SUDOKU

A

Mini Project Report
Submitted in partial fulfilment of the Requirements

for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

MALIGIREDDY CHANDRA KIRAN REDDY 1602-20-737-008

VINTHA HARSHA VARDHAN 1602-20-737-015

ANKAM PRANAY KUMAR 1602-20-737-028

Department of Information Technology Vasavi

College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University and Approved by AICTE)

Ibrahim Bagh, Hyderabad-31

2022

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University and Approved by AICTE)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We, Maligireddy Chandra Kiran Reddy, Vintha Harsha Vardhan and Ankam Pranay Kumar, bearing hall ticket numbers, **1602-20-737-008**, **1602-20-737-015** and **1602-20-737-028**, hereby declare that the project report entitled "**SUDOKU**" is submitted in partial fulfilment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology.

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

Maligireddy Chandra Kiran Reddy

1602-20-737-008

Vintha Harsha Vardhan

1602-20-737-015

Ankam Pranay Kumar

1602-20-737-028

(Faculty In-Charge)

Dr. K. Ram Mohan Rao
Professor & HOD
Dept. of IT

ACKNOWLEDGMENT

We extend our sincere thanks to Dr. S. V. Ramana, Principal, Vasavi College of Engineering for his encouragement.

We express our sincere gratitude to Dr. K. Ram Mohan Rao, Professor & Head, Department of Information Technology, Vasavi College of Engineering, for introducing the Mini-Project module in our curriculum, and for his suggestions, motivation, and co-operation for the successful completion of our Mini Project.

We also want to thank and convey our gratitude towards our mini project coordinators Divya Lingineni and RajyaLakshmi, for guiding us in understanding the process of project development & giving us timely suggestions at every phase.

We would also like to sincerely thank the project reviewers for their valuable inputs and suggestions

ABSTRACT

The main objective of the project is to provide a logic-based game. This project is useful for any kind of people especially young minds who wants to improve their problem-solving skills. And we have also implemented a sudoku solver which takes a sudoku board as input and finds solution if it is solvable.

When the game starts, a menu is shown. The player must select whether to solve a puzzle or would like to find a solution for one.

If player selects to solve a puzzle, then a puzzle is generated and is shown to player. If player selects to find a solution, then an empty puzzle is generated, and the player must give input and the solution will be displayed right away.

The modules, we used for building the GUI for our mini project are pygame and tkinter from python.

TABLE OF CONTENTS

INTRODUCTION

a. Overview

The Main Objective of this mini project is to develop a sudoku game which helps user to play and find solution to puzzle.

b. Features

i. Generates a sudoku puzzle
ii. Finds solution to puzzle entered by user

c. Scope

Playing logic-based games can improve other brain functions, such as attention, concentration, and focus. Combinatorial games give space to critical thinking and that helps children nurture their attention to detail.

Calculating the possibility is key to playing this game and playing them often will improve function in brain. Player can learn the importance of thinking ahead and plotting their next choice.

Technology

a. Software Requirements

- Windows 7 & newer or MacOS 11 & newer
- Modules Required Pre-Installed: tkinter, pygame
- Runtime Environment: PyCharm or IDLE

b. Hardware Requirements

- x86 64-bit CPU Processor
- 4 GB RAM
- 5 GB of free disk space

PROPOSED WORK

a. Design

Use Cases:

i. Generate
ii. Resolve

Use-Case 01:

Name: Generate

Actors: Player, System

Description: Generates a sudoku puzzle and displays it

Precondition: None

Postcondition: A sudoku is displayed

Player	System
	Generates a random sudoku puzzle and checks if a number is repeated in a row or column or in a square.
	If repeated, generates another sudoku puzzle. Else displays the generated puzzle
Enters the number	
	Checks whether the entered number is correct or not If not displays the correct answer

Use-Case 02:

Name: Resolve

Actors: Player, System

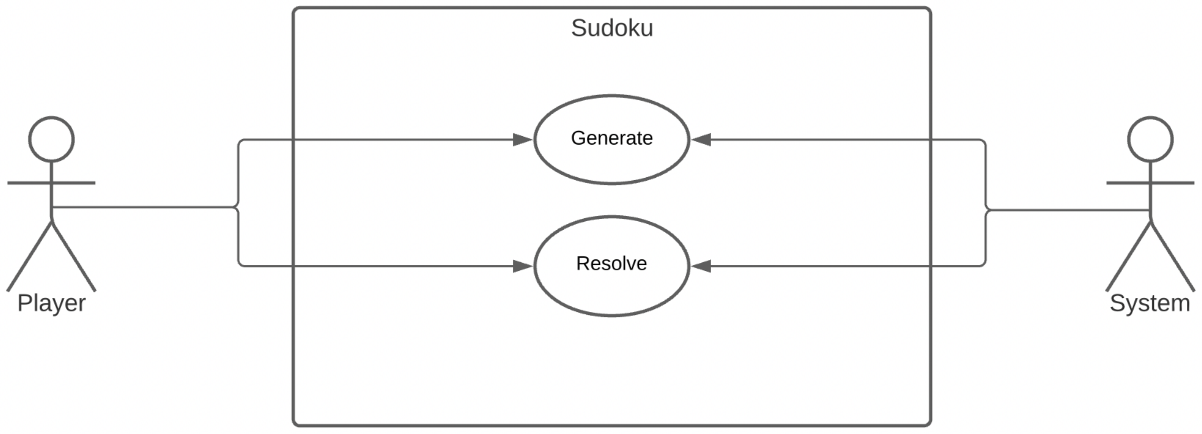
Description: Reads a sudoku puzzle from the player and displays the solution

