Project report

On

**SUDOKU SOLVER using backtracking algorithm**

Design and Analysis of Algorithms

By

2010030512 - L.Keerthana

2010030517 -M.Manideepa

2010030526 - D.dedeepya

2010030530 - K.Sri Teja

under the supervision of

**Udaya Rani**



Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.

March, 2022

**Declaration**

The Project Report entitled **SUDOKU SOLVER using backtracking algorithm** is a record of bona fide work of 2010030512-L.Keerthana,2010030517-Manideepa,2010030526-Dedeepya,2010030530-Sri Teja submitted as a requirement for the completion of the course **Design and Analysis of Algorithms** in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/University/Institute.

2010030512-L.Keerthana

2010030517M.Manideepa

2010030526-D.Dedeepya

2010030530-K.Sri Teja

## Certificate

This is to certify that the Project Report entitled **SUDOKU SOLVER using backtracking algorithm** is being submitted by 2010030512-L.Keerthana,2010030517-Manideepa,2010030526-Dedeepya,2010030530-Sri Teja as a requirement for the completion of the course **Design and Analysis of Algorithms** in the Department of Computer Science and Engineering, K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

## 

## Signature of the Supervisor

Name and Designation

**Signature of the HOD**

**Signature of the Examiner**

**ACKNOWLEDGEMENT**

First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

We take grateful opportunity to thank our beloved Founder and Chairman who has given constant encouragement during our course and motivated us to do this project. We are grateful to our Principal **Dr. L. Koteswara Rao** who has been constantly bearing the torch for all the curricular activities undertaken by us.

We pay our grateful acknowledgement & sincere thanks to our Head of the Department **Dr. Chiranjeevi Manike** for his exemplary guidance, monitoring and constant encouragement throughout the course of the project. We thank **Dr.Deepthi Kalavala** of our department who has supported throughout this project holding a position of supervisor.

We whole heartedly thank all the teaching and non-teaching staff of our department without whom we won’t have made this project a reality. We would like to extend our sincere thanks especially to our parents, our family members and friends who have supported us to make this project a grand success.

**ABSTARCT**

* We present the detailed development and implementation of simple sudoku game. The sudoku game consists of graphical user interface ,solver and implemented using python
* The solver finds the solution to the puzzles entered by the user. This project gives an insight to the different aspects of python programming

**WORK ALLOCATION For Project**

CODING AND TESTING – Sriteja ,Dedeepya

ALGORITHM AND FLOWCHART- Keerthana

ANALYZE DATA AND TECHNIQUES- Manideepa

## 

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| 1 | Abstract | 6 |
| 2 | Introduction | 7 |
| 3 | Literature Survey | 8 - 9 |
| 4 | Flow Diagram | 10 |
| 5 | System requirements – Software & Hardware | 11 |
| 6 | Methods | 12 |
| 7 | Implementation  7.1 Code  7.2 Output | 13 - 19 |
| 8 | References | 20 |
| 9 | Conclusion & Future Work | 21 |

**INTRODUCTION**

Sudoku is a combinational number-placement puzzle. The objective of sudoku is to fill a 9×9 grid with some digits in such a way so that each column, row, and nine 3×3 subgrids contain all of the digits from 0-9.

We’ll use an algorithm called backtracking. Backtracking is an algorithm for finding solutions to some computational problem. It tries to find a solution until it validates all criteria.

In sudoku, the algorithm will put a number in an empty space and then move on to the next empty space, if anywhere some criteria fail then it will turn back to some previous solutions and changes it according to the criteria. It continues until it finds a final solution that matches all the criteria.

**1.1 PROBLEM STATEMENT:-**

* The aim of the puzzle is to enter a numerical digit from 1 through 9 in each cell of a 9x9 grid made up of 3×3 subsquares or subgrids, starting with various digits given in some cells; each row, column, and subsquares region must contain each of the numbers 1 to 9 exactly once.
* A sudoku solution must satisfy all of the following rules:-

1. Each horizontal row contains each digit exactly once

2. Each vertical column contains each digit exactly once

3. Each subgrid or region contains each digit exactly once

**LITERATURE SURVEY:-**

**2.1 EXISTING SOLUTIONS:-**

* Sudoku puzzles are NP-complete problems and as such, their solution can be found by performing an exhaustive search.

**1) Pen and Paper Algorithm** :- a solution based on some strategies used by humans when solving the puzzle, therefore, it is called pencil-and-paper algorithm.

* As there are puzzles with different types of difficulty, the easy and medium
* puzzles can be solved using some simple techniques such as unique missing method, naked singles

