Appendix

Sectioned Code

The code below is updated to fix faults found during testing.

Login System

(login.py)

```
#imports

from tkinter import*

from tkmacosx import Button

import PIL.ImageTk
import PIL.Image

import mysql.connector
import cv2
import numpy as np
import HandTrackingModule as htm
import time
import autopy
import threading

#main loop launches program window and loops infinitely until program is
terminated
def main():
    window = Tk()
    app = LoginWindow(window)
    window.mainloop()

#initialises object attributes and formats size and name of login window
class LoginWindow:
    def __init__(self, root):
        self.root = root
        self.root.title("Login")
        self.root.geometry("1550x800+0+0")

#sets background image(retrieves, resizes and sets position of image)
      self.bg=PIL.ImageTk.PhotoImage(file=r"/Users/rakesharavind/Downloads/
pastel-memphis-blog-banner-template/nea_bg2.jpg")
        lbl_bg = Label(self.root, image = self.bg)
        lbl_bg.place(x=0, y=0, relwidth=1, relheight=1)

#creates light blue box
        frame = Frame(self.root, bg="light blue")
        frame.place(x=550, y=100, width=340, height=530)

#sets profile icon
    img1=PIL.Image.open(r"/Users/rakesharavind/Downloads/person_icon3.png")
```

```
        img1 = img1.resize((100,100), PIL.Image.ANTIALIAS)
        self.photoimage1 = PIL.ImageTk.PhotoImage(img1)
        lbl_img1 = Label(image=self.photoimage1, bg="light blue",
borderwidth=0)
        lbl_img1.place(x=667, y=105, width=100, height=100)

#sets text
        texttop = Label(frame,text="Get Started", font=("veranda", 30,
"bold"), fg="black", bg="light blue")
        texttop.place(x=85, y=100)

        username = Label(frame,text="Username", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        username.place(x=15, y=150)

#creates and positions box where user can input text (username)
        self.textuser = ttk.Entry(frame, font=("veranda", 25))
        self.textuser.place(x=15, y=190, width=300)

#sets text
        password = Label(frame, text="Password", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        password.place(x=15, y=245)

#creates and positions box where user can input text (password)
        self.textpass = ttk.Entry(frame, font=("veranda", 25))
        self.textpass.place(x=15, y=285, width=300)

#creates, positions and defines buttons (design, location and function
called when button is clicked)
        LoginBtn = Button(frame, text="Login", command=self.login,
font=("veranda", 25, "bold"), fg="black", bg="#3895d3",
activeforeground="white", activebackground="#9e7bb5")
        LoginBtn.place(x=110, y=350, width=120, height=50)

        RegisterBtn = Button(frame, text="New User Registeration",
command=self.registerWindow, font=("veranda", 15, "bold"), anchor="w",
fg="black", bg="light blue", activeforeground="white",
activebackground="#3895d3")
        RegisterBtn.place(x=15, y=440, width=180, height=30)

        ForgotBtn = Button(frame, text="Forgot Password?",
command=self.forgotPassword, font=("veranda", 15, "bold"), anchor="w",
fg="black", bg="light blue", activeforeground="white",
activebackground="#3895d3")
        ForgotBtn.place(x=15, y=475, width=140, height=30)

#specifies that register window is to open above the login window when the
function is called
    def registerWindow(self):
        self.newWindow = Toplevel(self.root)
        self.app = Register(self.newWindow)
```

```
#validates login username & password
    def login(self):
        if self.textuser.get() == "" or self.textpass.get() == "":
            messagebox.showerror("Error", "All fields are required")
#checks that fields/input boxes are not left empty
        else:
            conn = mysql.connector.connect(host="localhost", user="root",
        password="Test@123", database="mydata")#connects to MySQL database
#creates object 'cursor' which is used to execute query
            myCursor = conn.cursor()
#executes query to retrieve username and password from database
#replaces placeholder '%s' with data user inputs into input box; data is
stored in the global variables self.textuser and self.textpass and fetched
using the get() function
            myCursor.execute("select * from register where Username=%s and
        Password=%s", (
                self.textuser.get(),
                self.textpass.get()
            ))
            row = myCursor.fetchone()

#if username or password is invalid, pop up error message tells the user
            if row == None:
                messagebox.showerror("Error", "Invalid username or
password")
#if validated, open sudoku game and virtual mouse
#pop up message box asks users if they want to start playing the Sudoku
game
else:
                openMain = messagebox.askyesno("Play", "Start playing now?")