Precondition: None

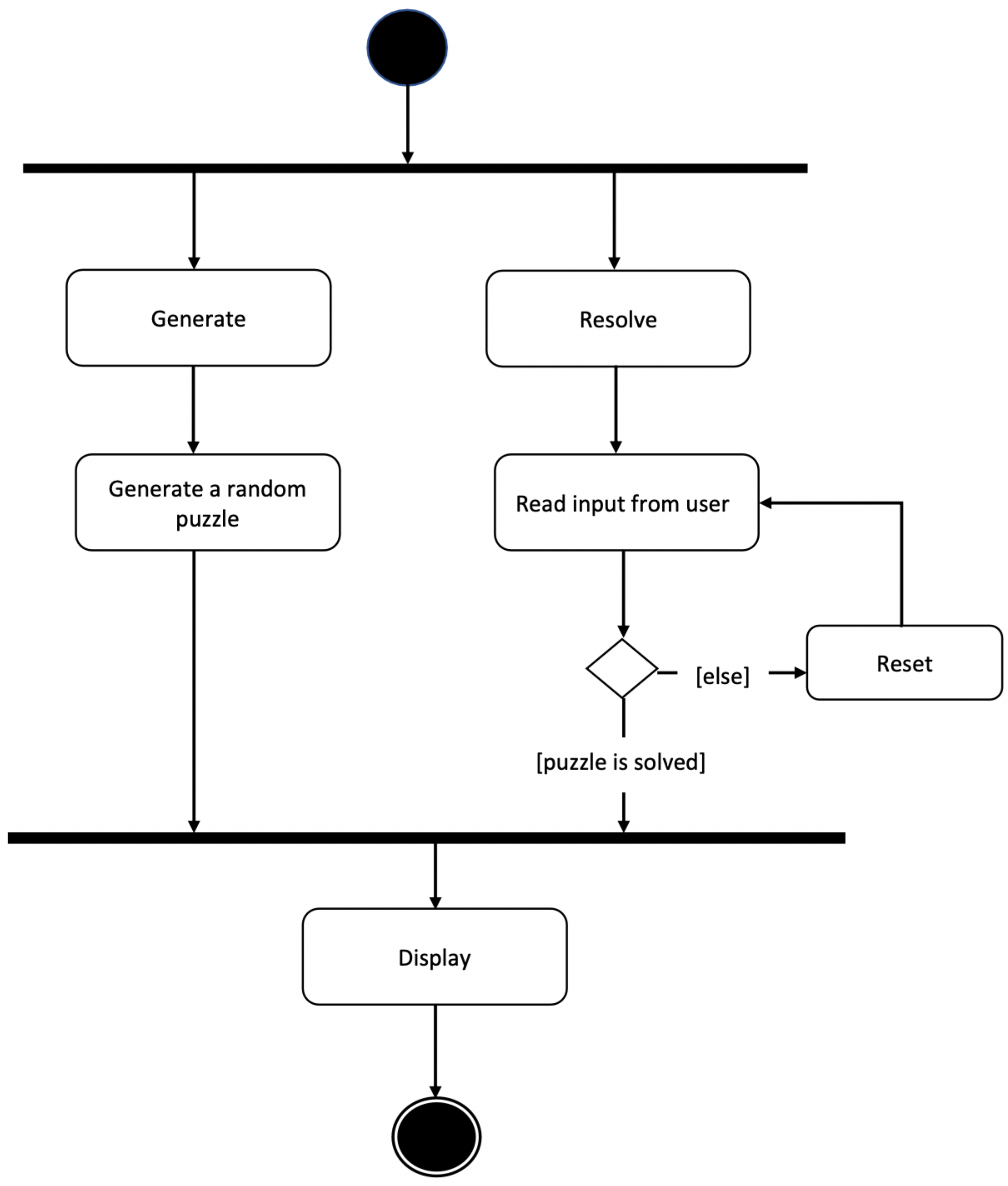
Postcondition: Solution for Puzzle given by player is displayed

Player	System
Enters the sudoku Puzzle	
	Reads a sudoku puzzle from Player and checks if a number is repeated in a row or column or in a square.
	If repeated, reset the puzzle. Else find the solution of puzzle and display it.

Use-Case Diagram:



Activity diagram:



b. Implementation

Module-wise Code:

i. Generate:

```
from copy import deepcopy
from sys import exit
import pygame
import time
import random

pygame.init()

def find_empty (board):
    for i in range (9):
        for j in range (9):
            if board[i][j] == 0:
                return i,j

def valid(board, pos, num):
    for i in range(9):
        if board[i][pos[1]] == num and (i, pos[1])
!= pos:
            return False

    for j in range(9):
        if board[pos[0]][j] == num and (pos[0], j)
!= pos:
            return False

    start_i = pos[0] - pos[0] % 3
    start_j = pos[1] - pos[1] % 3
    for i in range(3):
        for j in range(3):
            if board[start_i + i][start_j + j] ==
num and (start_i + i, start_j + j) != pos:
                return False
    return True

def solve(board):

    empty = find_empty(board)
    if not empty:
        return True

    for nums in range(9):
        if valid(board, empty,nums+1):
            board[empty[0]][empty[1]] = nums+1

            if solve(board):
                return True
            board[empty[0]][empty[1]] = 0
    return False
def generate():

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit()
        board = [[0 for i in range(9)] for j in
range (9)]

        for i in range(9):
            for j in range(9):
                if random.randint(1, 10) >= 5:
                    board[i][j] = random.randint(1, 
9)
                    if valid(board, (i, j), 
board[i][j]):
                        continue
                    else:
                        board[i][j] = 0
        partialBoard = deepcopy(board)
        if solve(board):
            return partialBoard
class Board:

    def __init__(self, window):
        self.board = generate()
        self.solvedBoard = deepcopy(self.board)
        solve(self.solvedBoard)
        self.tiles = [[Tile(self.board[i][j], 
window, i*60, j*60) for j in range(9)] for i in 
range(9)]
        self.window = window

    def draw_board(self):

        for i in range(9):
            for j in range(9):
                if j%3 == 0 and j != 0:
                    pygame.draw.line(self.window, 
(0, 0, 0), ((j//3)*180, 0), ((j//3)*180, 540), 4)
```

```
                if i%3 == 0 and i != 0:
                    pygame.draw.line(self.window,
(0, 0, 0), (0, (i//3)*180), (540, (i//3)*180), 4)

                self.tiles[i][j].draw((0,0,0), 1)

                if self.tiles[i][j].value != 0:

self.tiles[i][j].display(self.tiles[i][j].value,
(21+(j*60), (16+(i*60))), (0, 0, 0))
        pygame.draw.line(self.window, (0, 0, 0),
(0, ((i+1) // 3) * 180), (540, ((i+1) // 3) * 180),
4)

    def deselect(self, tile):

        for i in range(9):
            for j in range(9):
                if self.tiles[i][j] != tile:
                    self.tiles[i][j].selected =
False

    def redraw(self, keys, wrong, time):
        self.window.fill((255,255,255))
        self.draw_board()
        for i in range(9):
            for j in range(9):
                if self.tiles[j][i].selected:
                    self.tiles[j][i].draw((50, 205,
50), 4)
```

```
                elif self.tiles[i][j].correct:
                    self.tiles[j][i].draw((34, 139,
34), 4)

                elif self.tiles[i][j].incorrect:
                    self.tiles[j][i].draw((255, 0,
0), 4)

        if len(keys) != 0:
            for value in keys:

self.tiles[value[0]][value[1]].display(keys[value],
(21+(value[0]*60), (16+(value[1]*60))), (128, 128,
128))

        if wrong > 0:
            font = pygame.font.SysFont('Bauhaus
93', 40)
            text = font.render('X', True, (255, 0,
0))
            self.window.blit(text, (10, 554))

            font = pygame.font.SysFont('Calibri
Bold', 40)
            text = font.render(str(wrong), True,
(0, 0, 0))
            self.window.blit(text, (32, 554))

        font = pygame.font.SysFont('Bahnschrift',
40)
        text = font.render(str(time), True, (0, 0,
0))
        self.window.blit(text, (388, 554))
        pygame.display.flip()

    def visualSolve(self, wrong, time):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit()

        empty = find_empty(self.board)
        if not empty:
            return True

        for nums in range(9):
            if valid(self.board,
(empty[0],empty[1]), nums+1):
                self.board[empty[0]][empty[1]] =
nums+1

self.tiles[empty[0]][empty[1]].value = nums+1

self.tiles[empty[0]][empty[1]].correct = True
                pygame.time.delay(63)
                self.redraw({}, wrong, time)

                if self.visualSolve(wrong, time):
                    return True

                self.board[empty[0]][empty[1]] = 0

self.tiles[empty[0]][empty[1]].value = 0
self.tiles[empty[0]][empty[1]].incorrect = True

self.tiles[empty[0]][empty[1]].correct = False
                pygame.time.delay(63)
                self.redraw({}, wrong, time)

    def hint(self, keys):
        while True:
            i = random.randint(0, 8)
            j = random.randint(0, 8)
            if self.board[i][j] == 0:
                if (j,i) in keys:
                    del keys[(j,i)]
                self.board[i][j] =
self.solvedBoard[i][j]
                self.tiles[i][j].value =
self.solvedBoard[i][j]
                return True

            elif self.board == self.solvedBoard:
                return False
class Tile:
    def __init__(self, value, window, x1, y1):
        self.value = value
        self.window = window
        self.rect = pygame.Rect(x1, y1, 60, 60)
        self.selected = False
        self.correct = False
        self.incorrect = False
    def draw(self, color, thickness):
        pygame.draw.rect(self.window, color,
self.rect, thickness)

    def display(self, value, position, color):
        font = pygame.font.SysFont('lato', 45)
        text = font.render(str(value), True, color)
        self.window.blit(text, position)

    def clicked(self, mousePos):
        if self.rect.collidepoint(mousePos):
            self.selected = True
        return self.selected

def mains():

    screen = pygame.display.set_mode((540, 590))
    screen.fill((255, 255, 255))
    pygame.display.set_caption("Sudoku")

    font = pygame.font.SysFont('Courier New Bold',
40)
    text = font.render("Generating", True, (0, 0,
0))
    screen.blit(text, (175, 245))

    font = pygame.font.SysFont('Courier New Bold',
40)
    text = font.render("Puzzle", True, (0, 0, 0))
    screen.blit(text, (200, 290))
    pygame.display.flip()
    wrong = 0
    board = Board(screen)
    selected = -1,-1
    keyDict = {}
    running = True
    startTime = time.time()
    while running:
        elapsed = time.time() - startTime
        passedTime = time.strftime("%H:%M:%S", 
time.gmtime(elapsed))

        if board.board == board.solvedBoard:
            for i in range(9):
                for j in range(9):
                    board.tiles[i][j].selected = 
False
                    running = False

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit()

            elif event.type == 
pygame.MOUSEBUTTONUP:
                mousePos = pygame.mouse.get_pos()
                for i in range(9):
                    for j in range (9):
                        if 
board.tiles[i][j].clicked(mousePos):
                            selected = i,j
```

```
board.deselect(board.tiles[i][j])

            elif event.type == pygame.KEYDOWN:
                if
board.board[selected[1]][selected[0]] == 0 and
selected != (-1,-1):
                    if event.key == pygame.K_1:
                        keyDict[selected] = 1

                    if event.key == pygame.K_2:
                        keyDict[selected] = 2

                    if event.key == pygame.K_3:
                        keyDict[selected] = 3

                    if event.key == pygame.K_4:
                        keyDict[selected] = 4

                    if event.key == pygame.K_5:
                        keyDict[selected] = 5

                    if event.key == pygame.K_6:
                        keyDict[selected] = 6

                    if event.key == pygame.K_7:
                        keyDict[selected] = 7

                    if event.key == pygame.K_8:
                        keyDict[selected] = 8
                    if event.key == pygame.K_9:
                        keyDict[selected] = 9

                    elif event.key ==
pygame.K_BACKSPACE or event.key == pygame.K_DELETE:
                        if selected in keyDict:

board.tiles[selected[1]][selected[0]].value = 0
                            del keyDict[selected]

                    elif event.key ==
pygame.K_RETURN:
                        if selected in keyDict:
                            if keyDict[selected] !=
board.solvedBoard[selected[1]][selected[0]]:
                                wrong += 1

board.tiles[selected[1]][selected[0]].value = 0
                                del
keyDict[selected]
                                break

board.tiles[selected[1]][selected[0]].value =
keyDict[selected]

board.board[selected[1]][selected[0]] =
keyDict[selected]
                            del keyDict[selected]

                if event.key == pygame.K_TAB:
                    board.hint(keyDict)
                if event.key ==pygame.K_r:
                    mains()
                if event.key ==pygame.K_q:
                    pygame.quit()
                if event.key == pygame.K_ESCAPE:
                    for i in range(9):
                        for j in range(9):

board.tiles[i][j].selected = False
                    keyDict = {}
                    board.visualSolve(wrong,
passedTime)
                    for i in range(9):
                        for j in range(9):

board.tiles[i][j].correct = False

board.tiles[i][j].incorrect = False
                    running = False
        board.redraw(keyDict, wrong, passedTime)

mains()
pygame.quit()
```

