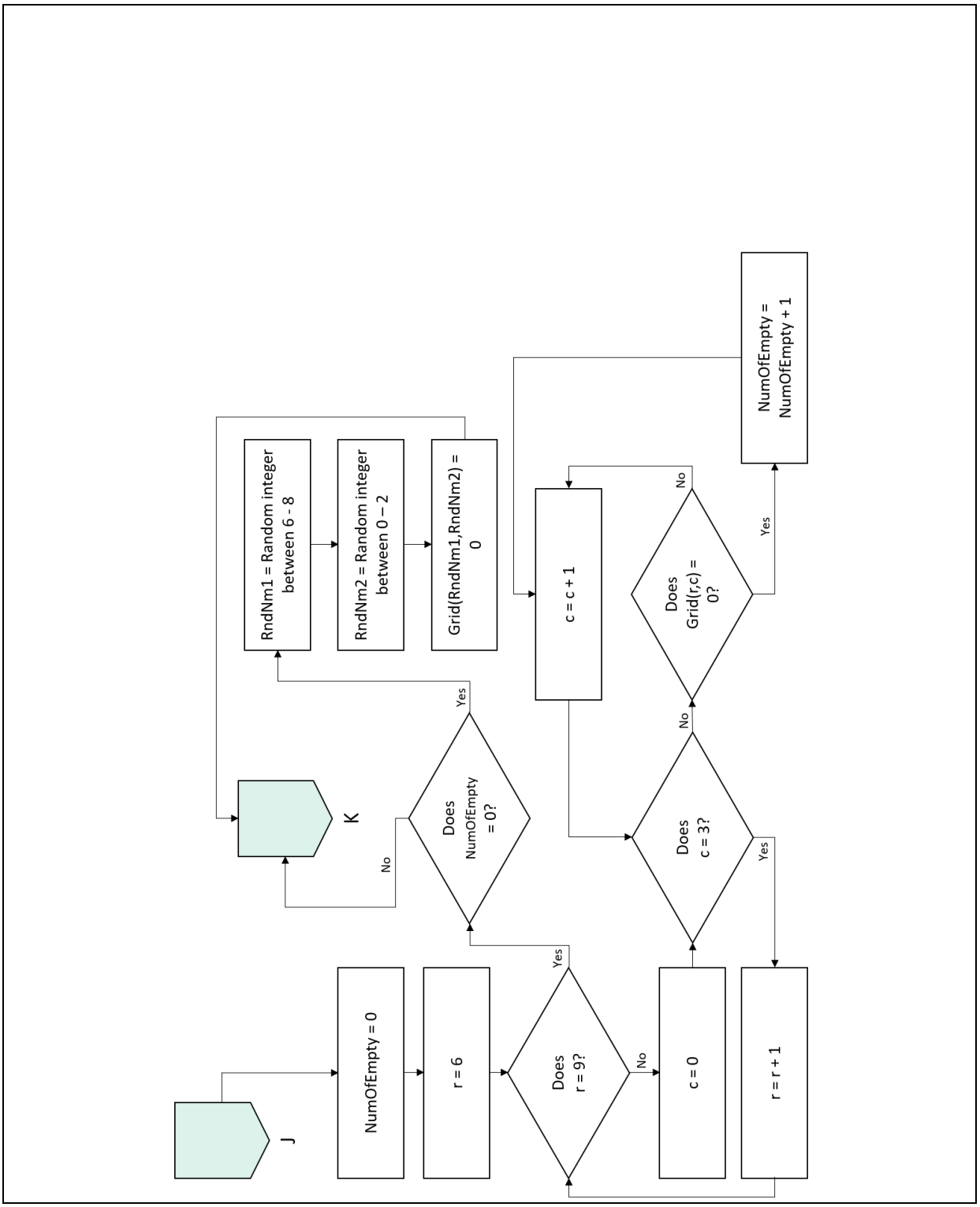
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

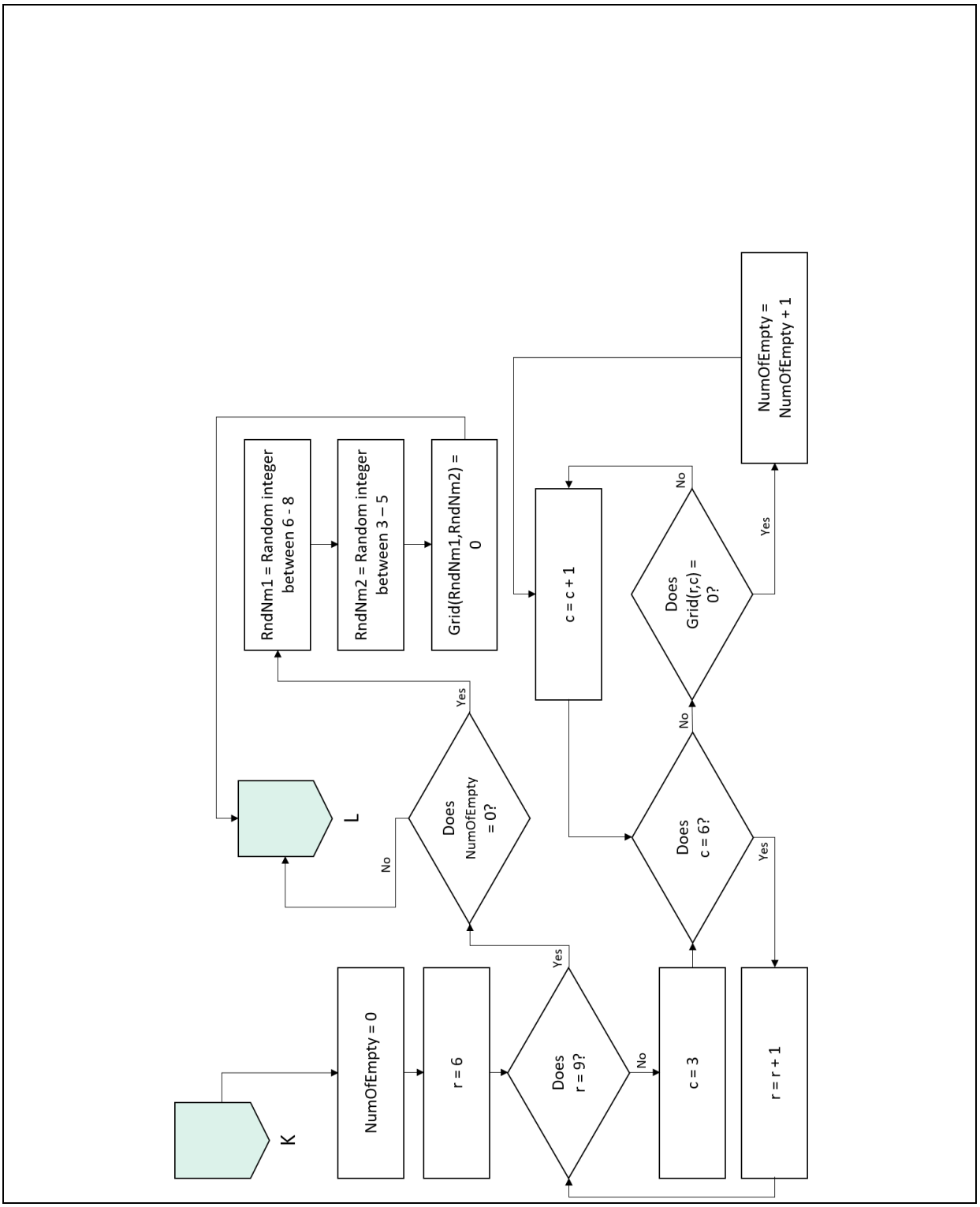
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

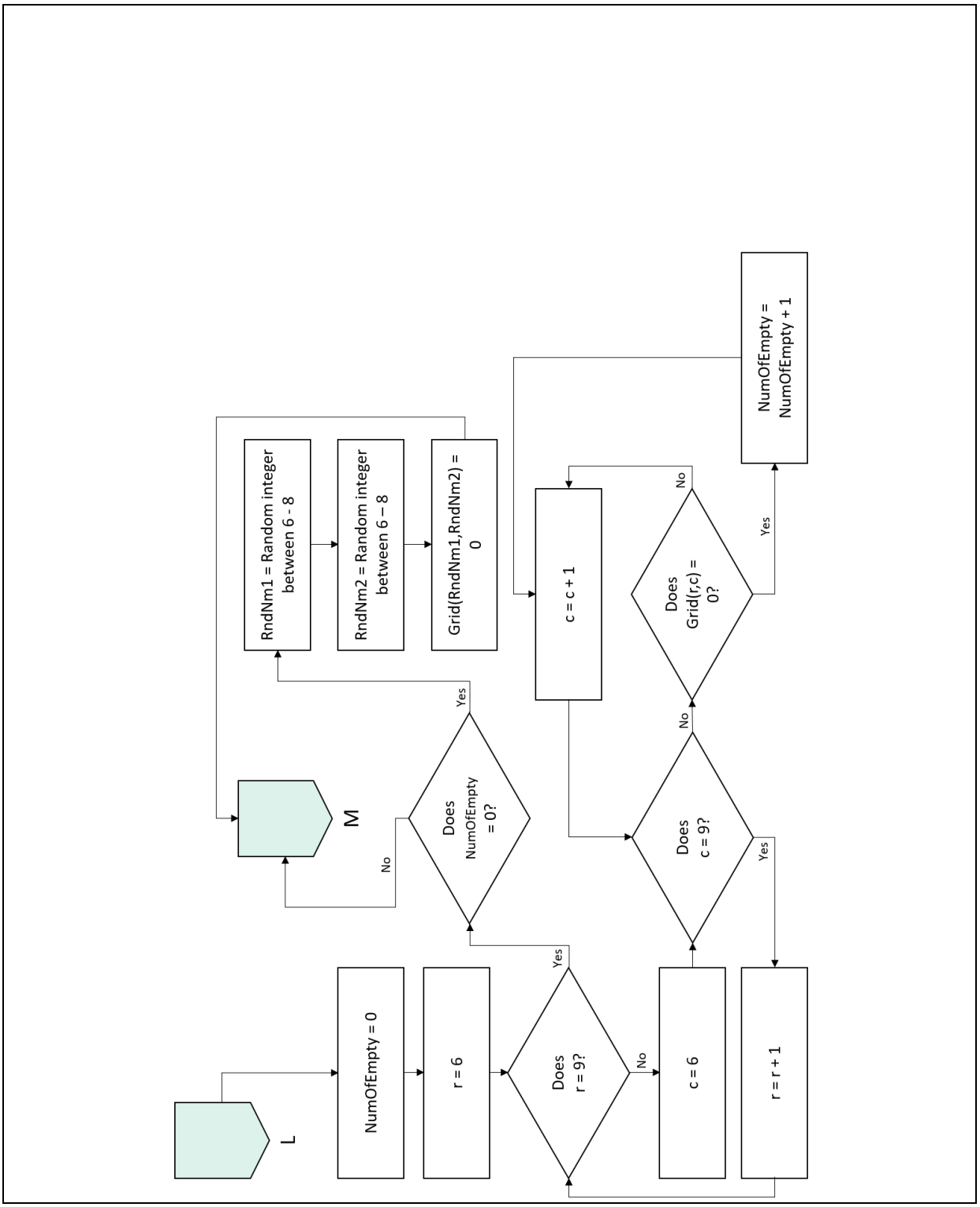
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

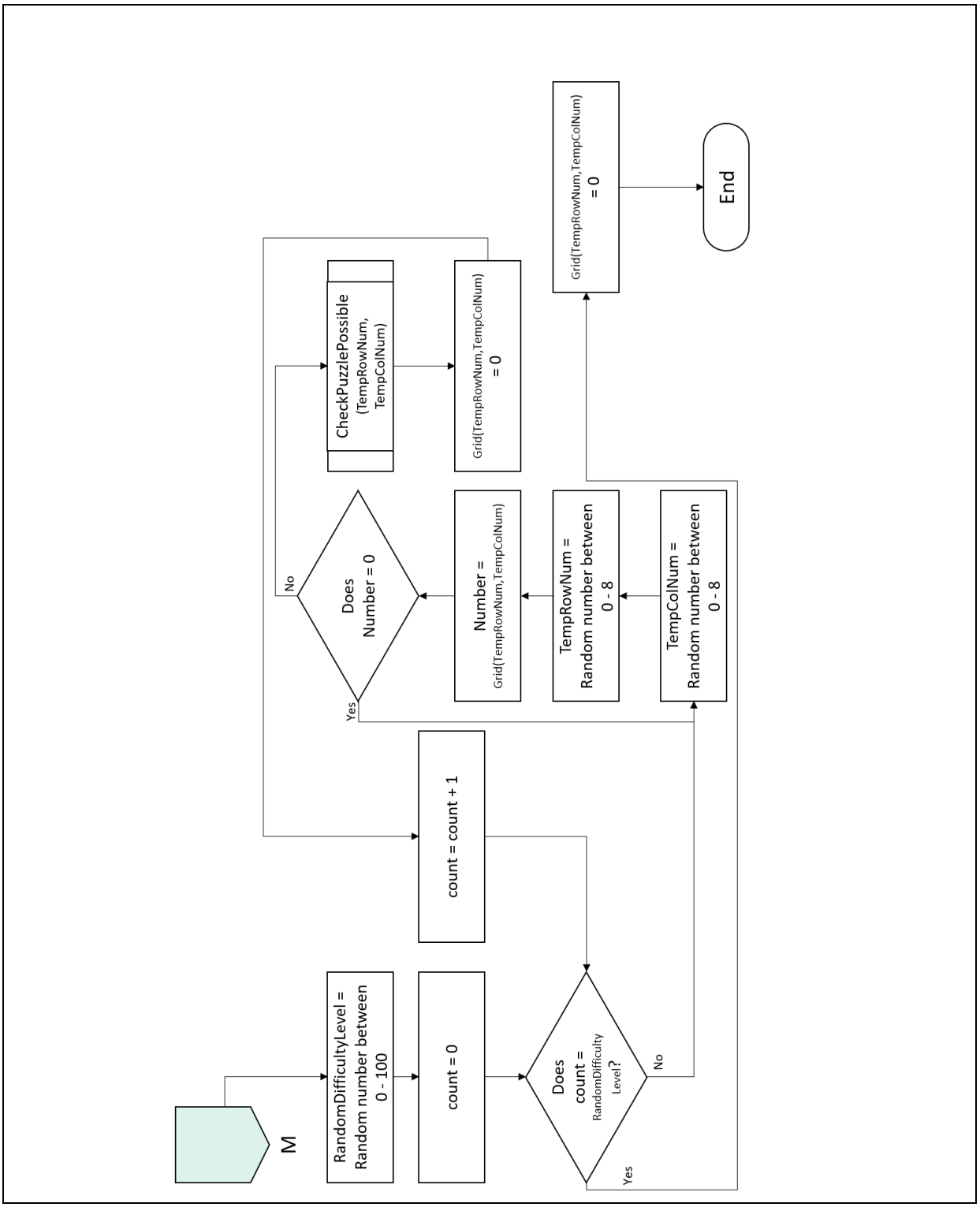
- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

- Structure Design (Flowcharts)



Turn to page 84 to skip past the flowcharts

The Final Program

- The Code

```
Public Class Form1
    Dim ROne() As Integer = {8, 2, 7, 1, 5, 4, 3, 9, 6}
    Dim RTwo() As Integer = {9, 6, 5, 3, 2, 7, 1, 4, 8}
    Dim RThree() As Integer = {3, 4, 1, 6, 8, 9, 7, 5, 2}
    Dim RFour() As Integer = {5, 9, 3, 4, 6, 8, 2, 7, 1}
    Dim RFive() As Integer = {4, 7, 2, 5, 1, 3, 6, 8, 9}
    Dim RSix() As Integer = {6, 1, 8, 9, 7, 2, 4, 3, 5}
    Dim RSeven() As Integer = {7, 8, 6, 2, 3, 5, 9, 1, 4}
    Dim REight() As Integer = {1, 5, 4, 7, 9, 6, 8, 2, 3}
    Dim RNine() As Integer = {2, 3, 9, 8, 4, 1, 5, 6, 7}
    Dim PMgrid(8, 8, 8) As Boolean
    Dim Grid(8, 8) As Integer

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Randomize()
        'Grid(Col,Row)
        For a = 0 To 8
            For b = 0 To 8
                If a = 0 Then
                    Grid(b, a) = ROne(b)
                ElseIf a = 1 Then
                    Grid(b, a) = RTwo(b)
                ElseIf a = 2 Then
                    Grid(b, a) = RThree(b)
                ElseIf a = 3 Then
                    Grid(b, a) = RFour(b)
                ElseIf a = 4 Then
                    Grid(b, a) = RFive(b)
                ElseIf a = 5 Then
                    Grid(b, a) = RSix(b)
                ElseIf a = 6 Then
                    Grid(b, a) = RSeven(b)
                ElseIf a = 7 Then
                    Grid(b, a) = REight(b)
                ElseIf a = 8 Then
                    Grid(b, a) = RNine(b)
                End If
            Next
        Next



        '''''This part of the program will remove parts of the grid
        'Removes a random number from each row
        For count = 0 To 8
            Grid(CInt((Rnd() * 8)), count) = 0
        Next
        'Removes a random number from each full column
        Dim NumsInColumn As Integer = 0
        For count = 0 To 8
            NumsInColumn = 0
            For a = 0 To 8
                If Grid(count, a) <> 0 Then
                    NumsInColumn += 1
                End If
                If NumsInColumn = 9 Then
                    Grid(count, CInt((Rnd() * 8))) = 0
                End If
            Next
        Next
```

