Sudoku Solver

Math Research Project

By Maansey Rishi

Abstract

A Sudoku puzzle is a logic-based number placement puzzle. The objective is to fill a 9 by 9 grid with the digits 1-9 such that each column, each row, and each of the nine 3 by 3 sub-grids that compose the grid contains each number only once. Each Sudoku puzzle has a unique solution. The smallest number of starting clues a Sudoku puzzle can contain is 17, and there are exactly 6, 670, 903, 752, 021, 072, 936, 960 possible solutions to Sudoku. The goal of my math research project was to make a computer program that efficiently solves these Sudoku puzzles. My computer program solves these puzzles by logically going through possible solutions and eliminating choices sequentially until it reaches a solution. This essay will explain the computer program I have made and the reasoning behind it.

Introduction

When I first started riding airplanes I noticed that several of the passengers were always engrossed in the back section of the newspaper. They were always staring at a grid-like figure with random numbers scattered all over it. I later found that the name of this puzzle was Sudoku but I still didn't understand why people spent so much time sitting in their seats looking so puzzled. When I found an old Sudoku puzzle book in an old drawer in my house, I decided to get to the bottom of this mystery puzzle. I decided to start with an easier puzzle, and before I knew it I was filling out numbers into the grid laid out in front of me. I picked the numbers I believed to make the most sense, while closely following the rules given to me. As I continued I realized most of my guesses contradicted my other guesses as I got further into the puzzle. After ten minutes I gave up, and my life continued as if nothing had ever happened.

A couple years later, I was assigned a math research project and I was told that I could pick any topic in math that interested me. It took me a while for me to think of something, but I finally recalled my nine-year-old self giving up on the Sudoku puzzle in my basement. I then realized that this would be an interesting direction for me to take for my math project. As I researched more about Sudoku I learned the rules and some very interesting fun facts:

The word Sudoku is short for "Su-fi wa dokushin ni kagiru" which means "the numbers must be single". The roots of the Sudoku puzzle come from Switzerland where Leonhard Euler created Latin squares which is very similar to Sudoku puzzles. However, the first real Sudoku was published in 1979 and was invented by Howard Garns, an American architect.

A Sudoku puzzle consists of 81 cells which are divided into nine columns, rows, and 3 by 3 mini grids. The objective of Sudoku is to fill the 9 by 9 grid with the digits 1-9 such that each column, each row, and each of the nine 3 by 3 sub-grids that compose the grid contains each number only once. Each Sudoku puzzle has a unique solution. Mathematicians have found that the smallest number of starting clues a puzzle can contain is 17, and there are exactly 6, 670, 903, 752, 021, 072, 936, 960(About 6.5 Sextillion) possible solutions to Sudoku.

After more research, I decided to pursue the problem of Sudoku puzzles for my math research project. I also determined that my new-found knowledge of computer science would be a great way to put my algorithm to the test.

The goal of my math research project was to make a computer program that efficiently solves Sudoku puzzles. My computer program solves these puzzles by eliminating answers, and checking each spot on the grid for unique solutions. My computer program is able to solve the majority of Sudoku puzzles.

This essay will explain the computer program I have made and the reasoning behind it. I have also made several flowcharts which explain and simplify my very complicated computer code. Each program includes a simple explanation which outlines the flowchart in simple English.

Rules of Sudoku

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

The objective of Sudoku is to fill a 9 by 9 grid with the digits 1-9 such that each column, each row, and each of the nine 3 by 3 sub-grids that compose the grid contains each number only once. Each Sudoku puzzle has a unique solution.

Concept Behind Algorithm used in the Computer Program

For these next couple of pages the concept behind the computer program I made will be explained.

Next Few Steps

1. The computer systematically runs through the steps previously shown for each of the empty indexes.
2. It fills in the indexes with only one possible choice.
3. The computer repeatedly runs through these steps until there are no more changes left to be made.
4. Some times, these steps alone are enough to solve a Sudoku puzzle, but the last step is often needed.
5. Since each number(1-9) must appear exactly once in each row, column, and mini-grid, if one of the guesses of an empty index does not appear as a guess for any other indexes in the same row/column/mini-grid this means that the guess has to be the number that takes the place of the index because if no other index can hold that number in the same row/column/mini-grid there has to be at least one index that does.

Arrays

Java Arrays are data structures containing objects of the same type. The arrays can be one, two, or three dimensional. My computer program generates a 9 by 9 by 9 three dimensional array to mimic the 9 by 9 by 9 matrix described previously. It is important to understand arrays before understanding the flowcharts that I have made.

Description of the Actual Computer Code

Main Program Overview: *solvePuzzle*

The computer first sets up the window with the Sudoku grid. The computer then asks the user to type in numbers into the grid which are part of a Sudoku puzzle that the user wants the computer to solve.