ii. Call:

```
import os
def create(n):
    if (n==1):
        print()
    if (n==0):
        import Generate
```

iii. Solve:

```
from tkinter import *
from tkinter import messagebox
import os
entries = []

def initialize(top,arr):
    E = entries[0]
    m=1
    for i in range(9):
        for j in range(9):
            if(not E.get()):
                arr[i][j] = 0
            else:
                try:
                    if int(E.get()):
                         arr[i][j] = int(E.get())
                    else:

messagebox.showwarning("Warning",  "Enter Integers
From 1 to 9 Only")
                       button_reset.invoke()
                except ValueError:

messagebox.showwarning("Warning",  "Enter Integers
From 1 to 9 Only ")
                    button_reset.invoke()
            if(m<=80):
                E = entries[m]
                m+=1
    checker(top,arr)
    squarecheck(top,arr)

def print_maze(arr):
    clean_Mess()
    E = entries[0]
    m=1
    for i in range(9):
        for j in range(9):
            E.insert(1,arr[i][j])
            if(m<=80):
                E = entries[m]
                m+=1

def find_empty_location(arr,l):
    for row in range(9):
        for col in range(9):
            if(arr[row][col]==0):
                l[0]=row
                l[1]=col
                return True
    return False
def used_in_row(arr,row,num):
    for i in range(9):
        if(arr[row][i] == num):
            return True
    return False
def used_in_col(arr,col,num):
    for i in range(9):
        if(arr[i][col] == num):
            return True
    return False
```

```
def used_in_box(arr,row,col,num):
    for i in range(3):
        for j in range(3):
            if(arr[i+row][j+col] == num):
                return True
    return False
def check_location_is_safe(arr,row,col,num):
    return not used_in_row(arr,row,num) and not
used_in_col(arr,col,num) and not
used_in_box(arr,row - row%3,col - col%3,num)

def solve_sudoku(arr):
    l=[0,0]
    if(not find_empty_location(arr,l)):
        return True
    row=l[0]
    col=l[1]
    for num in range(1,10):

if(check_location_is_safe(arr,row,col,num)):
            arr[row][col]=num
            if(solve_sudoku(arr)):
                return True
            arr[row][col] = 0
    return False

def createGUI(maze):
    top = Tk()
    top.title("Sudoku")
    canvas = Canvas(top, height=320, width =350)
    createRow(canvas)
    createCol(canvas)
    createEntry(top)
    createButtons(top,maze)
    canvas.pack(side = 'top')
    top.mainloop()

def createButtons(top,maze):
    button_solve = Button(top, text="Solve",
justify='left', default='active', command =
lambda: play_Game(top,maze))
    button_reset = Button(top, text="Reset",
justify='right', command = lambda: clean_Mess())
    button_solve.place(x=70, y=275, height=30,
width=60)
    button_reset.place(x=230, y=275, height=30,
width=60)

def clean_Mess():
    for e in entries:
        e.delete(0, END)

def checker(top,arr):
    c=0
    for i in range(9):
        for j in range(9):
            if arr[i][j]!=0:
                for k in range(i+1,9):
                    if(arr[i][j]==arr[i][k]):
                        c=c+1
                    if(arr[i][j]==arr[k][j]):
                        c=c+1
    if c!=0:
        messagebox.showinfo("Information",
"Numbers have repeated within the row or column")
        button_reset.invoke()

def check(arr,startRow,startCol):
    st = set()
    for row in range(0, 3):
        for col in range(0, 3):
            curr = arr[row + startRow][col +
startCol]
            if curr in st:
                return False
            if curr != 0:
                st.add(curr)

    return True

def squarecheck(top,arr):
    sa=0
    for i in range(9):
        for j in range(9):
            if arr[i][j]!=0:
                if not check(arr,i,j):
                    sa=sa+1
    if sa!=0:
        messagebox.showinfo("Information",
"Numbers have repeated within the square")
        button_reset.invoke()
```

```
def play_Game(top,maze):
    initialize(top,maze)
    if(solve_sudoku(maze)):
        print_maze(maze)
    else:
        print ("No solution found")

def createEntry(top):
    p,q=41.4,41.4
    for i in range(9):
        for j in range(9):
            E = Entry(top, width=3, font = 'BOLD')
            E.grid(row=i, column=j)
            E.place(x=p, y=q, height=20, width=25)
            entries.append(E)
            p+=30.0
        q+=24.5
        p=41.2

def createRow(canvas):
    i,j=40,40
    p=40
    q=260
    for m in range(10):
        if(m%3==0):
            canvas.create_line(i,j,p,q,width=2.5)
        else:
            canvas.create_line(i,j,p,q,width=2)
        i+=30
        p+=30
```

```
def createCol(canvas):
    i,j=40,40
    p,q=310,40
    for m in range(10):
        canvas.create_line(i,j,p,q,width=2.3)
        j+=24.5
        q+=24.5
if __name__=="__main__":
    maze=[[0 for x in range(9)]for y in range(9)]
    createGUI(maze)
```