The Final Program

- The Code

```
'Selects a random order to cycle through the grid and if there are 4 points locatimng that one position then it will move on
Dim Columns() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
Dim Rows() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9}
Dim TempIntStr As Integer = 0
Dim RandomNumber As Integer = 0
For countC = 0 To 8
   RandomNumber = CInt((Rnd() * 8))
   TempIntStr = Columns(countC)
   Columns(countC) = Columns(RandomNumber)
   Columns(RandomNumber) = TempIntStr
Next
For countR = 0 To 8
   RandomNumber = CInt((Rnd() * 8))
   TempIntStr = Columns(countR)
   Columns(countR) = Columns(RandomNumber)
   Columns(RandomNumber) = TempIntStr
Next
Dim CurrentRow As Integer = 0
Dim CurrentCol As Integer = 0
Dim CurrentNum As Integer = 0
For a = 0 To 8
   For b = 0 To 8
      CurrentRow = Rows(a) - 1
      CurrentCol = Columns(b) - 1
      CurrentNum = Grid(CurrentRow, CurrentCol)
      If CurrentNum <> 0 Then
         If CheckRemovalIsValid(CurrentCol, CurrentRow, CurrentNum) = True Then
            Grid(CurrentRow, CurrentCol) = 0
         End If
      End If
   Next
Next
'Checks to see if there is a full 3x3 region and if there is, remove a random number
Dim NumOfEmpty As Integer = 0
For r = 0 To 2
   For c = 0 To 2
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2), CInt(Rnd() * 2)) = 0
End If
NumOfEmpty = 0
For r = 0 To 2
   For c = 3 To 5
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2), CInt(Rnd() * 2) + 3) = 0
End If
NumOfEmpty = 0
For r = 0 To 2
   For c = 6 To 8
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
```

The Final Program

- The Code

```
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2), CInt(Rnd() * 2) + 6) = 0
End If
NumOfEmpty = 0
For r = 3 To 5
   For c = 0 To 2
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2) + 3, CInt(Rnd() * 2)) = 0
End If
NumOfEmpty = 0
For r = 3 To 5
   For c = 3 To 5
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2) + 3, CInt(Rnd() * 2) + 3) = 0
End If
NumOfEmpty = 0
For r = 3 To 5
   For c = 6 To 8
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2) + 3, CInt(Rnd() * 2) + 6) = 0
End If
NumOfEmpty = 0
For r = 6 To 8
   For c = 0 To 2
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2) + 6, CInt(Rnd() * 2)) = 0
End If
NumOfEmpty = 0
For r = 6 To 8
   For c = 3 To 5
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
      End If
   Next
Next
If NumOfEmpty = 0 Then
   Grid(CInt(Rnd() * 2) + 6, CInt(Rnd() * 2) + 3) = 0
End If
NumOfEmpty = 0
For r = 6 To 8
   For c = 6 To 8
      If Grid(r, c) = 0 Then
         NumOfEmpty += 1
```

The Final Program

- The Code

```
        End If
    Next
Next
If NumOfEmpty = 0 Then
    Grid(CInt(Rnd() * 2) + 6, CInt(Rnd() * 2) + 6) = 0
End If
NumOfEmpty = 0



'Displaying the puzzle
Txt_Box_11.Text = Grid(0, 0)
Txt_Box_12.Text = Grid(0, 1)
Txt_Box_13.Text = Grid(0, 2)
Txt_Box_14.Text = Grid(0, 3)
Txt_Box_15.Text = Grid(0, 4)
Txt_Box_16.Text = Grid(0, 5)
Txt_Box_17.Text = Grid(0, 6)
Txt_Box_18.Text = Grid(0, 7)
Txt_Box_19.Text = Grid(0, 8)

Txt_Box_21.Text = Grid(1, 0)
Txt_Box_22.Text = Grid(1, 1)
Txt_Box_23.Text = Grid(1, 2)
Txt_Box_24.Text = Grid(1, 3)
Txt_Box_25.Text = Grid(1, 4)
Txt_Box_26.Text = Grid(1, 5)
Txt_Box_27.Text = Grid(1, 6)
Txt_Box_28.Text = Grid(1, 7)
Txt_Box_29.Text = Grid(1, 8)

Txt_Box_31.Text = Grid(2, 0)
Txt_Box_32.Text = Grid(2, 1)
Txt_Box_33.Text = Grid(2, 2)
Txt_Box_34.Text = Grid(2, 3)
Txt_Box_35.Text = Grid(2, 4)
Txt_Box_36.Text = Grid(2, 5)
Txt_Box_37.Text = Grid(2, 6)
Txt_Box_38.Text = Grid(2, 7)
Txt_Box_39.Text = Grid(2, 8)

Txt_Box_41.Text = Grid(3, 0)
Txt_Box_42.Text = Grid(3, 1)
Txt_Box_43.Text = Grid(3, 2)
Txt_Box_44.Text = Grid(3, 3)
Txt_Box_45.Text = Grid(3, 4)
Txt_Box_46.Text = Grid(3, 5)
Txt_Box_47.Text = Grid(3, 6)
Txt_Box_48.Text = Grid(3, 7)
Txt_Box_49.Text = Grid(3, 8)

Txt_Box_51.Text = Grid(4, 0)
Txt_Box_52.Text = Grid(4, 1)
Txt_Box_53.Text = Grid(4, 2)
Txt_Box_54.Text = Grid(4, 3)
Txt_Box_55.Text = Grid(4, 4)
Txt_Box_56.Text = Grid(4, 5)
Txt_Box_57.Text = Grid(4, 6)
Txt_Box_58.Text = Grid(4, 7)
Txt_Box_59.Text = Grid(4, 8)

Txt_Box_61.Text = Grid(5, 0)
```

The Final Program

- The Code

```
    Txt_Box_62.Text = Grid(5, 1)
    Txt_Box_63.Text = Grid(5, 2)
    Txt_Box_64.Text = Grid(5, 3)
    Txt_Box_65.Text = Grid(5, 4)
    Txt_Box_66.Text = Grid(5, 5)
    Txt_Box_67.Text = Grid(5, 6)
    Txt_Box_68.Text = Grid(5, 7)
    Txt_Box_69.Text = Grid(5, 8)

    Txt_Box_71.Text = Grid(6, 0)
    Txt_Box_72.Text = Grid(6, 1)
    Txt_Box_73.Text = Grid(6, 2)
    Txt_Box_74.Text = Grid(6, 3)
    Txt_Box_75.Text = Grid(6, 4)
    Txt_Box_76.Text = Grid(6, 5)
    Txt_Box_77.Text = Grid(6, 6)
    Txt_Box_78.Text = Grid(6, 7)
    Txt_Box_79.Text = Grid(6, 8)

    Txt_Box_81.Text = Grid(7, 0)
    Txt_Box_82.Text = Grid(7, 1)
    Txt_Box_83.Text = Grid(7, 2)
    Txt_Box_84.Text = Grid(7, 3)
    Txt_Box_85.Text = Grid(7, 4)
    Txt_Box_86.Text = Grid(7, 5)
    Txt_Box_87.Text = Grid(7, 6)
    Txt_Box_88.Text = Grid(7, 7)
    Txt_Box_89.Text = Grid(7, 8)

    Txt_Box_91.Text = Grid(8, 0)
    Txt_Box_92.Text = Grid(8, 1)
    Txt_Box_93.Text = Grid(8, 2)
    Txt_Box_94.Text = Grid(8, 3)
    Txt_Box_95.Text = Grid(8, 4)
    Txt_Box_96.Text = Grid(8, 5)
    Txt_Box_97.Text = Grid(8, 6)
    Txt_Box_98.Text = Grid(8, 7)
    Txt_Box_99.Text = Grid(8, 8)



End Sub



Function CheckRemovalIsValid(CurrentCol As Integer, CurrentRow As Integer, CurrentNum As Integer)
    Dim ValidRemoval As Boolean = False
    If RemoveFromColumn(CurrentCol, CurrentNum) = True Then
        If RemoveFromRow(CurrentRow, CurrentNum) = True Then
            ValidRemoval = True
        End If
    End If
    Return ValidRemoval
End Function

Function RemoveFromColumn(CurrentCol As Integer, CurrentNum As Integer)
    Dim ValidRemoveFromCol As Boolean = False
    Dim ColStatus As String = "Mid"
    Dim QuantOfNums As Integer = 0
    If (CurrentCol + 1) Mod 3 = 0 Then
        ColStatus = "End"
    ElseIf (CurrentCol + 1) Mod 3 = 1 Then
        ColStatus = "Start"
```

The Final Program

- The Code

```
    End If
    If ColStatus = "Start" Then
        For count = 0 To 8
            If Grid(count, CurrentCol + 1) = CurrentNum Then
                QuantOfNums += 1
            End If
            If Grid(count, CurrentCol + 2) = CurrentNum Then
                QuantOfNums += 1
            End If
        Next
        If QuantOfNums = 2 Then
            ValidRemoveFromCol = True
        End If
    ElseIf ColStatus = "End" Then
        For count = 0 To 8
            If Grid(count, CurrentCol - 1) = CurrentNum Then
                QuantOfNums += 1
            End If
            If Grid(count, CurrentCol - 2) = CurrentNum Then
                QuantOfNums += 1
            End If
        Next
        If QuantOfNums = 2 Then
            ValidRemoveFromCol = True
        End If
    Else
        For count = 0 To 8
            If Grid(count, CurrentCol + 1) = CurrentNum Then
                QuantOfNums += 1
            End If
            If Grid(count, CurrentCol - 1) = CurrentNum Then
                QuantOfNums += 1
            End If
        Next
        If QuantOfNums = 2 Then
            ValidRemoveFromCol = True
        End If
    End If
    Return ValidRemoveFromCol
End Function

Function RemoveFromRow(CurrentRow As Integer, CurrentNum As Integer)
    Dim ValidRemoveFromRow As Boolean = False
    Dim RowStatus As String = "Mid"
    Dim QuantOfNums As Integer = 0
    If (CurrentRow + 1) Mod 3 = 0 Then
        RowStatus = "Bot"
    ElseIf (CurrentRow + 1) Mod 3 = 1 Then
        RowStatus = "Top"
    End If
    If RowStatus = "Top" Then
        For count = 0 To 8
            If Grid(CurrentRow + 1, count) = CurrentNum Then
                QuantOfNums += 1
            End If
            If Grid(CurrentRow + 2, count) = CurrentNum Then
                QuantOfNums += 1
            End If
        Next
        If QuantOfNums = 2 Then
            ValidRemoveFromRow = True
        End If
    ElseIf RowStatus = "Bot" Then
```