This grid is moved into a 3D array called *puzzleArray*, and all of the spots not filled out by the user are given a value of zero and are put in the second dimension of the array. The third dimension of the array will later be used to store the computer's guesses of possible numbers that could fit in the Sudoku grid.

To solve the puzzle, the computer uses the help of a program called *solvePuzzle* which has several sub-programs that help solve the puzzle the user typed into *puzzleArray*.

solvePuzzle first takes in *puzzleArray* and gives it three dimensions, 9 by 9 by 9. The first two dimensions represent the numbers in the Sudoku puzzle that the user will eventually see, and the third dimension behind the second dimension will consist of the computer's guesses of possible numbers that could fit into its corresponding spot in the second dimension.

solvePuzzle consists of several sub-programs, one of which is a Boolean called *checkArray*. Booleans are programs that return a true or false value. In this case, *checkArray* is returns true if all of the spots in the second dimension are filled in the array, but if one of the indexes equal zero then *checkArray* will return false. solvePuzzle will only continue if *checkArray* returns false, because that means the Sudoku puzzle is not completed yet.

The computer then runs through all of the numbers in the second dimension of *puzzleArray*. If one of the indexes equal zero then the computer calls on Boolean *checkNumber* which takes in *puzzleArray*, an index in *puzzleArray*, and a number x(1-9). *checkNumber* will check if x is a possible number that could represent the index in the Sudoku puzzle. If it is, *checkNumber* will return true and x will be stored as a guess in the third dimension. *checkNumber* runs through each index with a value of zero in the second dimension, and checks the numbers 1-9 for all of them.

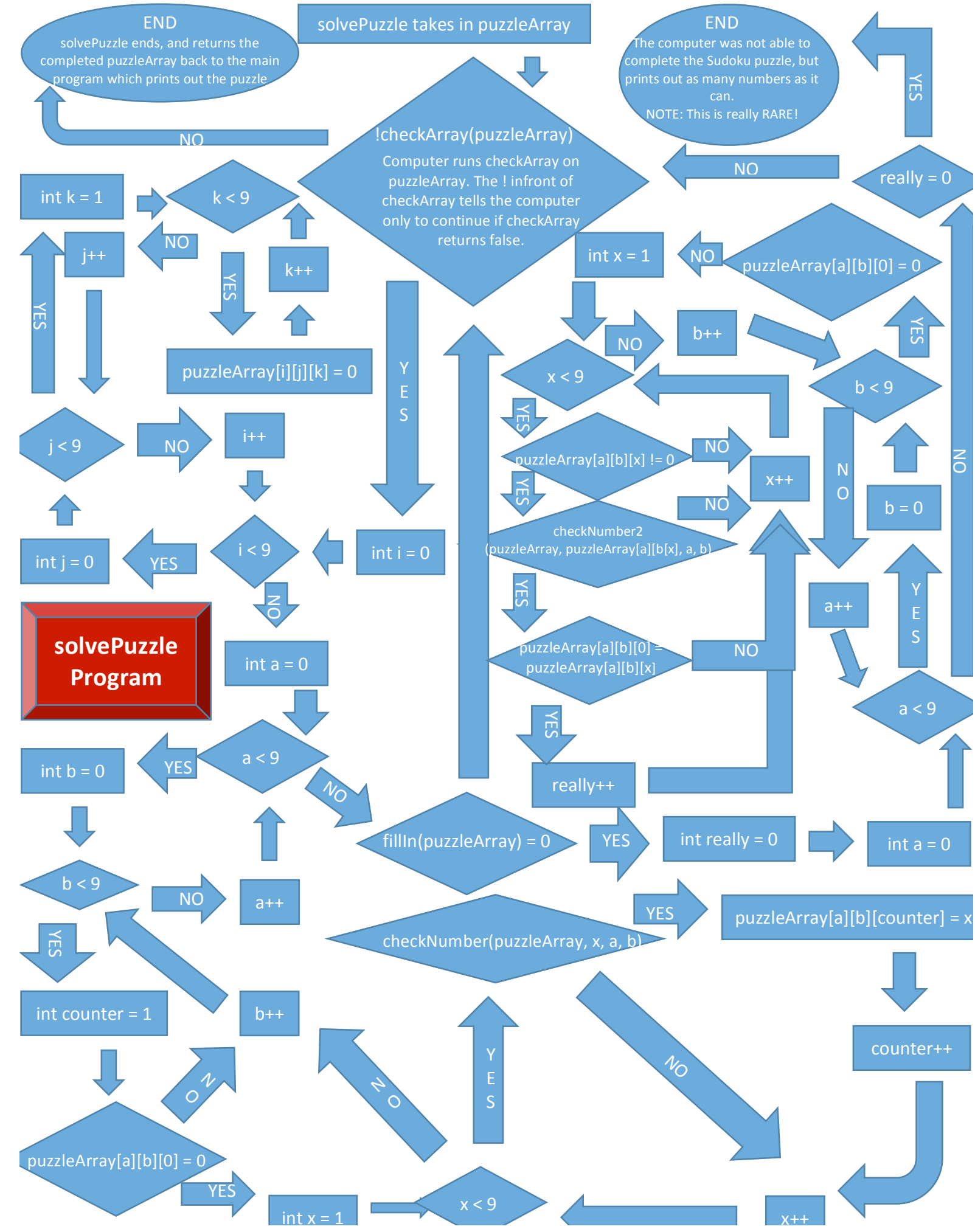
After the computer runs through all of the indexes, *solvePuzzle* gives puzzleArray to another sub-program called *fillIn*. *fillIn* runs through all of the spots in puzzleArray that have a zero in the second dimension, and if it has only one number that the computer placed in the third dimension as a guess then that guess will take the place of the zero in the second dimension.

