VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Department of Computer Engineering

OPEN SOURCE TECH LAB (OSTL)

Mini Project Report on

SUDOKU (GAME & SOLVER)

Submitted in partial fulfilment of the requirements

of Second Year Computer Engineering

By

Kedar Kharde(D7B-29)

Kunal Kotkar(D7B-32)

Parthesh Pawar(D7B-47)

Supervisor:

Mrs. MANSI TALREJA

DEPARTMENT OF COMPUTER ENGINEERING

V.E.S INSTITUTE OF TECHNOLOGY

2019-20

TITLE : SUDOKU – GAME & SOLVER

AIM: Implementation of Sudoku game and solver using GUI

PROBLEM STATEMENT: Solving the Sudoku by attempting the sample questions or inputting the question of sudoku and get your answer solved.

REQUIREMENT OF PROJECT(MODULES OF PYTHON USED): Project is done on Python – IDE: **PYCHARM**. Inbuilt modules used are as follows:

a) tkinter - for creating GUI application and widgets.

b) numpy – for storing and displaying the 9x9 entries of sudoku block.

c) tkinter.font – for setting the font size of labels and entries.

d) copy – for creating a copy of list/array.

e) messagebox - to give a popup messages.

ABSTRACT: In this game, user has two choices :

1) Try some sample questions : Here the user can attempt 9 sudoku questions. 3 of easy level, 3 of medium level, 3 of hard level respectively. User can check(confirm) the correct answer.

2) Solver: Here, the user has to enter the proper question as input and will get the correct solved answer.

THEORY:

Python GUI – tkinter

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter app:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

tkinter module is imported using following statement:

```
import tkinter
```

OVERVIEW OF WORKING : User has two options: 1)**Solver** 2) **Try some Questions**

1) **Solver**: In this the user has to enter the entries(question). The entry fields of sudoku block are accessed by their respective objects which are stored in **global list: 'e[ ]'** . With the help of this list all the entries from user are copied into a **global multidimensional list : 'gridn[ ]'**. All 'gridn[ ]' elements are set initially to zero. User entries are then copied into 'gridn[ ]'. Further operations are performed on this 'gridn[ ]'. The inputs are checked, whether they are according to the sudoku rules. After checking everything, the answer is found and stored into 'gridn[ ]' . Again with the help of objects in list : 'e[ ]', the answers in 'gridn[ ]' are displayed in the sudoku block(screen/layout).

2) **Questions for User:** Here user can attempt 9 sudoku questions. 3 each of **easy, medium** and **hard** levels respectively. All this 9 questions are stored in 9 separate multidimensional lists. Also there is a global multidimensional **list: 'grid1[ ]'** . Initially, this list has all elements as zero. According to selection of question by user, the list belonging to that particular question is copied into the global list: 'grid1[ ]' . And the question from 'grid1[ ]' (i.e the elements of list) is displayed on sudoku (layout) with the help objects in **global list :'e [ ]'**. Now, if user wants to check the answer, he can click on 'solve' button that will find the answer for that question as per the procedures mentioned above in the 'Solver' part.

- All the widgets(except frames) in project are been arranged using "GRID" layout manager.
- Frames are packed using "PACK" layout manager.

DETAILED THEORY OF MODULES AND FUNCTIONS:

User Defined Modules in Project:

a) GUI.py module : This is the starting module, say module where the first window is created. The window is created along with initialization of font size. Basically, all **GUI part and the navigation part** and their functions are created here. At first p1() function is called. The module imports following things:

Buitlin : tkinter, tkinter.font.

From second.py : list e[] and function create()

From sudoku.py : inputa() function

From question.py : qdisplay() function

Functions in GUI module:

- **p1( )** : This is the first function to be called. This creates a frame, named as Sudoku.

  Other widgets:

  1 Label: Welcoming the Users

  2 Buttons: START and CLOSE

  After clicking the START button p3() function is called navigating to different frame. After clicking CLOSE dest() function is called, destroying the window.

- **p3(frame)** : This creates a frame, named as Sudoku, destroying the previous one.

  This frame contains MENU, where three options are given to user:

  SOLVER, TRY SOME QUESTIONS, BACK

  Other Widgets:

  1 Label: Displaying the MENU options.

  3 Buttons : SOLVER, TRY SOME QUESTIONS, BACK

  SOLVER will take user to the frame where the user has to input the question to be solved. p2() function is called.

  TRY SOME QUESTIONS will take the user to choose the sample questions and its difficulty level to be solved. p4() function is called.

  BACK will call the back() function that will navigate the user to p1() function.

- **p2(frame)** : This will destroy the previous frame, creating new one and adjusting the size of window. The function will call create() function from module : second.py

  This will create a 9x9 entry field of sudoku. User has to enter the question and click on SOLVE to get answer.

  Other widgets:

  1 Label: Information

  4 Buttons: SOLVE, RESET, MENU, EXIT

  SOLVE button will call the inputa() function from module: sudoku, this will return the answer of the inputted sudoku.

  RESET button will call the reset() function from GUI module, this will reset all the entry fields to Empty.

  MENU button will take the user to MENU field, by calling p3() function

  EXIT button will destroy the window by calling dest() function.

- **p4(frame)** : This will destroy the previous frame, creating new one and adjusting the size of window. This frame contains the difficulty options for the questions. There are 3 options: EASY MEDIUM & HARD. User has to choose any one of these and will be taken to choose the question.

  Other widgets:

  1 Label: Difficulty Level

  5 Buttons: EASY, MEDIUM, HARD, BACK, EXIT

EASY button will take user to the easy questions, by calling difficult() function. While calling difficulty() function, a signal variable '1' is passed, indicating user has choosen Easy level.

MEDIUM button will take user to the medium questions, by calling difficult() function. While calling difficulty() function, a signal variable '2' is passed, indicating user has choosen Medium level.

HARD button will take user to the hard questions, by calling difficult() function. While calling difficulty() function, a signal variable '3' is passed, indicating user has choosen Hard level.

BACK button will take user to MENU, calling p3() function.

EXIT button destroys the window, calling dest() function.

- **difficulty(frame, signal variable)** : This will destroy the previous frame, creating new one and adjusting the size of window. This frame contains Level chosen and their respective questions. User has to choose any of the three questions for solving.

  Other widgets:

  1 Label: Level Name (Based on signal variable from parameter)

  5 buttons : QUESTION 1,2,3 BACK & EXIT

  QUESTION 1,2,3 is used for selecting questions. On click event, qdisplay() function from module : question will be called. Parameters contains the difficulty() function, window, signal variable , question number.

  BACK button will call the p4() function taking user back to difficulty selection frame.

  EXIT will call dest() function, destroying the window.

- **reset(frame)** : This function will clear the list that contains the object of 81 Entry fields, then will call the p2(). Thus clearing all the previous data in grid and list.