**2) Brute Force Search/BackTracking/Depth For Search** :-

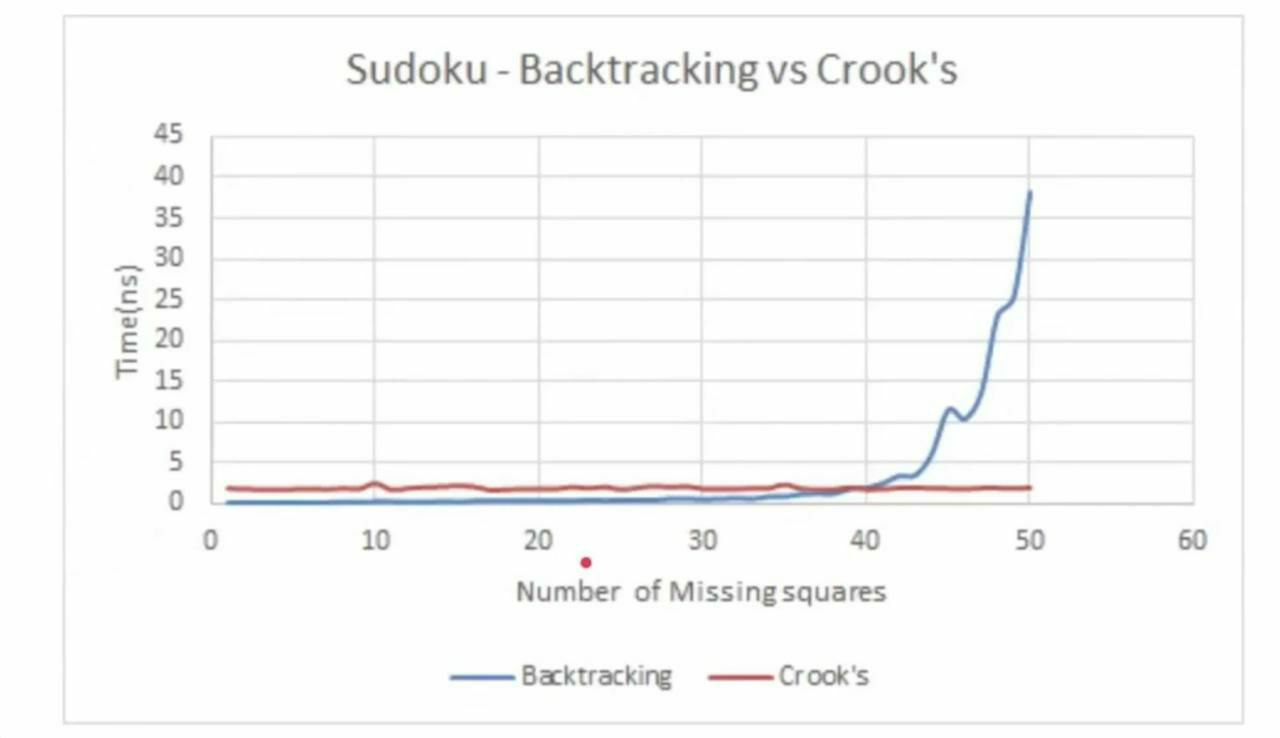
* Backtracking checks every possible outcome that can occur on the board and is terminated when correct output occurs in one of these outcomes.
* Backtråcking is also known as "Complete Brute Force Search" OR "Depth For Search Traversal Approach in Grid"

**3)** **CROOKS ALGORITHM** :-

* Crooks algorithm, which is an imitation of pencil and paper algorithm, is a simple algorithm which we use to solve the sudoku on our own.
* This algorithm sounds simple, but it has its limitations. Sometimes, we get incomplete answer to the sudoku because of multiple values that can occur on a particular empty space.

**2.2 COMPARISION:-**

* According to the data, crooks algorithm is much faster than backtracking algorithm as it eliminates the impossible outcomes and then solves the algorithm, as compared to backtracking which generates all possible sudoku matrices which takes more time.
* On the other hand, complete version of Crooks algorithm is that, solve the puzzle, and if it is incomplete, complete it with backtracking.



**2.3 WHICH IS BEST ALGORITHM:-**

* Crook's on its own is an incomplete algorithm, but it imitates solving by pencil and paper like us. It Might Get Stuck Somewhere due to Own Wrong Assumption
* Backtracking checks all possible combinations but it provides correct solution. Although, it Takes More Time, But is Able to Provide Correct Answer
* Best Algorithm => Backtracking Approach/ DFS Approach

**Hardware and Software Requirements :-**

* 1. **SOFTWARE REQUIREMENTS** : python(pycharm) ,windows operating system (can’t use mac coz have compatibility issues )
  2. **HARDWARE REQUIREMENTS** :mouse or trackpad

**FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS**

2.4 Non Functional Requirements

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&

2.4 Non Functional Requirements

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&:

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&:

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&:

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&:

**4.1 Non Functional Requirements**

**4.1.1 Performance:-**

The puzzle generator should be able to generate distinct puzzles every time based on given difficulty.The puzzle solver should be fast enough to solve even the hardest puzzles .

**4.1.2 Reliability**

The product should not crash under any circumstance such as user entering invalid values, user trying to load unsupported files etc.Also the solver should be able to solve any kind of puzzle and should be able to detect an illegal or individual puzzle.

**4.1.3 Portability**

Our product will be portable to carry and will run in any machine provided it runs a Windows Operating System.

**Proposed Algorithm Design Technique**

**5.1 BACKTRACKING ALGORITHM** :-

* In simple terms, we first start with the first empty space on the sudoku board. We add the first element that can come in that space, let's say 1.
* Next, we move on to the second empty space, and add the first element, 1, to it.
* If this is valid, it will go to next space, or else the element will become 2. This repeats for every element that can be placed in that space.
* This process is repeated for every empty space and if the sudoku is not solved, it goes to the first empty space and places next possible element, that is, 2, in our case.

This whole process is repeated until the sudoku is solved

**5.2 ALGORITHM**

Find an empty cell with coordinates (row, col).

if no cell is found then

return true;

end

for digits x from 1 to 9 do

if we can place x at position (row,col) and the puzzle remains valid then recursively continue to fill the rest of the puzzle:

if recursion succeeds i.e. returns true then

return true

end

else

free cell and try with the next digit;

end

end

end

if none of the digits lead to a solution then

return false

end

**5.3 BACKTRACKING –ANALYSIS**

* The time complexity for solving the sudoku is O(n^m), where m = nxn. Where n is the number of squares in one side of the sudoku square. In classical sudoku, n = 9. Thus, time complexity for classical sudoku is 0(9^81).
* The recurrence relation for backtracking algorithm for classical sudoku can be written as T(m) = 9\*T(m-1) + O(1)
* This is because, we have to check 9 values in a particular empty space.

