**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“Jnana Sangama”, Belagavi-560018, Karnataka**



**DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY**

**WITH MINI PROJECT-** **19IS4DLADA**

**REPORT**

**On**

**“SUDOKU SOLVER”**

**BACHELOR OF ENGINEERING**

In

**INFORMATION SCIENCE AND ENGINEERING**

Submitted by

**Anitha P (1DS19IS017)**

**Apoorva S V (1DS19IS020)**

**Kruthi R Aithal (1DS19IS052)**

**Manasa B (1DS19IS054)**

**LAB IN-CHARGE**

Mrs. Bindu Bhargavi S M

Mrs. Shilpa Shree S

Mrs. Sreevidya B S



**2020-21**

**Department of Information Science and Engineering**

**DAYANANDA SAGAR COLLEGE OF ENGINEERING**

**SHAVIGE MALLESHWARA HILLS, KUMARASWAMY LAYOUT**

**Bengaluru-560078**

**DAYANANDA SAGAR COLLEGE OF ENGINEERING**

**Shavige Malleshwara Hills, Kumaraswamy Layout-560078**

**Department of Information Science and Engineering**



**2020-21**

**CERTIFICATE**

This is to certify that Design and Analysis of Algorithms Laboratory with Mini Project Work (19IS4DLADA ) entitled **“ Sudoku Solver ”** is a bonafide work carried out by  **Anitha P (1DS19IS017), Apoorva S V (1DS19IS020), Kruthi R Aithal (1DS19IS052), Manasa B (1DS19IS054)** in the partial fulfillment for the 4th semester of **Bachelor of Engineering in Information Science and** **Engineering** of theVisvesvaraya Technological University, Belgaviduring the year 2020-21.

|  |  |  |
| --- | --- | --- |
|  |  | *Signature of Lab-Incharge* |
|  |  |  |

Name of the Examiners Signature with Date

1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

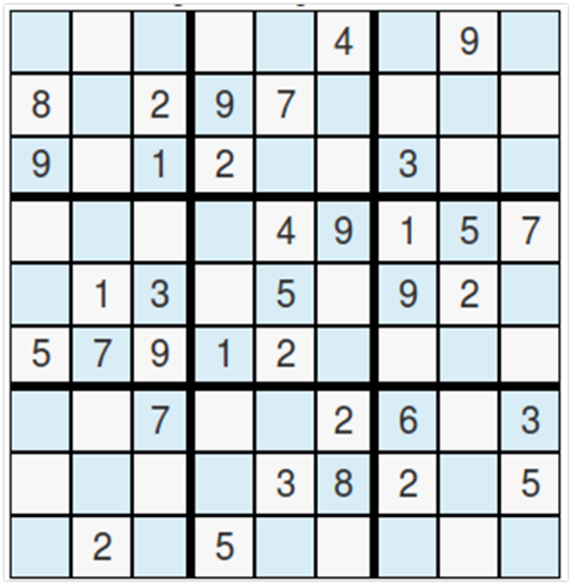
**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Topics** | **Page No.** |
|  | Introduction   * 1. What is a Sudoku?   2. Sudoku Solver   3. Brute Force Algorithm   4. Recursion   5. Backtracking Algorithm | 1-2 |
|  | System Requirements | 2 |
|  | Algorithm  3.1 Algorithm of Sudoku  3.2 Backtracking Algorithm for Sudoku | 3 |
|  | Implementation |  |
|  | 1. Pseudocode | 4-5 |
|  | 1. Snapshots of Results | 6-7 |
|  | Time Complexity | 7-8 |

**1. INTRODUCTION**

**1.1 What is a Sudoku?**

Sudoku, also known as Su Doku, is a very popular number game. Sudoku consists of a 9 × 9 grid with numbers appearing in some of the squares. The object of the puzzle is to fill the remaining squares, using all the numbers 1–9 exactly once in each row, column, and the nine 3 × 3 sub grids.