Next, the computer will clear out all of the numbers in the third dimension, and run through *checkNumber* and *fillIn* all over again until there are no more changes left to be made. Once this happens, the computer runs through all of the numbers in the puzzleArray, and if one of the indexes equal zero then the computer calls on the Boolean checkNumber2. checkNumber2 is programmed to take an index in puzzleArray and one of its possible numbers/guesses in the third dimension, and then check if it is the correct number by running through all of the other guesses in the third dimension of the same row, column, and mini grid of the index. If the guess is not equal to another guess in the same row, column, or mini grid then that guess will take the place of the index in the second dimension. For example, if index[2][4][0] has a guess of 2, and no other index in its row, column, or mini grid has a

possibility of being 2 then index[2][4][0] must equal 2 because there has to be at least one of each number(1-9) in each row, column, and mini grid in Sudoku puzzles.

Once the computer runs checkNumber2 on all of the indexes in the second dimension equal to zero, *solvePuzzle* clears all of the indexes in the third dimension and sets them to zero. Then *solvePuzzle* runs *checkNumber* and fillIn on puzzleArray, and then runs through *checkNumber2* again if needed. This whole process, will be repeated as many times as it takes to solve puzzleArray.

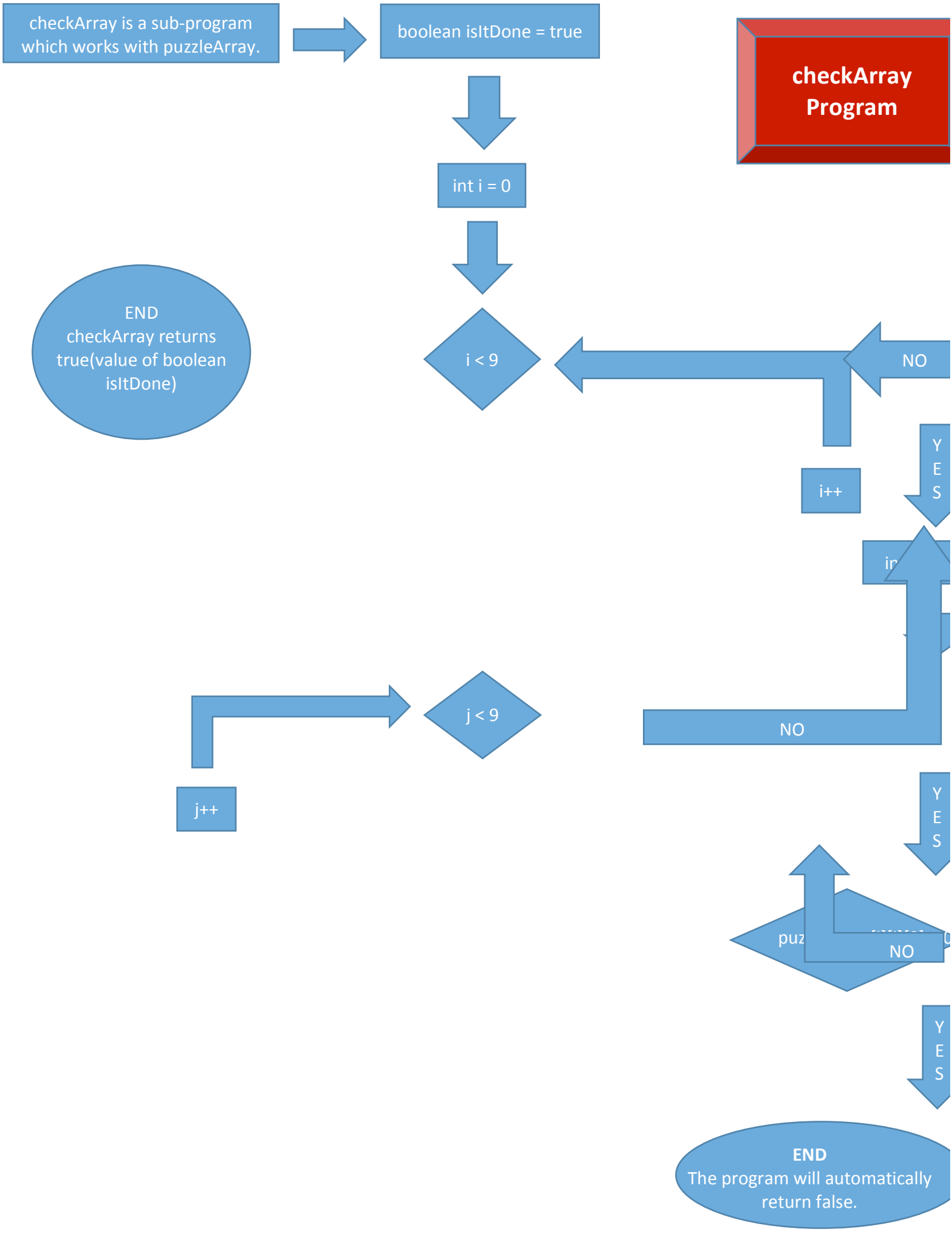
Finally, when *checkArray* returns true, and all of the numbers in the second dimension are filled in, then the main program prints the numbers out into the grid that the user is able to see on the computer screen.