- **back(frame) :** This function will destroy the parameter frame and will call the p1() function

- **dest(frame) :** This function is inbuilt function and is used to destroy the current window/frame/canvas.

b) **second.py module :** This module is for **creating the 9x9 sudoku layout**. It has a global list named **"e [ ] "** which stores the object of 81 Entry widgets that are created in sudoku layout. With the help of this list, the entry fields can be accessed. Grid layout manager is used. The module imports tkinter module.

Functions in second module:

- **create(frame) :** This function creates the sudoku grid(layout) in the frame which is been passed as parameter. Before that it clears the list. As the Entry field is created it's object is appended into list. Adjacent 3x3 grids have different background colours. Widgets: It has 81 Entry Fields.

c) **question.py module :** This module is responsible for generating the **GUI for sample questions**. As the user choses to solve the questions, following by choosing difficulty levels and question number, the corresponding question is displayed in sudoku grid.

All 9 questions, 3 of each 3 levels are stored in 9 different globally declared multidimensional lists(grid). It also has a global grid: 'grid1[ ]' with all entries as 0(null grid), this is used to find the answers for the question. The question grid is copied into this null grid and this null grid is sent to sudoku module where answer for the question is generated. Finally the answers generated in null grid are then displayed onto sudoku layout using Entry field objects stored in e[]. The module imports following things:

Buitlin : tkinter, copy

From second.py : list e[] and function create()

From sudoku.py : inputa() function

Functions in question module:

- **qdisplay(window, frame, signal variable, question number , difficulty funcion) :** It destroys the previous frame and creates a new one. Based on the paramters passed(signal variable, question number), the particular grid which contains the question will be copied to null grid. Then the function calls the create() function from second.py which creates a sudoku layout. It clears the sudoku grid first and enters the values at particular entries as per the chosen question. Thus the question is displayed to user in sudoku grid.

  Widgets: 1 Label - Info

  3 Buttons – SOLVE, BACK & EXIT

SOLVE button click, will call the inputa() function from sudoku.py module. This returns the answer for the question and is displayed on sudoku layout.

BACK button click, will take the user to question selection page, i.e the difficulty() function from GUI.py which is been passed while calling the qdisplay() function is called.

EXIT will destroy the frame and window, by calling go() function.

- **dest (frame) :** This will destroy the frame passed as parameter.
- **go (frame, window) :** This will destroy the frame and window passed as parameter.

d) **useless.py module:** This module contains the functions which are **responsible for checking whether the user has entered the inputs as per sudoku laws**. As soon as the out() function is called the check starts. For row checking rcheck() function is called, for column checking ccheck() function is called and for every block of 3x3 bcheck() function is called. If any of these three check functions return the value 1, then the out function will return the value -1, indicating that inputs are not entered correctly. If all check function returns 0, the out() function will return 0, indicating that inputs are correctly entered. Modules imported: copy.

Functions in useless.py module :

- **out(grid) :** The out function takes grid as parameter. The grid contains the input of user. The grid is then passed for row, column and block checking functions. The out function will return 0, if all these check function return 1, indicating correct inputs. Else out() will return -1, if any of these check functions return 1, out() will return -1, indicating wrong inputs.

- **finderror ( ) :** This is the most important function of this module. Every check function will call this function. The function uses the global list named dup[].

  For every row check, the elements of row are copied into this global list and the finderror() is called. The finderror() checks whether any number in the list apart from 0(default input in grid) is repeated more than once or not. If yes, then finderror() will return 1, indicating input error. If No, then function will return 0, indicating the list has no duplicates, thus inputs for a particular row is correct.

  The same happens for checking of every column and every 3x3 block.

- **rcheck (grid) :** This function checks all the 9 rows. Each row is checked separately. Each row inputs are added to the global list 'dup[]' and passed for checking duplicates to finderror(). If return value is 1, rcheck() will return 1 to out() function, indicating input error. If return value is 0, execution continues for next checking.

- **ccheck (grid) :** This function checks all the 9 columns. Each column is checked separately. Each column inputs are added to the global list 'dup[]' and passed for checking duplicates to finderror(). If return value is 1, ccheck() will return 1 to out() function, indicating input error. If return value is 0, execution continues for next checking.

- **bcheck (grid) :** This function checks all the 9 blocks of 3x3 order. Each block is checked separately. Each block inputs are added to the global list 'dup[]' and passed for checking duplicates to finderror(). If return value is 1, bcheck() will return 1 to out() function, indicating input error. If return value is 0, execution continues for next checking.

e) **sudoku.py module :** This is the **main module of this project**. It takes all the inputs from the sudoku block and copies it into a global multidimensional-list named: gridn[]. The inputs in grid are by-default set to 0. According to user input, changes are made into 'gridn'. This 'gridn' is first check for correction of input values, which is done by passing the 'gridn' list to out() function in useless.py module. After checking, solve() function is called which **finds the answer for sudoku** using possible() function. After getting a solution, it is stored in that global list('gridn'). Using the output() function and the global list 'e[]' which stores the objects for Entry fields of sudoku layout, all the numbers from the 'gridn' are displayed to screen(sudoku layout).

Module imports: tkinter, global list e[] from second.py, out() from useless.py, messagebox, numpy, copy.

Functions in sudoku.py module:

- **inputa (frame) :** This function firstly, reset the list 'gridn' to default by setting all elements as zero, by calling setintial(). Then using the objects of Entry field which are stored in global list 'e[]', the inputs from user are entered into 'gridn' accordingly. This also includes the check on what kind of inputs user has given. E.g. if user has entered an alphabet or any number less than 1 and greater than 9, then the error message is displayed. After this the out() function is called for checking whether the inputs are entered according to rules of sudoku. If yes, then solve function is called, else the error message is displayed.

- **solve ( ) :** The "solve" function solve the given Sudoku problem. It iterates over entire Sudoku matrix(i.e the global list 'gridn[ ]') and finds perfect value to be inserted at blank position which is represented by value 0.

❖ The two for loops iterates one row at a time, and checks blank position in that row.

❖ If blank position is found which has value 0, it finds value from 1 to 9 which is possible to enter.

❖ After entering the value the "solve" function does a recursive call to itself, and solves for next blank position saving the previous result. If that value is giving correct result for next blank positions it will continue to execute. If the recursive calling is giving dead end for result i.e. it is not possible to enter any digit for next blank position. If that is the case then we backtrack and again set value for that position to 0.

❖ **BACKTRACKING ALGORITHM** : Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time. Backtracking algorithms rely on the use of a **recursive function**. A recursive function is a function that calls itself until a condition is met.

❖ We traverse each position of the gridn[ ], check which digit can be placed in the blank position , repeat till we have a solution. If a value(digit) cannot be placed in the current blank position, we go to the previous value from which we started and try with placing another value in that blank position.

❖ This basically means, whenever we arrive at a dead-end, we go back to the previous choice we made and change our choice.

- **possible (y,x,n) :** The "possible" function takes 3 parameters as input 'y','x','n' , y represents row number, x represents column number of empty position which have value 0 and n is value to be inserted at position(y, x).The function returns true if value is possible to enter and false otherwise.