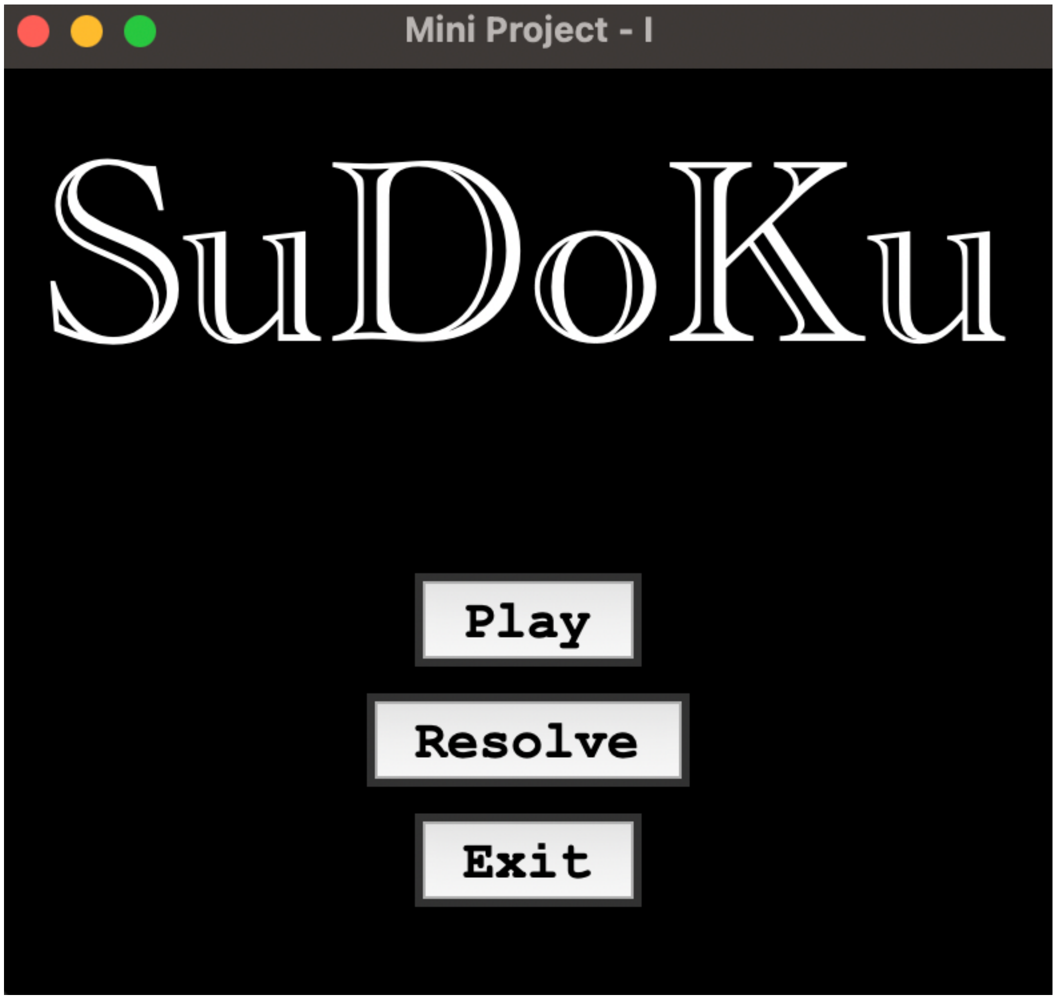
iv. Run:

```
from tkinter import *
from tkinter import messagebox
from Solve import *
from Call import *
import pygame
import os
def ex():
    window.destroy()
    os._exit(0)
def play():
    messagebox.showinfo("showinfo", "Thanks for
Playing \n Press 'TAB' for getting a Hint \n Press
'ESCAPE' to Auto Solve the Puzzle \n Press 'q' to
Go back to Main Menu \n Press 'r' to Reset the
Puzzle \n")
    create(0)
def solver():
    maze=[[0 for x in range(9)]for y in range(9)]
    createGUI(maze)
```

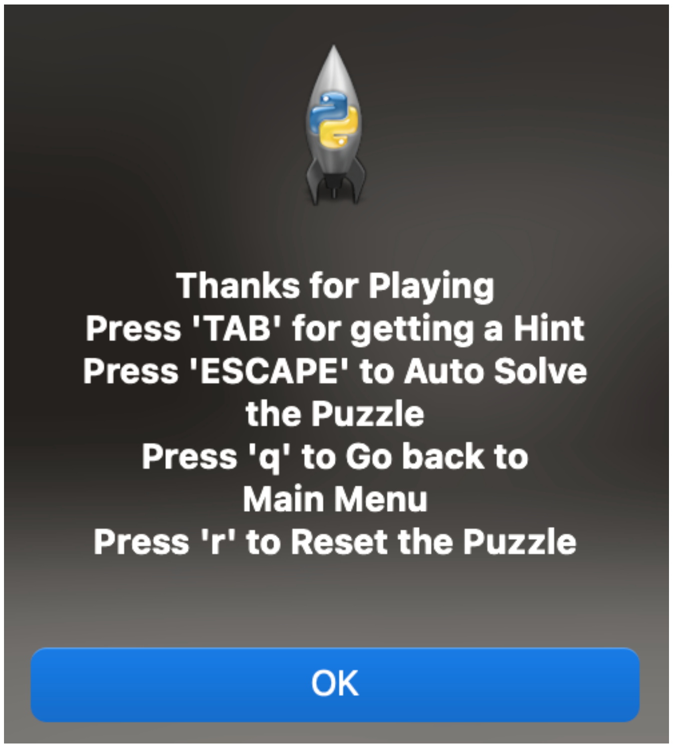
```
window = Tk()
window.title("Mini Project - I")
window.geometry('400x350')
window.config(background="black")
text=Label(window,text="SuDoKu",foreground="white"
,background="black",font=('Academy Engraved
LET',100))
text.pack(pady=30)
b1 = Button(window,text='Play',command=play,
font=("Courier New Bold", 20))
b1.pack(pady=5)
b2 = Button(window,text='Resolve',command =
solver,font=("Courier New Bold", 20))
b2.pack(pady=5)
b3 = Button(window,text='Exit',command =
ex,font=("Courier New Bold", 20))
b3.pack(pady=5)

window.mainloop()
```

RESULT

After running the code, a new page is popped up with title “Mini Project - I”



If we click on play button, a message box is popped up stating all the keys used.


Thanks for Playing
Press 'TAB' for getting a Hint
Press 'ESCAPE' to Auto Solve the Puzzle
Press 'q' to Go back to Main Menu
Press 'r' to Reset the Puzzle

Then, we must click on ‘ok’ button and then sudoku puzzle is displayed.