Row

3 x 3 Block

Column

Fig1.1: Simple sudoku grid

**1.2 Sudoku Solver**

In this project, we apply the brute force, backtracking and recursion algorithms to create a solution for a simple 3 x 3 x 9 sudoku (9 matrices of 3 x 3 dimension).

**1.3 Brute Force Algorithm**

Brute Force algorithm is a straightforward method of solving a problem that rely on sheer computing power and trying every possibility rather than advanced techniques to improve efficiency.

In this problem the Brute force approach tries out all the possible solutions and chooses the desired/best solutions.

**1.4 Recursion**

Recursive algorithm is a method of simplification that divides the problem into sub-problems of the same nature. The result of one recursion is the input for the next recursion. The algorithm calls itself with smaller input values and obtains the results by simply performing the operations on these smaller values.

**1.5 Backtracking Algorithm**

A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output.

Backtracking is used to solve problems in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criteria.

The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.

**2. SYSTEM REQUIREMENTS**

**Hardware requirement** :

Min RAM Size : 4 MB

Min Hard Drive Space : 25 MB

Min Processor Type : Intel 386 or higher

**Software Requirement**:

OS Required : Microsoft DOS, Microsoft Windows 3.1 or later, PC DOS

OS Family : Windows

**3. ALGORITHM**

**3.1 Algorithm of Sudoku**

STEP 1: Make a list of all the empty spots.

STEP 2: Select a spot and place a number, between 1 and 9, in it and validate the sub grids. Sub grids are the horizontal row, vertical column, and the 3x3 grid associated with that spot.

STEP 3: If any of the constraints fail, abandon that solution by backtracking to the previous state and repeat step 2 with the next number. Otherwise, check if the goal is reached.

STEP 4: If a solution is found, stop searching. Otherwise, repeat steps 2 to 4.

**3.2 Backtracking Algorithm for Sudoku**

Find the position i.e., row, column of an unassigned cell

if there is none

return true

Otherwise For digits from 1-9

if there is no conflict for a digit at a row, column and the 3 x 3 grid, then assign to

row, column and recursively try to fill in the rest of the grid.

if recursion is successful

return true

else

remove digit and try another

**4. IMPLEMENTATION**

**4.1 Pseudocode**

// *Checking If Sudoku Is Solved*

int isSolved (int prob[][3][3]) {

int flag = 1, b1, sum, b, i, j, sudo[3] = { 0, 3, 6 };

for (b1 = 0; b1 <= 2; b1++)

for (i = 0; i <= 2; i++) {

for (sum = 0, b = sudo[b1]; b <= sudo[b1] + 2; b++)

for (j = 0; j <= 2; sum += prob[b][i][j], j++)

;

// *if sum is 45*

sum != 45 ? flag = 0 : NULL;

}

return flag;

}

*// Backtracking Algorithm*

*//assigns values to all unsigned locations for Sudoku solution*

int solve (int prob [][3][3], int block) {

int i, j, row = 999, col = 999;

*//unassigned locations are marked as NULL*

*//if there is no unassigned location, we are done*

if (block == 9) {

isSolved(prob)? print(prob): NULL;

return 0;

}

for (i = 0; i <= 2; i++)

for (j = 0; j <= 2; j++)

if (prob[block][i][j] == 0) {

*//making tentative assignment if the location looks promising*

row = i;

col = j;

goto xx;

}

xx: if (row == 999 && col == 999) {

solve (prob, block + 1);

return 0;

}

else {

for (i = 1; i <= 9; i++)

if (canPut (prob, block, row, col, i)) {

prob[block][row][col] = i;

solve (prob, block);

prob[block][row][col] = 0;