❖ **CHECKING IN SAME ROW:** First for loop checks whether value to be inserted is present in given row; if value is present in a row then we return false.

❖ **CHECKING IN SAME COLUMN:** Second for loop checks whether value to be inserted is present in given column; if that is the case it returns false.

❖ **CHECKING IN SMALL BOXES:** Finally we check if value is present in small boxes with respect to that position, if that is the case we returns false.

If none of the above conditions satisfy then function returns true that means it is possible to enter the value

- **setintial ( ) :** This function sets the value of all elements in global grid 'gridn' to zero.
- **output ( ) :** This function uses the Entry field objects from global list 'e[]' to display the correct answers into sudoku layout(screen). The solution stored in 'gridn' list is displayed in sudoku layout.

CODE:

GUI.py MODULE: run this file → python GUI.py

```
import tkinter
import tkinter as tk
from tkinter import *
import tkinter.font as font
from second import create
from second import e
from sudoku import inputa
import question
from question import qdisplay

def dest(f): #TO DESTROY FRAME
    f.destroy()

def p1():  #STRATING FRAME/SCREEN

    window.geometry('770x570')

    frame1 = LabelFrame(window, text="Sudoku",
padx=150, pady=150, fg="green")

    l1 = Label(frame1, text="WELCOME TO SUDOKU
SOLVER!!!", fg="red", padx=40, pady=40,
font="Arial", bg="yellow")
    l1.grid(row=0, column=0, padx=10, pady=10)
    l1['font'] = myFont

    b11 = Button(frame1,
text="START",command=lambda: p3(frame1),
bg="green", fg="pink")
    b11.grid(row=1, column=0)
    b11['font'] = myFont
    b12 = Button(frame1,
text="CLOSE",command=lambda: dest(window),
bg="orange", fg="brown")
    b12.grid(row=2, column=0, padx=15, pady=5)
    b12['font'] = myFont

    frame1.pack(padx=10, pady=10)

def p3(previousf):  # MENU OPTION FRAME
    dest(previousf)
    window.geometry('570x580')
    frame3 = LabelFrame(window, text="Sudoku",
padx=100, pady=80, fg="green")
    frame3.grid(padx=30, pady=10)

    l31 = Label(frame3, text="***  MENU  ***",
fg="red", padx=60, pady=20, font="Arial", bg="deep
sky blue")
    l31.grid(row=2, column=0, rowspan=2,
columnspan=55, padx=20, pady=30)
    l31['font'] = myFont

    b31 = Button(frame3, text="SOLVER",
command=lambda: p2(frame3), bg="SpringGreen2",
fg="purple3")
    b31.grid(padx=10, pady=20, row=25, column=3,
columnspan=4)
    b31['font'] = myFont

    b32 = Button(frame3, text="TRY SOME
QUESTIONS",command=lambda: p4(frame3),
bg="SpringGreen2", fg="purple3",padx=25)
    b32.grid(padx=10, pady=20, row=26, column=3,
columnspan=4)
    b32['font'] = myFont

    b33 = Button(frame3, text="BACK",
command=lambda: back(frame3), bg="SpringGreen2",
fg="black")
    b33.grid(padx=10, pady=20, row=27, column=3,
columnspan=4)
    b33['font'] = myFont

def p2(previousf): # SOLVER FRAME

    dest(previousf)
    window.geometry('750x800')
    frame2 = LabelFrame(window,text="Sudoku",
padx=100, pady=80, fg="green")
    frame2.grid(padx=30, pady=10)

    create(frame2)

    l21 = Label(frame2, text="Enter you inputs and then
click solve",fg="red")
    l21.grid(row=2,column=0,
rowspan=2,columnspan=55,padx=20,pady=30)
```

```
    l21['font'] = myFont

    b21 = Button(frame2,
text="SOLVE",command=lambda: inputa(frame2),
bg="yellow", fg="purple")

b21.grid(padx=10,pady=20,row=25,column=3,columns
pan=4)
    b21['font'] = myFont

    b22 = Button(frame2, text="RESET",
command=lambda: reset(frame2), bg="khaki1",
fg="black")
    b22.grid(padx=10, pady=20, row=25,
column=8,columnspan=4)
    b22['font'] = myFont

    b23 = Button(frame2, text="MENU",
command=lambda: p3(frame2), bg="alice blue",
fg="blue2")
    b23.grid(padx=10, pady=20, row=26, column=3,
columnspan=4)
    b23['font'] = myFont

    b24 = Button(frame2, text="EXIT",
command=lambda: dest(window), bg="hot pink",
fg="red")
    b24.grid(padx=10, pady=20, row=26, column=8,
columnspan=4)
    b24['font'] = myFont

    frame2.pack(padx=10, pady=10)

def p4(previousf): # QUESTION DIFFICULTY
CHOICE FRAME
    dest(previousf)

    window.geometry('600x560')
    frame4 = LabelFrame(window, text="Sudoku",
padx=100, pady=30, fg="green")
    frame4.grid(padx=30, pady=10)

    l41 = Label(frame4, text="DIFFICULTY LEVEL",
fg="gray9",padx=60, pady=20, font="Arial", bg="green
yellow")
    l41.grid(row=2, column=4, rowspan=2,
columnspan=55, padx=20, pady=30)
    l41['font'] = myFont

    b41 = Button(frame4, text="EASY",
command=lambda: difficulty(frame4, 1), bg="peach
puff", fg="magenta2",width=20)
    b41.grid(padx=10, pady=20, row=25, column=14,
columnspan=10)
    b41['font'] = myFont

    b42 = Button(frame4,
text="MEDIUM",command=lambda: difficulty(frame4,
2), bg="AntiqueWhite3", fg="red3",width=20)
    b42.grid(padx=10, pady=20, row=26, column=14,
columnspan=10)
    b42['font'] = myFont

    b43 = Button(frame4,
text="HARD",command=lambda: difficulty(frame4, 3),
bg="cornflower blue", fg="black", width=20)
    b43.grid(padx=10, pady=20, row=27, column=14,
columnspan=10)
    b43['font'] = myFont

    b44 = Button(frame4, text="BACK",
command=lambda: p3(frame4), bg="DarkOrchid1",
fg="cyan", width=12)
    b44.grid(padx=10, pady=20, row=28, column=8,
columnspan=10)
    b44['font'] = myFont

    b45 = Button(frame4, text="EXIT",
command=lambda: dest(window), bg="DarkOrchid1",
fg="red2", width=12)
    b45.grid(padx=10, pady=20, row=28, column=22,
columnspan=10)
    b45['font'] = myFont

def difficulty(frame,v): # QUESTION CHOICE
FRAME
    dest(frame)

    window.geometry('570x745')
    frame5 = LabelFrame(window, text="Sudoku",
padx=100, pady=80, fg="green")
    frame5.grid(padx=30, pady=10)

    if v==1:
        le1 = Label(frame5, text="LEVEL : EASY",
fg="blue", padx=60, pady=20, font="Arial",
bg="yellow")
        le1.grid(row=0, column=6, rowspan=2,
columnspan=55, padx=20, pady=30)
        le1['font'] = myFont
    elif v==2:
        le1 = Label(frame5, text="LEVEL : MEDIUM",
fg="green", padx=60, pady=20, font="Arial",
bg="yellow")
        le1.grid(row=0, column=6, rowspan=2,
columnspan=55, padx=20, pady=30)
        le1['font'] = myFont
    elif v==3:
        le1 = Label(frame5, text="LEVEL : HARD",
fg="red", padx=60, pady=20, font="Arial",
bg="yellow")
        le1.grid(row=0, column=6, rowspan=2,
columnspan=55, padx=20, pady=30)
        le1['font'] = myFont

    le2 = Label(frame5, text="SELECT THE
QUESTION", fg="orange")
    le2.grid(row=3, column=6, rowspan=2,
columnspan=55, padx=20, pady=30)
    le2['font'] = myFont
```

```
    be1 = Button(frame5, text="QUESTION 1",
command=lambda: qdisplay(window, frame5, v, 1,
difficulty), bg="cyan", fg="deep pink", width=25)
    be1.grid(padx=10, pady=20, row=25, column=11,
columnspan=10)
    be1['font'] = myFont

    be2 = Button(frame5, text="QUESTION 2",
command=lambda: qdisplay(window, frame5, v, 2,
difficulty), bg="deep pink", fg="yellow", width=25)
    be2.grid(padx=10, pady=20, row=26, column=11,
columnspan=10)
    be2['font'] = myFont

    be3 = Button(frame5, text="QUESTION 3",
command=lambda: qdisplay(window, frame5, v, 3,
difficulty), bg="OliveDrab1", fg="blue2", width=25)
    be3.grid(padx=10, pady=20, row=27, column=11,
columnspan=10)
    be3['font'] = myFont

    be4 = Button(frame5, text="BACK",
command=lambda: p4(frame5), bg="orchid1",
fg="black", width=10)
    be4.grid(padx=10, pady=20, row=28, column=5,
columnspan=10)
    be4['font'] = myFont

    be5 = Button(frame5, text="EXIT",
command=lambda: dest(window), bg="orchid1",
fg="red", width=10)
    be5.grid(padx=10, pady=20, row=28, column=18,
columnspan=10)
    be5['font'] = myFont

def reset(frame): #RESET THE SUDOKU ENTRIES
    global e
    e.clear()
    p2(frame)

def back(frame): #BACK TO FIRST SCREEN OF
WINDOW
    dest(frame)
    p1()

window= Tk()
window.geometry('770x570')
window.title("Sudoku")
myFont = font.Font(size=15)

p1()

window.mainloop()
```