**5.4 Data Structures needed**

**3D array or matrix**

* Sudoku is a 2D grid game; so this data structure comes in mind intuitively.
* Mostly the operation involves reading the value of cell and writing value to the cell. So, these operations must be as fast as possible. With the use of 2D array, these operations will be O(1).
* So this is the best data structure which we can use.
* Assuming that you will use backtracking, you might check what are the possible values for a cell. To keep track of all values, you will store them. So it will be way easier to store them in Z-direction for each cell. Hence we need the third dimension**.**

2.4 Non Functional Requirements

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

86$3&0600 6#$!0&&"%!!#%46&

2.4 Non Functional Requirements

2.4.1 Performance

6??"!&06377&"!&3$0&$!&46??09#

&$%703!"$9!3$ $6&#:46??090637 0&!6"&

099!&30&46??06!30!3:

2.4.2 Reliability

436&063!&  0  6!3 !#$6%0&!06060

!&$!"$!9$3960+ 60 &#$!"&3 6!0644&3 $0&:0 &

0906377&09!#2$!3 46??!306377&3&&

!$"$!9$346??:



2.4.3 Portability

6436& 1$ 7 4&7&# !3  1$6!$! !#%$!

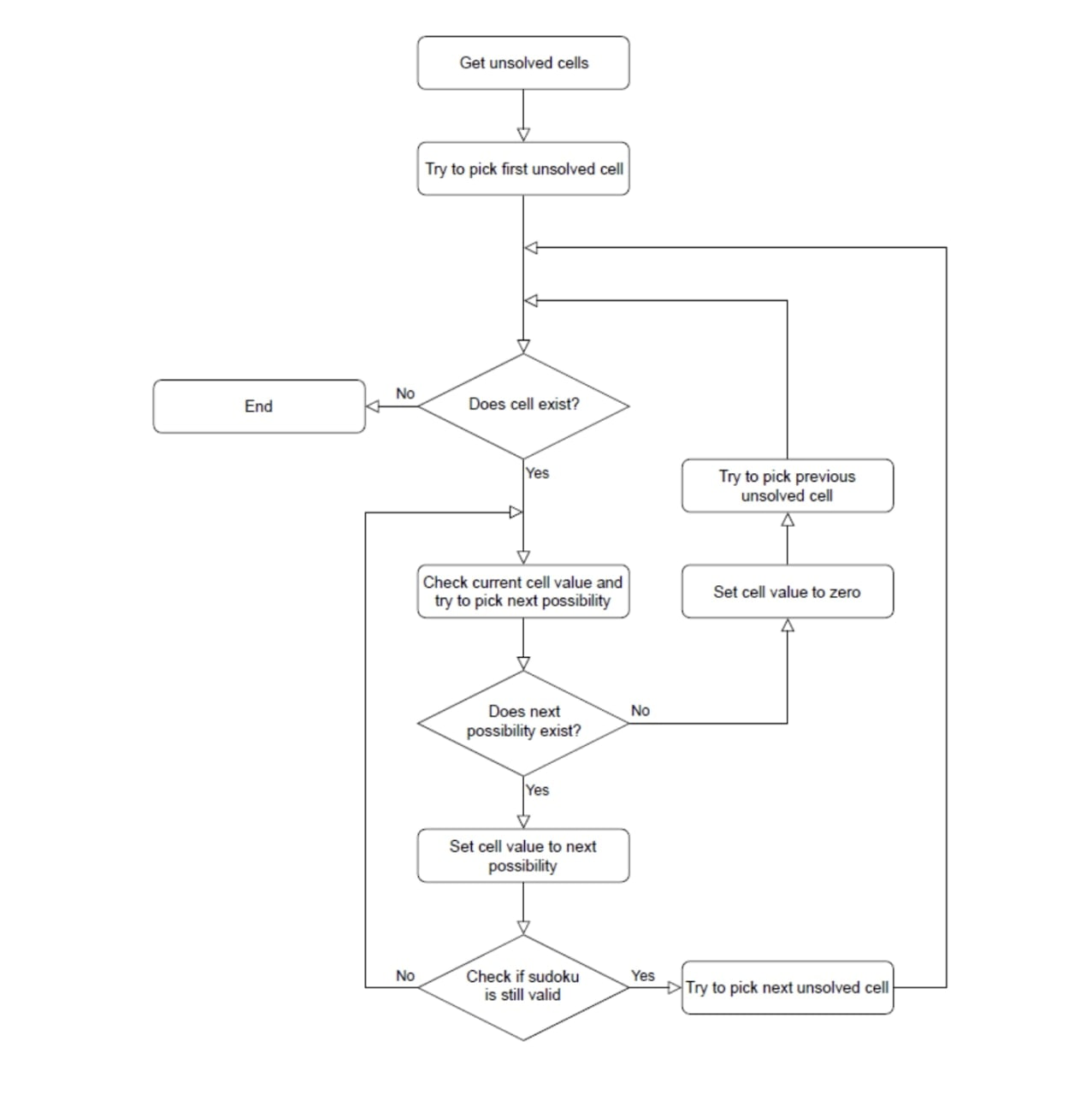
49$33$&6!0;$!3104&$!"#0&%:;9&3!$!0&

1$  %4$0    $0  $!&    0$!"  E6&7  ':%0$\*:  !#  &$0  $  $0

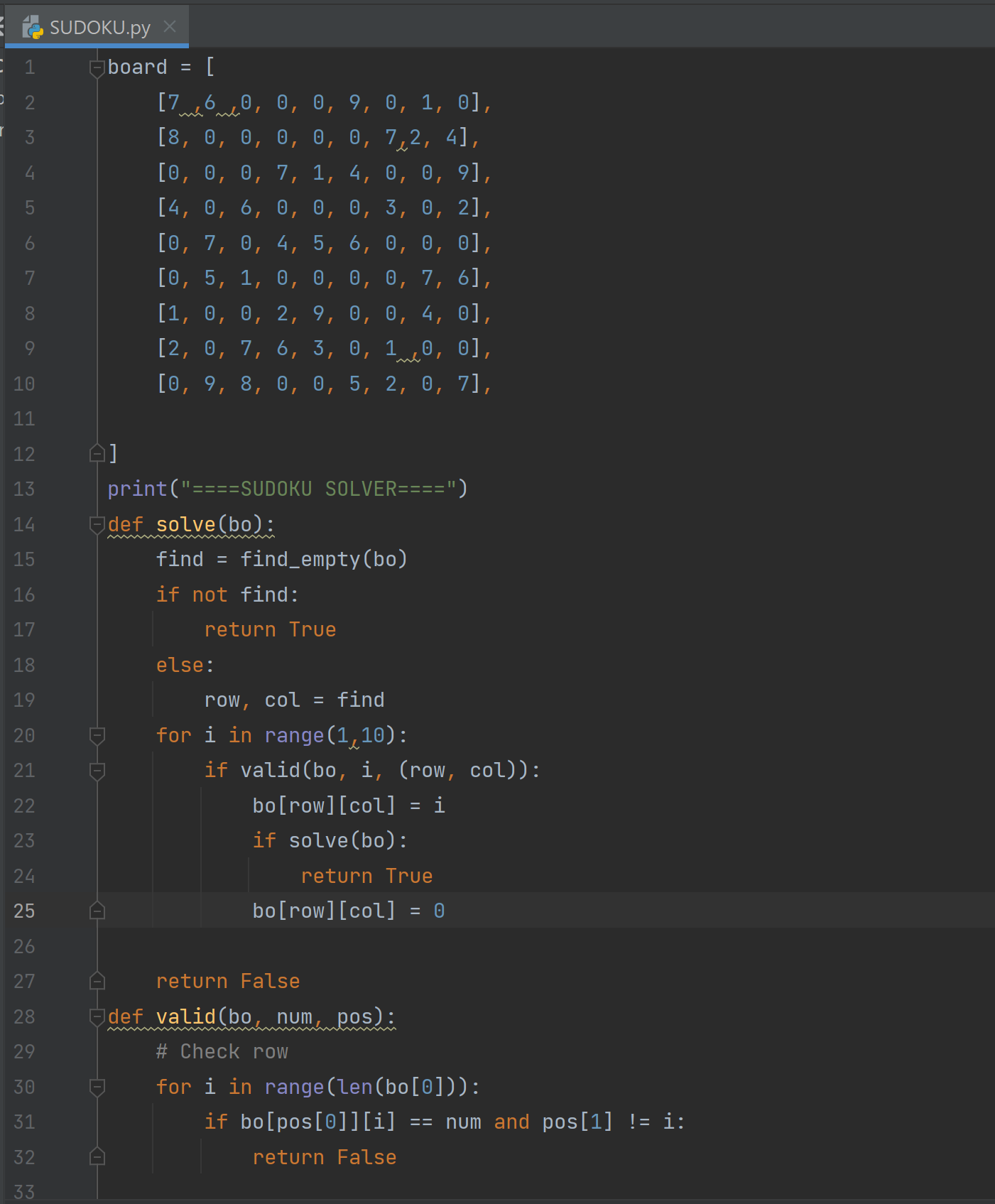
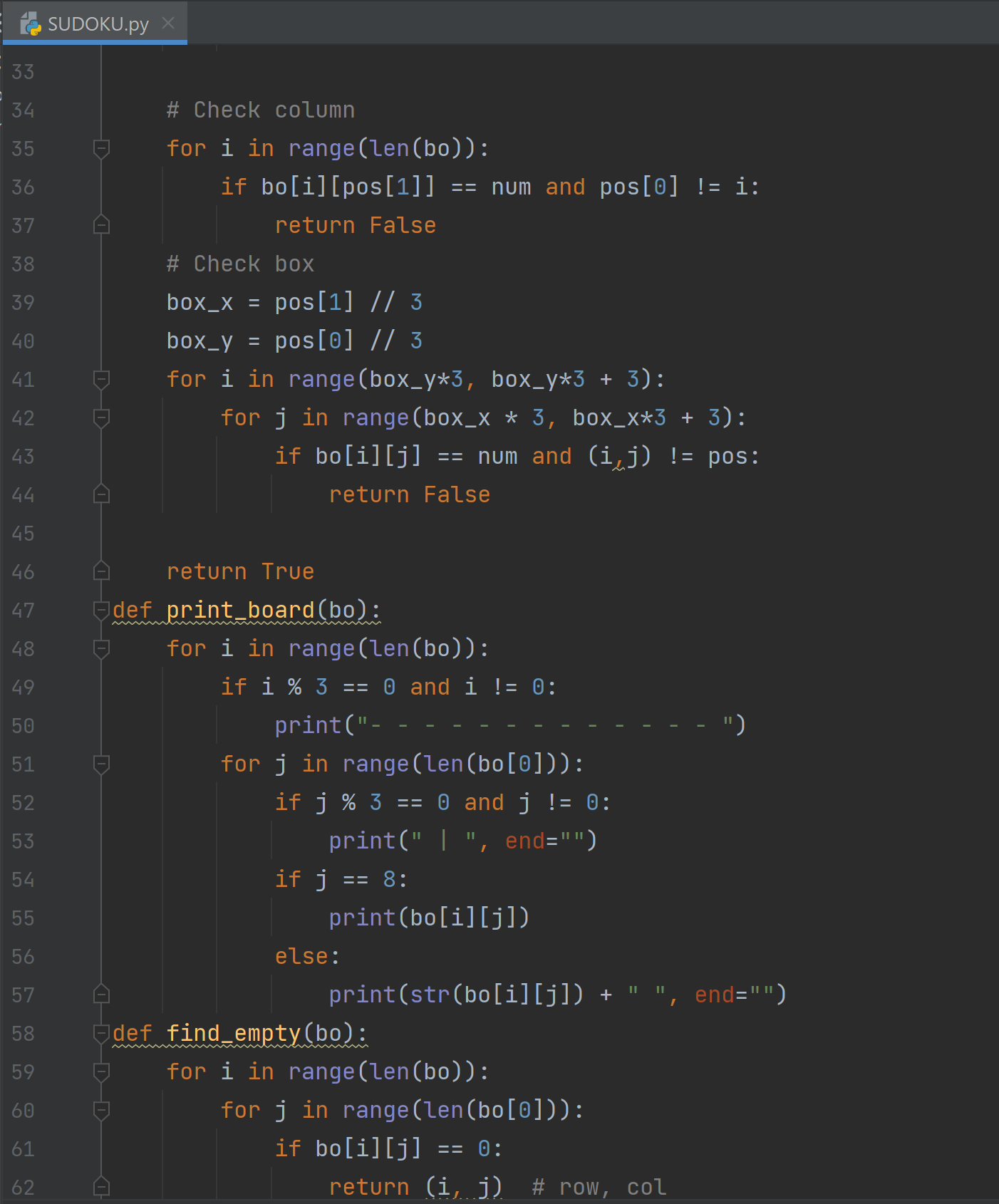
86$3&0600 6#$!0&&"%!!#%46&

