DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

DISCRETE STRUCTURES -[IT-205]

PROJECT REPORT

Submitted by
Aman Yadav
(2K19/IT/014)

Submitted to
Ms. Swati Sharda
(Faculty IT- dept.)

CANDIDATE'S DECLARATION

I, Aman Yadav (2K19/IT/014) student of B.Tech.(Department of Information Technology), hereby declare that the project Dissertation titled ***Real Timer Sudoku Solver and Game*** Project Bachelor of Technology, is original and not copied from any which is submitted by us to the Department of Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Name: Aman Yadav

Roll no: (2K19/IT/014)

Place: Delhi

DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "***Real Timer Sudoku Solver and Game***" which is submitted by Aman Yadav(2K19/IT/014) and Information Technology, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology, is a record of the project work carried out by the students under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Ms. Swati Sharda
(supervisor)

Place: Delhi

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations.

I would like to extend my sincere thanks to all of them. We are highly indebted to Ms. Swati Sharda for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & members of (DTU)for their kind cooperation and encouragement which helped us in completion of this project.

I would like to express my special gratitude and appreciation also to our colleagues in developing the project and people who have willingly helped us out with their abilities.

Abstract

In this report, we present the detailed development and implementation of interactive sudoku game along with its real time solution option. The Sudoku game consists of graphical user interface, input of hard copy via webcam, the solver and generator is implemented using efficient algorithm. The solver finds the solution to the puzzles displayed on the webcam as well as puzzles generated by the generator. Generator uses a database for generating different sudoku puzzles. This project gives an insight into the different aspects of python programming.

CONTENTS

INTRODUCTION

Topic : Real Timer Sudoku Solver and Game

Problem : To make a real time Sudoku solver and an interactive playing platform.

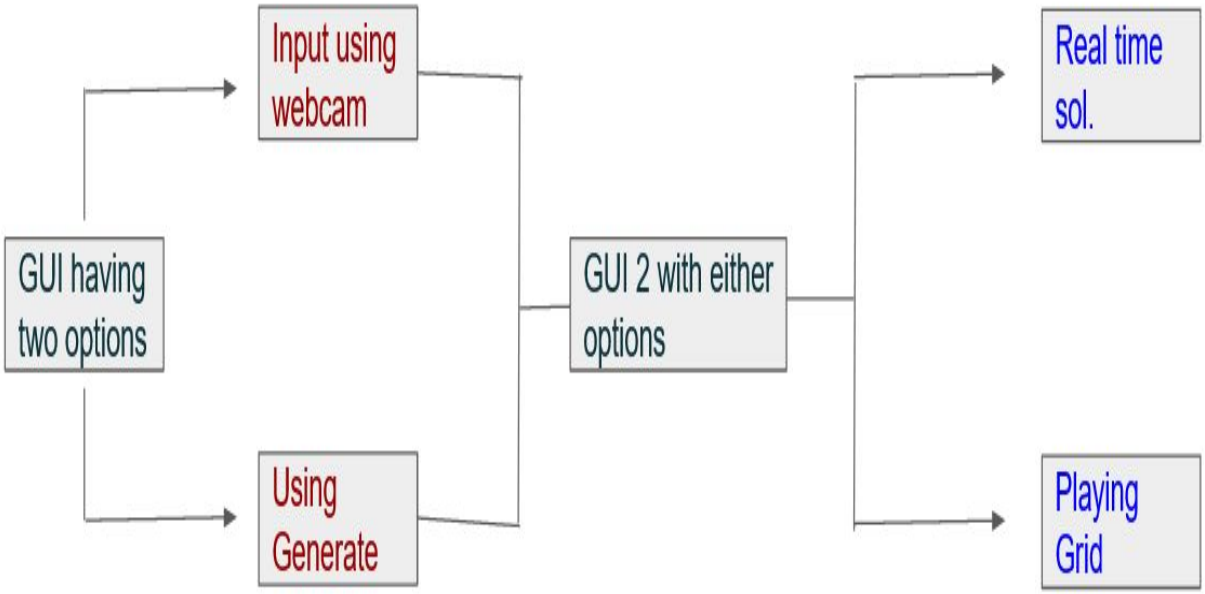
Special Features :

- ❏ Input by webcam
- ❏ Real time solution
- ❏ Interactive grid to play
- ❏ All in GUI

Approach :

- Input using Webcam or Generate method:
  Reading Image -> Processing -> Extracting digits(using OCR) -> Forming grid
- Sudoku solving module:
  Using Backtracking algo for solving.
- Real time solution:
  Using Opencv for masking solution on real time image.
- Playing Grid:
  GUI using tkinter along with HINT, CHECK, CLEAR, SOLUTION buttons.