solvePuzzle Program
solvePuzzle takes in puzzleArray
!checkArray(puzzleArray)
Computer runs checkArray on puzzleArray. The ! infront of checkArray tells the computer only to continue if checkArray returns false.
END
solvePuzzle ends, and returns the completed puzzleArray back to the main program which prints out the puzzle
END
The computer was not able to complete the Sudoku puzzle, but prints out as many numbers as it can.
NOTE: This is really RARE!
int k = 1
k < 9
j++
k++
puzzleArray[i][j][k] = 0
j < 9
i++
int j = 0
i < 9
int i = 0
int a = 0
int b = 0
a < 9
b < 9
a++
int counter = 1
b++
puzzleArray[a][b][0] = 0
int x = 1
x < 9
b++
b < 9
puzzleArray[a][b][x] != 0
x++
checkNumber2 (puzzleArray, puzzleArray[a][b[x], a, b)
puzzleArray[a][b][0] = puzzleArray[a][b][x]
really++
fillIn(puzzleArray) = 0
really = 0
int really = 0
int a = 0
a++
a < 9
b = 0
checkNumber(puzzleArray, x, a, b)
puzzleArray[a][b][counter] = x
counter++
int x = 1
x < 9
x++
YES
NO

checkArray Explanation

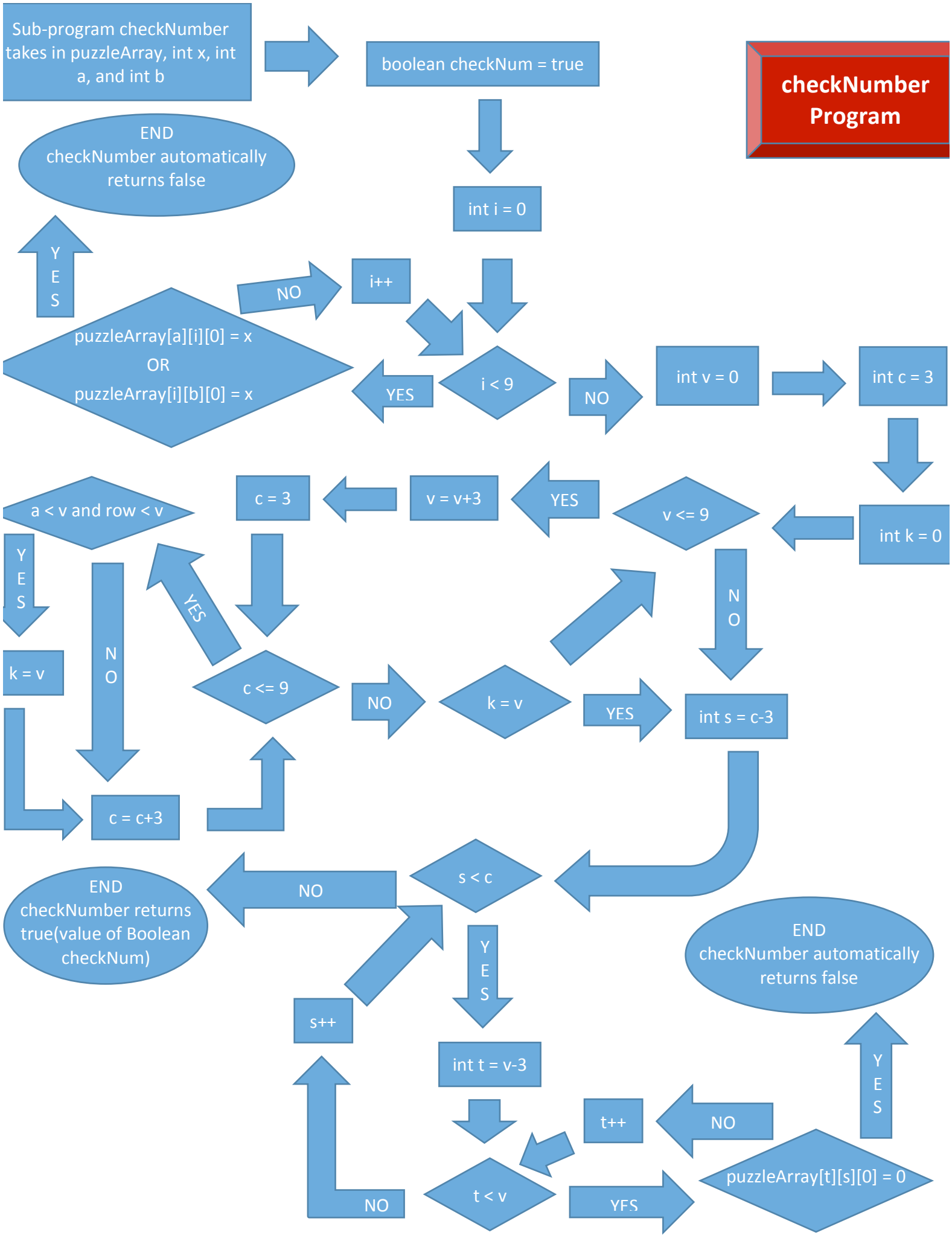
checkArray is a sub-program of the *solvePuzzle* program*. solvePuzzle* uses checkArray to find out if the computer is done solving the puzzle or if there are still more parts of the program left to be filled. *checkArray* returns a Boolean at the end of the program. A Boolean has a true or false value. If *checkArray* returns true, it means that all of the spots of the Sudoku puzzle are filled in the second dimension. However, if *checkArray* returns false, it means that the computer still has empty indexes.