second.py MODULE:

```
import tkinter
from tkinter import *
e = [] # STORES THE OBJECTS OF 81 ENTRY
FIELDS OF LAYOUT

def create(frame): #CREATES THE SUDOKU 9X9
LAYOUT
    global e
    e.clear()
    for r in range(10, 19):
        for c in range(3, 12):
            if r in range(10, 13) and c in range(6, 9):
                l = Entry(frame, width='3', font=('Times New
Roman', 25), bg="gray78",justify='center')
                l.grid(row=r, column=c)
                e.append(l)
            elif r in range(13, 16) and c in range(3, 6):
                l = Entry(frame, width='3', font=('Times New
Roman', 25), bg="gray78",justify='center')
                l.grid(row=r, column=c)
                e.append(l)
            elif r in range(13, 16) and c in range(9, 12):
                l = Entry(frame, width='3', font=('Times New
Roman', 25), bg="gray78",justify='center')
                l.grid(row=r, column=c)
                e.append(l)
            elif r in range(16, 19) and c in range(6, 9):
                l = Entry(frame, width='3', font=('Times New
Roman', 25), bg="gray78",justify='center')
                l.grid(row=r, column=c)
                e.append(l)
            else:
                l = Entry(frame, width='3', font=('Times New
Roman', 25), borderwidth='2', bg='white
smoke',justify='center')
                l.grid(row=r,column=c)
                e.append(l)
```