#if user clicks 'yes', Sudoku Puzzle window opens ontop of the Login window
                if openMain > 0:
                    self.newWindow = Toplevel(self.root)
#the App class is instantiated and run, opening the Sudoku Puzzle window
                    self.app = App()
                    self.app.run()

#threading is used to allow the Sudoku puzzle and Virtual Mouse window to
open and run at the same time
                    thread = threading.Thread(target=self.app.run())
                    thread.start()

#the Mouse class is instantiated and run, opening the Sudoku Puzzle window
ontop of the Login window
                    self.newWindow = Toplevel(self.root)
                    self.app = Mouse(self.newWindow)

#if the user clicks 'no', no action is taken
                else:
                    if not openMain:
                        return
```

```
#confirms/commits to the aforementioned transaction (necessary since Python
does not auto-commit)
            conn.commit()
            conn.close()#closes connection to database

#gives the user an option to reset their password
    def resetPassword(self):

#ensures no fields are left blank
        if self.comboQuestions.get() == "Select":
            messagebox.showerror("Error", "Select your security question",
parent=self.root2)
        elif self.textsecurityans.get() == "":
            messagebox.showerror("Error", "Please enter the answer to your
security question", parent=self.root2)
        elif self.textnewpass.get() == "":
            messagebox.showerror("Error", "Please enter your new password",
parent=self.root2)

#validates input data by comparing it to data stored in database
        else:
            conn = mysql.connector.connect(host="localhost", user="root",
password="Test@123", database="mydata")
            myCursor = conn.cursor()
            query2 = ("select * from register where Username=%s and
SecurityQ=%s and SecurityA=%s")
            value2 = (self.textuser.get(), self.comboQuestions.get(),
self.textsecurityans.get())
            myCursor.execute(query2, value2)
            row = myCursor.fetchone()

#if no identical records are found in the database, the user input data is
invalid and an error message is shown to convey this information
              if row == None:
                messagebox.showerror("Error", "Answer to security question
is incorrect", parent=self.root2)
#otherwise, user input data is valid and the register table is updated with
the new password
            else:
                query = ("update register set Password=%s where
Username=%s")
                value = (self.textnewpass.get(), self.textuser.get())
                myCursor.execute(query, value)

                conn.commit()
                conn.close()
                messagebox.showinfo("Reset Password", "Your password has
been reset. Please login with your new password.", parent=self.root2)
                self.root2.destroy()

#initialises and formats forgot password window
    def forgotPassword(self):
```

```
 #ensures username field is not left blank
      if self.textuser.get()=="":
           messagebox.showerror("Error", "Please enter your username to
reset password")

#validates username by comparing it to usrenames stored in database
      else:
           conn = mysql.connector.connect(host="localhost", user="root",
password="Test@123", database="mydata")
           myCursor = conn.cursor()
           query = ("select * from register where Username=%s")
           value = (self.textuser.get(),)
           myCursor.execute(query, value)
           row = myCursor.fetchone()

            if row == None:
               messagebox.showerror("Error", "Please enter a valid
username")

#specifies that Forgot Password window is to appear above the Login window
           else:
               conn.close()
               self.root2 = Toplevel()
               self.root2.title("Forgot Password")
               self.root2.geometry("340x530+550+130")

#sets frames, text and input/combo boxes that make up the design of the
Forgot Password window
               frame2 = Frame(self.root2, bg="light blue")
               frame2.place(width=340, height=530)

               texttop2 = Label(self.root2, text="Reset Password",
font=("veranda", 30, "bold"), fg="black", bg="light blue")
               texttop2.place(x=0, y=10, relwidth=1)

               securityQ = lbl = Label(self.root2, text="Security
Question", font=("veranda", 25, "bold"), fg="black",
                                       bg="light blue")
               securityQ.place(x=50, y=80)

               self.comboQuestions = ttk.Combobox(self.root2,
font=("veranda", 15),state="readonly")
               self.comboQuestions["values"] = (
               "Select", "In what city were you born?", "What is the name
of your favourite pet?",
               "What high school did you attend?", "What was your favourite
food as a child?",
               "What was the make of your first car?")