In *checkArray* the computer runs through all of the indexes in the second dimension and checks if any of them have a zero in their spot. If even one zero is detected, the computer returns false. However, if the computer is able to run through the whole program without finding any zeros, *checkArray* returns true.


checkArray is a sub-program which works with puzzleArray.
boolean isItDone = true
checkArray Program
int i = 0
END
checkArray returns true(value of boolean isItDone)
i < 9
NO
i++
YES
NO
j < 9
j++
YES
NO
YES
END
The program will automatically return false.

checkNumber Explanation

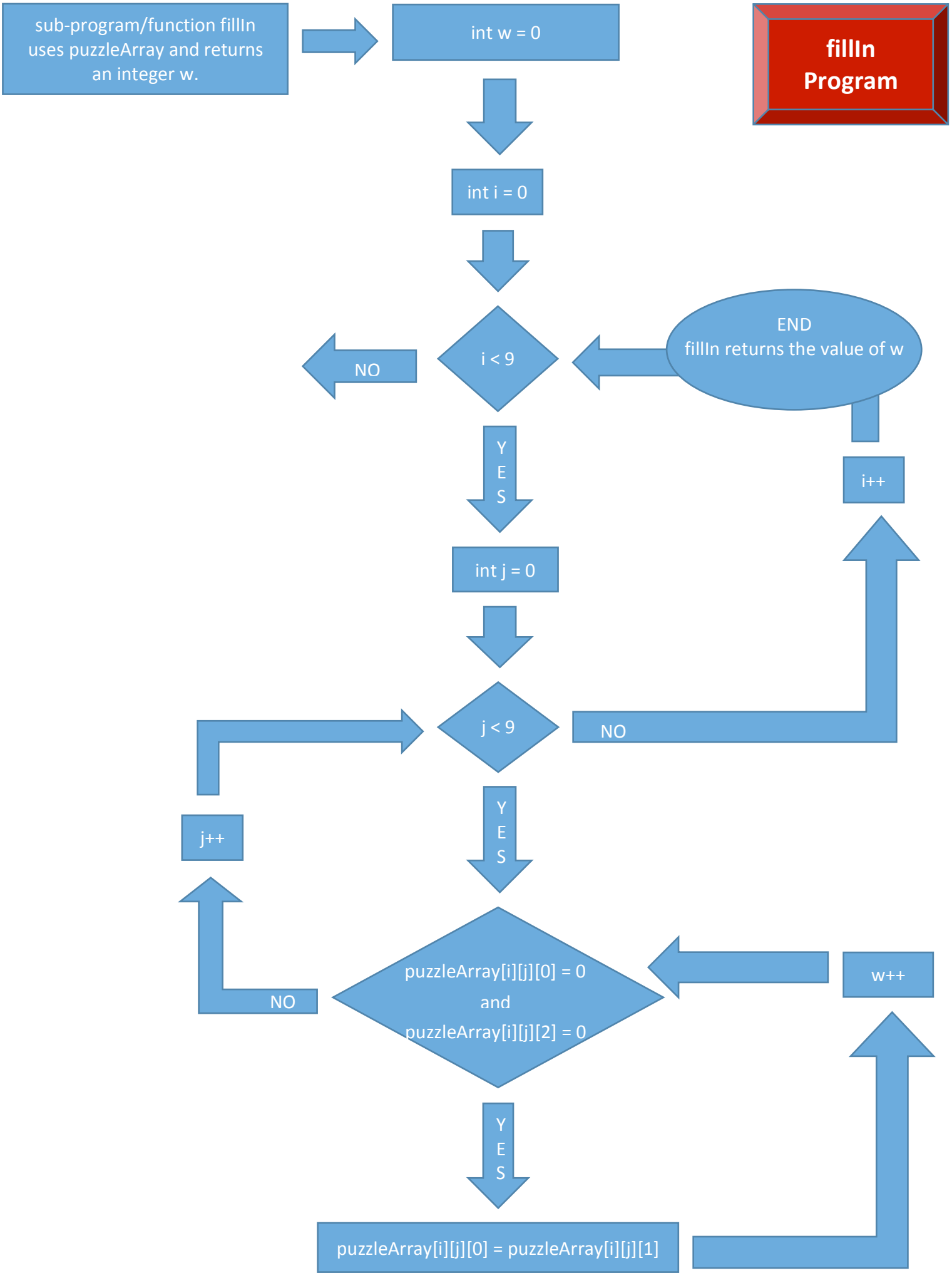
Boolean *checkNumber* takes in one index in puzzleArray and an integer. The goal of the program is to determine whether the integer is a possible candidate for the empty index in puzzleArray. The computer does this by running through all of the other numbers in the same column, row, and mini-grid in the second dimension of puzzleArray of the index given. If the integer given is already present in the same column, row, or mini-grid then *checkNumber* returns false, but if the computer does not detect the same integer in the column, row, or mini-grid of the index then the program will return true.