 # question.py MODLUE :

```
import tkinter
from tkinter import *
from sudoku import inputa
from second import e,create
import copy

#EASY QUESTIONS
gride1 =  [[6, 0, 0, 1, 0, 0, 0, 0, 2],
           [8, 0, 1, 0, 9, 0, 0, 0, 0],
           [0, 7, 5, 0, 8, 4, 0, 0, 0],
           [4, 3, 0, 0, 2, 0, 5, 6, 1],
           [5, 1, 8, 7, 0, 0, 4, 0, 9],
           [0, 9, 6, 4, 1, 0, 3, 0, 0],
           [0, 0, 0, 0, 7, 0, 0, 0, 0],
           [0, 6, 0, 0, 3, 1, 0, 5, 0],
           [7, 0, 2, 5, 4, 0, 6, 0, 3]]

gride2 =  [[0, 0, 1, 9, 8, 4, 7, 6, 0],
           [6, 0, 0, 0, 5, 7, 0, 0, 0],
           [8, 0, 7, 0, 1, 0, 0, 0, 0],
           [9, 6, 0, 3, 0, 8, 1, 0, 5],
           [1, 8, 5, 0, 2, 0, 0, 7, 3],
           [3, 0, 0, 0, 0, 0, 2, 0, 8],
           [2, 1, 0, 0, 0, 0, 0, 3, 6],
           [0, 0, 0, 1, 0, 0, 0, 0, 4],
           [0, 9, 6, 0, 0, 2, 5, 1, 0]]

gride3 =  [[0, 0, 0, 6, 0, 0, 1, 0, 7],
           [6, 8, 0, 9, 5, 1, 3, 0, 0],
           [0, 0, 3, 0, 0, 2, 5, 6, 8],
           [0, 4, 0, 8, 1, 0, 0, 2, 0],
           [0, 0, 0, 0, 0, 0, 8, 5, 0],
           [0, 9, 0, 0, 6, 5, 0, 7, 3],
           [4, 0, 9, 0, 0, 3, 0, 8, 5],
           [1, 6, 2, 0, 0, 9, 0, 3, 0],
           [5, 0, 0, 7, 0, 6, 0, 0, 0]]

#MEDIUM QUESTIONS
gridm1=  [[0, 0, 0, 4, 0, 0, 2, 0, 0],
          [0, 0, 2, 0, 0, 0, 0, 1, 8],
          [5, 0, 6, 9, 0, 0, 0, 3, 0],
          [0, 6, 9, 0, 0, 0, 3, 0, 0],
          [0, 5, 0, 0, 0, 0, 0, 2, 1],
          [8, 0, 0, 1, 5, 7, 6, 0, 9],
          [0, 0, 0, 0, 3, 0, 9, 6, 0],
          [9, 0, 0, 6, 0, 2, 0, 5, 0],
          [0, 0, 0, 0, 0, 0, 7, 0, 2]]

gridm2=  [[0, 0, 0, 0, 0, 0, 6, 0, 9],
          [1, 0, 0, 0, 0, 4, 0, 0, 0],
          [0, 0, 5, 3, 0, 6, 8, 2, 1],
          [0, 0, 4, 6, 7, 0, 0, 5, 0],
          [0, 0, 7, 0, 0, 0, 9, 0, 0],
          [0, 0, 0, 5, 4, 0, 0, 0, 0],
          [3, 7, 0, 4, 0, 5, 2, 0, 6],
          [0, 0, 0, 0, 0, 0, 5, 1, 0],
          [0, 6, 0, 0, 2, 0, 0, 3, 7]]

gridm3=  [[2, 5, 0, 0, 0, 3, 0, 9, 1],
          [3, 0, 9, 0, 0, 0, 7, 2, 0],
          [0, 0, 1, 0, 0, 6, 3, 0, 0],
          [0, 0, 0, 0, 6, 8, 0, 0, 3],
          [0, 1, 0, 0, 4, 0, 0, 0, 0],
          [6, 0, 3, 0, 0, 0, 0, 5, 0],
          [1, 3, 2, 0, 0, 0, 0, 7, 0],
          [0, 0, 0, 0, 0, 4, 0, 6, 0],
          [7, 6, 4, 0, 1, 0, 0, 0, 0]]

#HARD QUESTIONS
gridh1=  [[5, 8, 6, 0, 7, 0, 0, 0, 0],
          [0, 0, 0, 9, 0, 1, 6, 0, 0],
          [0, 0, 0, 6, 0, 0, 0, 0, 0],
          [0, 0, 7, 0, 0, 0, 0, 0, 0],
          [9, 0, 2, 0, 1, 0, 3, 0, 5],
          [0, 0, 5, 0, 9, 0, 0, 0, 0],
          [0, 9, 0, 0, 4, 0, 0, 0, 8],
          [0, 0, 3, 5, 0, 0, 0, 6, 0],
          [0, 0, 0, 0, 2, 0, 4, 7, 0]]

gridh2=  [[0, 0, 7, 0, 0, 0, 3, 0, 2],
          [2, 0, 0, 0, 0, 5, 0, 1, 0],
          [0, 0, 0, 8, 0, 1, 4, 0, 0],
          [0, 1, 0, 0, 9, 6, 0, 0, 8],
          [7, 6, 0, 0, 0, 0, 0, 4, 9],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 3, 0, 0, 0],
          [8, 0, 1, 0, 6, 0, 0, 0, 0],
          [0, 0, 0, 7, 0, 0, 0, 6, 3]]

gridh3=  [[0, 0, 4, 8, 6, 0, 0, 3, 0],
          [0, 0, 1, 0, 0, 0, 0, 9, 0],
          [8, 0, 0, 0, 0, 9, 0, 6, 0],
          [5, 0, 0, 2, 0, 6, 0, 0, 1],
          [0, 2, 7, 0, 0, 1, 0, 0, 0],
          [0, 0, 0, 0, 4, 3, 0, 0, 6],
          [0, 5, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 9, 0, 0, 0, 4, 0, 0],
          [0, 0, 0, 4, 0, 0, 0, 1, 5]]

#GLOBAL GRID TO PUT THE PARTICULAR
QUESTION ONTO LAYOUT(SCREEN)
grid1 = [[0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 0, 0, 0, 0]]


def dest(f): # DESTROYS FRAME
   f.destroy()
```

```
def go(frame,window): # DESTROYS WINDOW, FRAME
    dest(frame)
    window.destroy()

def qdisplay(window, previousf, v, q, difficulty): # DISPLAYS THE CHOSEN QUESTION IN SUDOKU LAYOUT
    dest(previousf)

    global grid1, gride1, gride2, gride3, gridm1, gridm2, gridm3, gridh1, gridh2, gridh3

    if v==1 and q==1:
        grid1 = copy.deepcopy(gride1)
    elif v==1 and q==2:
        grid1= copy.deepcopy(gride2)
    elif v==1 and q==3:
        grid1= copy.deepcopy(gride3)
    elif v==2 and q==1:
        grid1= copy.deepcopy(gridm1)
    elif v==2 and q==2:
        grid1= copy.deepcopy(gridm2)
    elif v==2 and q==3:
        grid1= copy.deepcopy(gridm3)
    elif v==3 and q==1:
        grid1= copy.deepcopy(gridh1)
    elif v==3 and q==2:
        grid1= copy.deepcopy(gridh2)
    elif v==3 and q==3:
        grid1= copy.deepcopy(gridh3)

    window.geometry('750x770')
    frameq = LabelFrame(window, text="Sudoku", padx=100, pady=80, fg="green")
    frameq.grid(padx=30, pady=10)
    create(frameq)
    k=0
    for i in range (0,9):
        for j in range (0,9):
            e[k].delete(0,END)
            if grid1[i][j]!=0:
                e[k].insert(INSERT,str(grid1[i][j]))
            k += 1

    lque = Label(frameq, text="To check the right answer click Solve!!", fg="red")
    lque.grid(row=2, column=0, rowspan=2, columnspan=55, padx=20, pady=30)
    lque['font'] = 15

    bque= Button(frameq, text="SOLVE", command=lambda: inputa(frameq), bg="yellow", fg="black",width=7)
    bque.grid(padx=10, pady=20, row=25, column=6, columnspan=3)
    bque['font'] = 15

    bque = Button(frameq, text="BACK", command=lambda: difficulty(frameq, v), bg="yellow", fg="black",width=7)
    bque.grid(padx=10, pady=20, row=25, column=3, columnspan=3)
    bque['font'] = 15

    bque = Button(frameq, text="EXIT", command=lambda: go(frameq,window), bg="yellow", fg="red",width=7)
    bque.grid(padx=10, pady=20, row=25, column=9, columnspan=3)
    bque['font'] = 15

    frameq.pack(padx=10, pady=10)
```

useless.py MODULE:

```
import copy

dup = []

def out(grid):
    r= rcheck(grid)
    if r==1:
        return -1
    r=ccheck(grid)
    if r==1:
        return -1
    r=bcheck(grid)
    if r==1:
        return -1

    return 0

def finderror(): #CHECKS DUPLICITY IN THE LIST : dup
    global dup

    for i in dup:
        if i==0:
            pass
        else:
            if dup.count(i)>1:
                return 1

    return 0

#ROW CHECK
```