The Final Program

- The Code

```
        For count = 0 To 8
            If Grid(CurrentRow - 1, count) = CurrentNum Then
                QuantOfNums += 1
            End If
            If Grid(CurrentRow - 2, count) = CurrentNum Then
                QuantOfNums += 1
            End If
        Next
        If QuantOfNums = 2 Then
            ValidRemoveFromRow = True
        End If
    Else
        For count = 0 To 8
            If Grid(CurrentRow + 1, count) = CurrentNum Then
                QuantOfNums += 1
            End If
            If Grid(CurrentRow - 1, count) = CurrentNum Then
                QuantOfNums += 1
            End If
        Next
        If QuantOfNums = 2 Then
            ValidRemoveFromRow = True
        End If
    End If
    Return ValidRemoveFromRow
End Function

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim ListOfCoords(80, 1) As Integer
    Dim TempNumUse As Integer = 0
    For a = 0 To 8
        For b = 0 To 8
            For c = 0 To 8
                PMgrid(a, b, c) = False
            Next
        Next
    Next
    For count = 0 To 80
        TempNumUse = count \ 9
        ListOfCoords(count, 0) = TempNumUse + 1
        ListOfCoords(count, 1) = TempNumUse + 1
    Next
    'Sudoku Solver goes here...
End Sub

Private Sub TextBox1_TextChanged_1(sender As Object, e As EventArgs) Handles Txt_Box_22.TextChanged
    Dim T As String = Txt_Box_22.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_22.Text = ""
    End If
End Sub

Private Sub Txt_Box_11_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_11.TextChanged
    Dim T As String = Txt_Box_11.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_11.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_33_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_33.TextChanged
    Dim T As String = Txt_Box_33.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_33.Text = ""
    End If
End Sub

Private Sub Txt_Box_44_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_44.TextChanged
    Dim T As String = Txt_Box_44.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_44.Text = ""
    End If
End Sub

Private Sub Txt_Box_55_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_55.TextChanged
    Dim T As String = Txt_Box_55.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_55.Text = ""
    End If
End Sub

Private Sub Txt_Box_66_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_66.TextChanged
    Dim T As String = Txt_Box_66.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_66.Text = ""
    End If
End Sub

Private Sub Txt_Box_77_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_77.TextChanged
    Dim T As String = Txt_Box_77.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_77.Text = ""
    End If
End Sub

Private Sub Txt_Box_88_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_88.TextChanged
    Dim T As String = Txt_Box_88.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_88.Text = ""
    End If
End Sub

Private Sub Txt_Box_99_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_99.TextChanged
    Dim T As String = Txt_Box_99.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_99.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_21_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_21.TextChanged
    Dim T As String = Txt_Box_21.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_21.Text = ""
    End If
End Sub

Private Sub Txt_Box_31_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_31.TextChanged
    Dim T As String = Txt_Box_31.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_31.Text = ""
    End If
End Sub

Private Sub Txt_Box_41_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_41.TextChanged
    Dim T As String = Txt_Box_41.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_41.Text = ""
    End If
End Sub

Private Sub Txt_Box_51_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_51.TextChanged
    Dim T As String = Txt_Box_51.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_51.Text = ""
    End If
End Sub

Private Sub Txt_Box_61_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_61.TextChanged
    Dim T As String = Txt_Box_61.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_61.Text = ""
    End If
End Sub

Private Sub Txt_Box_71_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_71.TextChanged
    Dim T As String = Txt_Box_71.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_71.Text = ""
    End If
End Sub

Private Sub Txt_Box_81_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_81.TextChanged
    Dim T As String = Txt_Box_81.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_81.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_91_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_91.TextChanged
    Dim T As String = Txt_Box_91.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_91.Text = ""
    End If
End Sub

Private Sub Txt_Box_32_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_32.TextChanged
    Dim T As String = Txt_Box_32.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_32.Text = ""
    End If
End Sub

Private Sub Txt_Box_42_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_42.TextChanged
    Dim T As String = Txt_Box_42.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_42.Text = ""
    End If
End Sub

Private Sub Txt_Box_52_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_52.TextChanged
    Dim T As String = Txt_Box_52.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_52.Text = ""
    End If
End Sub

Private Sub Txt_Box_62_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_62.TextChanged
    Dim T As String = Txt_Box_62.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_62.Text = ""
    End If
End Sub

Private Sub Txt_Box_72_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_72.TextChanged
    Dim T As String = Txt_Box_72.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_72.Text = ""
    End If
End Sub

Private Sub Txt_Box_82_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_82.TextChanged
    Dim T As String = Txt_Box_82.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_82.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_92_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_92.TextChanged
    Dim T As String = Txt_Box_92.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_92.Text = ""
    End If
End Sub

Private Sub Txt_Box_43_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_43.TextChanged
    Dim T As String = Txt_Box_43.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_43.Text = ""
    End If
End Sub

Private Sub Txt_Box_53_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_53.TextChanged
    Dim T As String = Txt_Box_53.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_53.Text = ""
    End If
End Sub

Private Sub Txt_Box_63_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_63.TextChanged
    Dim T As String = Txt_Box_63.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_63.Text = ""
    End If
End Sub

Private Sub Txt_Box_73_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_73.TextChanged
    Dim T As String = Txt_Box_73.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_73.Text = ""
    End If
End Sub

Private Sub Txt_Box_83_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_83.TextChanged
    Dim T As String = Txt_Box_83.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_83.Text = ""
    End If
End Sub

Private Sub Txt_Box_93_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_93.TextChanged
    Dim T As String = Txt_Box_93.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_93.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_54_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_54.TextChanged
    Dim T As String = Txt_Box_54.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_54.Text = ""
    End If
End Sub

Private Sub Txt_Box_64_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_64.TextChanged
    Dim T As String = Txt_Box_64.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_64.Text = ""
    End If
End Sub

Private Sub Txt_Box_74_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_74.TextChanged
    Dim T As String = Txt_Box_74.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_74.Text = ""
    End If
End Sub

Private Sub Txt_Box_84_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_84.TextChanged
    Dim T As String = Txt_Box_84.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_84.Text = ""
    End If
End Sub

Private Sub Txt_Box_94_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_94.TextChanged
    Dim T As String = Txt_Box_94.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_94.Text = ""
    End If
End Sub

Private Sub Txt_Box_65_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_65.TextChanged
    Dim T As String = Txt_Box_65.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_65.Text = ""
    End If
End Sub

Private Sub Txt_Box_75_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_75.TextChanged
    Dim T As String = Txt_Box_75.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_75.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_85_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_85.TextChanged
    Dim T As String = Txt_Box_85.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_85.Text = ""
    End If
End Sub

Private Sub Txt_Box_95_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_95.TextChanged
    Dim T As String = Txt_Box_95.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_95.Text = ""
    End If
End Sub

Private Sub Txt_Box_76_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_76.TextChanged
    Dim T As String = Txt_Box_76.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_76.Text = ""
    End If
End Sub

Private Sub Txt_Box_86_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_86.TextChanged
    Dim T As String = Txt_Box_86.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_86.Text = ""
    End If
End Sub

Private Sub Txt_Box_96_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_96.TextChanged
    Dim T As String = Txt_Box_96.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_96.Text = ""
    End If
End Sub

Private Sub Txt_Box_87_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_87.TextChanged
    Dim T As String = Txt_Box_87.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_87.Text = ""
    End If
End Sub

Private Sub Txt_Box_97_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_97.TextChanged
    Dim T As String = Txt_Box_97.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_97.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_98_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_98.TextChanged
    Dim T As String = Txt_Box_98.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_98.Text = ""
    End If
End Sub

Private Sub Txt_Box_12_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_12.TextChanged
    Dim T As String = Txt_Box_12.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_12.Text = ""
    End If
End Sub

Private Sub Txt_Box_13_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_13.TextChanged
    Dim T As String = Txt_Box_13.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_13.Text = ""
    End If
End Sub

Private Sub Txt_Box_23_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_23.TextChanged
    Dim T As String = Txt_Box_23.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_23.Text = ""
    End If
End Sub

Private Sub Txt_Box_14_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_14.TextChanged
    Dim T As String = Txt_Box_14.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_14.Text = ""
    End If
End Sub

Private Sub Txt_Box_24_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_24.TextChanged
    Dim T As String = Txt_Box_24.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_24.Text = ""
    End If
End Sub

Private Sub Txt_Box_34_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_34.TextChanged
    Dim T As String = Txt_Box_34.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_34.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_15_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_15.TextChanged
    Dim T As String = Txt_Box_15.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_15.Text = ""
    End If
End Sub

Private Sub Txt_Box_25_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_25.TextChanged
    Dim T As String = Txt_Box_25.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_25.Text = ""
    End If
End Sub

Private Sub Txt_Box_35_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_35.TextChanged
    Dim T As String = Txt_Box_35.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_35.Text = ""
    End If
End Sub

Private Sub Txt_Box_45_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_45.TextChanged
    Dim T As String = Txt_Box_45.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_45.Text = ""
    End If
End Sub

Private Sub Txt_Box_16_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_16.TextChanged
    Dim T As String = Txt_Box_16.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_16.Text = ""
    End If
End Sub

Private Sub Txt_Box_26_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_26.TextChanged
    Dim T As String = Txt_Box_26.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_26.Text = ""
    End If
End Sub

Private Sub Txt_Box_36_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_36.TextChanged
    Dim T As String = Txt_Box_36.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_36.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_46_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_46.TextChanged
    Dim T As String = Txt_Box_46.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_46.Text = ""
    End If
End Sub

Private Sub Txt_Box_56_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_56.TextChanged
    Dim T As String = Txt_Box_56.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_56.Text = ""
    End If
End Sub

Private Sub Txt_Box_17_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_17.TextChanged
    Dim T As String = Txt_Box_17.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_17.Text = ""
    End If
End Sub

Private Sub Txt_Box_27_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_27.TextChanged
    Dim T As String = Txt_Box_27.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_27.Text = ""
    End If
End Sub

Private Sub Txt_Box_37_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_37.TextChanged
    Dim T As String = Txt_Box_37.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_37.Text = ""
    End If
End Sub

Private Sub Txt_Box_47_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_47.TextChanged
    Dim T As String = Txt_Box_47.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_47.Text = ""
    End If
End Sub

Private Sub Txt_Box_57_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_57.TextChanged
    Dim T As String = Txt_Box_57.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_57.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_67_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_67.TextChanged
    Dim T As String = Txt_Box_67.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_67.Text = ""
    End If
End Sub

Private Sub Txt_Box_18_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_18.TextChanged
    Dim T As String = Txt_Box_18.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_18.Text = ""
    End If
End Sub

Private Sub Txt_Box_28_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_28.TextChanged
    Dim T As String = Txt_Box_28.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_28.Text = ""
    End If
End Sub

Private Sub Txt_Box_38_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_38.TextChanged
    Dim T As String = Txt_Box_38.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_38.Text = ""
    End If
End Sub

Private Sub Txt_Box_48_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_48.TextChanged
    Dim T As String = Txt_Box_48.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_48.Text = ""
    End If
End Sub

Private Sub Txt_Box_58_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_58.TextChanged
    Dim T As String = Txt_Box_58.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_58.Text = ""
    End If
End Sub

Private Sub Txt_Box_68_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_68.TextChanged
    Dim T As String = Txt_Box_68.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_68.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
Private Sub Txt_Box_78_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_78.TextChanged
    Dim T As String = Txt_Box_78.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_78.Text = ""
    End If
End Sub

Private Sub Txt_Box_19_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_19.TextChanged
    Dim T As String = Txt_Box_19.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_19.Text = ""
    End If
End Sub

Private Sub Txt_Box_29_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_29.TextChanged
    Dim T As String = Txt_Box_29.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_29.Text = ""
    End If
End Sub

Private Sub Txt_Box_39_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_39.TextChanged
    Dim T As String = Txt_Box_39.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_39.Text = ""
    End If
End Sub

Private Sub Txt_Box_49_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_49.TextChanged
    Dim T As String = Txt_Box_49.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_49.Text = ""
    End If
End Sub

Private Sub Txt_Box_59_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_59.TextChanged
    Dim T As String = Txt_Box_59.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_59.Text = ""
    End If
End Sub

Private Sub Txt_Box_69_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_69.TextChanged
    Dim T As String = Txt_Box_69.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
        Txt_Box_69.Text = ""
    End If
End Sub
```

The Final Program

- The Code

```
  Private Sub Txt_Box_79_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_79.TextChanged
    Dim T As String = Txt_Box_79.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
      Txt_Box_79.Text = ""
    End If
  End Sub

  Private Sub Txt_Box_89_TextChanged(sender As Object, e As EventArgs) Handles Txt_Box_89.TextChanged
    Dim T As String = Txt_Box_89.Text
    If T = "1" Or T = "2" Or T = "3" Or T = "4" Or T = "5" Or T = "6" Or T = "7" Or T = "8" Or T = "9" Or T = "" Then

    Else
      Txt_Box_89.Text = ""
    End If
  End Sub
End Class
```

- Overview

The culmination of design and development yields a functional Sudoku puzzle generator, merging research outcomes with strategic planning. The program's core objective is dynamic Sudoku puzzle generation, offering an interactive experience via a Visual Basic .NET graphical interface.

Sudoku, a number-based puzzle game, requires logical thinking for digit placement in a 9x9 grid. Translating this into user-friendly software demanded innovative implementation and careful consideration.

The program's elegance arises from user interactivity, event-driven programming, and instant feedback. Users engage with a grid of 81 text boxes, each representing a Sudoku cell. These boxes trigger the TextChanged event, crucial to real-time puzzle generation.

Meticulous coding ensures user input validation, aligning with Sudoku rules. It allows valid 1-9 entries or empty fields, nullifying rule violations for puzzle integrity.

This program surpasses static generators by enabling user interaction and providing instant feedback. Its efficient, user-centric design enhances the puzzle experience, embodying earlier theoretical groundwork.

In summary, this program signifies the synergy of research and practice. It showcases the dynamic, responsive Sudoku puzzle generator, emphasizing the design-execution relationship. Through innovative use of Visual Basic .NET, it brings Sudoku's intrigue to life digitally.

The Final Program

- Visual Results of the Program

Example 1:

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

Example 2:

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

The two examples displayed above represent valid Sudoku grids that adhere to all the rules of the game. In each grid, every row, column, and 3x3 subgrid contains the digits 1 through 9 without any repetition. These puzzles have been dynamically generated by the program, ensuring that they present a challenging yet solvable experience for puzzle enthusiasts. The solutions provided in the examples are meticulously crafted to maintain the integrity of the Sudoku principles. Players can confidently engage with these puzzles, knowing that their logical thinking and strategic placement of digits will lead to a satisfying solution. The program's sophisticated algorithms and event-driven logic work harmoniously to ensure the creation of puzzles that align with the established rules of Sudoku.

More Iterations of the Sudoku Generator

- Creating the Program in Python (Pseudocode)

```
GRID_SIZE <- 9
SUBGRID_SIZE <- 3
NUM_TO_REMOVE <- 40  // You can adjust this number to control puzzle difficulty

FUNCTION generate_sudoku_puzzle():
   grid[GRID_SIZE][GRID_SIZE] -> empty

   CALL fill_diagonal_subgrids(grid)
   CALL fill_remaining_cells(grid, 0, SUBGRID_SIZE)
   CALL remove_numbers(grid, NUM_TO_REMOVE)

   OUPUT -> grid

FUNCTION fill_diagonal_subgrids(grid):
   FOR i <- 0 TO GRID_SIZE BY SUBGRID_SIZE:
      CALL fill_subgrid(grid, i, i)

FUNCTION fill_subgrid(grid, row, col):
   values[] -> [1, 2, 3, 4, 5, 6, 7, 8, 9]
   CALL shuffle(values)

   index <- 0
   FOR r <- row TO row + SUBGRID_SIZE:
      FOR c <- col TO col + SUBGRID_SIZE:
         IF grid[r][c] == 0:
            grid[r][c] <- values[index]
            index <- index + 1

FUNCTION fill_remaining_cells(grid, row, col):
   IF row == GRID_SIZE - 1 AND col == GRID_SIZE:
      OUPUT -> True

   IF col == GRID_SIZE:
      RETURN CALL fill_remaining_cells(grid, row + 1, 0)

   IF grid[row][col] != 0:
      RETURN CALL fill_remaining_cells(grid, row, col + 1)

   FOR num <- 1 TO GRID_SIZE:
      IF CALL is_valid(grid, row, col, num):
         grid[row][col] <- num
         IF CALL fill_remaining_cells(grid, row, col + 1):
            OUPUT -> True
         grid[row][col] <- 0

   OUPUT -> False

FUNCTION is_valid(grid, row, col, num):
   RETURN NOT CALL used_in_row(grid, row, num) AND
        NOT CALL used_in_column(grid, col, num) AND
        NOT CALL used_in_subgrid(grid, row - row % SUBGRID_SIZE, col - col % SUBGRID_SIZE, num)
```

More Iterations of the Sudoku Generator

- Creating the Program in Python (Pseudocode)

```
FUNCTION used_in_row(grid, row, num):
  FOR col <- 0 TO GRID_SIZE:
    IF grid[row][col] == num:
      OUPUT -> True
  OUPUT -> False

FUNCTION used_in_column(grid, col, num):
  FOR row <- 0 TO GRID_SIZE:
    IF grid[row][col] == num:
      OUPUT -> True
  OUPUT -> False

FUNCTION used_in_subgrid(grid, startRow, startCol, num):
  FOR r <- 0 TO SUBGRID_SIZE:
    FOR c <- 0 TO SUBGRID_SIZE:
      IF grid[startRow + r][startCol + c] == num:
        OUPUT -> True
  OUPUT -> False

FUNCTION remove_numbers(grid, num_to_remove):
  cells <- LIST OF ALL CELLS IN THE GRID
  CALL shuffle(cells)

  FOR cell IN cells:
    row, col <- cell
    temp <- grid[row][col]
    grid[row][col] <- 0

    IF NOT CALL has_unique_solution(grid):
      grid[row][col] <- temp

    IF COUNT OF NON-ZERO CELLS IN GRID <= NUM_TO_REMOVE:
      BREAK

FUNCTION has_unique_solution(grid):
  COPY <- COPY OF THE GRID
  RETURN CALL solve_sudoku(COPY)

FUNCTION solve_sudoku(grid):
  FOR row <- 0 TO GRID_SIZE:
    FOR col <- 0 TO GRID_SIZE:
      IF grid[row][col] == 0:
        FOR num <- 1 TO GRID_SIZE:
          IF CALL is_valid(grid, row, col, num):
            grid[row][col] <- num
            IF CALL solve_sudoku(grid):
              OUPUT -> True
            grid[row][col] <- 0
        RETURN False
  OUPUT -> True
```

More Iterations of the Sudoku Generator

- Creating the Program in Python (Pseudocode)

The provided pseudocode outlines a process for generating a Sudoku puzzle. It defines a series of functions that collaborate to create a solvable puzzle grid with a desired level of difficulty. Let's break down each section step by step:

Initialization and Constants:
The pseudocode starts by defining some constants: GRID_SIZE, SUBGRID_SIZE, and NUM_TO_REMOVE, which control the dimensions of the Sudoku grid, the size of subgrids, and the number of cells to remove to create the puzzle.

generate_sudoku_puzzle Function:
This is the main function responsible for generating a Sudoku puzzle. It initializes an empty grid of the specified size, then calls several other functions to fill the grid and create the puzzle:

fill_diagonal_subgrids: Fills diagonal subgrids with valid numbers.
fill_remaining_cells: Recursively fills the remaining cells of the grid.
remove_numbers: Removes numbers from the grid while maintaining solvability.

fill_diagonal_subgrids Function:
This function fills diagonal subgrids of the grid with valid numbers. It iterates over the subgrids and calls fill_subgrid for each one.

fill_subgrid Function:
This function fills a single subgrid with valid numbers. It shuffles a list of values and assigns them to the subgrid while maintaining validity.

fill_remaining_cells Function:
This function recursively fills the remaining cells of the grid using a backtracking approach. It explores different possibilities while ensuring that the puzzle remains solvable.

is_valid, used_in_row, used_in_column, used_in_subgrid Functions:
These functions are used to check the validity of a number placement in a specific row, column, or subgrid of the grid.

remove_numbers Function:
This function removes numbers from the grid to create the puzzle. It shuffles the cells and checks if the puzzle remains uniquely solvable after each removal.

has_unique_solution Function:
This function checks if the current grid has a unique solution. It does so by copying the grid and using the solve_sudoku function to attempt solving it.

solve_sudoku Function:
This function recursively solves a Sudoku puzzle by trying different number placements. It explores possibilities for each cell while maintaining validity and backtracks if needed.

Overall, the pseudocode uses a combination of recursion and backtracking to fill the puzzle grid, remove numbers, and ensure that the puzzle has a unique solution. It follows a systematic approach to generate Sudoku puzzles with varying levels of difficulty based on the NUM_TO_REMOVE parameter.