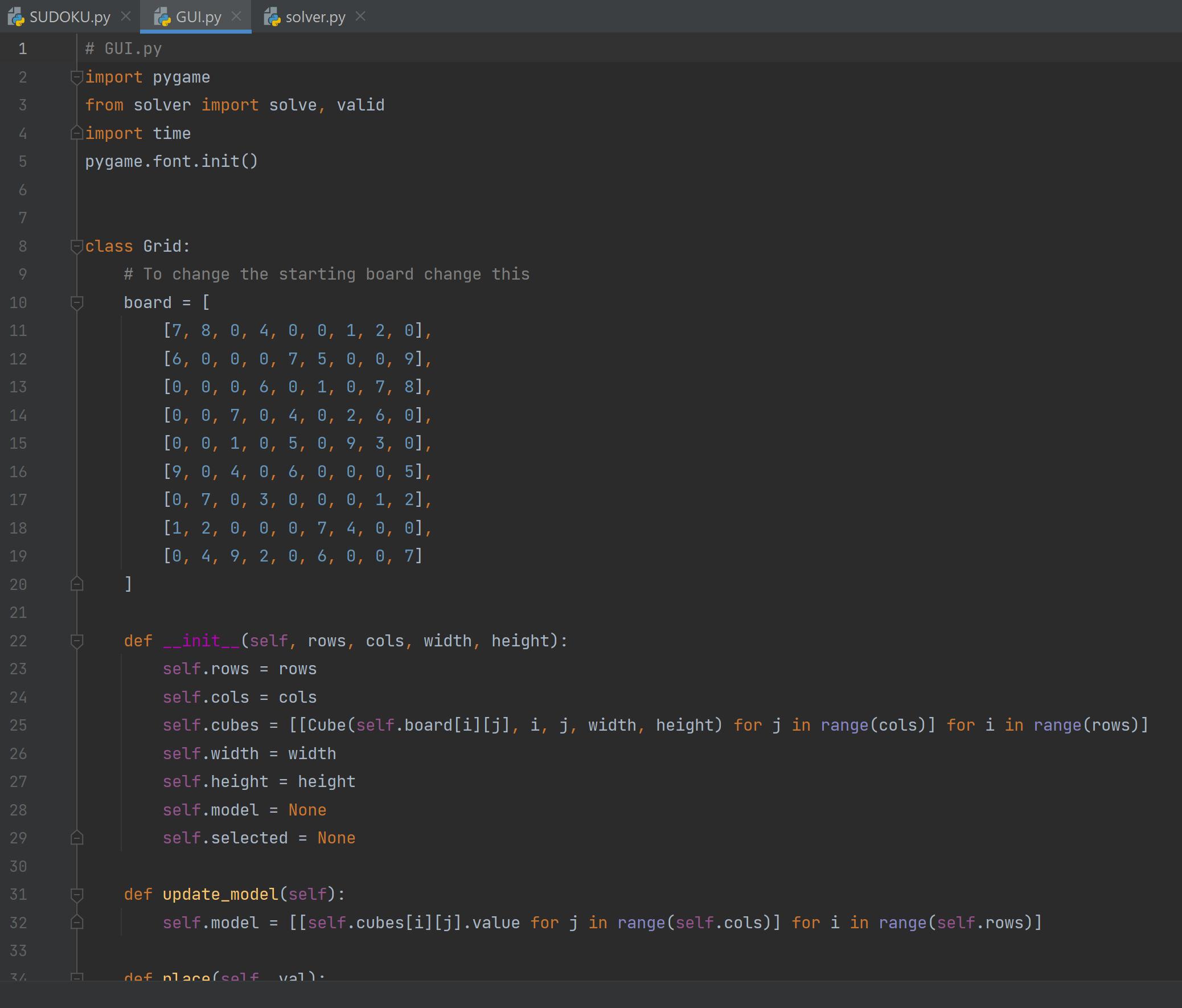