}

}

return 0; *// this triggers backtracking*

}

**4.2 Snapshots of Results**

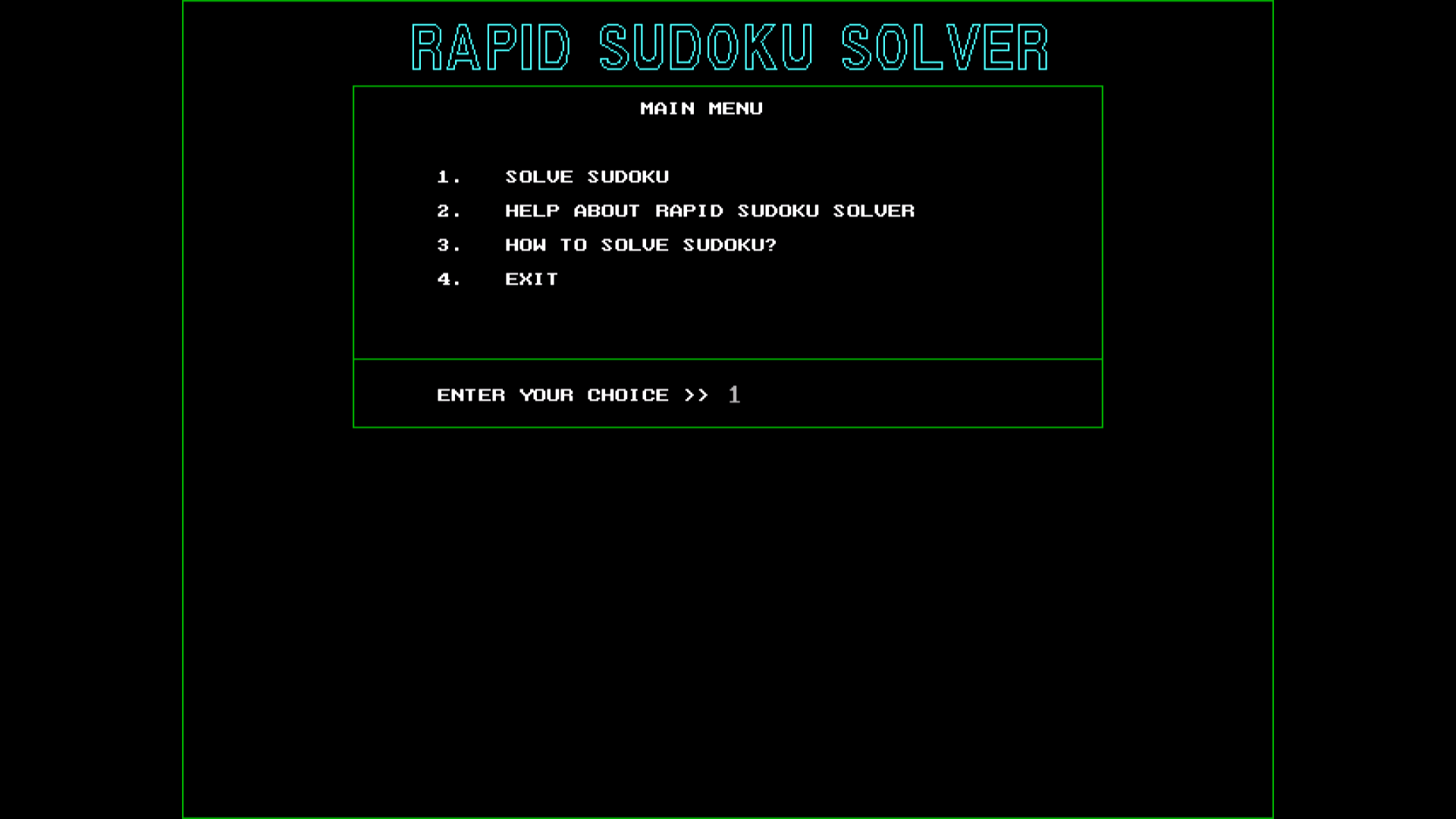


Fig 4.2 (a) : Main menu

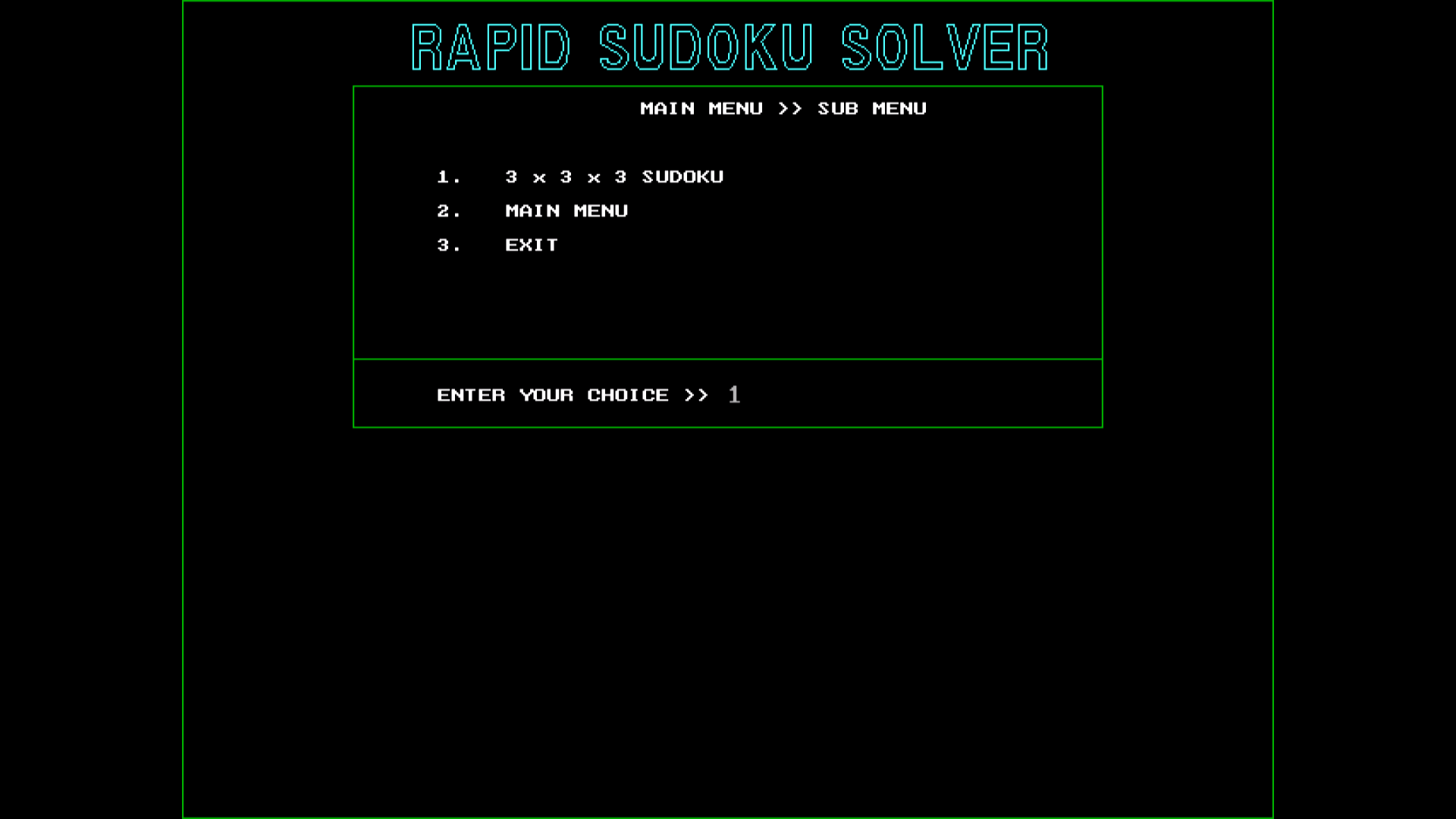


Fig 4.2 (b) : Sub menu after entering option 1 in main menu

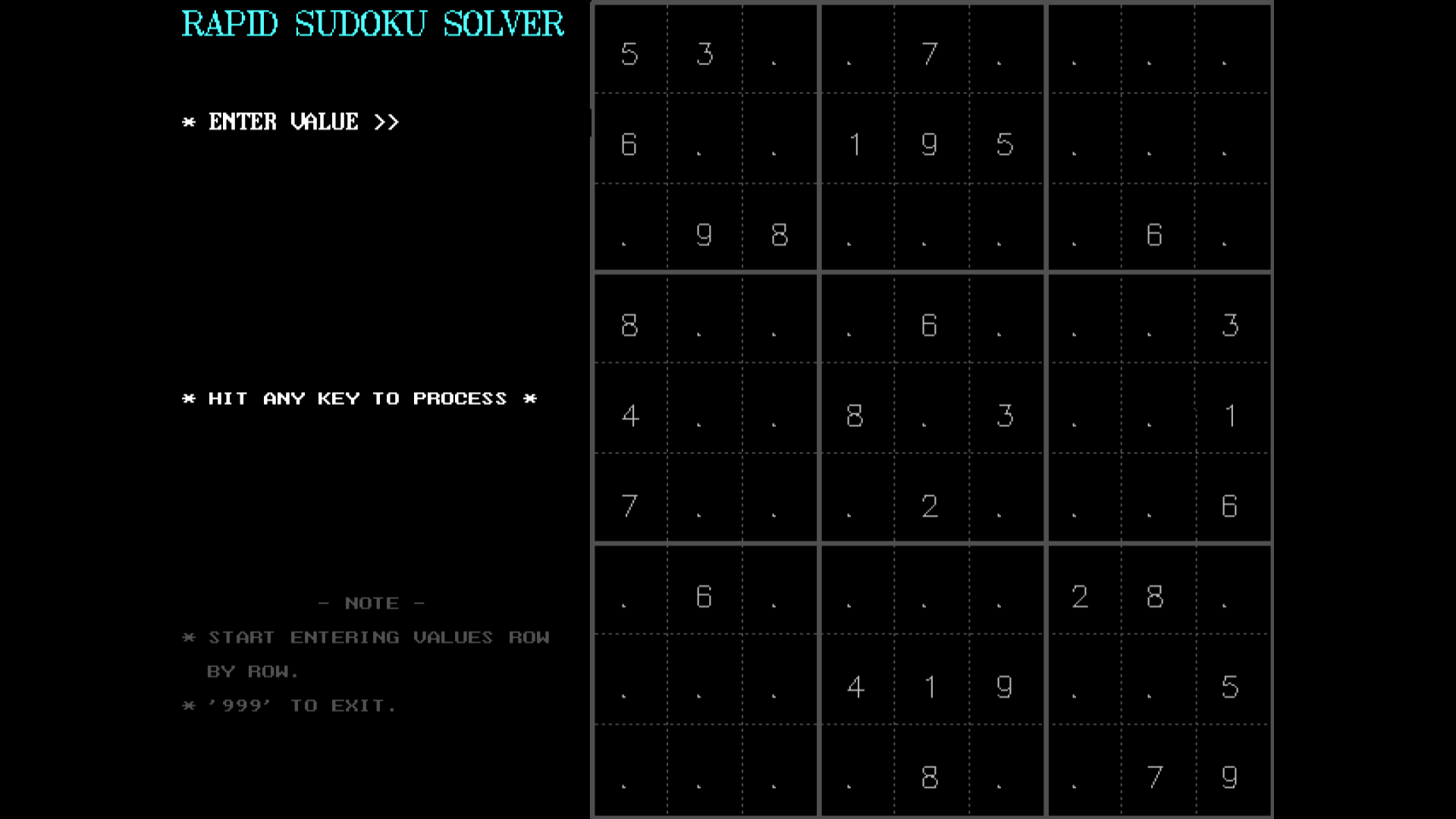


Fig 4.2 (c) : After entering the input matrix



Fig 4.2 (d) : Final output

**5.TIME COMPLEXITY**

**Time Complexity:**

O (n ^ m) where n is the number of possibilities for each square (i.e., 9 in classic Sudoku) and m is the number of spaces that are blank. The recurrence equation can be written as

T(M) = 9\*T(M-1) + O (1)

where T(N) is the running time of the solution for a problem size of N. Solving this recurrence will yield, O(9^M).

**Space complexity:**

It’s the recursion stack that is used as an auxiliary space which is N\*N step deep. Remember we need to fill in 81 cells in a 9\*9 sudoku and at each level, only one cell is filled. So, space complexity would be O(M).