checkNumber Program
Sub-program checkNumber takes in puzzleArray, int x, int a, and int b
boolean checkNum = true
int i = 0
i < 9
YES
NO
i++
puzzleArray[a][i][0] = x
OR
puzzleArray[i][b][0] = x
YES
END
checkNumber automatically returns false
int v = 0
int c = 3
int k = 0
v <= 9
YES
v = v+3
c = 3
a < v and row < v
YES
NO
k = v
c <= 9
YES
NO
NO
k = v
int s = c-3
c = c+3
s < c
NO
END
checkNumber returns true(value of Boolean checkNum)
YES
s++
int t = v-3
t < v
NO
YES
t++
NO
puzzleArray[t][s][0] = 0
YES
END
checkNumber automatically returns false

fillIn Explanation

The goal of sub-program *fillIn* is to fill in as many integers as it can into the second dimension of puzzleArray. *fillIn* first runs through all of the indexes in the second dimension. If one of the indexes has only one guess in the the third dimension, then that guess takes the place of the zero in that index.

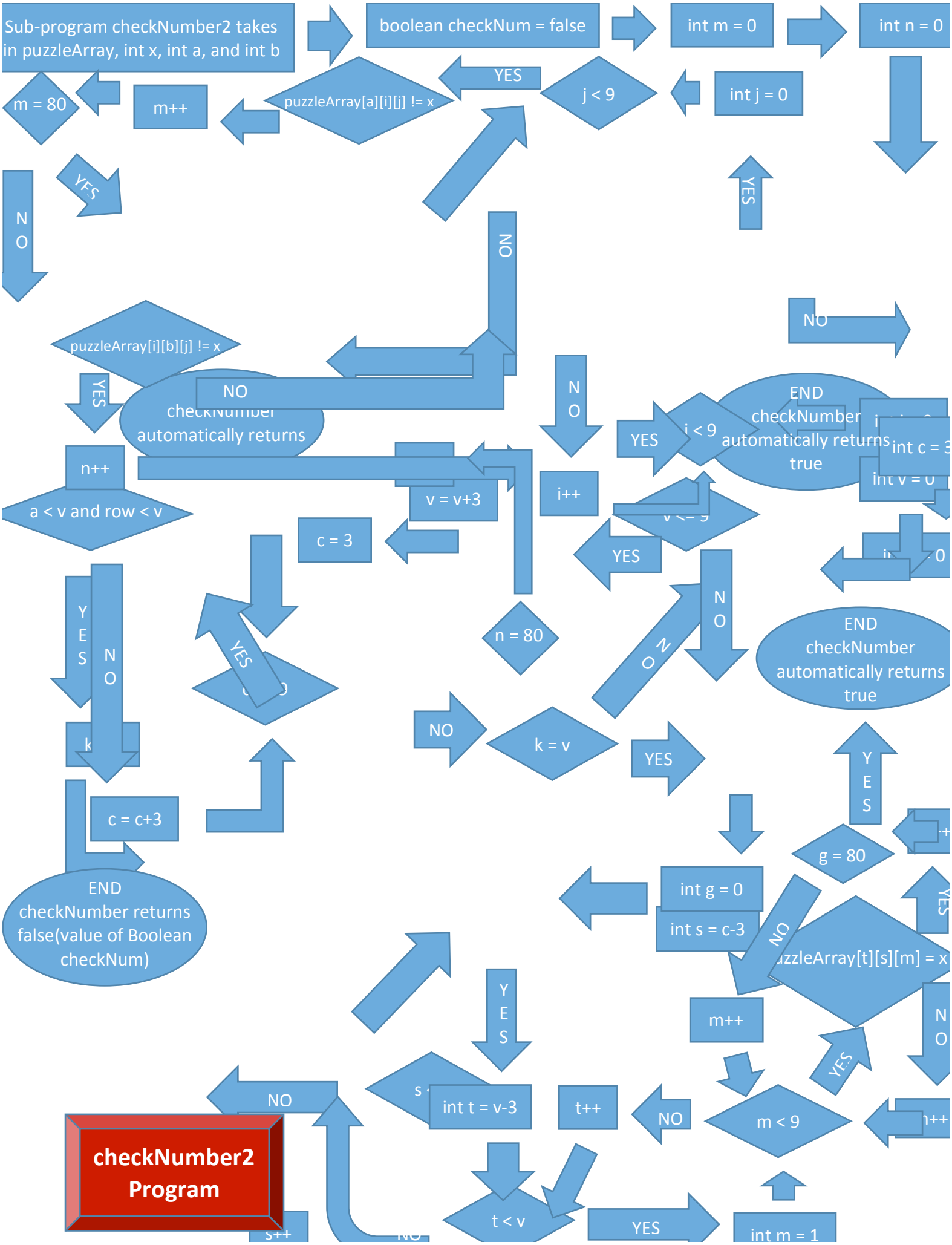
The computer also keeps track of how many times a zero in the second dimension is replaced by a guess. If no changes to puzzleArray are made after *fillIn* then the computer will decide to move onto the next step involving another method to fill in indexes in puzzleArray. But if a few changes are made by *fillIn* in puzzleArray then the computer will redo *checkNumber* and *fillIn* because there may be more changes left to be made using this method.


fillIn Program
sub-program/function fillIn uses puzzleArray and returns an integer w.
int w = 0
int i = 0
i < 9
NO
END
fillIn returns the value of w
YES
i++
int j = 0
j < 9
NO
YES
j++
puzzleArray[i][j][0] = 0
and
puzzleArray[i][j][2] = 0
NO
w++
YES
puzzleArray[i][j][0] = puzzleArray[i][j][1]

checkNumber2 Explanation

Boolean *checkNumber2* is similar to *checkNumber* but has a different method in returning a true or false value. *checkNumber2* also takes in puzzleArray, one index in the second dimension of puzzleArray, and a guess of the same index in the third dimension. checkNumber2 will test to see if this guess is the only possible number to replace the zero in the index originally given to the program.