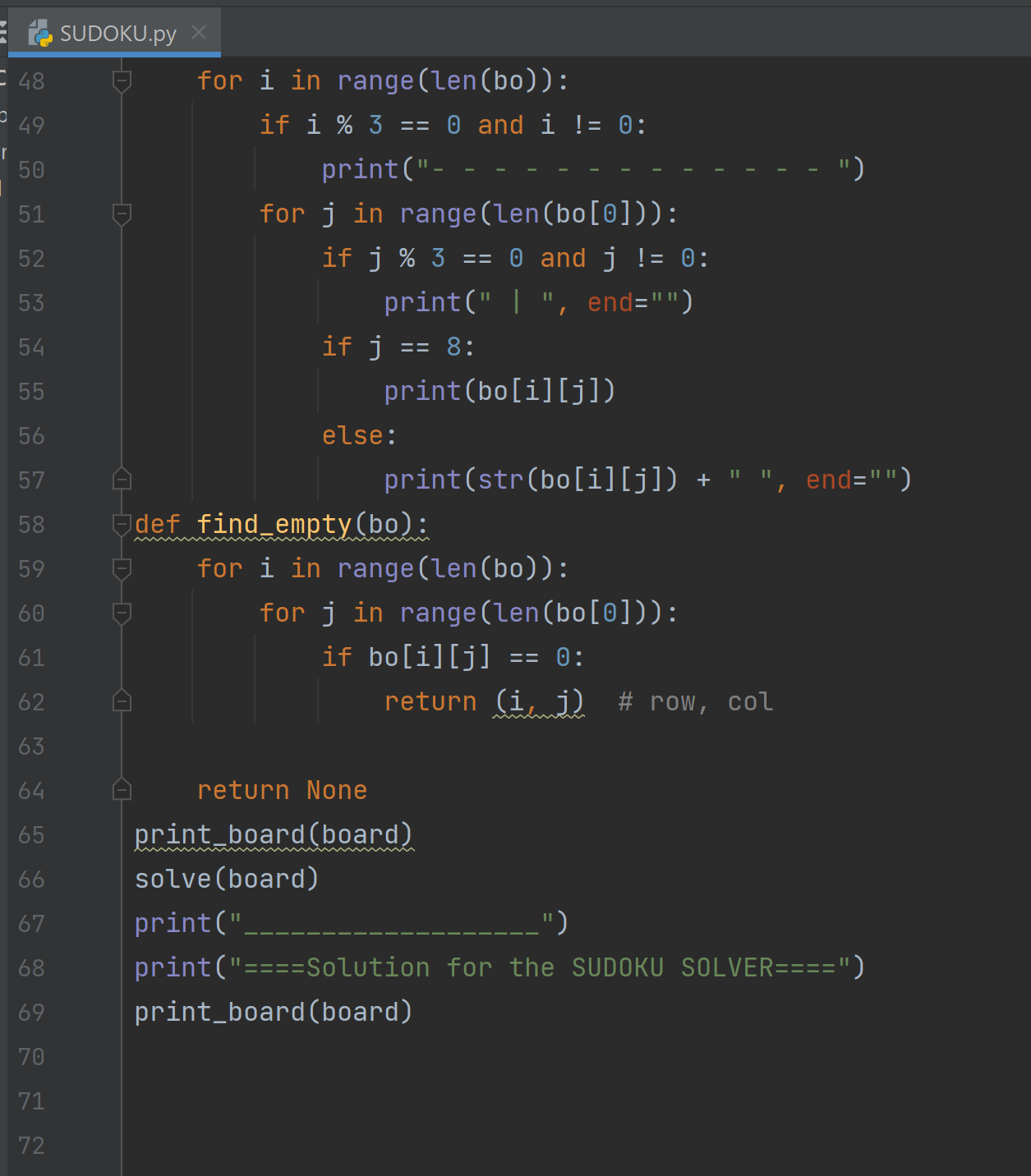
**IMPLEMENTATION**