FLOW CHART


Input using webcam
Real time sol.
GUI having two options
GUI 2 with either options
Using Generate
Playing Grid

CODE SNIPPETS

Programming Language: Python 3.7

Python Modules:

1. **OpenCV -> for image reading, writing, processing on image.**
2. **Numpy -> for performing operations on images to reduce noises.**
3. **Tkinter -> for GUI**
4. **Pytesseract -> for OCR**
5. **copy,os,glob,PIL,random -> for deep copy, file handling, removing folder data, reading image, randint() respectively.**

User Defined Classes and Functions:

- `class sudokucls(Frame)` `#GUI CLASS`
  - `def __incls(self):` *#Playing GUI window*
  - `def draw_grid(self):` *#drawing grids of 9x9*
  - `def draw_fill(self, fill_flag):` *#filling in grids*
  - `def highlight_cell(self):` *#highlighting cell in focus*
  - `def cell_clicked(self, event):` *#detect click on canvas*
  - `def key_pressed(self, event):` *#detect pressed key*
  - `def check(self):` *#checking inputs (Button on GUI)*
  - `def clear_ans(self):` *#clearing inputs (Button on GUI)*
  - `def hint(self):` *#printing a row as hint (Button on GUI)*
  - `def solution(self):` *#show whole solution (Button on GUI)*
  - `def victory(self):` *#last message display*

- `def solving_module():` *#main solving module*

```
        def find_blank(row, col)        finding voids
        def is_allow(num, row, col):    checking for possibility
```

- `def cam():` Getting grid from webcam

```
        def blurring_mod(img):          image smoothing module
        def proper_order(ar):           check the order of points
        def chopping_grid(pt):          chopping 0f cells
```

- `def gen():` function for generating Sudoku
- `def separate_space():` separating blanks and numbers
- `def remove_cell_noice(num_pos):` removing cell's noises and saving

```
        def autocrop(image):            Black edge cropping
        def ocr_detector(x,num): detect and place digit in grid
```

- `def print_grid(sudo):` solution printing function in output window
- `def virsol(num_pos):` real time solution display function
- `def rts():` function for assigning digits
- `def quit():` closing GUI windows
- `def close_fol(Path):` removing data from folder for future use
- ***GUI Window code***

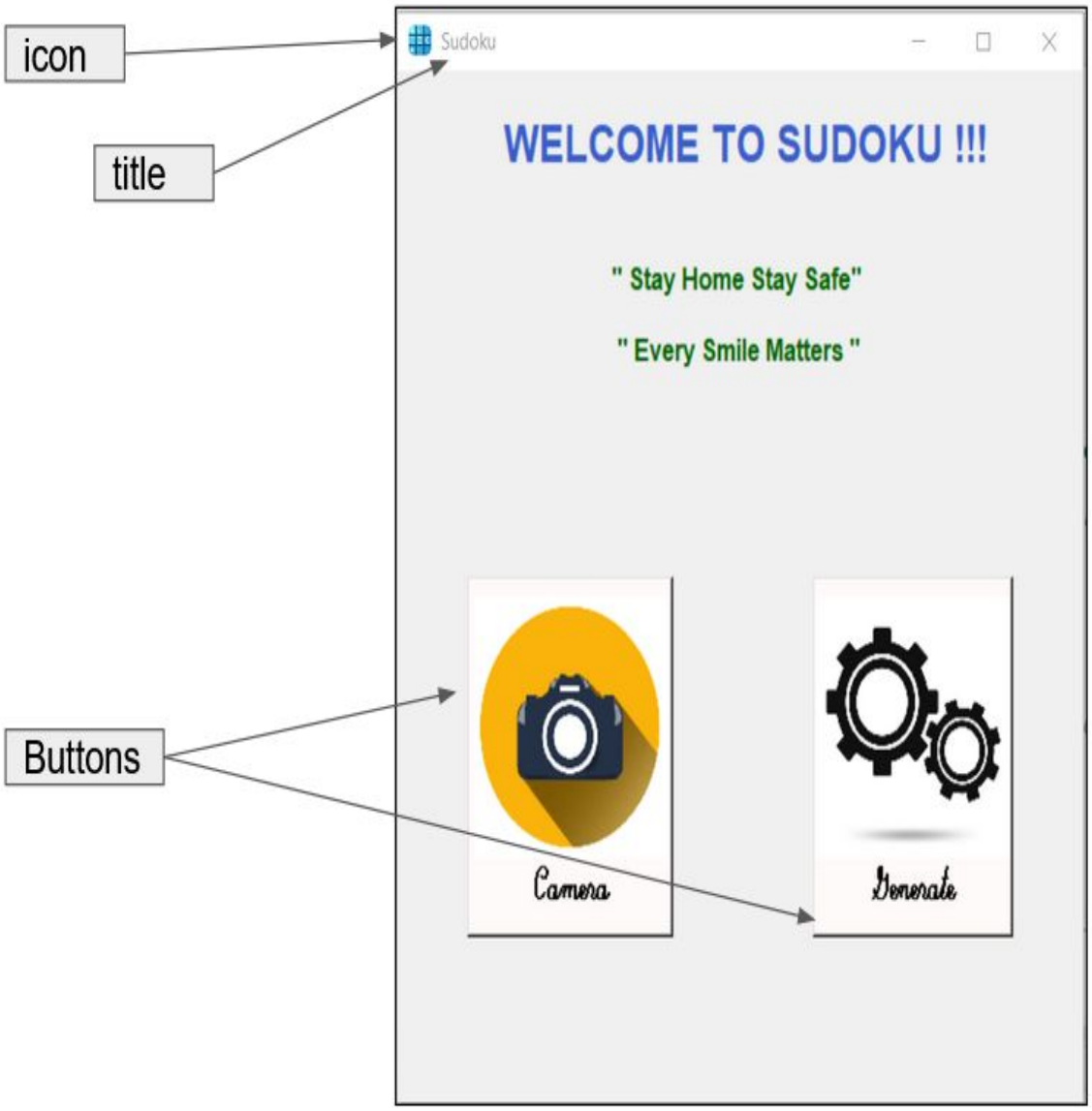
Folders Used:

- **Binary** (for storing Binary image of cells with digits)
- **cells** (for storing all image 81 cells)
- **data_base** (contain images of sudoku used in gen())
- **icons** (contain different icons to used in GUI)

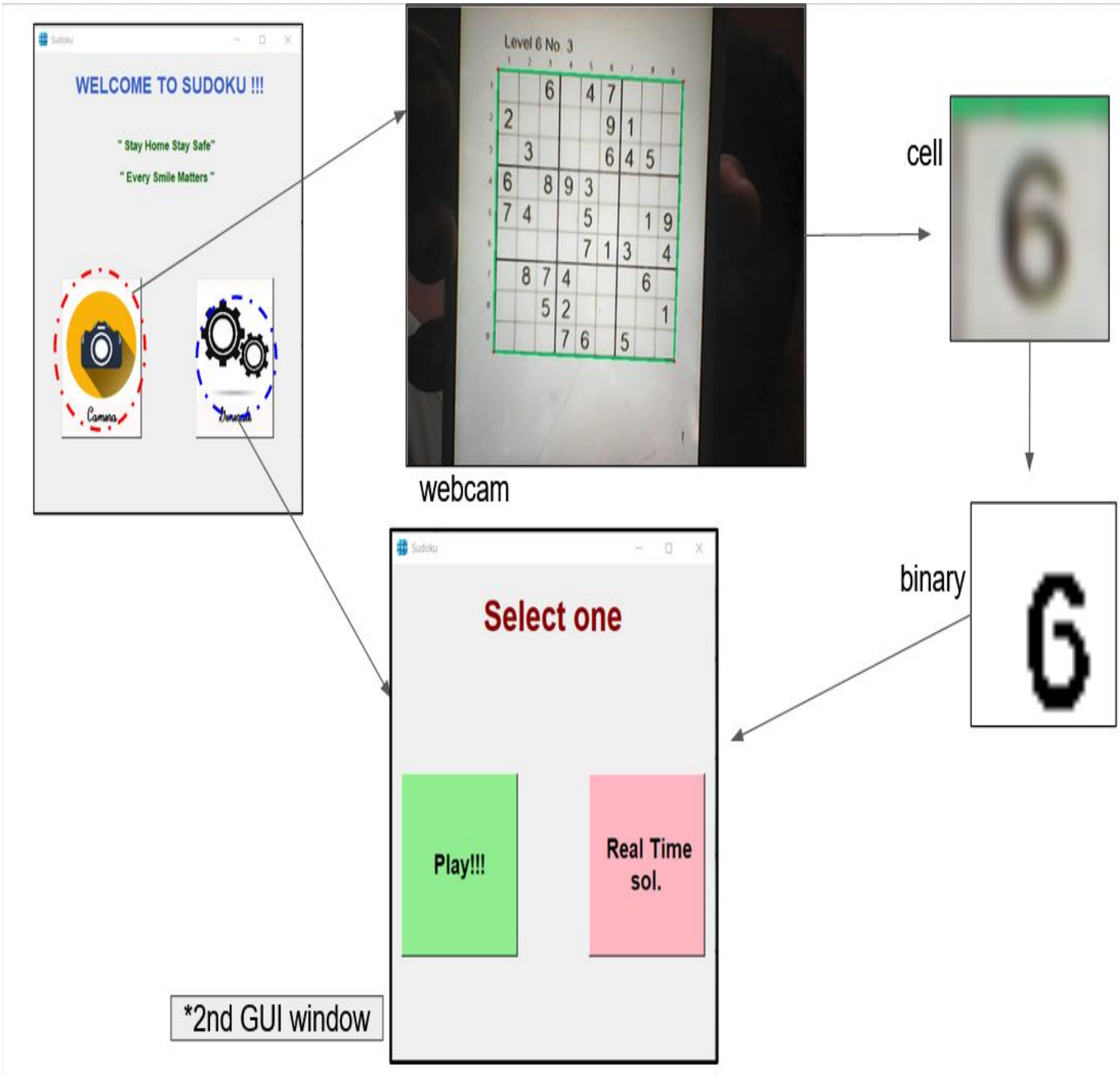
Additional software Used:

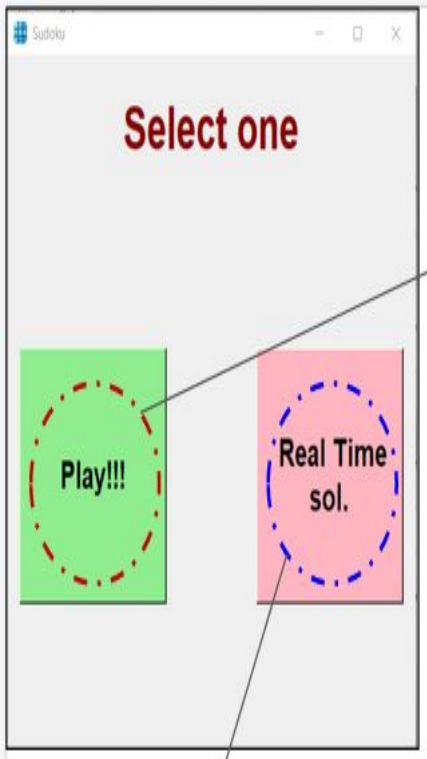
- **tesseract**

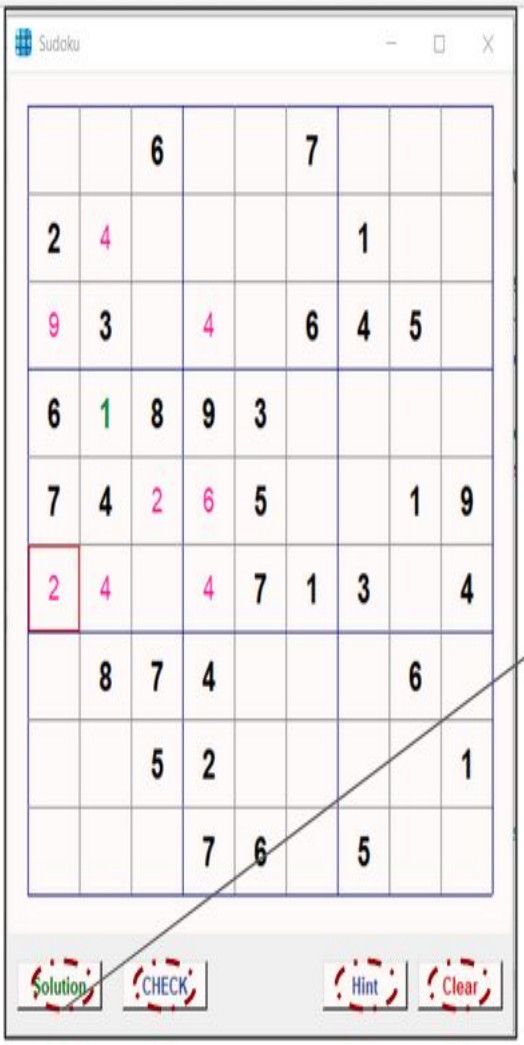
OUTPUT SCREENS

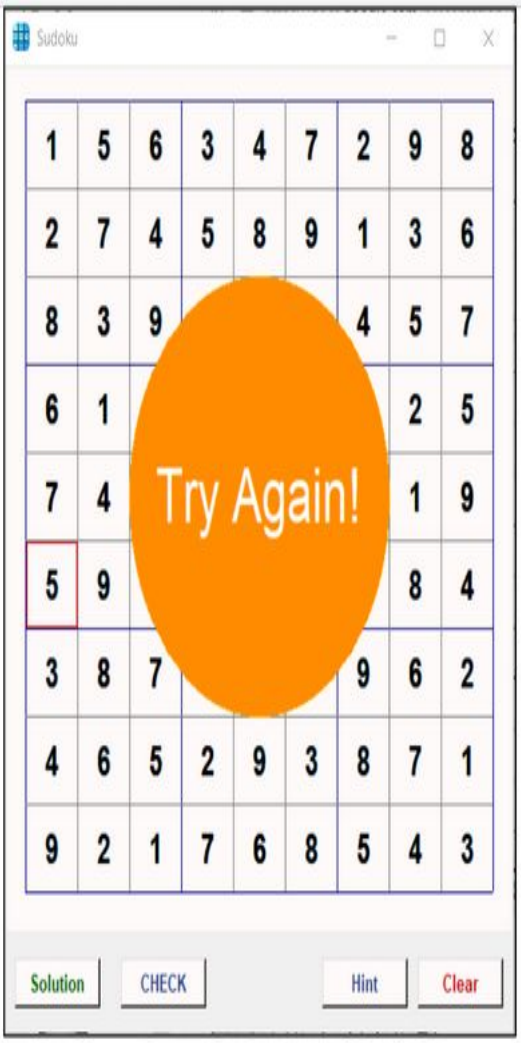


*1st GUI window


WELCOME TO SUDOKU !!!
" Stay Home Stay Safe"
" Every Smile Matters "
Camera
Level 6 No. 3
webcam
cell
binary
Select one
Play!!!
Real Time sol.
*2nd GUI window







*playing platform

*masked real time sol.