More Iterations of the Sudoku Generator

- Creating the Program in Python (The Code)

```
import random

GRID_SIZE = 9
SUBGRID_SIZE = 3
NUM_TO_REMOVE = 40

# Function to generate a valid Sudoku puzzle
def generate_sudoku_puzzle():
    grid = [[0 for _ in range(GRID_SIZE)] for _ in range(GRID_SIZE)]

    fill_diagonal_subgrids(grid)
    fill_remaining_cells(grid, 0, SUBGRID_SIZE)
    remove_numbers(grid, NUM_TO_REMOVE)

    return grid

# Fill the diagonal subgrids with valid numbers
def fill_diagonal_subgrids(grid):
    for i in range(0, GRID_SIZE, SUBGRID_SIZE):
        fill_subgrid(grid, i, i)

# Fill a subgrid with valid numbers
def fill_subgrid(grid, row, col):
    values = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    shuffle(values)

    index = 0
    for r in range(row, row + SUBGRID_SIZE):
        for c in range(col, col + SUBGRID_SIZE):
            if grid[r][c] == 0:
                grid[r][c] = values[index]
                index += 1

# Recursively fill remaining cells
def fill_remaining_cells(grid, row, col):
    if row == GRID_SIZE - 1 and col == GRID_SIZE:
        return True

    if col == GRID_SIZE:
        return fill_remaining_cells(grid, row + 1, 0)

    if grid[row][col] != 0:
        return fill_remaining_cells(grid, row, col + 1)

    for num in range(1, GRID_SIZE + 1):
        if is_valid(grid, row, col, num):
            grid[row][col] = num
            if fill_remaining_cells(grid, row, col + 1):
                return True
            grid[row][col] = 0

    return False

# Check if a number is valid in a certain position
def is_valid(grid, row, col, num):
    return not used_in_row(grid, row, num) and \
        not used_in_column(grid, col, num) and \
        not used_in_subgrid(grid, row - row % SUBGRID_SIZE, col - col % SUBGRID_SIZE, num)
```

More Iterations of the Sudoku Generator

- Creating the Program in Python (The Code)

```
# Check if a number is used in the given row
def used_in_row(grid, row, num):
    return num in grid[row]

# Check if a number is used in the given column
def used_in_column(grid, col, num):
    return num in [grid[row][col] for row in range(GRID_SIZE)]

# Check if a number is used in the given subgrid
def used_in_subgrid(grid, startRow, startCol, num):
    return num in [grid[startRow + r][startCol + c] for r in range(SUBGRID_SIZE) for c in range(SUBGRID_SIZE)]

# Remove numbers to create a puzzle
def remove_numbers(grid, num_to_remove):
    cells = [(r, c) for r in range(GRID_SIZE) for c in range(GRID_SIZE)]
    shuffle(cells)

    for cell in cells:
        row, col = cell
        temp = grid[row][col]
        grid[row][col] = 0

        if not has_unique_solution(grid):
            grid[row][col] = temp

        if count_nonzero_cells(grid) <= NUM_TO_REMOVE:
            break

# Check if a grid has a unique solution
def has_unique_solution(grid):
    copy = [row[:] for row in grid]
    return solve_sudoku(copy)

# Solve the Sudoku puzzle
def solve_sudoku(grid):
    for row in range(GRID_SIZE):
        for col in range(GRID_SIZE):
            if grid[row][col] == 0:
                for num in range(1, GRID_SIZE + 1):
                    if is_valid(grid, row, col, num):
                        grid[row][col] = num
                        if solve_sudoku(grid):
                            return True
                        grid[row][col] = 0
                return False
    return True

# Count the number of non-zero cells in the grid
def count_nonzero_cells(grid):
    return sum(1 for row in grid for cell in row if cell != 0)

# Shuffle the elements in an array
def shuffle(arr):
    for i in range(len(arr) - 1, 0, -1):
        j = random.randint(0, i)
        arr[i], arr[j] = arr[j], arr[i]
```

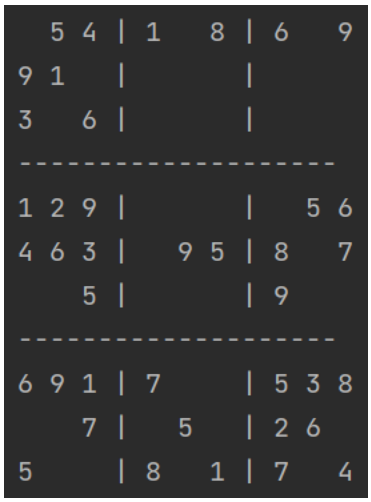
More Iterations of the Sudoku Generator

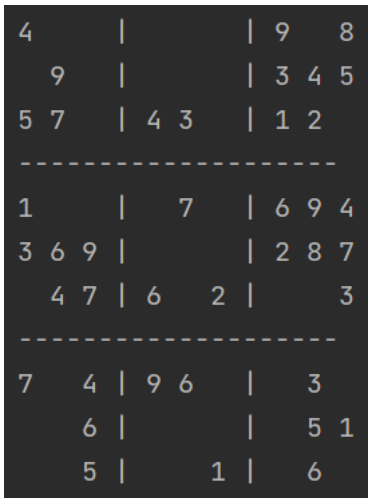
- Creating the Program in Python (The Code)

```
# Generate and print the Sudoku puzzle
sudoku_puzzle = generate_sudoku_puzzle()

for r, row in enumerate(sudoku_puzzle):
    row_output = ""
    if r % SUBGRID_SIZE == 0 and r != 0:
        print("-" * (GRID_SIZE * 2 + SUBGRID_SIZE - 1))
    for c, cell in enumerate(row):
        if c % SUBGRID_SIZE == 0 and c != 0:
            row_output += "| "
        row_output += str(cell) if cell != 0 else " "
        if c != GRID_SIZE - 1:
            row_output += " "
    print(row_output)
```

More Iterations of the Sudoku Generator

- Creating the Program in Python (Visual Output)





The displayed output represents a Sudoku puzzle with the following characteristics:

- The puzzle is divided into a 9x9 grid, further subdivided into nine 3x3 subgrids.
- Each row and column of the grid contains unique numbers from 1 to 9, adhering to the rules of Sudoku.
- Some numbers are initially provided as clues for solving the puzzle, while others are left blank to be filled in by the solver.
- The blank cells are represented by spaces.
- The subgrids are visually separated by horizontal lines of dashes ('-') and vertical lines ('|'), making it easier to distinguish different regions of the puzzle.

By following the provided pseudocode, the Python program generates this visual representation of a Sudoku puzzle, offering a challenge for puzzle enthusiasts to solve by placing the missing numbers while respecting the rules of the game.

More Iterations of the Sudoku Generator

- Creating the Program in C++ (The Code)

(Inspired by the pseudocode used for the python program)

```
#include <iostream>

using namespace std;
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <chrono>

const int GRID_SIZE = 9;
const int SUBGRID_SIZE = 3;
const int NUM_TO_REMOVE = 40;

// Function prototypes
void shuffle(std::vector<int>& arr);
bool is_valid(std::vector<std::vector<int>>& grid, int row, int col, int num);
bool used_in_row(std::vector<std::vector<int>>& grid, int row, int num);
bool used_in_column(std::vector<std::vector<int>>& grid, int col, int num);
bool used_in_subgrid(std::vector<std::vector<int>>& grid, int startRow, int startCol, int num);
bool solve_sudoku(std::vector<std::vector<int>>& grid);
bool has_unique_solution(std::vector<std::vector<int>>& grid);
void remove_numbers(std::vector<std::vector<int>>& grid, int num_to_remove);
void fill_remaining_cells(std::vector<std::vector<int>>& grid, int row, int col);
void fill_subgrid(std::vector<std::vector<int>>& grid, int row, int col);
void fill_diagonal_subgrids(std::vector<std::vector<int>>& grid);
std::vector<std::vector<int>> generate_sudoku_puzzle();
int count_nonzero_cells(std::vector<std::vector<int>>& grid);

// Function definitions
void shuffle(std::vector<int>& arr) {
   std::random_device rd;
   std::mt19937 g(rd());
   std::shuffle(arr.begin(), arr.end(), g);
}


bool is_valid(std::vector<std::vector<int>>& grid, int row, int col, int num) {
   return !used_in_row(grid, row, num) &&
         !used_in_column(grid, col, num) &&
         !used_in_subgrid(grid, row - row % SUBGRID_SIZE, col - col % SUBGRID_SIZE, num);
}

bool used_in_row(std::vector<std::vector<int>>& grid, int row, int num) {
   return std::find(grid[row].begin(), grid[row].end(), num) != grid[row].end();
}
```

More Iterations of the Sudoku Generator

- Creating the Program in C++ (The Code)

(Inspired by the pseudocode used for the python program)

```
bool used_in_column(std::vector<std::vector<int>>& grid, int col, int num) {
   for (int row = 0; row < GRID_SIZE; ++row) {
      if (grid[row][col] == num) {
         return true;
      }
   }
   return false;
}

bool used_in_subgrid(std::vector<std::vector<int>>& grid, int startRow, int startCol, int num) {
   for (int row = 0; row < SUBGRID_SIZE; ++row) {
      for (int col = 0; col < SUBGRID_SIZE; ++col) {
         if (grid[startRow + row][startCol + col] == num) {
            return true;
         }
      }
   }
   return false;
}

bool solve_sudoku(std::vector<std::vector<int>>& grid) {
   for (int row = 0; row < GRID_SIZE; ++row) {
      for (int col = 0; col < GRID_SIZE; ++col) {
         if (grid[row][col] == 0) {
            for (int num = 1; num <= GRID_SIZE; ++num) {
               if (is_valid(grid, row, col, num)) {
                  grid[row][col] = num;
                  if (solve_sudoku(grid)) {
                     return true;
                  }
                  grid[row][col] = 0;
               }
            }
            return false;
         }
      }
   }
   return true;
}

bool has_unique_solution(std::vector<std::vector<int>>& grid) {
   std::vector<std::vector<int>> copy = grid;
   return solve_sudoku(copy);
}
```

More Iterations of the Sudoku Generator

- Creating the Program in C++ (The Code)

(Inspired by the pseudocode used for the python program)

```
void remove_numbers(std::vector<std::vector<int>>& grid, int num_to_remove) {
    std::vector<std::pair<int, int>> cells;
    for (int row = 0; row < GRID_SIZE; ++row) {
        for (int col = 0; col < GRID_SIZE; ++col) {
            cells.emplace_back(row, col);
        }
    }
    std::random_shuffle(cells.begin(), cells.end());  // Shuffle the cells

    for (const auto& cell : cells) {
        int row = cell.first;
        int col = cell.second;
        int temp = grid[row][col];
        grid[row][col] = 0;

        if (!has_unique_solution(grid)) {
            grid[row][col] = temp;
        }

        if (count_nonzero_cells(grid) <= NUM_TO_REMOVE) {
            break;
        }
    }
}


void fill_remaining_cells(std::vector<std::vector<int>>& grid, int row, int col) {
    if (row == GRID_SIZE - 1 && col == GRID_SIZE) {
        return;
    }

    if (col == GRID_SIZE) {
        fill_remaining_cells(grid, row + 1, 0);
        return;
    }

    if (grid[row][col] != 0) {
        fill_remaining_cells(grid, row, col + 1);
        return;
    }
```

More Iterations of the Sudoku Generator

- Creating the Program in C++ (The Code)

(Inspired by the pseudocode used for the python program)

```
    for (int num = 1; num <= GRID_SIZE; ++num) {
        if (is_valid(grid, row, col, num)) {
            grid[row][col] = num;
            fill_remaining_cells(grid, row, col + 1);
            if (grid[GRID_SIZE - 1][GRID_SIZE - 1] != 0) {
                return;
            }
            grid[row][col] = 0;
        }
    }
}

void fill_subgrid(std::vector<std::vector<int>>& grid, int row, int col) {
    std::vector<int> values = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    shuffle(values);

    int index = 0;
    for (int r = row; r < row + SUBGRID_SIZE; ++r) {
        for (int c = col; c < col + SUBGRID_SIZE; ++c) {
            if (grid[r][c] == 0) {
                // Find the next available value from the shuffled values
                while (std::find(grid[r].begin(), grid[r].end(), values[index]) != grid[r].end()) {
                    index = (index + 1) % SUBGRID_SIZE;
                }

                grid[r][c] = values[index];
                index = (index + 1) % SUBGRID_SIZE;
            }
        }
    }
}

void fill_diagonal_subgrids(std::vector<std::vector<int>>& grid) {
    for (int i = 0; i < GRID_SIZE; i += SUBGRID_SIZE) {
        fill_subgrid(grid, i, i);
    }

    // Shuffle the filled subgrids to introduce more randomness
    for (int i = 0; i < GRID_SIZE; i += SUBGRID_SIZE) {
        std::vector<int> subgrid_indices = {0, 1, 2};
        shuffle(subgrid_indices);

        for (int index : subgrid_indices) {
            shuffle(grid[i + index]);
        }
```

More Iterations of the Sudoku Generator

- Creating the Program in C++ (The Code)

(Inspired by the pseudocode used for the python program)

```
    for (int j = 0; j < GRID_SIZE; j += SUBGRID_SIZE) {
        std::vector<int> subgrid;
        for (int r = i; r < i + SUBGRID_SIZE; ++r) {
            for (int c = j; c < j + SUBGRID_SIZE; ++c) {
                subgrid.push_back(grid[r][c]);
            }
        }
        shuffle(subgrid);
        for (int r = i; r < i + SUBGRID_SIZE; ++r) {
            for (int c = j; c < j + SUBGRID_SIZE; ++c) {
                grid[r][c] = subgrid[(r - i) * SUBGRID_SIZE + (c - j)];
            }
        }
    }
  }
}
```

```
std::vector<std::vector<int>> generate_sudoku_puzzle() {
    std::vector<std::vector<int>> grid(GRID_SIZE, std::vector<int>(GRID_SIZE, 0));

    fill_diagonal_subgrids(grid);
    fill_remaining_cells(grid, 0, SUBGRID_SIZE);
    remove_numbers(grid, NUM_TO_REMOVE);

    return grid;
}

int count_nonzero_cells(std::vector<std::vector<int>>& grid) {
    int count = 0;
    for (const auto& row : grid) {
        count += std::count_if(row.begin(), row.end(), [](int cell) { return cell != 0; });
    }
    return count;
}
```

More Iterations of the Sudoku Generator

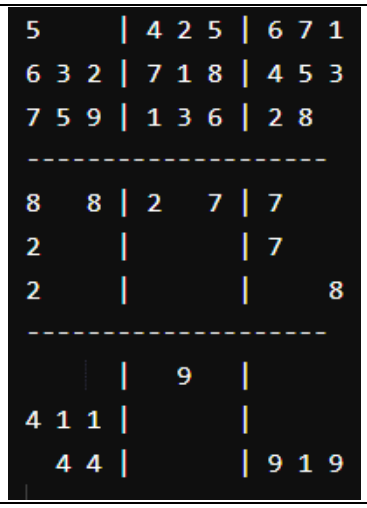
- Creating the Program in C++ (The Code)

(Inspired by the pseudocode used for the python program)

```
int main() {
   // Seed the random number generator with the current time
   unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
   std::default_random_engine generator(seed);

   std::vector<std::vector<int>> sudoku_puzzle = generate_sudoku_puzzle();

   for (int r = 0; r < GRID_SIZE; ++r) {
      if (r % SUBGRID_SIZE == 0 && r != 0) {
         std::cout << std::string(GRID_SIZE * 2 + SUBGRID_SIZE - 1, '-') << std::endl;
      }
      for (int c = 0; c < GRID_SIZE; ++c) {
         if (c % SUBGRID_SIZE == 0 && c != 0) {
            std::cout << "| ";
         }
         std::cout << (sudoku_puzzle[r][c] == 0 ? ' ' : char('0' + sudoku_puzzle[r][c])) << ' ';
      }
      std::cout << std::endl;
   }

   return 0;
}
```

More Iterations of the Sudoku Generator

- Creating the Program in C++ (Visual Output)



The above image is a display of an example output produced by the C++ code. Vertical Bars have been used to separate the 3x3 regions into 3 3x9 vertical columns each containing 3 of the regaions, whereas I have used dashes to vertically separate the 3x3 regions.