**6.1 Flow chart of the proposed solution**

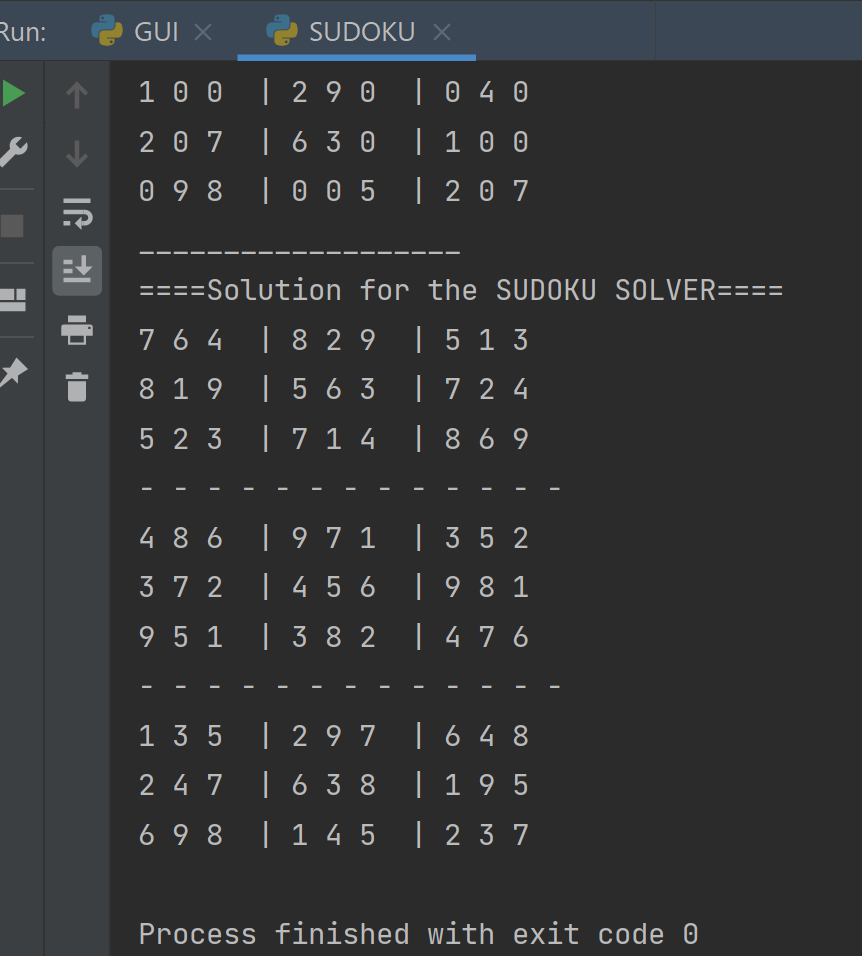


**6.2 CODE**

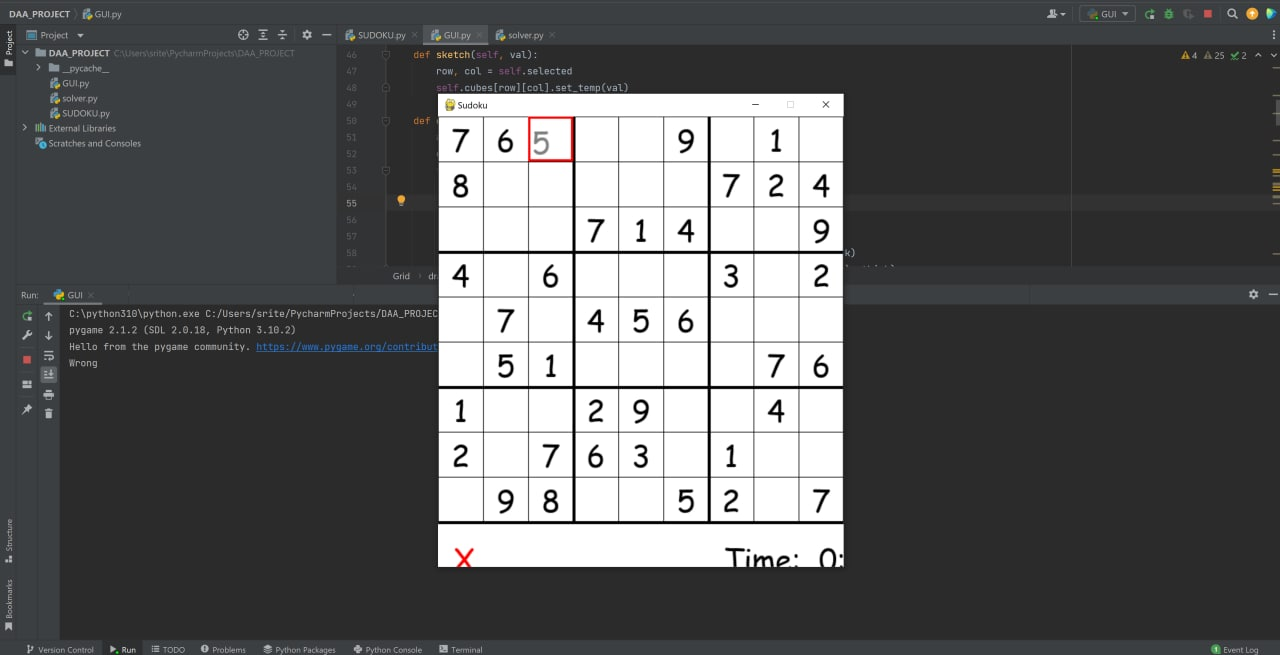
** **

****

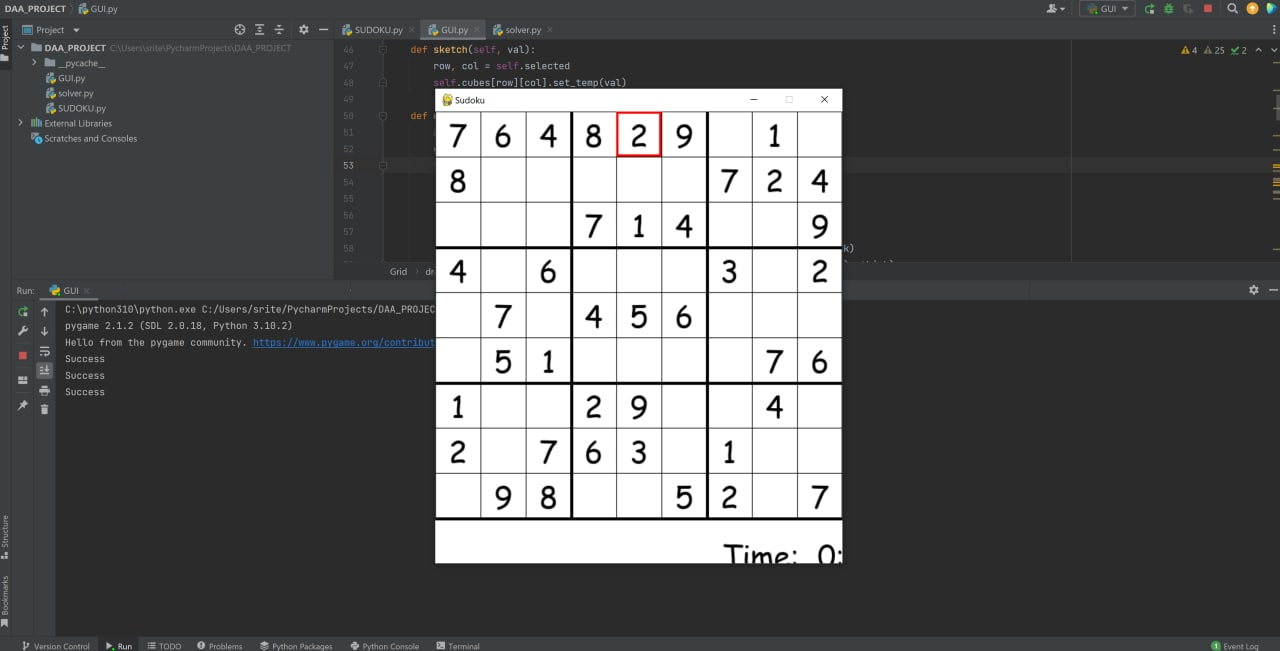
**Results**

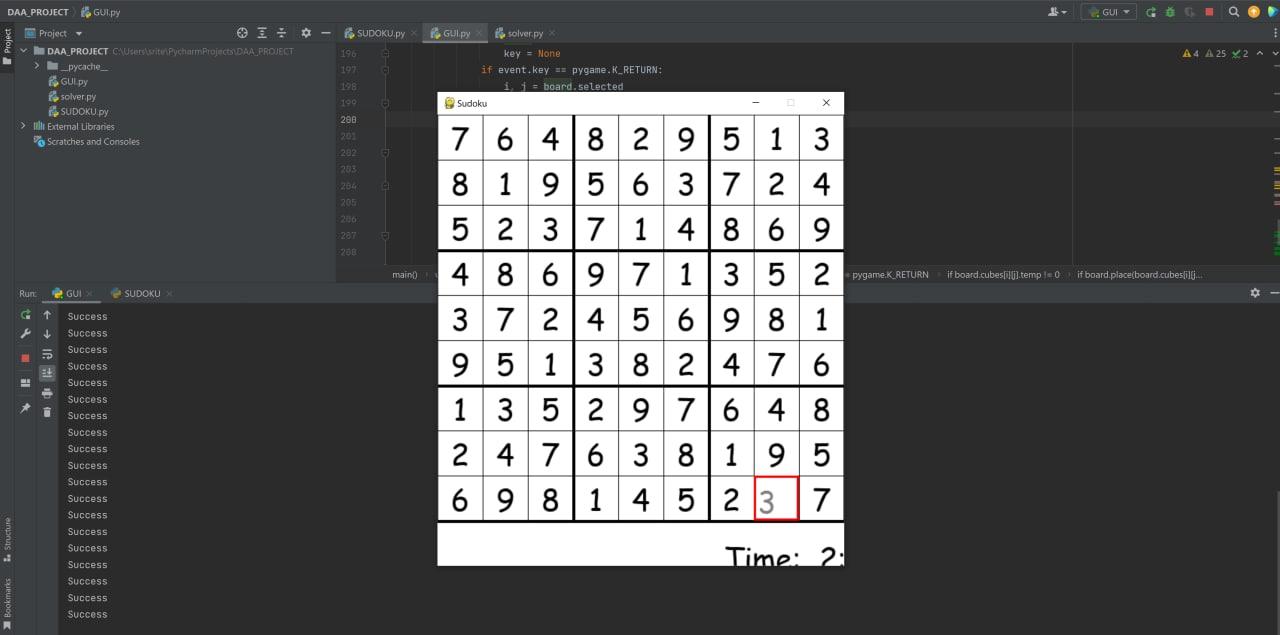


WRONG



SUCCESS





**CONCLUSION And Future Work**

* Our project mainly focuses on solving sudoku problems using Backtracking algorithm.
* In our first review, we’ve explored different algorithms used for solving sudoku.
* Based on our research we’ve decided to use a backtracking algorithm as it is more convenient to solve sudoku.
* Now , we have successfully implemented sudoku using console input.
* Further developments in this field should consider the uniqueness of solution in relation to the minimum number of input data necessary for this condition.
* Further work could explore more direct puzzle generation algorithm that

makes use of the unique nature of our difficulty metric. By applying the logical strategies we consider in reverse to selectively remove given from a completed grid, a puzzle that requires certain strategies to solve could be constructed, effectively tailoring the puzzle to conform to a desired difficulty. This could potentially generate puzzles of a desired difficulty more efficiently.

**REFERRENCES**

<https://www.geeksforgeeks.org/sudoku-backtracking-7/>

<https://en.wikipedia.org/wiki/Sudoku_solving_algorithms#:~:text=A%20Sudoku%20(top)%2eing%20solved,against%20the%20brute%20force%20algorithm>.

<https://dev.to/christinamcmahon/use-backtracking-algorithm-to-solve-sudoku-270>

<https://www.101computing.net/backtracking-algorithm-sudoku-solver/>