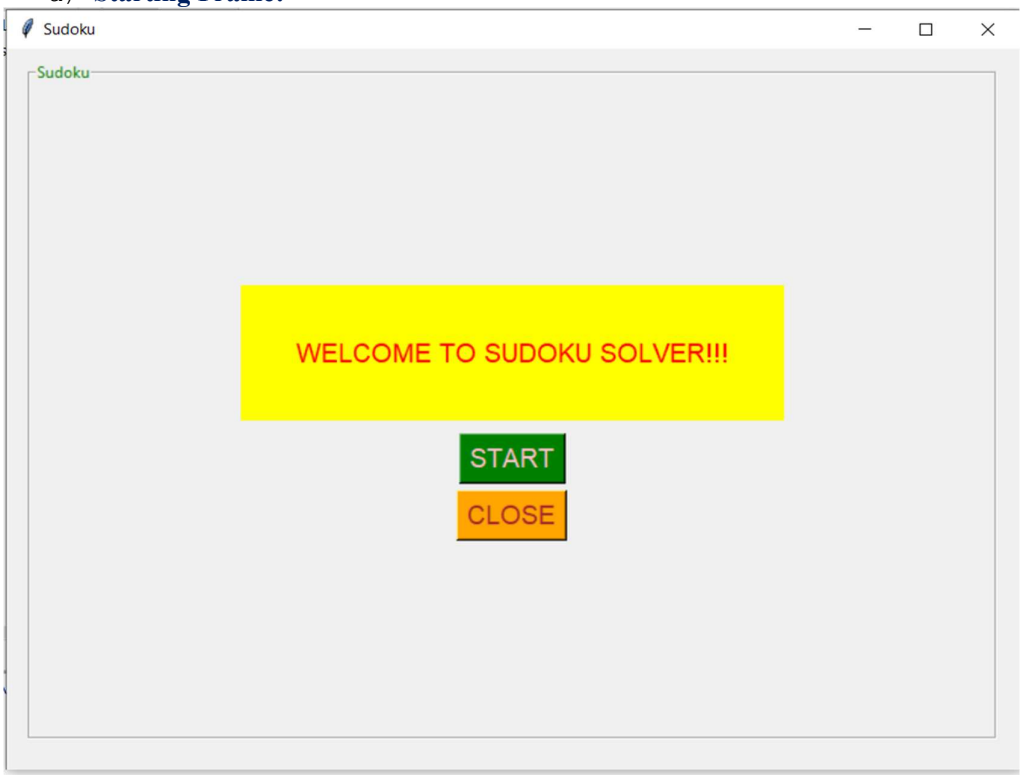
```
def rcheck(grid):
    global dup
    dup.clear()
    for l in range(0, 9):
        k = grid[l]
        dup = copy.deepcopy(k)
        j = finderror()
        if (j == 0):
            pass
        else:
            return 1
    dup.clear()
    return 0


#COLUMN CHECK
def ccheck(grid):
    global dup
    dup.clear()
    for l in range(0, 9):
        for i in grid:
            dup.append(i[l])
        j = finderror()
        if (j == 0):
            pass
        else:
            return 1
        dup.clear()
    dup.clear()
    return 0



def bcheck(grid):
    # BLOCK 1 2 3 CHECK
    global dup
    dup.clear()
    p = [0, 3, 6]
    for m in p:
        for l in range(m, m + 3):
            c = 0
            for i in grid:
                dup.append(i[l])
                c += 1
                if (c > 2):
                    break
        j = finderror()
        if j == 0:
            pass
        else:
            return 1
        dup.clear()

    # BLOCK 4 5 6 CHECK
    dup.clear()
    p = [0, 3, 6]
    for m in p:
        for l in range(m, m + 3):
            c = 0
            for i in grid:
                if c == 0 or c == 1 or c == 2:
                    pass
                else:
                    dup.append(i[l])
                    if c > 4:
                        break
                c += 1
        j = finderror()
        if j == 0:
            pass

        else:
            return 1

        dup.clear()

    # BLOCK 7 8 9 CHECK
    dup.clear()
    p = [0, 3, 6]
    for m in p:
        for l in range(m, m + 3):
            c = 0
            for i in grid:
                if (c == 0) or (c == 1) or (c == 2) or (c == 3)
or (c == 4) or (c == 5):
                    pass
                else:
                    dup.append(i[l])
                    if c > 8:
                        break
                c += 1

        j = finderror()
        if j == 0:
            pass
        else:
            return 1

        dup.clear()

    return 0
```

 # sudoku.py MODULE:

```
import tkinter
from tkinter import *
from second import e
from useless import out
from tkinter import messagebox
import numpy as np
import copy

#GLOBALLY DECLARE GRID ON WHICH OPERATIONS ARE PERFORMED FOR FINDING ANSWERS OF SUDOKU
gridn = [[0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]]


def setinitial(): # SETS THE GLOBAL GRID TO INTITAL STATE, WITH ALL ELEMENTS AS ZERO
    global gridn
    for i in range(0,9):
        for j in range(0,9):
            gridn[i][j]=0


def inputa(frame): # TAKES INPUT FROM THE SUDOKU LAYOUT(USER INPUT) & CHECKS CORRECTNESS
    global gridn, e
    setinitial()
    key=0
    k=0
    for i in range(0,9):
        for j in range(0,9):
            if e[k].get()=='':
                pass
            else:
                try:
                    if int(e[k].get(), 10) in range(1,10):
                        gridn[i][j]=(int(e[k].get(),10))
                    else:
                        messagebox.showinfo("Error!!", "Invalid Inputs")
                        key = 1
                        break
                except ValueError as w:
                    messagebox.showinfo("Error!!", "Invalid Inputs")
                    key = 1
                    break
            k+=1

        if key==1:
            break
    if key==0:

        l=out(gridn) # TO CHECK WHETHER THE INPUTS ARE PROPERLY ENTERED
        if l==-1:
            messagebox.showinfo("Error!!", "Invalid Inputs")
        else:
            solve()

def possible(y, x, n):# CHECK IF WE CAN PUT A NUMBER, y-verical , x-horizontal
    global gridn
    for i in range(0, 9):
        if gridn[y][i] == n:
            return False
    for i in range(0, 9):
        if gridn[i][x] == n:
            return False
    x0 = (x // 3) * 3
    y0 = (y // 3) * 3
    for i in range(0, 3):
        for j in range(0, 3):
            if gridn[y0 + i][x0 + j] == n:
                return False
    return True


def solve(): # TRYING DIFFERENT COMBINATIONS TO GET ANSWER(USES BACKTRACKING)
    global gridn, e

    try:
        for y in range(9):
            for x in range(9):
                if gridn[y][x] == 0:
                    for n in range(1, 10):
                        if possible(y, x, n):
                            gridn[y][x] = n
                            solve()
                            gridn[y][x] = 0
                    return
        output()

    except Exception as t:
        messagebox.showinfo("Error!!", "Invalid Inputs")


def output(): # TO SHOW THE OUTPUT ON SUDOKU LAYOUT/SCREEN
    global gridn
    k = 0
    for i in range(0, 9):
        for j in range(0, 9):
            e[k].delete(0, END)
            e[k].insert(INSERT, str(gridn[i][j])
            k += 1
```