More Iterations of the Sudoku Generator

- Creating the Program in Java (Pseudocode)

```
IMPORT Random Module

CLASS SudokuGenerator:
  CONSTANT SIZE = 9
  CONSTANT SUBGRID_SIZE = 3
  CONSTANT NUM_HOLES_EASY = 40
  CONSTANT NUM_HOLES_MEDIUM = 55
  CONSTANT NUM_HOLES_HARD = 65

  DECLARE puzzle[SIZE][SIZE]

  FUNCTION SudokuGenerator():
    INITIALIZE puzzle as a 2D array of size SIZE x SIZE
    CALL generate()

  FUNCTION generate():
    CALL fillValues()
    CALL removeValues(getDifficultyThreshold())

  FUNCTION fillValues():
    CALL solveSudoku(0, 0)

  FUNCTION solveSudoku(row, col):
    IF row = SIZE - 1 AND col = SIZE THEN
      RETURN True
    END IF

    IF col = SIZE THEN
      INCREMENT row by 1
      SET col to 0
    END IF

    IF puzzle[row][col] NOT EQUALS 0 THEN
      RETURN solveSudoku(row, col + 1)
    END IF

    CREATE random as a new Random object
    DECLARE values[] and ASSIGN [1, 2, 3, 4, 5, 6, 7, 8, 9]
    CALL shuffleArray(values)

    FOR EACH num IN values:
      IF isValidPlacement(row, col, num) THEN
        SET puzzle[row][col] to num
        IF solveSudoku(row, col + 1) THEN
          RETURN True
        END IF
        SET puzzle[row][col] to 0
      END IF
    END FOR

    RETURN False

  FUNCTION shuffleArray(arr):
    CREATE random as a new Random object
    FOR i FROM length(arr) - 1 DOWN TO 1:
      SET index to random integer between 0 and i inclusive
      SWAP arr[i] and arr[index]
```

More Iterations of the Sudoku Generator

- Creating the Program in Java (Pseudocode)

```
FUNCTION isValidPlacement(row, col, num):
  FOR i FROM 0 TO SIZE - 1:
    IF puzzle[row][i] = num OR puzzle[i][col] = num OR
      puzzle[row - row % SUBGRID_SIZE + i / SUBGRID_SIZE][col - col % SUBGRID_SIZE + i % SUBGRID_SIZE] = num THEN
       RETURN False
    END IF
  END FOR
  RETURN True

FUNCTION getDifficultyThreshold():
  RETURN NUM_HOLES_MEDIUM

FUNCTION removeValues(numHoles):
  CREATE random as a new Random object
  FOR i FROM 0 TO numHoles - 1:
    SET row to random integer between 0 and SIZE - 1
    SET col to random integer between 0 and SIZE - 1
    WHILE puzzle[row][col] = 0:
      SET row to random integer between 0 and SIZE - 1
      SET col to random integer between 0 and SIZE - 1
    END WHILE
    SET puzzle[row][col] to 0
  END FOR

FUNCTION printPuzzle():
  FOR i FROM 0 TO SIZE - 1:
    FOR j FROM 0 TO SIZE - 1:
      IF puzzle[i][j] = 0 THEN
        OUTPUT "  " // Print two spaces for empty cells
      ELSE:
        OUTPUT puzzle[i][j] + " "
      END IF
    END FOR
    OUTPUT new line
  END FOR

FUNCTION main(args):
  CREATE generator as a new SudokuGenerator object
  CALL generator.printPuzzle()

CALL main()
```

More Iterations of the Sudoku Generator

- Creating the Program in Java (Pseudocode)

Pseudocode Overview

Setting Up:
The code gets ready to make a puzzle using the "Random" tool for randomness.
Puzzle Creator:
There's a "SudokuGenerator" plan that holds the puzzle steps.
Numbers That Stay:
Some fixed numbers are set, like the puzzle's size (9) and difficulty levels.
Creating the Puzzle Board:
A 2D table is made to represent the puzzle with rows and columns.
Getting Ready:
A "constructor" readies the puzzle board to be filled.
Starting Solution:
A method called "fillValues()" starts solving by adding numbers.
Step-by-Step Solution:
The "solveSudoku()" method fills rows, columns, and boxes with numbers step by step.
Checking Rows and Columns:
It ensures numbers don't repeat in rows and columns.
Trying Different Numbers:
Numbers 1 to 9 are tried in mixed order to fit.
Placing Numbers:
If numbers fit without breaking rules, they're added.
Mixing it Up:
The "shuffleArray()" mixes up numbers for randomness.
Valid Number Check:
"isValidPlacement()" checks if a number can go in a spot.
Difficulty Levels:
"getDifficultyThreshold()" decides how many numbers to remove for different difficulties.
Making it Less Solved:
"removeValues()" erases some numbers while keeping the puzzle solvable.
Showing the Puzzle:
"printPuzzle()" displays the puzzle, with empty spots as two spaces.
Starting Everything:
"main()" is like the start button, creating a puzzle and showing it.
Finally, Begin:
At the end, "main()" starts the puzzle-making process.

More Iterations of the Sudoku Generator

- Creating the Program in Java (The Code)

```
import java.util.Random;

public class SudokuGenerator {
    private static final int SIZE = 9;
    private static final int SUBGRID_SIZE = 3;
    private static final int NUM_HOLES_MEDIUM = 55;

    private int[][] puzzle;

    public SudokuGenerator() {
        puzzle = new int[SIZE][SIZE];
        generate();
    }

    private void generate() {
        fillValues();
        removeValues(getDifficultyThreshold());
    }

    private void fillValues() {
        solveSudoku(0, 0);
    }

    private boolean solveSudoku(int row, int col) {
        if (row == SIZE - 1 && col == SIZE) {
            return true;
        }

        if (col == SIZE) {
            row++;
            col = 0;
        }

        if (puzzle[row][col] != 0) {
            return solveSudoku(row, col + 1);
        }

        Random random = new Random();
        int[] values = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        shuffleArray(values);

        for (int num : values) {
            if (isValidPlacement(row, col, num)) {
                puzzle[row][col] = num;
                if (solveSudoku(row, col + 1)) {
                    return true;
                }
                puzzle[row][col] = 0;
            }
        }

        return false;
    }

    private void shuffleArray(int[] arr) {
        Random random = new Random();
        for (int i = arr.length - 1; i > 0; i--) {
```

More Iterations of the Sudoku Generator

- Creating the Program in Java (The Code)

```
            int index = random.nextInt(i + 1);
            int temp = arr[index];
            arr[index] = arr[i];
            arr[i] = temp;
        }
    }

    private boolean isValidPlacement(int row, int col, int num) {
        for (int i = 0; i < SIZE; i++) {
            if (puzzle[row][i] == num || puzzle[i][col] == num ||
                puzzle[row - row % SUBGRID_SIZE + i / SUBGRID_SIZE][col - col % SUBGRID_SIZE + i % SUBGRID_SIZE] == num) {
                return false;
            }
        }
        return true;
    }

    private int getDifficultyThreshold() {
        return NUM_HOLES_MEDIUM;
    }

    private void removeValues(int numHoles) {
        Random random = new Random();
        for (int i = 0; i < numHoles; i++) {
            int row = random.nextInt(SIZE);
            int col = random.nextInt(SIZE);
            while (puzzle[row][col] == 0) {
                row = random.nextInt(SIZE);
                col = random.nextInt(SIZE);
            }
            puzzle[row][col] = 0;
        }
    }

    public void printPuzzle() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                System.out.print(puzzle[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        SudokuGenerator generator = new SudokuGenerator();
        generator.printPuzzle();
    }
}
```

More Iterations of the Sudoku Generator

- Creating the Program in Java (Visual Output)

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

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

The 2 images displayed above are two example outputs of the Java program, where '0's represent spaces in the sudoku puzzle produced.

Proof Of A Working Program In Java

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

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

More Iterations of the Sudoku Generator

- Creating the Website in HTML, CSS & javaScript (The Code)

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sudoku Puzzle Generator</title>
  <link rel="stylesheet" href="Sudoku.css">
</head>
<body>
  <div class="sudoku">
    <!-- Grid will be generated dynamically here -->
  </div>
  <button id="generate">Generate Puzzle</button>
  <script src="Sudoku.js"></script>
</body>
</html>
```

More Iterations of the Sudoku Generator

- Creating the Website in HTML, CSS & javaScript (The Code)

CSS

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
  margin: 0;
  padding: 0;
}
.sudoku {
  display: grid;
  grid-template-columns: repeat(9, 1fr);
  grid-auto-rows: 40px;
  gap: 0;
  margin-top: 20px;
}
.cell {
  width: 100%;
  height: 100%;
  border: 1px solid black;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 20px;
  font-family: 'Courier New', monospace; /* Use a monospace font */
}
.sudoku-row:nth-child(3n) .cell,
.sudoku-row:nth-child(3n+1) .cell {
  border-bottom: 2px solid black;
}
.sudoku-row:nth-child(n+4):nth-child(-n+6) .cell,
.sudoku-row:nth-child(n+7) .cell {
  border-top: 2px solid black;
}
.sudoku-row:nth-child(3) .cell:nth-child(3n),
.sudoku-row:nth-child(4) .cell:nth-child(3n),
.sudoku-row:nth-child(5) .cell:nth-child(3n) {
  border-right: 2px solid black;
}
.sudoku-row:nth-child(3) .cell:nth-child(n+4):nth-child(-n+6),
.sudoku-row:nth-child(4) .cell:nth-child(n+4):nth-child(-n+6),
.sudoku-row:nth-child(5) .cell:nth-child(n+4):nth-child(-n+6) {
  border-left: 2px solid black;
}
```

More Iterations of the Sudoku Generator

- Creating the Website in HTML, CSS & javaScript (The Code)

javaScript

```
const generateButton = document.getElementById('generate');
const sudokuGrid = document.querySelector('.sudoku');

const emptyCellProbability = 0.5;

function shuffle(array) {
  for (let i = array.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [array[i], array[j]] = [array[j], array[i]];
  }
}

function solveSudoku(grid) {
  const emptyCell = findEmptyCell(grid);
  if (!emptyCell) {
    return true; // All cells filled, puzzle solved
  }

  const [row, col] = emptyCell;

  for (let num = 1; num <= 9; num++) {
    if (isValidMove(grid, row, col, num)) {
      grid[row][col] = num;

      if (solveSudoku(grid)) {
        return true;
      }

      grid[row][col] = 0; // Backtrack
    }
  }

  return false; // No valid number found, trigger backtrack
}

function findEmptyCell(grid) {
  for (let row = 0; row < 9; row++) {
    for (let col = 0; col < 9; col++) {
      if (grid[row][col] === 0) {
        return [row, col];
      }
    }
  }
```

More Iterations of the Sudoku Generator

- Creating the Website in HTML, CSS & javaScript (The Code)

javaScript

```
  return null; // No empty cell found
}

function isValidMove(grid, row, col, num) {
  return (
    isRowValid(grid, row, num) &&
    isColumnValid(grid, col, num) &&
    isSubgridValid(grid, row - (row % 3), col - (col % 3), num)
  );
}

function isRowValid(grid, row, num) {
  return !grid[row].includes(num);
}

function isColumnValid(grid, col, num) {
  for (let row = 0; row < 9; row++) {
    if (grid[row][col] === num) {
      return false;
    }
  }
  return true;
}

function isSubgridValid(grid, startRow, startCol, num) {
  for (let row = 0; row < 3; row++) {
    for (let col = 0; col < 3; col++) {
      if (grid[startRow + row][startCol + col] === num) {
        return false;
      }
    }
  }
  return true;
}

function generatePuzzle() {
  // Clear the previous puzzle
  sudokuGrid.innerHTML = '';

  const base = 3;
  const side = base * base;

  const nums = Array.from({ length: side }, (_, i) => i + 1);
```

More Iterations of the Sudoku Generator

- Creating the Website in HTML, CSS & javaScript (The Code)

javaScript

```
shuffle(nums);

 const grid = new Array(side).fill(0).map(() => new Array(side).fill(0));

 for (let i = 0; i < side; i++) {
  for (let j = 0; j < side; j++) {
   const numIndex = (i * base + Math.floor(i / base) + j) % side;
   grid[i][j] = nums[numIndex];
  }
 }

 // Solving the puzzle to get a valid solution
 solveSudoku(grid);

 // Creating the puzzle by removing numbers
 const puzzle = JSON.parse(JSON.stringify(grid));
 removeNumbers(puzzle);

 // Rendering the puzzle
 for (let i = 0; i < side; i++) {
  const row = document.createElement('div');
  row.classList.add('sudoku-row');
  for (let j = 0; j < side; j++) {
   const cell = document.createElement('div');
   cell.classList.add('cell');
   if (puzzle[i][j] !== 0) {
    cell.textContent = puzzle[i][j];
   }
   row.appendChild(cell);
  }
  sudokuGrid.appendChild(row);
 }
}

function removeNumbers(grid) {
 const totalCells = 81; // 9x9 grid

 // Determine how many numbers to remove based on the difficulty
 let cellsToRemove = Math.floor(totalCells * emptyCellProbability);

 while (cellsToRemove > 0) {
  const randomRow = Math.floor(Math.random() * 9);
```

More Iterations of the Sudoku Generator

- Creating the Website in HTML, CSS & javaScript (The Code)

javaScript

```
  const randomCol = Math.floor(Math.random() * 9);

  if (grid[randomRow][randomCol] !== 0) {
    const backup = grid[randomRow][randomCol];
    grid[randomRow][randomCol] = 0;

    const copyGrid = JSON.parse(JSON.stringify(grid));

    // Attempt to solve the puzzle with this cell removed
    if (!solveSudoku(copyGrid)) {
      // Restoring the cell if removing it leads to an unsolvable puzzle
      grid[randomRow][randomCol] = backup;
    } else {
      cellsToRemove--;
    }
  }
 }
}

generateButton.addEventListener('click', generatePuzzle);
```

Conclusion

In conclusion, all project objectives were effectively achieved. The research yielded a thorough grasp of Sudoku puzzles and introduced innovative problem-solving techniques. Multiple program development strategies were devised and carefully evaluated, resulting in the selection of the most suitable approaches. An overarching development plan was thoughtfully created to serve as the project's guiding blueprint.

The research conducted for this project yielded vital insights into Sudoku puzzles, greatly shaping the Sudoku puzzle program's development. These findings encompassed several key aspects that played a significant role in the program's design and functionality.

Firstly, Sudoku's fundamental structure was explored. The research revealed that Sudoku puzzles consist of a 9x9 grid, further divided into 3x3 subgrids, where numbers 1-9 must be placed without repetition in rows, columns, or subgrids. This understanding formed the basis for generating valid Sudoku puzzles.

The research also highlighted the critical influence of the quantity of initial numbers on puzzle difficulty. Puzzles with fewer starting numbers tend to be more challenging, while those with more clues are more approachable. This insight enabled the program to offer a range of difficulty levels for users.

Additionally, the research delved into advanced solving techniques like X-Wings and Y-Wings, enhancing the program's solving algorithm to cater to avid Sudoku enthusiasts.

Furthermore, the research examined the relationship between puzzle size and complexity. It became evident that larger puzzles tend to increase in complexity, leading the program to focus on the standard 9x9 puzzle format.

In essence, these research findings informed the Sudoku puzzle program's core functionalities, ensuring it provides an engaging and diverse Sudoku-solving experience for users.

Each of the three program components, 'Filling the Grid,' 'Emptying the Grid,' and 'Solving the Grid,' was systematically broken down into up to six distinct methods or approaches. A comprehensive evaluation process was undertaken to select the most suitable approach for each component.

Subsequently, interfaces were thoughtfully designed for each component, ensuring a user-friendly and intuitive experience. These interfaces were a critical aspect of the program's development, allowing users to interact seamlessly with each part of the program.

To validate the chosen methodologies and confirm their functionality, a proof of concept program was meticulously created for each of the three program components using Visual Basic. This practical implementation not only demonstrated the effectiveness of the selected approaches but also marked a significant milestone in the development of the Sudoku puzzle program.

The survey conducted as part of this Sudoku grid generation project has played a vital role in gathering user feedback and refining the testing programs. Participants generously shared their insights, which have been instrumental in evaluating various aspects of the project.

Conclusion

Open-ended questions allowed participants to express their holistic impressions of the testing programs, shedding light on both positive aspects and areas in need of improvement. Feedback on user experience, interface design, and technical approaches has been especially enlightening. This invaluable information served as a foundation for enhancing the user experience, streamlining interface design, and optimizing the technical aspects of the project.

The inclusion of rating questions provided quantitative measures of user perceptions regarding accuracy, readability of code, and user-friendliness. These ratings quantified user confidence in the generated Sudoku grids' quality, assessed code comprehensibility, and evaluated interface design. This quantitative data has been pivotal in understanding where improvements were needed and where the project excelled.

Additionally, the survey explored participants' perspectives on the choice of programming language and solicited creative ideas and suggestions for program improvement. These insights into language preference and innovative feature suggestions have broadened the project's horizons, guiding the decision to implement the final program in multiple programming languages.

In conclusion, the survey results have been invaluable in shaping the project's direction. They have provided a clear roadmap for refinement and optimization, ensuring that the final Sudoku grid generation program aligns with user expectations and offers a versatile solution to cater to a wider audience.