checkNumber2 is programmed to check if the integer is the correct number by running through all of the other guesses in the third dimension of the same row, column, and mini grid of the index. If the guess is not equal to another guess in the same row, column, or mini grid then that guess will take the place of the index in the second dimension. For example, if index[2][4][0] has a guess of 2, and no other index in its row, column, or mini grid has a possibility of being 2 then index[2][4][0] must equal 2 because there has to be at least one of each number(1-9) in each row, column, and mini grid in Sudoku puzzles.


checkNumber2 Program
Sub-program checkNumber2 takes in puzzleArray, int x, int a, and int b
boolean checkNum = false
int m = 0
int n = 0
int j = 0
j < 9
YES
puzzleArray[a][i][j] != x
m++
m = 80
NO
YES
int c = 3
c = 3
c = c+3
n++
a < v and row < v
v = v+3
i++
i < 9
n = 80
k = v
int g = 0
int s = c-3
g = 80
m++
m < 9
int t = v-3
t++
t < v
int m = 1
puzzleArray[i][b][j] != x
NO
checkNumber automatically returns
END checkNumber automatically returns true
END checkNumber returns false(value of Boolean checkNum)

New Directions and Questions for Further Research

Even though the Sudoku Solver solves the majority of Sudoku puzzles, it does not solve every single one. My computer program does not involve the guessing of numbers, but instead the computer solves the puzzle by eliminating possible numbers for each spot in the Sudoku grid and then finding spots with only one possible solution until all of the spots in the puzzle are filled. Even though this is an efficient and interesting way to solve a Sudoku puzzle it does not guarantee a solution. Therefore, if I ever have a chance to further improve my project and make my program more efficient I will definitely make sure I add a guessing component in my program so that the computer can solve all of puzzles and not just the majority.