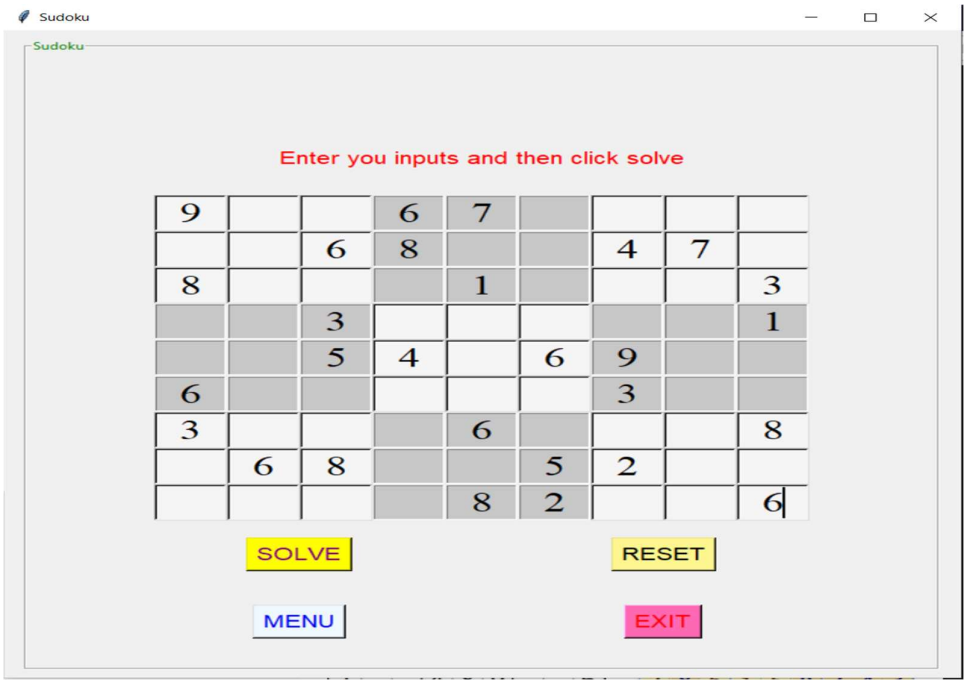
OUTPUTS:

a) **Starting Frame:**

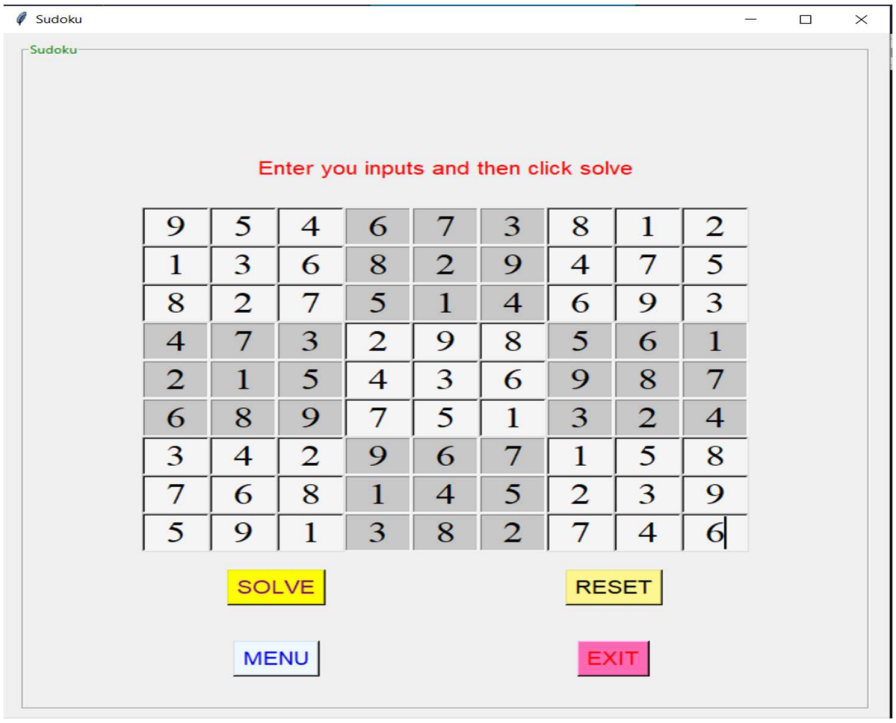


b) **MENU Frame:**

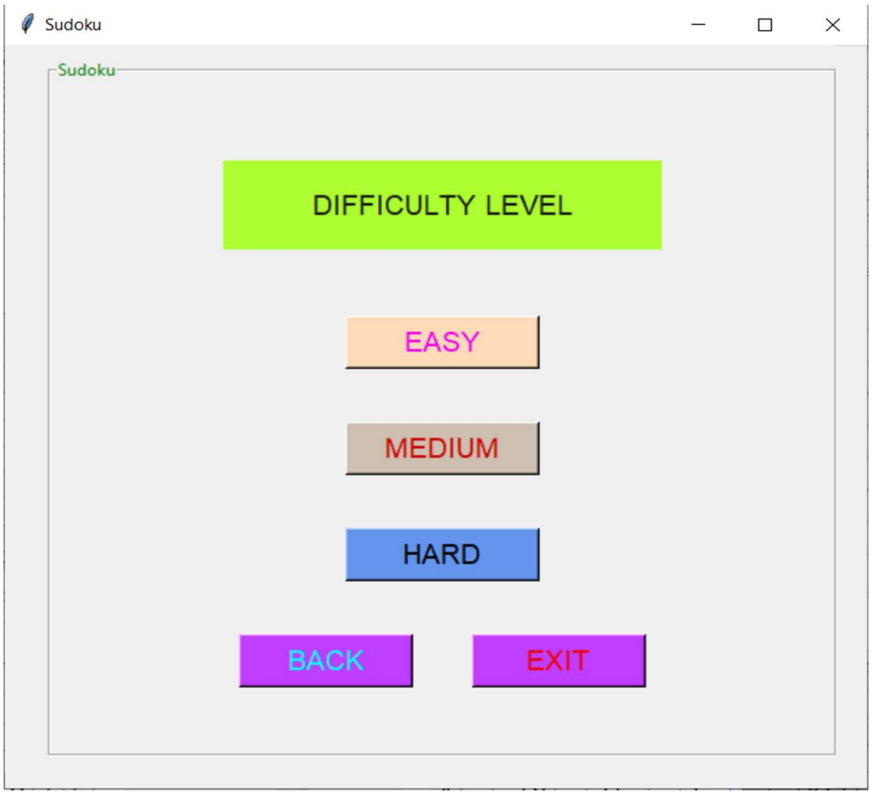
c) **Solver Frame (user has entered inputs):**