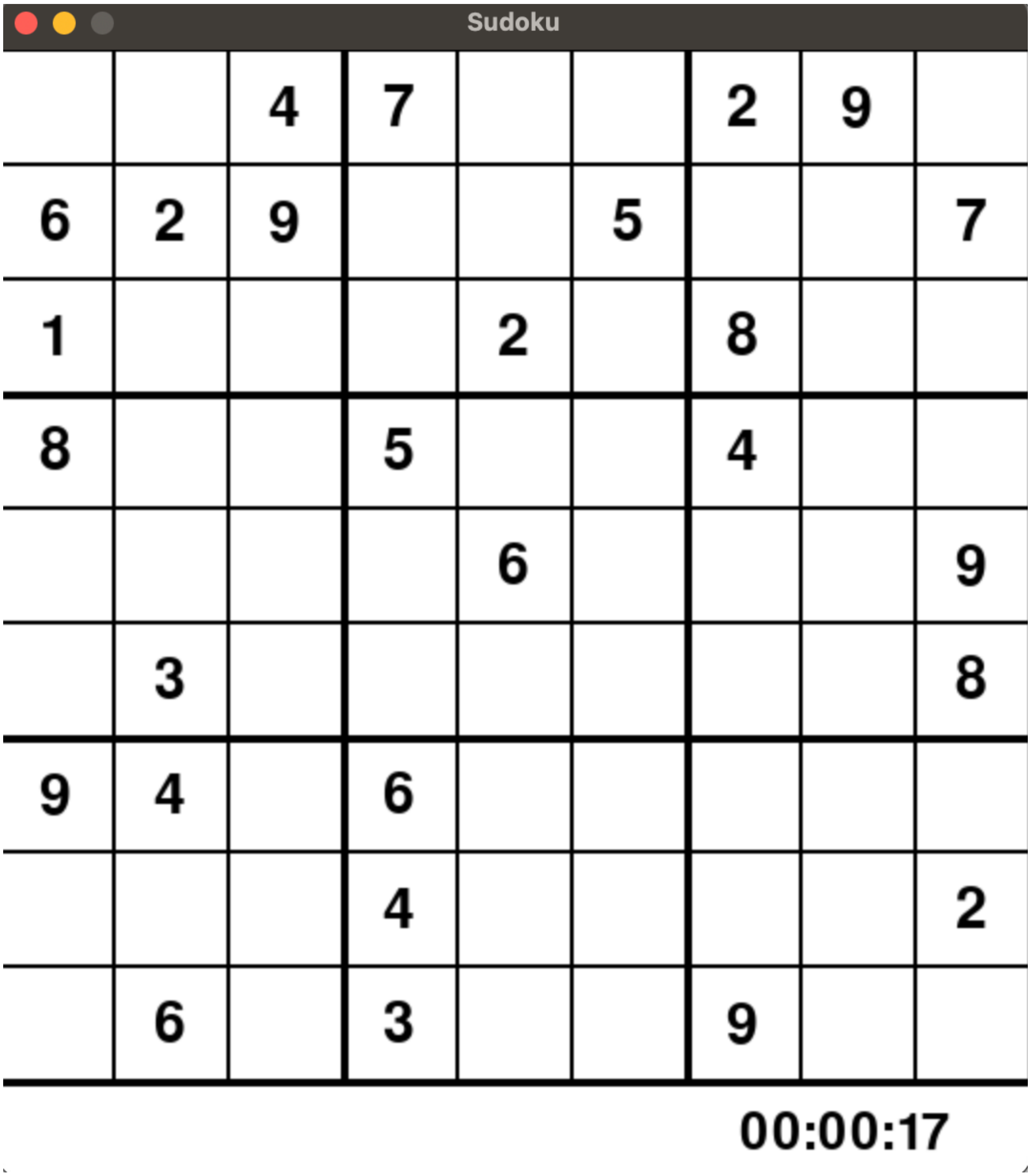
Player can start fill the puzzle and can check whether the entered number is correct or not by clicking the “ENTER” button.

Player can also get a random box filled as a hint by clicking the “TAB” button.

If player cannot solve the sudoku puzzle, then player can click the “ESCAPE” button and system automatically solves the puzzle.

Player can also rest the board by clicking “r” button.

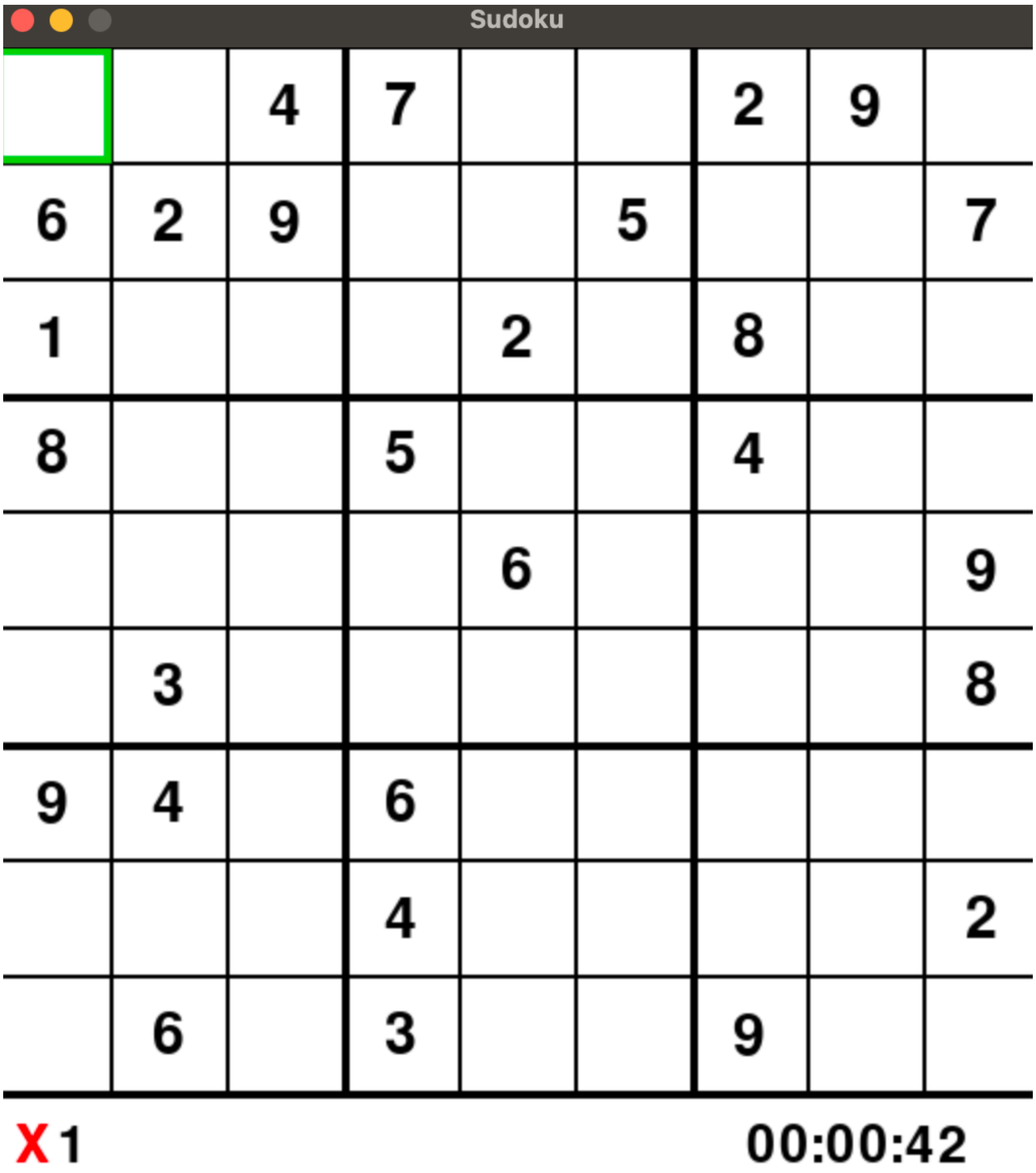
Player can also quit by clicking “q” button.