COMPLETE CODE

```
import cv2
import copy
import os
import glob
import random
import numpy as np
import pytesseract as pytes
from tkinter import*
from PIL import Image
pytes.pytesseract.tesseract_cmd="Tesseract-OCR\\tesseract.exe"

#******************************** GUI class*************************************

class sudokucls(Frame):

   def __init__(self, root):
       Frame.__init__(self, root)
       self.root = root
       self.row, self.col = -1, -1
       self.r = list()
       self.c = list()
       self.num = list()
       self.hint_i = 0
       self.exe = 0
       self.__incls()

   def __incls(self):                                  #Playing GUI window
       photo = PhotoImage(file="icons/icon.png")
       root.iconphoto(False, photo)
       self.root.title(" Sudoku ")
       self.pack(fill=BOTH)
       self.canvas = Canvas(self, width=490, height=490, bg="snow")

       self.f1 = Frame(self).pack(fill=BOTH, side=BOTTOM)
       self.B1 = Button(self.f1, text="Clear", font="Helvetica 10 bold",
fg="red3", bg="snow", height=1, width=9,
                        command=self.clear_ans) \
           .pack(side=RIGHT, padx=10, pady=10)
       self.B2 = Button(self.f1, text="Hint", font="Helvetica 10 bold",
fg="royalblue4", bg="snow", height=1, width=9,
                        command=self.hint) \
           .pack(side=RIGHT)
       self.B3 = Button(self.f1, text="Solution", font="Helvetica 10 bold",
fg="darkgreen", bg="snow", height=1,
                        width=9, command=self.solution) \
           .pack(side=LEFT, padx=10)
       self.B4 = Button(self.f1, text="CHECK", font="Helvetica 10 bold",
fg="royalblue4", bg="snow", height=1, width=9,
                        command=self.check) \
           .pack(side=LEFT, padx=10)

       self.draw_grid()                        #for drawing grids of 9x9
       self.canvas.pack(fill=BOTH, side=TOP)
       fill_flag = 0
       self.draw_fill(fill_flag)                  #for filling in grids
```

```
        self.canvas.bind("<Button-1>", self.cell_clicked)                #if any
left click then call cell_clicked()

        self.canvas.bind("<Key>", self.key_pressed)                      #if any
number key pressed then call key_pressed()


    def draw_grid(self):                                                 #drawing grids of
9x9
        for i in range(10):
            color = "navy" if i % 3 == 0 else "gray"

            x0 = 20 + i * 50
            y0 = 20
            x1 = 20 + i * 50
            y1 = 470
            self.canvas.create_line(x0, y0, x1, y1, fill=color)

            x0 = 20
            y0 = 20 + i * 50
            x1 = 470
            y1 = 20 + i * 50
            self.canvas.create_line(x0, y0, x1, y1, fill=color)

    def draw_fill(self, fill_flag):                  #filling in grids
        global Usudoku, sudoku

        if (fill_flag == 1):
            k = 0
            for (i, j) in zip(self.r, self.c):
                if self.num[k] != 0:
                    x = 20 + j * 50 + 25
                    y = 20 + i * 50 + 25
                    self.canvas.create_text(x, y, text=self.num[k],
font=("Purisa", 18, "bold"), fill="springGreen4")
                k += 1

        else:
            for i in range(9):
                for j in range(9):
                    if Usudoku[i][j] != 0:
                        x = 20 + j * 50 + 25
                        y = 20 + i * 50 + 25
                        self.canvas.create_text(x, y, text=Usudoku[i][j],
font=("purisa", 18, "bold"), fill="black")

    def highlight_cell(self):                    #highlighting cell in focus
        self.canvas.delete("cursor")
        if self.row >= 0 and self.col >= 0:
            x0 = 20 + self.col * 50 + 1
            y0 = 20 + self.row * 50 + 1
            x1 = 20 + (self.col + 1) * 50 - 1
            y1 = 20 + (self.row + 1) * 50 - 1
            self.canvas.create_rectangle(x0, y0, x1, y1, outline="red",
tags="cursor")
```

```
    def cell_clicked(self, event):
        x, y = event.x, event.y
        if (20 < x < 470 and 20 < y < 470):
            self.canvas.focus_set()

            # get row and col numbers from x,y coordinates
            row, col = (y - 20) // 50, (x - 20) // 50

            # if cell was selected already - deselect it
            if (row, col) == (self.row, self.col):
                self.row, self.col = -1, -1
            elif Usudoku[row][col] == 0:
                self.row, self.col = row, col
        else:
            self.row, self.col = -1, -1

        self.highlight_cell()                    #calling for highlithing cell

    def key_pressed(self, event):

        if self.row >= 0 and self.col >= 0 and event.char in "123456789":
            x = 20 + self.col * 50 + 25
            y = 20 + self.row * 50 + 25
            self.canvas.create_text(x, y, text=(event.char), font=("Purisa", 14),
tags="numbers", fill="deeppink")
            self.r.append(self.row)
            self.c.append(self.col)
            self.num.append(int(event.char))
            self.col, self.row = -1, -1

    def check(self):                        #checking inputs
        flag = 0
        for i in range(len(self.num)):
            if sudoku[self.r[i]][self.c[i]] == self.num[i]:
                Usudoku[self.r[i]][self.c[i]] = self.num[i]
                flag = 1
            else:
                self.num[i] = 0

        self.draw_fill(flag)
        self.r.clear()
        self.c.clear()
        self.num.clear()

        if (int(0) not in Usudoku and Usudoku == sudoku):
            self.victory()

    def clear_ans(self):                        #clearing inputs
        self.canvas.delete("numbers")

    def hint(self):                             #printing a row as hint
        self.exe += 1
        a = [3, 6, 8, 1, 2, 7, 5, 0, 4]
        i = a[self.hint_i]
        y = 20 + i * 50 + 25
        for j in range(9):
            x = 20 + j * 50 + 25
```

```
            if (Usudoku[i][j] & sudoku[i][j] == 0):
                Usudoku[i][j] = sudoku[i][j]
                self.canvas.create_text(x, y, text=sudoku[i][j], font=("Purisa",
18, "bold"), fill="deep sky blue")

        self.hint_i = (self.hint_i + 1) % 9

    def solution(self):                             #show whole solution
        self.exe = 4
        self.clear_ans()
        for i in range(9):
            for j in range(9):
                y = 20 + i * 50 + 25
                x = 20 + j * 50 + 25
                if (Usudoku[i][j] & sudoku[i][j] == 0):
                    Usudoku[i][j] = sudoku[i][j]
                    self.canvas.create_text(x, y, text=sudoku[i][j],
font=("Purisa", 18, "bold"), fill="deep sky blue")

    def victory(self):                              #last message
        x0 = y0 = 120
        x1 = y1 = 370
        self.canvas.create_oval(x0, y0, x1, y1, tags="victory", fill="dark
orange", outline="orange")
        # create text
        x = y = 20 + 4 * 50 + 25
        if self.exe == 0:
            ch = "You win!"
        elif self.exe in [2, 3]:
            ch = "Satisfactory!"
        else:
            ch = "Try Again!"
        self.canvas.create_text(x, y, text=ch, tags="victory", fill="white",
font=("Arial", 32))


#*************** sudoku solver functions module ****************************

def print_grid(sudo):
    for i in sudo:
        print (i)

#---------------------- finding voids ---------------------------------

def find_blank(row, col):
    flag=0
    for i in range(9):
        for j in range(9):
            if(sudoku[i][j]==0):
                row = i
                col = j
                flag= 1
                a=[row,col,flag]
                return a
    a=[-1,-1,flag]
    return a
```

```
#---------------------- checking for possibility-------------------------------

def is_allow(num, row, col):
   for i in range(9):                                  #checking row wise
       if(sudoku[row][i] == num):
           return False

   for i in range(9):                                  #checking col wise
       if(sudoku[i][col] == num):
           return False

   start_row = (row//3)*3
   start_col = (col//3)*3
   for i in range(start_row, start_row + 3):           #checking box wise
       for j in range(start_col, start_col + 3):
           if sudoku[i][j] == num:
               return False

   return True

#----------------------- main solving module ----------------------------

def solving_module():

 row=0
 col=0
 rec=find_blank(row, col)                             #calling for checking voids
 if (rec[2]==0):
     return True
 row=rec[0]
 col=rec[1]

 for num in range(1,10):
     if (is_allow(num, row, col)):                   #calling for possibility func.
        sudoku[row][col]=num

        if (solving_module()):                       #recursive calling
           return True
        sudoku[row][col]=0

 return False                                        #recursion trigger

#****************** Grid and digit detection module function *****************

#cleaning folders

def clean_fol(path):
   files = glob.glob(path)
   for f in files:
       os.remove(f)
#-------------------------------------------------------------------------------

#Black edge croping

def autocrop(image):
   for i in range(28):
       for j in range(28):
```

```
            if (i<=3 or j<=3 or i>=26 or j>=26) and image[i][j]==255:
                image[i][j]=0
    return image
#-------------------------------------------------------------------------------

#using ocr to detect and place digit in solving grid
def ocr_detector(x,num):
    pos=1
    k=0
    for i in range(9):
        for j in range(9):
            if pos == x:                        #getting correct position in
solving grid
                if is_allow(num,i,j):               #only checking for acurracy
                    Usudoku[i][j]=num

            pos+=1

#-------------------------------------------------------------------------------
# function to check the order of points

def proper_order(ar):
    ar1 = [[0, 0], [0, 0], [0, 0], [0, 0]]
    max = 0
    ymax = 0
    min = 99999
    for i in range(len(ar)):
        if (ar[i][0][0] + ar[i][0][1]) > max:
            max = ar[i][0][0] + ar[i][0][1]
            ar1[2][0] = ar[i][0][0]
            ar1[2][1] = ar[i][0][1]
        if (ar[i][0][0] + ar[i][0][1]) < min:
            min = ar[i][0][0] + ar[i][0][1]
            ar1[0][0] = ar[i][0][0]
            ar1[0][1] = ar[i][0][1]