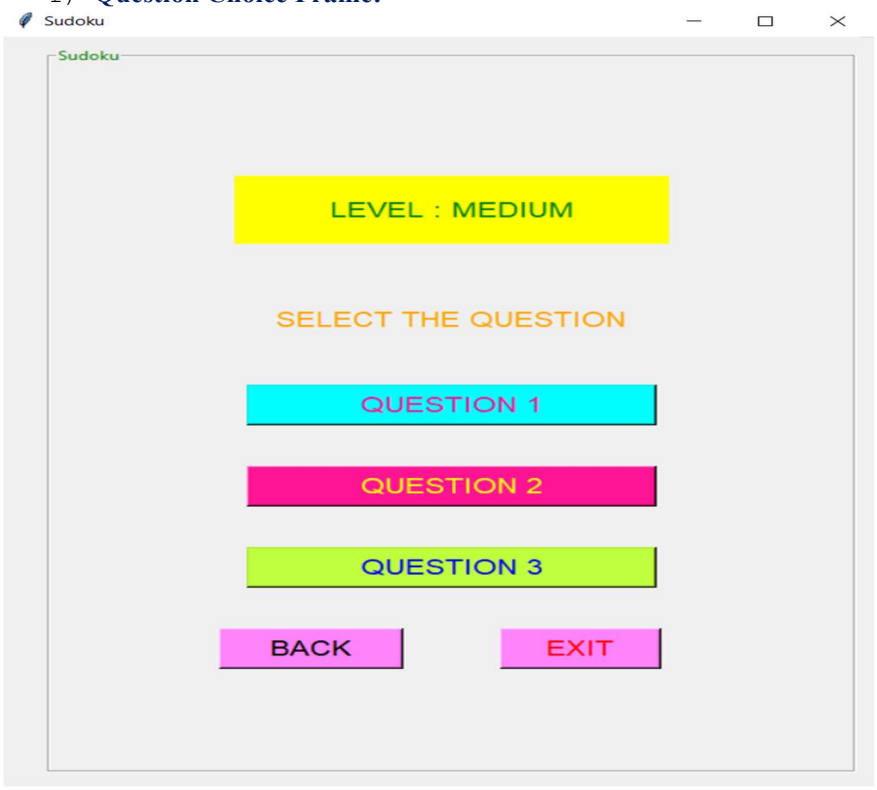
d) **Solver Frame (Answer for user's inputs):**



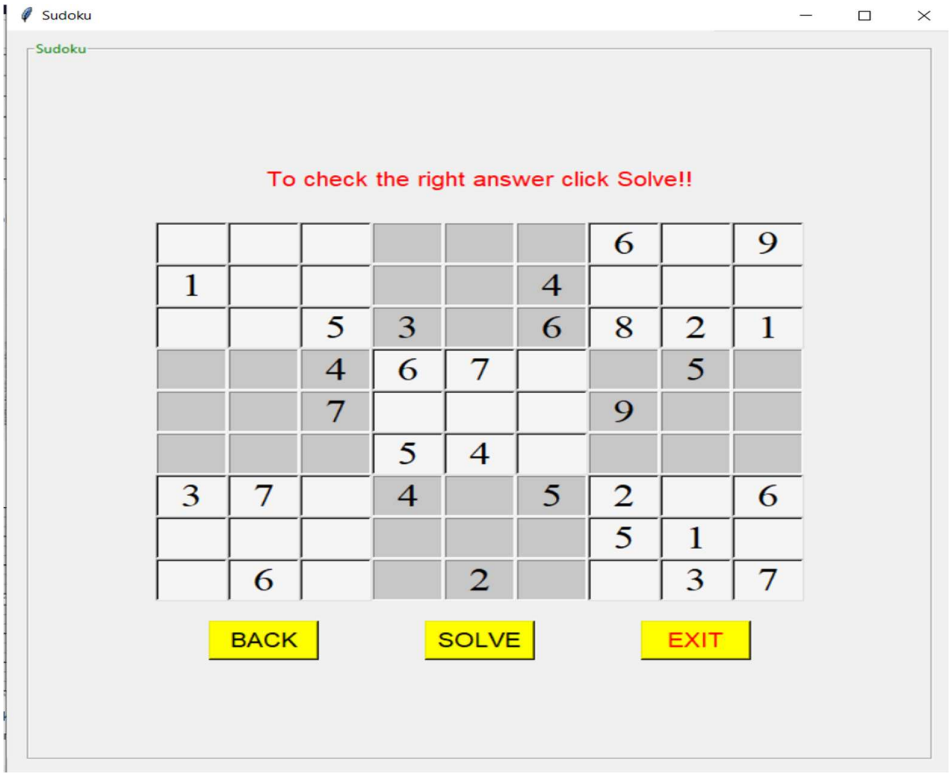
e) **Question difficulty Frame:**



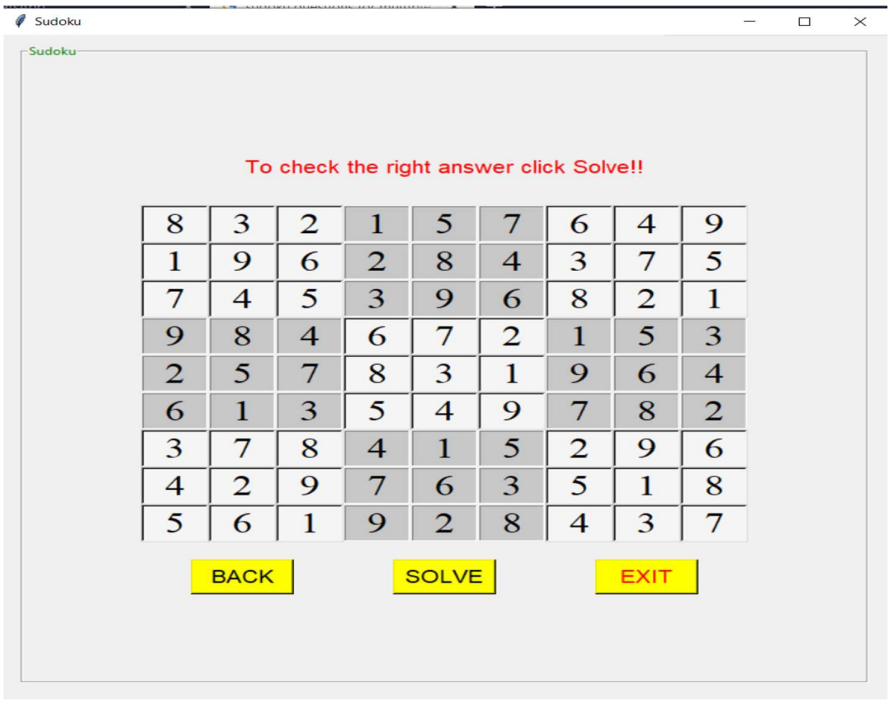
f) **Question Choice Frame:**



g) **Sample Inbuilt Question:**



h) **Answer for Sample Question (After user clicks SOLVE):**



CONCLUSION:

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution. Python helps in making coding simpler for sudoku game. With the help of tkinter module, GUI for the sudoku game is created. And because of Python's wide library and it's GUI content Sudoku game is developed.