Sudoku

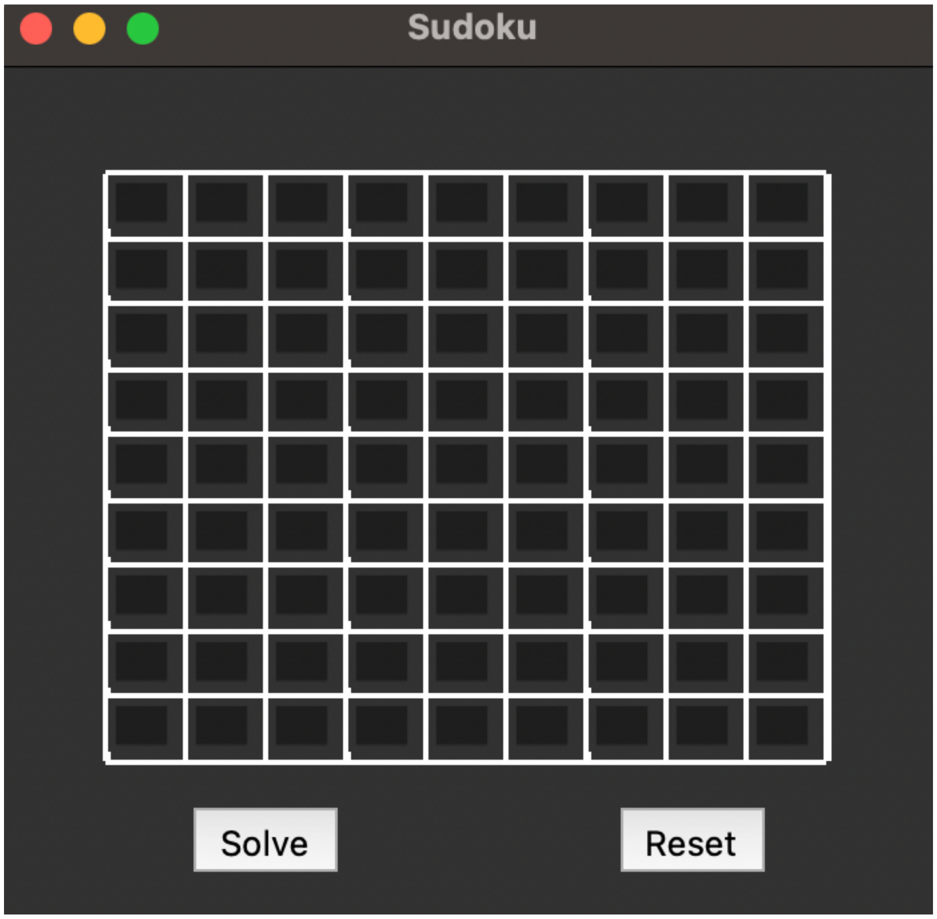
		4	7			2	9	
6	2	9			5			7
1				2		8		
8			5			4		
				6				9
	3							8
9	4		6					
			4					2
	6		3			9		