    for i in range(len(ar)):
        if (ar[i][0][0] + ar[i][0][1]) < max and (ar[i][0][0] + ar[i][0][1]) > min
and ar[i][0][1] > ymax:
            ar1[1][0] = ar[i][0][0]
            ar1[1][1] = ar[i][0][1]
            ymax = ar[i][0][1]
    for i in range(len(ar)):
        if (ar[i][0][0] + ar[i][0][1]) < max and (ar[i][0][0] + ar[i][0][1]) > min
and ar[i][0][1] < ymax:
            ar1[3][0] = ar[i][0][0]
            ar1[3][1] = ar[i][0][1]

    return ar

#----------------------- function smoothing module ------------------
#converting BGR -> grayscale -> blurs to remove noise

def blurring_mod(img):
  imgGray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  imgBlur=cv2.GaussianBlur(imgGray,(5,5),1)
  imgCanny=cv2.Canny(imgBlur,10,50)
```

```
  cv2.imshow("canny",imgCanny)
  return imgCanny

#---------------------- chopping 0f cells ----------------------------

def chopping_grid(pt):

   img = cv2.imread("frame.jpg", 1)  # reading captured grid
   flag = 1
   dim = np.float32([[0, 0], [0, 252], [252, 252], [252, 0]])  # converting it
into 28x3x3 grid
   pt = np.float32(pt)

   grid = cv2.getPerspectiveTransform(pt, dim)
   warp_img = cv2.warpPerspective(img, grid, dsize=(252, 252))

   cv2.imwrite('warp_frame.jpg', warp_img )
   cv2.imshow('warp-frame', warp_img)

   for i in range(9):  # loop for chopping cells
       for j in range(9):
           cell = warp_img[(i * 28):((i + 1) * 28), (j * 28):((j + 1) * 28)]
           cell = cv2.GaussianBlur(cell, (5, 5), 1)

           cv2.imwrite('cells/cell{}.png'.format(flag), cell)
           flag = flag + 1

#------------------- separating spaces and numbers ----------------------------

def separate_space():

   noNum = 0
   list_cell = [list()]        # list to store (pos,element)
   num_pos = list()           # list to store number's position

   for k in range(1, 82):
       sum = 0

       cell = cv2.imread("cells/cell{}.png".format(k), 0)

       cell = cv2.adaptiveThreshold(cell, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 3, 2)
       list_cell.append([k, cell])

       # separating module

       for i in range(10, 19):
           for j in range(10, 19):
               sum = sum + cell[i, j]

       j = 81 - (sum / 255)

       if j >= 10:         # separating condition
           noNum += 1
           num_pos.append(k)

   return (noNum,list_cell,num_pos)
```

```
#----------------- removing cell's noises and saving ----------------------------
def remove_cell_noice(num_pos):

   for i in num_pos:
       cell = cv2.imread("cells/cell{}.png".format(i), 0)
       # for sharpening of image

       cell = cv2.adaptiveThreshold(cell, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, 3, 2)
       f2, cell = cv2.threshold(cell, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

       cell = autocrop(cell)          # removing borders of cells

       # removing noise
       kernel = np.ones((2, 2), np.uint8)
       cell = cv2.dilate(cell, kernel, iterations=1)
       cell = cv2.erode(cell, kernel)

       max_con = [0]
       max_area = 0
       cell_copy = cell.copy()
       contours, hie = cv2.findContours(cell, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

       for j in range(len(contours)):
           if cv2.contourArea(contours[j]) > max_area:
               max_area = cv2.contourArea(contours[j])
               max_con = contours[j]

       cv2.drawContours(cell, [max_con], -1, (255, 255, 255), -1)
       f3, thresh = cv2.threshold(cell, 254, 255, cv2.THRESH_BINARY)

       cell = cv2.bitwise_and(cell_copy, cell_copy, mask=thresh)
       f4, cell = cv2.threshold(cell, 254, 255, cv2.THRESH_BINARY_INV)

       cell = cv2.dilate(cell, kernel)
       cell = cv2.erode(cell, kernel)

       cv2.imwrite("binary/cell_binary{}.png".format(i), cell)          # saving
of filtered cells

       if i == len(num_pos):
           break

# ---------------using pytesseract OCR to recognise the num-------------------
       img_ocr = Image.open("binary/cell_binary{}.png".format(i))
       string = pytes.image_to_string(img_ocr, lang='eng',config='--psm 10 --oem
3 -c tessedit_char_whitelist=0123456789')
       if string[0] >= '1' and string[0] <= '9':
           ocr_detector(i, int(string[0]))
```

```
#---------------- virtual solution display -----------------------------

def vir_sol(num_pos):

   dst = cv2.imread("warp_frame.jpg",1)

   font = cv2.FONT_HERSHEY_SCRIPT_COMPLEX    #font of text
   for i in range(9):
       for j in range(9):
           if (sudoku[i][j]!=0) and ((i*9)+j+1) not in num_pos:
               x = (j * 28 + 8)
               y = (i * 28 + 22)
               dig = str(sudoku[i][j])           #reading digits as string

               cv2.putText(dst,dig,(x,y),font,0.65,(200,100,50),1,2)
#putting text on image

           vir_img = cv2.resize(dst, None, fx=3, fy=3,
interpolation=cv2.INTER_CUBIC)          #resizing image
           cv2.imshow('SUDOKU!!!', vir_img)          #display window
           cv2.waitKey(40)
   cv2.waitKey(0)
   cv2.destroyAllWindows()



#------------------ Getting grid from webcam -----------------------------

def cam():
   # use of webcam to capture image of sudoku
   global pt
