



# The Sudoku Solver

Project by:

*Supreeth Ratam* 200100130

*Korimi Vennela* 20d100013



# Contents

- Abstract
- Rules of sudoku
- Description of algorithm
- Problems Faced
- Results



# Abstract

[Link for our Abstract](#)



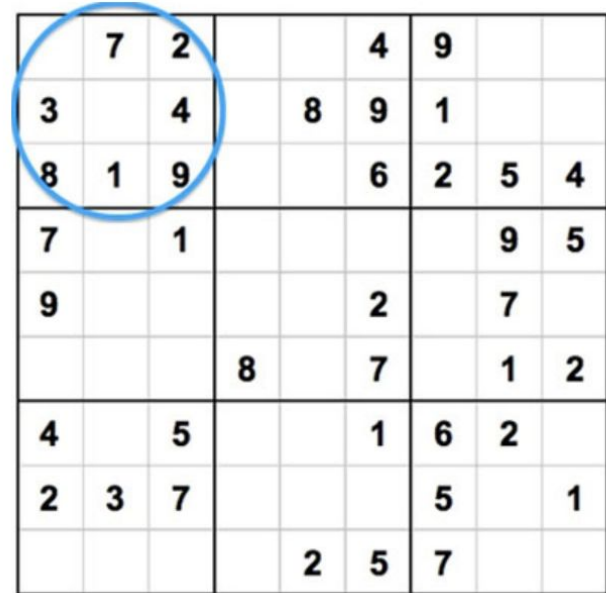
# About sudoku

## Overview

Sudoku is played on a grid of 9 x 9 spaces. Within the rows and columns are 9 “squares” (made up of 3 x 3 spaces). Each row, column, and square (9 spaces each) needs to be filled out with the numbers 1-9, without repeating any numbers within the row, column, or square.

## Game Rules

Each Sudoku grid comes with a few spaces already filled in; the more spaces filled in, the easier the game – the more difficult Sudoku puzzles have very few spaces that are already filled in. For example ;



A 9x9 Sudoku grid with a 3x3 subgrid circled in blue. The grid contains the following numbers:

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

in the upper left square (circled in blue), this square already has 7 out of the 9 spaces filled in. The only numbers missing from the square are 5 and 6. By seeing which numbers are missing from each square, row, or column, we can use the process of elimination and deductive reasoning to decide which numbers need to go in each blank space.

For example, in the upper left square, we know we need to add a 5 and a 6 to be able to complete the square but based on the neighboring rows and squares we cannot deduce which number to add in which space. Finally, sudoku is the game of patience, insights, and recognition of patterns



## Description of Algorithm

- We have to enter numbers from sudoku which we want to solve in *Input\_board.txt* file.
- One has to make sure to enter 0's in place of the blanks given in the sudoku board and have to write every row in a separate line with numbers separated with space.
- *Grid.py* file imports the board from input\_board.txt file and converts it an array
- *Sudoku\_solver.py* imports array from *Grid.py* and this is where the solving begins . This file has 3 functions to replace 0's in the array according to the sudoku rules.
- They are

*blank\_space* , *is\_possible* and *solver*

  
*def blank\_space ():*

This function is used to check 0's in the array

It iterates from left to right of every row and returns the row and column of the cell where there is a blank and returns false if there are no more blanks left in the array

In this way it also implies if the board is completely solved by returning false





```
def is_possible():
```

- This function is used to check if a number could satisfy all the rules to be in a given box.
- It returns **True** if the number is not in the given cell's row , column and the 3 x3 grid.
- If any of the above case is not valid then it returns **False** implying that number could not satisfy the rules of sudoku to be in that cell.



## *def solver():*

- This is the main function where everything comes into a place.
- The function first uses the *blank\_space* function to find an empty blank.
- Then it iterates all the numbers in the range of 1 to 9 including 9 and uses the *is\_possible* function to get a valid number for the blank.
- Then it replaces the 0 with a valid number and it checks if the board is further solvable.
- If it solvable then it runs again till every 0 in the board is replaced by a valid number.

## Sudoku\_board.py

Solver\_board.py returns a grid with all the 0's replaced by valid numbers.

Sudoku\_board.py will import the solved\_grid from Solver\_board.

We used pygame module to draw the grid and add numbers of the array on the screen.

`grid()` function is used to draw the basic sudoku board and then, the `filled_box()` function is used to fill colours in the boxes which have a predefined number, so that we can differentiate between the given and solved boxes of the board.

Lastly, we defined a `print_numbers` function to fill the board with the solved grid.



## Problems faced

The main problem we came across is when 2 or more numbers are possible for a same cell in the grid.

To overcome this issue we used the backtracking algorithm.

We used a recursion loop such that every time if there are multiple numbers for the same position we will consider first number as our solution and we will check if the board is further solvable, If it further not solvable we will return to the previous step and replaces the number with next valid number until there are no blanks left in the board.



## Problems faced

Another problem we faced if while filling colours for the grid.

We required an unsolved grid to fill colours for the cells with already a number present in it but since, so [Sudoku\\_solver](#) will run before printing the board the unsolved grid will be overwritten with solved grid and all the boxes of the board will be filled with colours.

To overcome this issue we made another array in **Grid.py** file while making the first grid and this will be further used to fill the boxes.

# Results

```
> input_sudoku board.txt  
0 5 0 1 7 9 3 8 0  
0 0 0 0 0 4 0 1 0  
0 1 9 0 0 0 7 2 0  
0 0 0 0 0 8 2 0 0  
2 6 0 0 4 0 0 0 5  
8 3 0 2 9 0 0 0 7  
0 4 3 0 6 0 0 7 8  
6 0 0 8 0 5 0 4 3  
1 9 0 4 3 0 0 0 2
```

input

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

OutPut



## Future works and Conclusion

- While writing code to write this project, we thought it will be a pain to enter the sudoku board manually into the input file and it would be better if we could get the array by directly scanning a photo of sudoku.
- So we thought of working on programmes like OpenCV and OCR.
- We were able to solve the sudoku successfully and we learnt a lot of new things and had a good exposure to pygame library and backtracking algorithm.