Conclusion

The Sudoku Solver is one of the many computer programs that involves the use of mathematics. Even though the original algorithm I made seems simple, the computer programming it takes to make it happen is very complicated and the code is very intricately designed. My program also uses several mathematical concepts such as three dimensional arrays, Booleans, several integers, for-loops, while-loops, and if statements.

Even though it was challenging to make all the small pieces finally fit together because in computer programming the smallest mistake could take hours to find and fix, I learned many things from the experience. Few of the many things I learned include how to make several programs work together to make one final product, how to take a complex problem/idea and make sense of it, and most importantly I learned that I want to further study and improve my knowledge of computer Science.

Computer Code

```
import javax.swing.JFrame;

public class solvePuzzle {
	public static int[][][] puzzleArray = new int[9][9][9];
	public static void main(String[] args) {

		myWindow window = new myWindow("Soduku Solver");
		window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}

	public static void solve(){
		while (!checkArray(puzzleArray)) {
			for (int i = 0; i < 9; i++) {
				for (int j = 0; j < 9; j++) {
					for (int k = 1; k < 9; k++) {
						puzzleArray[i][j][k] = 0;
					}
				}
			}

for (int a = 0; a < 9; a++) {
	for (int b = 0; b < 9; b++) {
		int counter = 1;
			if (puzzleArray[a][b][0] == 0) {
				for (int x = 9; x >= 1; x--) {
					if (checkNumber(puzzleArray, x, a, b)) {
						puzzleArray[a][b][counter] = x;
							counter++;
						}
					}

				}
			}
		}
			if (fillIn(puzzleArray) == 0) {
				int really = 0;
				for (int a = 0; a < 9; a++) {
					for (int b = 0; b < 9; b++) {
						if (puzzleArray[a][b][0] == 0) {
							for (int x = 1; x < 9; x++) {
					if (puzzleArray[a][b][x] != 0) {
								if (checkNumber2(puzzleArray,
puzzleArray[a][b][x], a, b)) {

	puzzleArray[a][b][0] = puzzleArray[a][b][x];
									really++;
									}
								}
								}

							}
```

```
                                    }
                              }
                        if (really == 0) {
                              break;
                        }
                        }
                  }
            }

      public static boolean checkNumber2(int[][][] array, int x, int column, int
row) {
            boolean checkNum = false;
            int countcolumn = 0;
            int countrow = 0;
            for (int i = 0; i < 9; i++) {
                  for(int j = 0; j < 9; j++) {
                  if (array[column][i][j] != x) {
                        countcolumn++;
                        if(countcolumn == 80) {
                              return true;
                        }
                  }
                  if (array[i][row][j] != x) {
                        countrow++;
                        if(countrow == 80) {
                              return true;
                        }
                  }
            }
            }
            int v = 0;
            int c = 3;
            int k = 0;
            int countgrid = 0;

            while (v <= 9) {
                  v += 3;
                  c = 3;
                  while (c <= 9) {
                        if (column < v && row < c) {
                              k = v;
                              break;
                        }
                        c += 3;
                  }
                  if (k == v)
                        break;
            }
            for (int s = c - 3; s < c; s++) {
                  for (int t = v - 3; t < v; t++) {
                        for(int m = 1; m < 9; m++) {
                        if (array[t][s][m] == x) {
                              countgrid++;
```

```
                        if(countgrid == 80)
                        return (true);
                  }
                  }

            }
      }
      return (checkNum);

}

public static boolean checkArray(int[][][] puzzleArray) {
      boolean isItDone = true;
      for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                  if (puzzleArray[i][j][0] == 0) {
                        return (false);
                  }
            }
      }
      return (isItDone);
}

public static boolean checkNumber(int[][][] array, int a, int column, int
row) {
      boolean checkNum = true;
      for (int i = 0; i < 9; i++) {
            if (array[column][i][0] == a) {
                  return (false);
            }
            if (array[i][row][0] == a) {
                  return (false);
            }
      }
      int v = 0;
      int c = 3;
      int k = 0;

      while (v <= 9) {
            v += 3;
            c = 3;
            while (c <= 9) {
                  if (column < v && row < c) {
                        k = v;
                        break;
                  }
                  c += 3;
            }
            if (k == v)
                  break;
      }
      for (int s = c - 3; s < c; s++) {
            for (int t = v - 3; t < v; t++) {
```

```
                        if (array[t][s][0] == a) {
                              return (false);

                        }

                  }
            }
            return (checkNum);
      }

      public static int fillIn(int[][][] array) {
            int x = 0;
            for (int i = 0; i < 9; i++) {
                  for (int j = 0; j < 9; j++) {
                        if (array[i][j][0] == 0 && array[i][j][2] == 0) {
                              array[i][j][0] = array[i][j][1];
                              x++;
                        }
                  }

            }
            return (x);
      }
}
```