   count = 0
   flag = 1

   cap = cv2.VideoCapture(0)  # web cam ON
   while (cap.isOpened()):

       f1, img = cap.read()  # reading frames from web cam

       imgCanny = blurring_mod(img)  # calling smoothing func
       # finding sudoku grid

       contours, hie = cv2.findContours(imgCanny, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

       max_con = [0]  # maxc for maximum contours
       max_area = 0  # max is variable for maximum area

       for i in range(len(contours)):  # loop to find largest contour in given
frame

           peri = cv2.arcLength(contours[i], True)  # finding perimeter of
contour
           approx = cv2.approxPolyDP(contours[i], 0.011 * peri, True)  # contour
approximation
           (x, y, w, h) = cv2.boundingRect(approx)
           # checking maximum contours
```

```
            if cv2.contourArea(contours[i]) > 10000 and
cv2.contourArea(contours[i]) > max_area \
                    and len(approx) == 4 and float(w) / h == 1:
                max_area = cv2.contourArea(contours[i])
                max_con = approx

        # check for four corners then drawing contours
        if len(max_con) == 4:
            count = count + 1
        else:
            count = 0
        if len(max_con) == 4:
            cv2.drawContours(img, [max_con], -1, (100, 200, 0), 2)
            cv2.drawContours(img, max_con, -1, (0, 0, 255), 4)

        flag = 0
        cv2.imshow('Contours', img)  # displaying contours
        cv2.waitKey(1)
        if count == 4:
            cv2.imwrite("frame.jpg", img)  # saving that frame
            pt = max_con
            flag = 1

        if flag == 1:
            break

    cv2.destroyAllWindows()
    cap.release()

    im = cv2.imread("icons/wait.png")
    cv2.imshow("Loading Image", im)
    cv2.waitKey(1)

    pt = proper_order(pt)  # function to set proper order of points

    chopping_grid(pt)  # function for chopping grid in 9x9

#------------- generating function ------------------------------------------

def gen():
    im=cv2.imread("icons/wait.png")
    cv2.imshow("Loading Image",im)
    cv2.waitKey(1)
    img = cv2.imread("data_base/{}.png".format(random.randint(1,40)))
    img = cv2.resize(img, (252, 252))
    cv2.imwrite("warp_frame.jpg", img)
    flag = 1
    for i in range(9):  # loop for chopping cells
        for j in range(9):
            cell = img[(i * 28):((i + 1) * 28), (j * 28):((j + 1) * 28)]
            cell = cv2.GaussianBlur(cell, (5, 5), 1)

            cv2.imwrite('cells/cell{}.png'.format(flag), cell)
            flag = flag + 1
```

```
def rts():
   global rts_var
   rts_var=1

def quit():
   root.destroy()

#------------ A beginning of new tour of learning ----------------------------


pt=None
rts_var=0
Usudoku = [[0 for x in range(9)] for y in range(9)]
sudoku = [[0 for x in range(9)] for y in range(9)]

root=Tk()
root.geometry("%dx%d" % (490,490+60))
photo = PhotoImage(file="icons/icon.png")
root.iconphoto(False, photo)
root.title(" Sudoku ")
frame=Frame().pack()
Label(frame,text=" WELCOME TO SUDOKU
!!!",font=("Comicsansms",20,"bold"),fg="royalblue3").pack(side=TOP,
fill="x",pady=20)
Label(frame,text='" Stay Home Stay Safe" \n\n" Every Smile Matters
"',font=("Comicsansms",12,"bold"),fg="darkgreen").pack(side=TOP,
fill="x",pady=20)
photo=PhotoImage(file="icons/cam.png")
photo=photo.subsample(3,3)
Button(frame,text="Camera",font=("script", 20,"bold"), image=photo,
compound=TOP,bg="snow",command=lambda:[cam(),quit()]).pack(side=LEFT, padx=50)
photo2=PhotoImage(file="icons/generate.png")
photo2=photo2.subsample(3,3)
Button(frame,text="Generate",font=("script", 20,"bold"), image=photo2,
compound=TOP,bg="snow",command=lambda:[gen(),quit()]).pack(side=LEFT,padx=50)
root.mainloop()

noNum, list_cell, num_pos = separate_space()      # function for separating
numbers

remove_cell_noice(num_pos)     # function to clean cell noice and save cell

sudoku=copy.deepcopy(Usudoku)

solving_module()          #calling solving module

cv2.destroyAllWindows()

root = Tk()
root.geometry("500x500")
photo = PhotoImage(file="icons/icon.png")
root.iconphoto(False, photo)
root.title(" Sudoku ")
Label(root,text=" Select one ",font=("Arial",
32,"bold"),fg="red4").pack(fill=BOTH, pady=25)
frame=Frame().pack(side=BOTTOM)
Button(frame,text=" Play!!! ",font=("Arial", 20,"bold"),relief=RAISED,bg="light
```

```
green",height=5,width=10,command=lambda:[quit()]).pack(side=LEFT,padx=15)
Button(frame,text=" Real Time\n sol. ",font=("Arial",
20,"bold"),relief=RAISED,bg="light
pink",height=5,width=10,command=lambda:[rts(),quit()]).pack(side=RIGHT,padx=15)
root.mainloop()

if(rts_var==1):
  vir_sol(num_pos)            #function for virtual display

elif(rts_var==0):
   root = Tk()
   root.geometry("%dx%d" % (490, 490 + 60))
   sudokucls(root)                #class of final GUI
   root.mainloop()


clean_fol("binary/*.png")           #removing data from file
clean_fol("cells/*.png")            #removing data from file
```

APPLICATIONS

1. *Real time solution*
2. *Input from hard sudoku puzzle*
3. *Making puzzles easy to solve as well as inveractive*

REFERENCES

1. *Geeks for Geeks*

2. *Opencv.org*

3. *Tkinter.org*

4. *Google Scholar*

5. *Stack Overflow*