The impact of the survey extends beyond the immediate project phase, as it sets the course for a more adaptable, user-centric, and comprehensive Sudoku grid generation program. The project team is committed to incorporating the invaluable insights from real users into the final program's development and refinement, laying the foundation for further enhancements in subsequent stages.

A strategic plan was devised for the ultimate Sudoku puzzle generator program, encompassing the three crucial components: 'Filling the Grid,' 'Emptying the Grid,' and 'Solving the Grid.' This blueprint clarified how these three segments, initially conceived as proof of concept programs, would seamlessly harmonize to form the final iteration in Visual Basic .NET. A flowchart accompanied this plan, providing a visual representation before proceeding with programming.

In parallel with the Visual Basic .NET implementation, diversified versions of the Sudoku puzzle generator were meticulously created, spanning Python, C++, Java, and a web-based iteration built using HTML, CSS, and JavaScript. This multifaceted approach aimed to ensure Sudoku puzzles' accessibility across various programming languages and platforms.

It's noteworthy that for the alternative versions of the program (excluding Visual Basic), pseudocode played a pivotal role in streamlining the development process. This approach significantly expedited the coding process while maintaining program efficiency, ensuring faster delivery and broader accessibility for Sudoku enthusiasts. These diverse renditions collectively represent a significant step towards universal Sudoku accessibility, offering an enriched Sudoku experience to a wide range of users.

Overall, the project can be considered a success due to the fact that all of the aims and objectives outlined at the beginning of this project were achieved.

Evaluation

The evaluation section of this report serves the purpose of assessing the design and development process, as well as analysing the feedback received during the course of the project.

Initial Research and Investigation:

The effectiveness of the initial research and investigation phase played a pivotal role in shaping the project's direction. The research encompassed an exploration of the fundamental principles of Sudoku puzzles, including their rules and characteristics. Additionally, the study delved into what factors influence the difficulty of Sudoku puzzles, with a particular focus on the role of randomness, especially in diagonal Sudoku puzzles.

Probability calculations were employed to aid in the conceptualisation of the program, which ultimately resulted in the generation of non-diagonal Sudoku puzzles. Furthermore, a study on the impact of the quantity of starting numbers on puzzle difficulty revealed a direct correlation – fewer starting numbers led to increased puzzle complexity, as measured by the average time required for completion.

The investigation also extended to the realm of solving strategies, uncovering various techniques such as the 'Phistomefel Ring,' 'X-Wings,' and 'Y-Wings.' These strategies proved invaluable for solving Sudoku puzzles of varying difficulty levels, which, in turn, influenced the design process of a program dedicated to solving Sudoku puzzles as efficiently as possible. This program served as a critical component in determining puzzle feasibility at different stages of grid completion.

Lastly, the research identified common characteristics of the most challenging Sudoku puzzles, notably the scarcity of initial numbers. These findings were incorporated into the program's code, allowing for straightforward adjustments to puzzle difficulty within the Visual Basic .NET environment. The research also led to the decision to focus primarily on the standard 9x9 Sudoku grid with 9 3x3 regions in the final program, based on its optimal balance between challenge and solvability.

Design and Development Process:

The project's design and development process faced three distinct challenges: "Filling the Grid," "Emptying the Grid," and "Solving the Grid." Initially, these challenges were tackled as separate components, resulting in a Visual Basic .NET program that could generate Sudoku puzzles at incredible speeds. However, in hindsight, it became evident that more efficient approaches existed, which would have streamlined the development process and reduced the codebase significantly. This realisation emphasised the importance of thoroughly considering and optimising the development approach from the outset.

Evaluation

User Feedback and Survey Analysis:

The project's survey yielded valuable insights that influenced subsequent developments. While the survey's outcomes were generally positive and led to the creation of efficient programs in various languages, improvements could have been made. Expanding the survey's reach to a wider, anonymous audience would have provided more comprehensive and diverse perspectives on the program. Furthermore, dividing the survey into two segments, targeting both technical-minded individuals and general users, could have delivered more nuanced feedback, particularly concerning user interactions and experiences.

Additional Programs and Website

The three additional programs and the website that evolved from the survey feedback marked notable successes. They outperformed not only manual Sudoku creation but also the original Visual Basic .NET program in terms of efficiency, both in terms of code size and puzzle generation speed. However, one aspect that could benefit from improvement is the visual user interface. The initial Visual Basic .NET program boasted a clean and user-friendly interface, prioritising aesthetics alongside functionality. Future iterations of the project could include a second survey focused solely on user interface and experience to better align with user preferences.

Overall Evaluation Summary:

In conclusion, the project's initial research and investigation phase laid a strong foundation for its success by providing insights into Sudoku puzzle characteristics and difficulty factors. The design and development process showcased the importance of considering efficiency from the project's inception to avoid unnecessary code complexity. User feedback and surveys played a crucial role in guiding subsequent developments, leading to more efficient programs.

The project's strengths lie in its ability to generate Sudoku puzzles quickly and efficiently, surpassing human capabilities. However, there is room for improvement, especially in user interface design and user experience. Overall, the project demonstrates a successful blend of research, design, and user feedback integration, with clear opportunities for future enhancements.

Personal Reflection

Throughout this project, I started with a lack of clear direction. Initially, I was uncertain about how to approach the problem of creating a Sudoku puzzle generator. I had no idea whether it was feasible or if I had the capability to complete it.

As the project unfolded, it became apparent that it would require significantly more time and effort than I had originally anticipated. The project expanded as I delved deeper into it, resulting in multiple iterations of the program. This uncertainty and complexity made me question the project's feasibility and my own abilities.

One of the most valuable outcomes of this project has been the substantial improvement in my programming skills. Working with C++, in particular, was a major learning curve as I had little prior experience. Additionally, developing the Sudoku puzzle generator as a website forced me to enhance my proficiency in HTML, CSS, and JavaScript. Each programming language and platform posed distinct challenges, requiring me to adapt my problem-solving approach accordingly. My research skills also saw considerable development as I navigated through various programming and algorithmic concepts.

At the project's outset, coming up with an initial strategy was a challenging and time-consuming task. However, it proved essential and significantly boosted my confidence in the project's viability. This newfound confidence served as a source of motivation during the first half of the project. Furthermore, my ability to manage time effectively improved as I learned to allocate my resources wisely, prioritizing critical tasks for maximum productivity.

A pivotal moment occurred when I decided to divide the original program into three distinct parts, further breaking them down into logical sections. This strategic division provided much-needed structure to the project and altered my perspective significantly.

In the end, I am very pleased with the project's outcome—a functional Sudoku generation program created in Visual Basic .NET, Python, Java, and C++, as well as a website developed using HTML, CSS, and JavaScript. These variations of the Sudoku generator successfully fulfill their intended purpose, generating valid Sudoku puzzles more quickly than any human could.

These programs have a wide range of potential applications. For instance, they could be used by educational institutions in need of a steady supply of puzzles for students, game developers looking to incorporate Sudoku puzzles into their software, or researchers investigating Sudoku as a computational problem. Additionally, having a Sudoku generator available as a website ensures accessibility and user-friendliness for anyone with an internet connection.

Hayden Manton

Bibliography

[1]Bertram Felgenhauer, Frazer Jarvis. (2005). Enumerating possible Sudoku grids. Available: http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf. Last accessed 20th Jul 2022.

[2]Sudoku Lovers. (2021). What Makes Sudoku Easy, Medium, Or Hard? Here's The Science Behind Sudoku Levels. Available: https://www.sudokulovers.com/what-makes-Sudoku-easy-medium-or-hard. Last accessed 20th Jul 2022.

[3]Green, D. (2006). Conceptis Sudoku difficulty levels explained. Available: https://www.conceptispuzzles.com/index.aspx?uri=info/article/2#:~:text=But%20before%20I%20start%20to%20explain%20I%E2%80%99d%20like,clues%20and%20easy%20puzzles%20with%20very%20few%20clues.. Last accessed 20th Jul 2022.

[4]Learn Sudoku. (2008). A note on Pencil Marks. Available: https://www.learn-sudoku.com/pencil-marks.html#:~:text=Although%20the%20act%20of%20making%20pencil%20marks%20itself,for%20all%20the%20cells%20that%20are%20still%20unsolved.. Last accessed 20th Jul 2022.

[5]How to Solve Sudoku with Pencil Marks. (2022). How to Solve Sudoku with Pencil Marks. Available: https://masteringsudoku.com/how-to-solve-sudoku-with-pencil-marks/. Last accessed 20th Jul 2022.

[6]Phistomefel Ring Explained & How To Solve Sudokus With It. (2022). Phistomefel Ring Explained & How To Solve Sudokus With It. Available: https://masteringsudoku.com/phistomefel-ring/ Last accessed 20th Jul 2022.

[7]The New Sudoku Players' Forum. (2020). Phistomefel's Theorem. Available: http://forum.enjoysudoku.com/phistomefel-s-theorem-t38410.html. Last accessed 20th Jul 2022.

[8]Mastering Sudoku. (2022). Sudoku X Wings – What They Are & How They Help Solve Sudokus. Available: https://masteringsudoku.com/x-wings/. Last accessed 20th Jul 2022.

[9]Mastering Sudoku. (2022). Sudoku Y Wings Explained – How To Solve Sudokus with Y Wings. Available: https://masteringsudoku.com/y-wings/#:~:text=Y%20Wings%20are%20a%20fairly%20advanced%20Sudoku%20solving,why%20it%E2%80%99s%20known%20as%20a%20candidate%20eliminator%20strategy.. Last accessed 20th Jul 2022.

[10]Stuart, A. (2014). Y-Wing Strategy. Available: https://www.sudokuwiki.org/Y_Wing_Strategy. Last accessed 20th Jul 2022.

[11]Ronan, P. (2012). The world's most difficult sudoku puzzle. Available: https://japanesetranslator.co.uk/2012/07/the-worlds-most-difficult-sudoku-puzzle/. Last accessed 20th Jul 2022.

Project Log

Date	Task	Time Taken	Expected Outcome	Actual Outcome	Successful? (Y/N)	Notes
03/07/23	Creating the Report Document and Designing the Style	1.5 hours	Design a professional-looking, simplistic report document with appropriate colour schemes and consistent layouts for easy readability.	The report document is successfully designed with a professional and clean look, ensuring easy navigation through titles and hierarchy.	Y	N/A
03/07/23	Font Selection	0.5 hours	Choose suitable fonts for the report to enhance readability and aesthetics.	Appropriate fonts are selected, enhancing the overall appearance of the report.	Y	N/A
04/07/23	Designing the Cover Page	1 hour	Create an eye-catching cover page with a captivating image.	The initial cover page design was unsuccessful due to the complexity of the image provided. A new cover page design is needed to replace the complex image.	N	The initial image was deemed too complex and distracting, making it unsuitable for the cover. A simpler and more visually appealing design is required.
04/07/23	Redesigning the Cover Page	1 hour	Redesign the cover page with a more suitable and visually pleasing image.	The cover page is successfully redesigned, incorporating a simpler and more visually appealing image.	Y	N/A
04/07/23	Planning the Report Structure	0.25 hours	Develop a rough order for the report's content to ensure logical flow.	A rough order for the report's content is determined, providing a clear structure for the project.	Y	N/A
05/07/23	Project Overview and Approach	0.5 hours	Outline a mental approach to the project, considering project goals and directions.	A mental approach is established to guide the project, addressing its purpose and intended outcomes.	Y	N/A
05/07/23	Writing the Introduction	0.5 hours	Craft an engaging introduction that captures the reader's interest and sets the stage for the report.	The initial introduction was not engaging, focusing too much on uncertainties. A second attempt is needed to	N	The first introduction was found to be lacking engagement, with excessive focus on uncertainties.

				create an engaging introduction.		A revised introduction is required to create a more captivating opening.
06/07/23	Revised Introduction	0.5 hours	Rewrite the introduction to be engaging and enticing to the reader.	The introduction is successfully rewritten, now captivating and inviting to the reader.	Y	N/A
06/07/23	Aims and Objectives	0.25 hours	Define the aims and objectives for the project, building upon the introduction.	Aims and objectives are successfully outlined, aligning with the revised introduction.	Y	N/A
07/07/23	Initial Research Planning	0.5 hours	Develop a preliminary plan for conducting research, including research sources and methodologies.	A preliminary research plan is successfully established, guiding the initial stages of research.	Y	N/A
08/07/23	Researching What is a Sudoku?	2.125 hours	Conduct research on the fundamentals of Sudoku puzzles.	Extensive research on the basics of Sudoku is completed.	Y	N/A
09/07/23	Creating an Accompanying Image (Sudoku Basics)	1.5 hours	Design an image to complement the research on the fundamental principles of Sudoku puzzles.	A suitable image is created to accompany the explanation of Sudoku basics.	Y	N/A
09/07/23	Researching What Makes A Sudoku Easy / Difficult?	2.5 hours	Research the factors that determine the difficulty level of Sudoku puzzles.	Thorough research on the elements influencing Sudoku puzzle difficulty is completed.	Y	N/A
09/07/23	Creating an Accompanying Image	1.25 hours	Design an image to illustrate the factors contributing to Sudoku puzzle difficulty.	A suitable image is created to complement the research on Sudoku puzzle difficulty.	Y	N/A
10/07/23	Calculating Probability for Valid Diagonal Sudoku Grid	1.5 hours	Calculate the probability of a randomly generated grid being valid for a diagonal Sudoku puzzle.	The probability calculation for valid diagonal Sudoku grids is successfully completed.	Y	N/A
10/07/23	Create Image For Probability Calculation	0.5 hours	Design an image to visually represent the probability calculation for diagonal Sudoku grids.	A suitable image is created to illustrate the probability calculation process.	Y	N/A
10/07/23	Calculating Number of Possible Diagonal Sudoku Grids	0.5 hours	Calculate the total number of possible diagonal Sudoku grids.	The calculation for the number of possible diagonal Sudoku grids is successfully completed.	Y	N/A

10/07/23	Creating an Image for Grid Count Calculation	0.25 hours	Design an image to visually represent the calculation of the total number of possible diagonal Sudoku grids.	A suitable image is created to illustrate the grid count calculation.	Y	N/A
10/07/23	Research Impact of Starting Numbers	2 hours	Investigate the relationship between the quantity of starting numbers in Sudoku puzzles and their impact.	Comprehensive research is conducted, exploring how the quantity of starting numbers affects Sudoku puzzles.	Y	N/A
11/07/23	Conducting Own Investigation on Quantity of Starting Numbers vs Time Taken To Complete	3.5 hours	Conduct an investigation into the relationship between the quantity of starting numbers and the time required to complete Sudoku puzzles.	The investigation is carried out, and relevant images, tables, and graphs are created to support the findings.	Y	N/A
11/07/23	Creating Table for Investigation Results	0.5 hours	Develop a table to visually display the results from the investigation.	A table is successfully created to present the investigation results.	Y	N/A
11/07/23	Creating Graph for Investigation Results	1.25 hours	Generate a graph to visually represent the relationship between quantity and time in Sudoku puzzles.	A graph is successfully created to visualize the investigation results.	Y	N/A
11/07/23	Analysing Investigation Results	1 hour	Analyse the results of the investigation and draw logical conclusions based on the data.	The investigation results are analysed, and conclusions are drawn in a logical manner.	Y	N/A
11/07/23	Researching Pencil Markings	1.75 hours	Conduct research on the use and functionality of pencil markings in Sudoku puzzles.	Research on pencil markings is successfully completed.	Y	N/A
11/07/23	Creating Scenario for Pencil Markings	1.5 hours	Create a scenario demonstrating the functionality of pencil markings and design images to illustrate this scenario.	A scenario showcasing the use of pencil markings is created, accompanied by explanatory images.	Y	N/A
12/07/23	Researching Advanced Techniques for Solving A Sudoku Puzzle.	2.25 hours	Research various advanced techniques for solving Sudoku puzzles.	Research on advanced solving techniques is successful, with three prominent	Y	Three prominent advanced solving techniques were identified

				techniques identified.		during the research.
12/07/23	Researching the Phistomefel Ring	2 hours	Successfully Research the Phistomefel Ring.	Successfully Researched the Phistomefel Ring.	Y	N/A
13/07/23	Understanding and Describing Pure Logic of the Phistomefel Ring	3.75 hours	Understand and describe the pure logic behind the Phistomefel Ring and create an image to complement the description.	The pure logic of the Phistomefel Ring is comprehended and described, with an accompanying illustrative image.	Y	N/A
13/07/23	Researching X-Wings	2.5 hours	Research the X-Wing technique for solving Sudoku puzzles.	Research on X-Wings is completed, understanding their application for puzzle solving.	Y	N/A
13/07/23	Creating Images for X-Wing Technique	1.75 hours	Generate a series of images illustrating how the X-Wing technique is used to progress in Sudoku puzzles.	Consecutive images are created, showing the application of the X-Wing technique in puzzle-solving scenarios.	Y	N/A
14/07/23	Researching Y-Wings	1.75 hours	Research the Y-Wing technique for solving Sudoku puzzles.	Research on Y-Wings is successfully completed, understanding their role in puzzle solving.	Y	N/A
14/07/23	Creating Images for Y-Wing Technique	2 hours	Create consecutive images illustrating how the Y-Wing technique is applied in Sudoku puzzle-solving situations.	Images are generated, highlighting the application of the Y-Wing technique in various puzzle scenarios.	Y	N/A
14/07/23	Researching Most Difficult Sudoku Puzzles and Solutions	1.75 hours	Research the concept of the most difficult Sudoku puzzle and document findings.	Extensive research is conducted, revealing varying opinions and disagreements on the most difficult Sudoku puzzles.	Y	There is no single consensus on the most difficult Sudoku puzzle, as opinions vary widely.
14/07/23	Creating Images To Accompany This	0.75 hours	Generate images showcasing difficult Sudoku puzzles alongside their solutions.	Images are created, displaying challenging puzzles paired with their respective solutions.	Y	N/A
14/07/23	Researching Different Sizes of Sudoku Puzzles	2 hours	Research the concept of Sudoku puzzles with varying grid sizes.	Research on different sizes of Sudoku puzzles is successfully completed.	Y	N/A