00:00:17

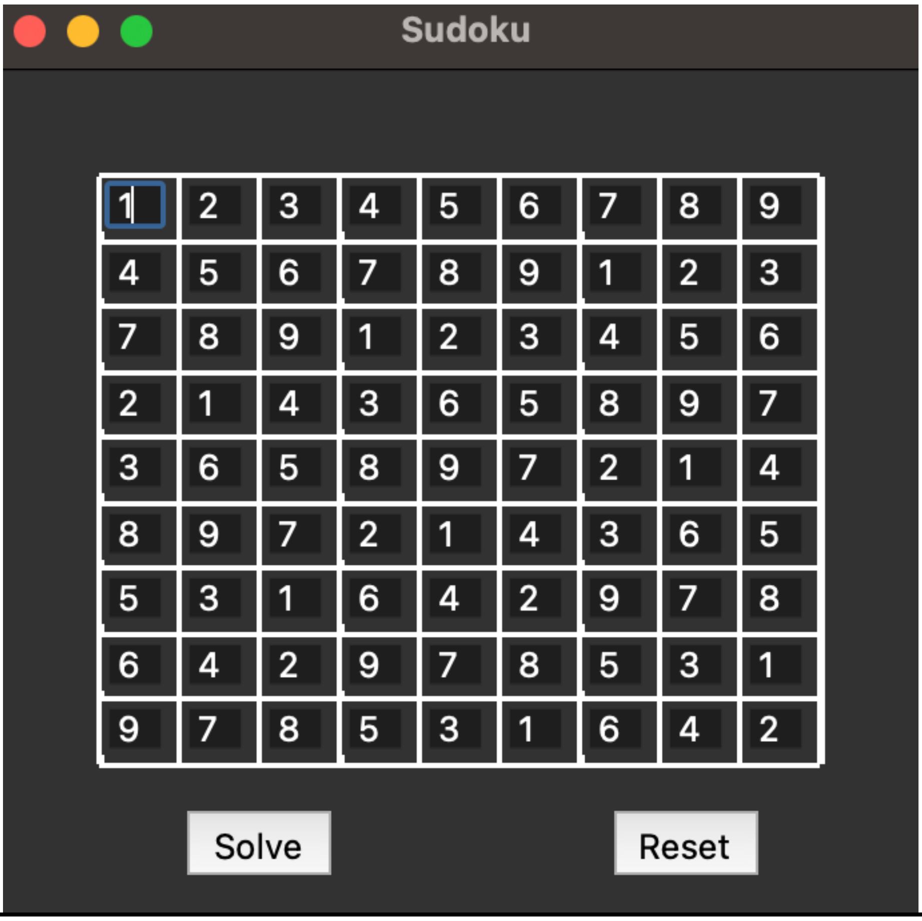


		4	7			2	9	
6	2	9			5			7
1				2		8		
8			5			4		
				6				9
	3							8
9	4		6					
			4					2
	6		3			9		

After quitting the play mode, the menu will appear and we will now click on resolve button, which helps in finding the solution for puzzle entered by players.



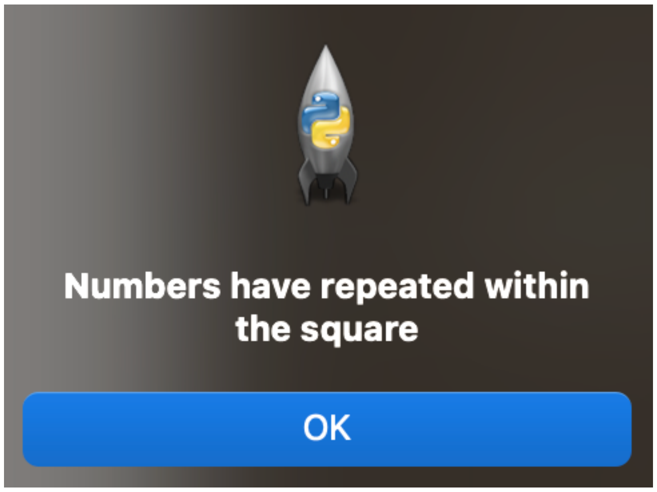
Here I have entered 1 and clicked on solve button. Then this is the output.

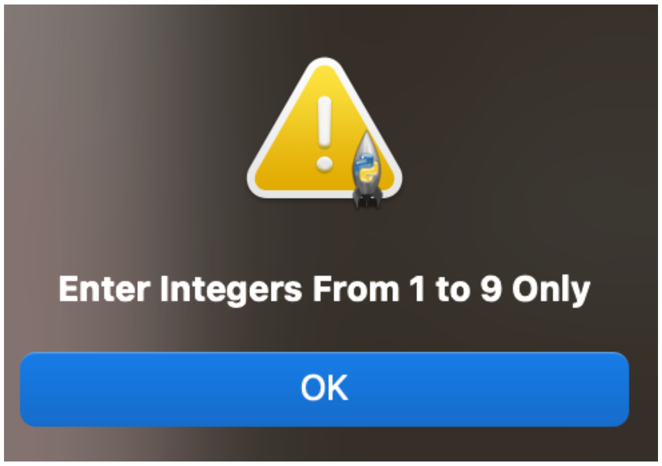


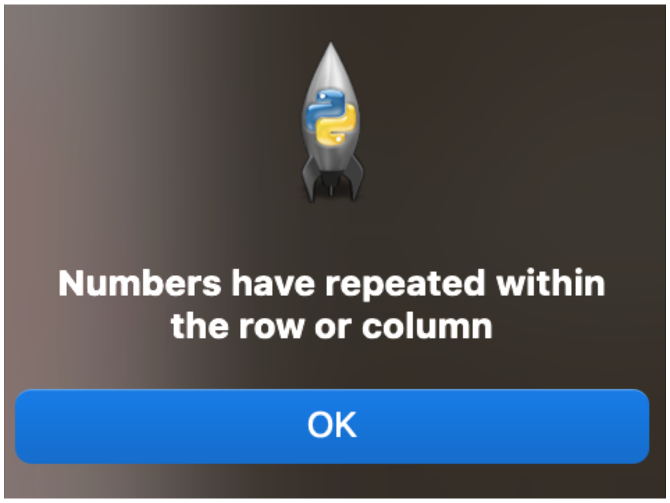
1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

Now if you enter a character, then a warning box is popped up.

If you enter a number same number in a row or column or in a square, then a warning box popped up.


Numbers have repeated within the square


Enter Integers From 1 to 9 Only


Numbers have repeated within the row or column

If we click on cross button, then we can come back to main menu again.

Then if we click on 'exit' button, program is terminated.

OUTCOMES

- I have discovered different kinds of libraries and modules in python and got to know their implementation and working.
- We have learnt how to implement GUI using python.
- We have also got to know about modules like pygame, os and tkinter and various components present in those modules.
- We have implemented GUI using pygame and tkinter. By doing so, we have learnt so much about GUI and error handling in GUI.

CONCLUSION

As we know Gaming is one of the largest segments of the entertainment industry. It is a demanding career in today's modern world. With the wide access to the internet has created a humongous demand for online games and with the advent of technology, this industry is growing at fast pace.
So, this kind of logic-based games still have a scope in future world.

FUTURE SCOPE

- Include levels like Easy, Medium and Hard for Play Option.
- Include AI for scanning and reading input from an image or directly from camera and find the solution respectively.
- Improve User Interface.
- Include Leaderboard based on time taken to solve the puzzle.

REFERENCES

1. https://www.pygame.org/docs/ref/event.html?highlight=keyword
2. https://docs.python.org/3/library/tk.html
3. Article: https://www.techwithtim.net/tutorials/python-programming/sudoku-solver-backtracking/
4. https://www.geeksforgeeks.org/sudoku-backtracking-7/