15/07/23	Conducting Own Investigation on Puzzle Size vs Difficulty	2.75 hours	Conduct an investigation into the difficulty of Sudoku puzzles as their sizes change, and display the results with accompanying images.	The investigation is conducted, and findings are presented with supporting images.	Y	N/A
15/07/23	Creating Basic Sudoku Scenarios	1.75 hours	Develop three basic Sudoku scenarios and visually explain them using created images.	Three basic Sudoku scenarios are created and visually explained through accompanying images.	Y	N/A
15/07/23	Creating Intermediate Sudoku Scenarios	2.75 hour	Develop two intermediate Sudoku scenarios and visually explain them using created images.	Two intermediate Sudoku scenarios are created and visually explained with accompanying images.	Y	N/A
16/07/23	Creating Advances Sudoku Scenarios	3.5 hours	Create an advanced Sudoku scenario and visually explain it using created images.	An advanced Sudoku scenario is developed and visually explained through accompanying images.	Y	N/A
16/07/23	Planning structure for Next Project Phase	1 hour	Outline the expected structure for the next phase of the project, considering the accumulated research and findings.	A structured plan is established for the upcoming project phase, taking into account existing research.	Y	N/A
17/07/23	Developing a Method for Sudoku Puzzle Creation	4.5 hours	Break down the process of creating a Sudoku puzzle into three parts.	A method for creating Sudoku puzzles is developed, with distinct phases outlined.	Y	N/A
18/07/23	Discussing Challenges in Filling a Grid	2 hours	Discuss the challenges involved in filling a Sudoku grid and why this process can be difficult.	Challenges in filling a Sudoku grid are comprehensively discussed, highlighting the difficulties involved.	Y	N/A
19/07/23	Discussing Challenges in Emptying a Grid	2.25 hours	Discuss the challenges and complexities associated with emptying a Sudoku grid.	Challenges in emptying a Sudoku grid are thoroughly discussed, highlighting the difficulties involved.	Y	N/A
19/07/23	Discussing Challenges in Solving a Grid	1.75 hours	Discuss the complexities and difficulties involved in solving a Sudoku grid.	Challenges in solving a Sudoku grid are comprehensively discussed, emphasizing the complexities.	Y	N/A

19/07/23 20/07/23	Forming Logical Methods for Filling a Grid	3.5 hours	Develop multiple logical approaches (at least six) for filling a Sudoku grid.	Six distinct approaches for filling a Sudoku grid are identified and documented.	Y	N/A
21/07/23	Explaining the 'Brute Force' Method	0.75 hours	Write about the 'Brute Force' method and explain how it works.	The 'Brute Force' method is described, providing insight into its functioning.	Y	N/A
21/07/23	Describing the Advantages and Disadvantages of the 'Brute Force' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Brute Force' method.	The pros and cons of the 'Brute Force' method are elaborated upon, providing a balanced view.	Y	N/A
21/07/23	Writing Pseudocode for the 'Brute Force' Method	1 hour	Create pseudocode to represent the implementation of the 'Brute Force' method.	Pseudocode for the 'Brute Force' method is successfully written, offering a step-by-step guide to its execution.	Y	N/A
21/07/23	Explaining the 'Using Grids and Regions' Method	1 hour	Write about the 'Using Grids and Regions' method and explain how it works.	The 'Using Grids and Regions' method is described, offering insights into its functionality.	Y	N/A
21/07/23	Describing the Advantages and Disadvantages of the 'Using Grids and Regions' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Using Grids and Regions' method.	The strengths and weaknesses of the 'Using Grids and Regions' method are outlined, providing a comprehensive evaluation.	Y	N/A
21/07/23	Creating Images for 'Using Grids and Regions' Algorithm	1.5 hours	Generate images illustrating how the algorithm of the 'Using Grids and Regions' method operates.	Images representing the functionality of the 'Using Grids and Regions' method are successfully created.	Y	N/A
22/07/23	Explaining the 'Backtracking With Stacks' Method	1.5 hours	Write about the 'Backtracking with Stacks' method and explain how it works.	The 'Backtracking with Stacks' method is described, providing insights into its operation.	Y	N/A
22/07/23	Describing the Advantages and Disadvantages of the 'Backtracking With Stacks' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Backtracking with Stacks' method.	The benefits and limitations of the 'Backtracking with Stacks' method are detailed, offering a balanced view.	Y	N/A
22/07/23	Writing Pseudocode for the	0.75 hours	Create pseudocode to represent the	Pseudocode for the 'Backtracking with Stacks'	Y	N/A

	'Backtracking With Stacks' Method		implementation of the 'Backtracking with Stacks' method.	method is successfully written, providing a clear guide for execution.		
22/07/23	Discussing how the 'Backtracking With Stacks' Method can be improved	0.25 hours	Discuss potential improvements to enhance the 'Backtracking with Stacks' method.	Ideas for improving the 'Backtracking with Stacks' method are explored, considering its efficiency and effectiveness.	Y	N/A
22/07/23	Explaining the 'Randomising Patterns' Method	0.25 hours	Write about the 'Randomizing Patterns' method and explain how it works.	The 'Randomizing Patterns' method is described, offering insights into its functioning.	Y	N/A
23/07/23	Describing the Advantages and Disadvantages of the 'Randomising Patterns' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Randomizing Patterns' method.	The strengths and weaknesses of the 'Randomizing Patterns' method are outlined, providing a comprehensive evaluation.	Y	N/A
23/07/23	Creating Images for 'Randomising Patterns' Algorithm	2.5 hours	Generate images illustrating how the algorithm of the 'Randomizing Patterns' method operates.	Images representing the functionality of the 'Randomizing Patterns' method are successfully created.	Y	N/A
23/07/23	Discussing Enhancement of Randomness in 'Randomising Patterns' Method	0.5 hours	Discuss ways to increase the randomness of outcomes generated by the 'Randomizing Patterns' method.	Strategies for enhancing the randomness of outcomes in the 'Randomizing Patterns' method are explored.	Y	N/A
23/07/23	Explaining the 'Pencil Mark Possibilities' Method	1 hour	Write about the 'Pencil Mark Possibilities' method and explain how it works.	The 'Pencil Mark Possibilities' method is described, offering insights into its operation.	Y	N/A
24/07/23	Describing the Advantages and Disadvantages of the 'Pencil Mark Possibilities' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Pencil Mark Possibilities' method.	The benefits and limitations of the 'Pencil Mark Possibilities' method are detailed, providing a comprehensive evaluation.	Y	N/A
24/07/23	Writing Pseudocode for the 'Pencil Mark Possibilities' Method	0.5 hours	Create pseudocode to represent the implementation of the 'Pencil Mark	Pseudocode for the 'Pencil Mark Possibilities' method is successfully written, offering	Y	N/A

			Possibilities' method.	a clear guide for execution.		
24/07/23	Discussing Improvement for 'Pencil Mark Possibilities' Method	0.25 hours	Discuss potential improvements to enhance the 'Pencil Mark Possibilities' method.	Ideas for improving the 'Pencil Mark Possibilities' method are explored, considering its efficiency and effectiveness.	Y	N/A
24/07/23	Explaining the 'Consecutive Pencil Markings' Method	0.5 hours	Write about the 'Consecutive Pencil Markings' method and explain how it works.	The 'Consecutive Pencil Markings' method is described, offering insights into its operation.	Y	N/A
24/07/23	Describing the Advantages and Disadvantages of the 'Consecutive Pencil Markings' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Consecutive Pencil Markings' method.	The strengths and weaknesses of the 'Consecutive Pencil Markings' method are outlined, providing a comprehensive evaluation.	Y	N/A
24/07/23	Creating Images for 'Consecutive Pencil Markings' Algorithm	1 hour	Generate images illustrating how the algorithm of the 'Consecutive Pencil Markings' method operates.	Images representing the functionality of the 'Consecutive Pencil Markings' method are successfully created.	Y	N/A
25/07/23	Comparing Filling the Grid Methods and choosing the most logical one	0.75 hours	Compare all identified methods using their advantages and disadvantages and select the most suitable one.	A thorough comparison is conducted, and the most logical method for filling the grid is chosen based on the evaluation.	Y	N/A
25/07/23	Discuss the Chosen Approach for Filling The Grid	2.25 hours	Integrate the chosen approach for filling the grid into the report and provide a rationale for its selection.	The selected method for filling the grid is added to the report, along with a detailed explanation of why it was chosen.	Y	N/A
25/07/23	Write Pseudocode for the Chosen Approach for Filling The Grid	1.5 hours	Create pseudocode to represent the implementation of the chosen filling approach for the Sudoku grid.	Pseudocode for the selected filling approach is successfully written, offering a clear guide for execution.	Y	N/A
26/07/23	Creating a Structure Chart for Filling The Grid	0.75 hours	Develop a structure chart illustrating the organization of the program for filling the Sudoku grid.	A structure chart is successfully created, providing a visual representation of the program's organization.	Y	N/A

26/07/23 27/07/23 28/07/23 29/07/23 30/07/23 31/07/23	Creating the 'Proof of Concept' Program to Fill the Sudoku Grid	26.5 hours	Successfully create a fully functioning program.	The program can be deemed a success due to the fact that it has the ability to fill a valid sudoku grid.	Y	N/A
31/07/23	Creating the User Interface for Filling The Grid	1.5 hours	Develop a user interface for the program to facilitate filling the Sudoku grid.	The user interface for filling the grid is successfully designed and implemented.	Y	N/A
31/07/23	Linking the Code and the User Interface	2.25 hours	Successfully modifying the code to produce a suitable output.	The code and UI have been successfully linked.	Y	N/A
31/07/23	Creating Images for the User Interface	1 hour	Generate images showcasing the user interface, highlighting important areas for user interaction.	Images of the user interface are created, emphasizing key interaction elements.	Y	N/A
31/07/23	Running the Program and Providing Evidence	1 hour	Execute the program and capture evidence in the form of images to demonstrate its functionality.	The program is successfully run, and images are provided as evidence of its proper functioning.	Y	N/A
01/08/23	Discussing Evidence of Program Functionality	1 hour	Discuss the captured evidence and explain how it serves as proof of a working program.	The evidence of the program's functionality is thoroughly discussed, highlighting its significance as a validation of the program's effectiveness.	Y	N/A
01/08/23	Generating Logical Methods for Emptying a Grid	2.5 hours	Develop multiple logical approaches (at least six) for emptying a Sudoku grid.	Six distinct approaches for emptying a Sudoku grid are identified and documented.	Y	N/A
01/08/23	Explaining the 'Removing 1 Number from Each Group' Method	0.5 hours	Write about the 'Removing 1 Number from Each Group' method and explain how it works.	The 'Removing 1 Number from Each Group' method is described, providing insights into its operation.	Y	N/A
01/08/23	Describing Advantages and Disadvantages of 'Removing 1 Number from Each Group' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Removing 1 Number from Each Group' method.	The pros and cons of the 'Removing 1 Number from Each Group' method are elaborated upon, offering a balanced view.	Y	N/A
02/08/23	Creating Images for 'Removing 1 Number from	1.25 hours	Generate images illustrating how the algorithm of the 'Removing 1	Images representing the functionality of the 'Removing 1	Y	N/A

	Each Group' Algorithm		Number from Each Group' method operates.	Number from Each Group' method are successfully created.		
03/08/23	Creating Images for an Alternative Order of Operations	0.5 hours	Generate images illustrating an alternative order of operations for the 'Removing 1 Number from Each Group' method.	Images are created to depict an alternative sequence of operations for the method.	Y	N/A
03/08/23	Explaining the 'Removing Random Numbers' Method	0.25 hours	Write about the 'Removing Random Numbers' method and explain how it works.	The 'Removing Random Numbers' method is described, offering insights into its functioning.	Y	N/A
03/08/23	Describing Advantages and Disadvantages of 'Removing Random Numbers' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Removing Random Numbers' method.	The benefits and limitations of the 'Removing Random Numbers' method are outlined, providing a comprehensive evaluation.	Y	N/A
03/08/23	Creating Images for the 'Removing Random Numbers' Method	0.5 hours	Generate images illustrating how the algorithm of the 'Removing Random Numbers' method operates.	Images representing the functionality of the 'Removing Random Numbers' method are successfully created.	Y	N/A
03/08/23	Writing Pseudocode for 'Removing Random Numbers' Method	0.5 hours	Create pseudocode to represent the implementation of the 'Removing Random Numbers' method.	Pseudocode for the 'Removing Random Numbers' method is successfully written, offering a step-by-step guide for execution.	Y	N/A
03/08/23	Explaining the 'Linearly Removing Numbers' Method	0.25 hours	Write about the 'Linearly Removing Numbers' method and explain how it works.	The 'Linearly Removing Numbers' method is described, offering insights into its operation.	Y	N/A
03/08/23	Describing Advantages and Disadvantages of 'Linearly Removing Numbers' Method	0.125 hours	Discuss the advantages and disadvantages of the 'Linearly Removing Numbers' method.	The strengths and weaknesses of the 'Linearly Removing Numbers' method are outlined, providing a comprehensive evaluation.	Y	N/A
05/08/23	Creating Image for	1.5 hours	Generate images illustrating how	Images representing the	Y	N/A

	'Linearly Removing Numbers' Algorithm		the algorithm of the 'Linearly Removing Numbers' method operates.	functionality of the 'Linearly Removing Numbers' method are successfully created.		
05/08/23	Writing Pseudocode for 'Linearly Removing Numbers' Method	0.5 hours	Create pseudocode to represent the implementation of the 'Linearly Removing Numbers' method.	Pseudocode for the 'Linearly Removing Numbers' method is successfully written, offering a clear guide for execution.	Y	N/A
05/08/23	Explaining the 'Removing Translated Groups' Method	0.375 hours	Write about the 'Removing Translated Groups' method and explain how it works.	The 'Removing Translated Groups' method is described, offering insights into its functioning.	Y	N/A
08/08/23	Describing Advantages and Disadvantages of 'Removing Translated Groups' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Removing Translated Groups' method.	The benefits and limitations of the 'Removing Translated Groups' method are outlined, providing a comprehensive evaluation.	Y	N/A
08/08/23	Creating Images for 'Removing Translated Groups' Algorithm	1.75 hours	Generate images illustrating how the algorithm of the 'Removing Translated Groups' method operates in two slightly different versions.	Images representing the functionality of the 'Removing Translated Groups' method are successfully created, encompassing two variations.	Y	N/A
08/08/23	Discussing the Two Versions for 'Removing Translated Groups'	0.75 hours	Discuss the differences between the two versions of the 'Removing Translated Groups' method.	A comparison between the two versions of the 'Removing Translated Groups' method is provided, highlighting their distinctions.	Y	N/A
09/08/23	Explaining the 'Removing Invalid Random Numbers' Method	0.25 hours	Write about the 'Removing Invalid Random Numbers' method and explain how it works.	The 'Removing Invalid Random Numbers' method is described, offering insights into its operation.	Y	N/A
09/08/23	Describing Advantages and Disadvantages of 'Removing Invalid Random Numbers' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Removing Invalid Random Numbers' method.	The strengths and weaknesses of the 'Removing Invalid Random Numbers' method are outlined, providing a comprehensive evaluation.	Y	N/A

09/08/23	Writing Pseudocode for 'Removing Invalid Random Numbers' Method	0.75 hours	Create pseudocode to represent the implementation of the 'Removing Invalid Random Numbers' method.	Pseudocode for the 'Removing Invalid Random Numbers' method is successfully written, providing a clear guide for execution.	Y	N/A
09/08/23	Explaining the Pseudocode for 'Removing Invalid Random Numbers' Method	0.5 hours	Provide an explanation for the pseudocode of the 'Removing Invalid Random Numbers' method.	The pseudocode for the 'Removing Invalid Random Numbers' method is explained, offering clarity on its implementation.	Y	N/A
09/08/23	Creating Images for 'Removing Invalid Random Numbers' Algorithm	0.75 hours	Generate images illustrating how the algorithm of the 'Removing Invalid Random Numbers' method operates.	Images representing the functionality of the 'Removing Invalid Random Numbers' method are successfully created.	Y	N/A
09/08/23	Explaining the 'Method 1 & 5 Together' Method	0.5 hours	Write about the 'Method 1 & 5 Together' method and explain how it works.	The 'Method 1 & 5 Together' method is described, providing insights into its operation.	Y	N/A
10/08/23	Describing Advantages and Disadvantages of 'Method 1 & 5 Together' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Method 1 & 5 Together' method.	The benefits and limitations of the 'Method 1 & 5 Together' method are outlined, providing a comprehensive evaluation.	Y	N/A
10/08/23	Creating Images for 'Method 1 & 5 Together' Algorithm	0.75 hours	Generate images illustrating the two halves of the algorithm for the 'Method 1 & 5 Together' method.	Images are created to depict the functioning of the 'Method 1 & 5 Together' method, representing both halves.	Y	N/A
10/08/23	Comparing Emptying Methods and Choosing the Most Logical One	1 hour	Compare all identified methods using their advantages and disadvantages and select the most suitable one.	A thorough comparison is conducted, and the most logical method for emptying the grid is chosen based on the evaluation.	Y	N/A
10/08/23	Adding Chosen Emptying Approach to the Report	1.25 hours	Integrate the selected method for emptying the grid into the report and provide a rationale for its selection.	The chosen method for emptying the grid is added to the report, along with a detailed explanation of why it was selected.	Y	N/A

10/08/23	Writing Pseudocode for Chosen Emptying Approach	0.75 hours	Create pseudocode to represent the implementation of the chosen emptying approach for the Sudoku grid.	Pseudocode for the selected emptying approach is successfully written, offering a clear guide for execution.	Y	N/A
10/08/23	Creating a Structure Chart for Emptying the Grid	0.25 hours	Develop a structure chart illustrating the organization of the program for emptying the Sudoku grid.	A structure chart is successfully created, providing a visual representation of the program's organization for grid emptying.	Y	N/A
11/08/23 12/08/23 13/08/23 14/08/23 15/08/23 16/08/23 17/08/23	Creating the 'Proof of Concept' Program to Empty the Sudoku Grid	26.5 hours	Successfully create a fully functioning program.	The program can be deemed a success due to the fact that it has the ability to empty any valid sudoku grid up until the point where it is no longer solvable.	Y	N/A
19/08/23	Creating the User Interface for Emptying The Grid	0.75 hours	Generate additional images showcasing the user interface, highlighting key elements and features.	Supplementary images of the user interface are successfully created, providing a comprehensive view of its design.	Y	I used the same UI as the one used for Filling The Grid
19/08/23	Running the Program and Providing Evidence	0.5 hours	Execute the program again and capture fresh evidence in the form of images to demonstrate its functionality.	The program is successfully re-run, and updated images are provided as evidence of its continued proper functioning.	Y	N/A
20/08/23	Discussing Evidence of Program Functionality	0.25 hours	Revisit and discuss the newly captured evidence, emphasizing how it continues to serve as proof of a working program.	The updated evidence of the program's functionality is thoroughly discussed, reaffirming its significance in validating the program's effectiveness.	Y	N/A
20/08/23	Generating Logical Methods for Solving a Grid	2.5 hours	Develop multiple logical approaches (at least three) for solving a Sudoku grid.	Three distinct approaches for solving a Sudoku grid are identified and documented.	Y	N/A
20/08/23	Explaining the 'Random Input Attempts' Method	1 hour	Write about the 'Random Input Attempts' method and explain how it works.	The 'Random Input Attempts' method is described, providing insights into its operation.	Y	N/A
20/08/23	Describing Advantages and	0.25 hours	Discuss the advantages and disadvantages of	The pros and cons of the 'Random Input	Y	N/A

	Disadvantages of 'Random Input Attempts' Method		the 'Random Input Attempts' method.	Attempts' method are elaborated upon, offering a balanced view.		
21/08/23	Creating Images for 'Random Input Attempts' Algorithm	1 hour	Generate images illustrating how the algorithm of the 'Random Input Attempts' method operates.	Images representing the functionality of the 'Random Input Attempts' method are successfully created.	Y	N/A
21/08/23	Writing about the 'Undo if no Inputs' Method	1 hour	Write about the 'Undo if no Inputs' method and explain how it works.	The 'Undo if no Inputs' method is described, offering insights into its operation.	Y	N/A
21/08/23	Describing Advantages and Disadvantages of 'Undo if no Inputs' Method	0.25 hours	Discuss the advantages and disadvantages of the 'Undo if no Inputs' method.	The benefits and limitations of the 'Undo if no Inputs' method are outlined, providing a comprehensive evaluation.	Y	N/A
21/08/23	Creating Images for 'Undo if no Inputs' Algorithm	0.75 hours	Generate images illustrating how the algorithm of the 'Undo if no Inputs' method operates.	Images representing the functionality of the 'Undo if no Inputs' method are successfully created.	Y	N/A
21/08/23	Writing about the 'Attempting Different Approaches' Method	1 hour	Write about the 'Attempting Different Approaches' method and explain how it works.	The 'Attempting Different Approaches' method is described, providing insights into its operation.	Y	N/A
21/08/23	Describing Advantages and Disadvantages of 'Attempting Different Approaches' Method	0.5 hours	Discuss the advantages and disadvantages of the 'Attempting Different Approaches' method.	The strengths and weaknesses of the 'Attempting Different Approaches' method are outlined, offering a comprehensive evaluation.	Y	N/A
21/08/23	Creating Images for 'Attempting Different Approaches' Algorithm	0.5 hours	Generate images illustrating how the algorithm of the 'Attempting Different Approaches' method operates.	Images representing the functionality of the 'Attempting Different Approaches' method are successfully created.	Y	N/A
21/08/23	Comparing Solving Methods and Choosing the Most Logical One	0.5 hours	Compare all identified methods using their advantages and disadvantages and select the most suitable one.	A thorough comparison is conducted, and the most logical method for solving the grid is chosen based on the evaluation.	Y	N/A

31/08/23	Adding Chosen Solving Approach to the Report	1 hour	Integrate the selected method for solving the grid into the report and provide a rationale for its selection.	The chosen method for solving the grid is added to the report, along with a detailed explanation of why it was selected.	Y	N/A
31/08/23 01/09/23 02/09/23 03/09/23 04/09/23 05/09/23 06/09/23 07/09/23 08/09/23	Creating the 'Proof of Concept' Program to Solve the Sudoku Grid	26.5 hours	Successfully create a fully functioning program.	The program can be deemed a success due to the fact that it has the ability to solve any valid sudoku grid.	Y	N/A
09/09/23	Creating the User Interface for Solving The Grid	0.75 hours	Generate additional images showcasing the user interface, highlighting key elements and features.	Supplementary images of the user interface are successfully created, providing a comprehensive view of its design.	Y	I used the same UI as the one used for Filling and Emptying The Grid
09/09/23	Running the Program and Providing Evidence	0.5 hours	Execute the program again and capture fresh evidence in the form of images to demonstrate its functionality.	The program is successfully re-run, and updated images are provided as evidence of its continued proper functioning.	Y	N/A
09/09/23	Discussing Evidence of Program Functionality	0.25 hours	Revisit and discuss the newly captured evidence, emphasizing how it continues to serve as proof of a working program.	The updated evidence of the program's functionality is thoroughly discussed, reaffirming its significance in validating the program's effectiveness.	Y	N/A
09/09/23	Coming up with Suitable Open-Ended Questions for the Survey	1 hour	Develop open-ended survey questions that encourage respondents to provide detailed, written responses.	Three open-ended questions are created to gather detailed written feedback from survey participants.	Y	N/A
10/09/23	Coming up with Suitable Rating Out of 10 Questions for the Survey	1 hour	Create rating questions on a scale of 1 to 10 to assess various aspects of the project.	Three rating questions on a scale of 1 to 10 are designed to gauge participants' perceptions of different project aspects.	Y	N/A
10/09/23	Coming up with Suitable Feedback Questions for the Survey	1 hour	Develop feedback questions to gather respondents'	Four feedback questions are formulated to solicit opinions and suggestions	Y	N/A

			opinions and suggestions regarding the project.	from survey participants about the project.		
10/09/23	Explaining Each of the 10 Questions for the Survey	2.5 hours	Provide explanations and context for all 10 survey questions, ensuring clarity for respondents.	Each of the 10 survey questions is explained in detail, providing context and ensuring respondent clarity.	Y	N/A
10/09/23	Discussing the Benefits of the Survey and the Expected Outcomes	1.25 hours	Discuss the benefits of conducting the survey and outline the expected outcomes.	The benefits of the survey are discussed, and anticipated outcomes are outlined, emphasizing the value of survey results.	Y	N/A
11/09/23	Discussing the Methodology of the Survey	0.25 hours	Describe the methodology of the survey, including how it will be administered and data collected.	The survey methodology is described, detailing how it will be administered, and data collection methods are outlined.	Y	N/A
11/09/23 12/09/23 13/09/23	Conduct the Survey	7 hours	Successfully conduct a survey.	The survey was successfully conducted	Y	N/A
14/09/23 15/09/23	Discussing the Results of Question 1 to Question 10	5.5 hours	Provide detailed discussions of the results obtained from each of the 10 survey questions.	Results of each survey question are discussed comprehensively, presenting findings and insights.	Y	N/A
15/09/23	Writing an Overview of the Results from the Survey	0.25 hours	Summarise the key findings and trends observed in the survey results.	An overview of the survey results is written, summarizing key findings and highlighting any notable trends.	Y	N/A
16/09/23	Writing a Plan for the Final Program	1.25 hours	Develop a plan outlining the steps and timeline for completing the final program based on survey feedback and research.	A detailed plan for the final program is created, specifying steps and timelines informed by survey feedback and research.	Y	N/A
17/09/23 18/09/23 19/09/23 20/09/23	Create a Flowchart for the Whole Program	13.5 hours	Design a comprehensive flowchart that illustrates the entire program's logic and functionality.	A detailed flowchart is created, providing a visual representation of the entire program's logic and functionality.	Y	N/A
21/09/23	Creating the Final Program in Visual Basic .NET	2.5 hours	Successfully create the final program in Visual Baic .NET	The final program meets all the requirements to be deemed	Y	The reason this final program had such a quick turnaround is

				successful as a valid sudoku grid can be produced visually by the program in very little time.		due to most of the components already being complete prior to this point.
21/09/23	Write an Overview of the Code for the Final Program in Visual Basic .NET	0.5 hours	Provide an overview of the code written for the final program in Visual Basic .NET, summarizing its key components and functionality.	A comprehensive overview of the code for the final program in Visual Basic .NET is written, highlighting its essential elements.	Y	N/A
21/09/23	Display the Visual Results of the Program in Visual Basic .NET in the Report	0.25 hours	Include visual representations of the program's results in Visual Basic .NET within the report, showcasing two valid Sudoku puzzles.	Visual results of the program in Visual Basic .NET, featuring two valid Sudoku puzzles, are successfully incorporated into the report.	Y	N/A
21/09/23	Discuss the Visual Results of the Program	0.25 hours	Provide an in-depth discussion of the visual results obtained from the program in Visual Basic .NET.	Visual results from the program in Visual Basic .NET are thoroughly discussed, offering insights into the generated Sudoku puzzles.	Y	N/A
22/09/23	Write Pseudocode for Creating the Program in Python	3.5 hours	Develop pseudocode outlining the logic and structure of the program to be created in Python.	Pseudocode for the Python program is successfully created, serving as a clear guide for its implementation.	Y	N/A
22/09/23	Writing an Overview of the Pseudocode Written for the Program in Python	1.25 hours	Provide an overview of the pseudocode written for the Python program, summarizing its key logic and structure.	A comprehensive overview of the pseudocode for the Python program is written, highlighting its essential components.	Y	N/A
22/09/23	Creating the Sudoku Puzzle Generator in Python	1.5 hours	Develop the Sudoku puzzle generator program in Python based on the provided pseudocode.	The Sudoku puzzle generator is successfully implemented in Python, following the provided pseudocode.	Y	N/A
23/09/23	Displaying the Visual Results of the Program in Python and Explaining the Output	1 hour	Include visual representations of the program's results in Python within the report and provide an explanation of the output.	Visual results of the program in Python, along with an explanation of the output, are effectively integrated into the report.	Y	N/A

23/09/23 24/09/23	Creating the Sudoku Puzzle Generator in C++	5.5 hours	Develop the Sudoku puzzle generator program in C++.	The Sudoku puzzle generator is successfully implemented in C++, providing another programming language option.	Y	N/A
25/09/23	Displaying the Visual Results of the Program in C++ and Explaining the Output	0.5 hours	Include visual representations of the program's results in C++ within the report and provide an explanation of the output.	Visual results of the program in C++, along with an explanation of the output, are effectively integrated into the report.	Y	N/A
25/09/23	Write Pseudocode for Creating the Program in Java	1.25 hours	Develop pseudocode outlining the logic and structure of the program to be created in Java.	Pseudocode for the Java program is successfully created, serving as a clear guide for its implementation.	Y	N/A
25/09/23	Writing an Overview of the Pseudocode Written for the Program in Java	0.75 hours	Provide an overview of the pseudocode written for the Java program, summarizing its key logic and structure.	A comprehensive overview of the pseudocode for the Java program is written, highlighting its essential components.	Y	N/A
26/09/23	Creating the Sudoku Puzzle Generator in Java	2 hours	Develop the Sudoku puzzle generator program in Java based on the provided pseudocode.	The Sudoku puzzle generator is successfully implemented in Java, offering another programming language option.	Y	N/A
26/09/23	Displaying the Visual Results of the Program in Java and Explaining the Output	1.25 hours	Include visual representations of the program's results in Java within the report and provide an explanation of the output.	Visual results of the program in Java, along with an explanation of the output, are effectively integrated into the report.	Y	N/A
27/09/23 28/09/23	Creating the Sudoku Puzzle Generator as a Website using HTML, CSS, and JavaScript	6.75 hours	Develop a web-based Sudoku puzzle generator using HTML, CSS, and JavaScript.	The Sudoku puzzle generator is successfully created as a website, demonstrating proficiency in web development technologies.	Y	N/A
29/09/23	Writing the Conclusion for the Report	2.75 hours	Summarise the key findings, achievements, and insights from the project and provide a conclusion for the report.	A comprehensive conclusion for the report is written, summarizing the project's outcomes and contributions.	Y	N/A
29/09/23	Writing the Evaluation for the Report	2 hours	Evaluate the project's successes, challenges, and areas for	A thoughtful evaluation of the project is presented, addressing	Y	N/A

			improvement, providing an honest assessment.	successes, challenges, and opportunities for improvement.		
30/09/23	Writing a Personal Reflection for the Report	1.5 hours	Share a personal reflection on the project, including lessons learned, personal growth, and insights gained.	A personal reflection on the project is included, highlighting personal growth and insights gained throughout the project journey.	Y	N/A
30/09/23	Create a Contents Page	1.25 hours	Create a suitable and accurate contents table that is accurate to the report with supporting headings and page numbers.	The contents table is easy to follow and fully correct.	Y	N/A
30/09/23	Writing an Abstract for the Report	1 hour	Create an abstract summarizing the key aspects of the report, providing a concise overview.	An abstract for the report is crafted, offering a brief summary of its key elements and findings.	Y	N/A
30/09/23	Coming up with a Suitable Title for the Report	0.5 hours	Propose a suitable and compelling title for the report that accurately reflects its content and purpose.	Fitting and descriptive title for the report is suggested, capturing its essence and purpose effectively.	Y	N/A

Total Time Taken To Complete The Project

1.698660714 Weeks 285.375 Hours 17122.5 Minutes 1,027,350 Seconds