               self.comboQuestions.place(x=50, y=120, width=200)
               self.comboQuestions.current(0)

               securityAns = lbl = Label(self.root2, text="Answer",
font=("veranda", 25, "bold"), fg="black",
```

```
                                        bg="light blue")
                securityAns.place(x=50, y=180)

                self.textsecurityans = ttk.Entry(self.root2,
font=("veranda", 25))
                self.textsecurityans.place(x=50, y=220, width=200)

                newPass = lbl = Label(self.root2, text="New Password",
font=("veranda", 25, "bold"), fg="black",
                                        bg="light blue")
                newPass.place(x=50, y=280)

                self.textnewpass = ttk.Entry(self.root2, font=("veranda",
25))
                self.textnewpass.place(x=50, y=320, width=200)

#creates reset button that will call resetPassword() function when clicked
                ResetBtn = Button(self.root2, text="Reset",
command=self.resetPassword, font=("veranda", 25, "bold"), fg="black",
bg="#3895d3", activeforeground="white", activebackground="#9e7bb5")
                ResetBtn.place(x=110, y=400, width=120, height=50)

#creates a new class
#initialises object attributes and formats size and name of register window
class Register:
    def __init__(self, root):
        self.root = root
        self.root.title("Register")
        self.root.geometry("1600x900+0+0")

#StringVar() function ensures all inputs are string types so they can be
stored in the database
        self.varFname = StringVar()
        self.varLname = StringVar()
        self.varSecurityQ = StringVar()
        self.varSecurityA = StringVar()
        self.varUsername = StringVar()
        self.varPassword= StringVar()
        self.varConfpassword = StringVar()

#retrieves and resizes background image
        self.bg = PIL.ImageTk.PhotoImage(
file=r"/Users/rakesharavind/Downloads/pastel-memphis-blog-banner-template/n
ea_bg3.jpg")
        lbl_bg2 = Label(self.root, image=self.bg)
        lbl_bg2.place(x=0, y=0, relwidth=1, relheight=1)

#sets frames, text and input/combo boxes that make up the design of the
Forgot Password window
        frame = Frame(self.root, bg="light blue")
        frame.place(x=370, y=80, width=700, height=570)
```

```
        RegisterLabel = Label(frame, text="Register Here", font=("veranda",
30, "bold"), fg="black", bg="light blue")
        RegisterLabel.place(x=250, y=20)

        fname = lbl = Label(frame, text="First Name", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        fname.place(x=15, y=100)

        self.textfname = ttk.Entry(frame, textvariable=self.varFname,
font=("veranda", 25))
        self.textfname.place(x=15, y=140, width=300)

        lname = lbl = Label(frame, text="Last Name", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        lname.place(x=385, y=100)

        self.textlname = ttk.Entry(frame, textvariable=self.varLname,
font=("veranda", 25))
        self.textlname.place(x=385, y=140, width=300)

        user = lbl = Label(frame, text="Username", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        user.place(x=15, y=195)

        self.textuserinput = ttk.Entry(frame,textvariable=self.varUsername,
font=("veranda", 25))
        self.textuserinput.place(x=15, y=235, width=300)

        passw = lbl = Label(frame, text="Password", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        passw.place(x=15, y=290)

        self.textpassinput = ttk.Entry(frame, textvariable=self.varPassword,
font=("veranda", 25))
        self.textpassinput.place(x=15, y=330, width=300)

        confpass = lbl = Label(frame, text="Confirm Password",
font=("veranda", 25, "bold"), fg="black", bg="light blue")
        confpass.place(x=385, y=290)

        self.textconfpass = ttk.Entry(frame,
textvariable=self.varConfpassword, font=("veranda", 25))
        self.textconfpass.place(x=385, y=330, width=300)

        securityQ = lbl = Label(frame, text="Security Question",
font=("veranda", 25, "bold"), fg="black", bg="light blue")
        securityQ.place(x=15, y=385)

        self.comboQuestions = ttk.Combobox(frame,
textvariable=self.varSecurityQ, font=("veranda", 15), state="readonly")
        self.comboQuestions["values"] = ("Select", "In what city were you
born?", "What is the name of your favourite pet?", "What high school did
```

```
you attend?", "What was your favourite food as a child?", "What was the
make of your first car?")
        self.comboQuestions.place(x=15, y=425, width=300)
        self.comboQuestions.current(0)

        securityAns = lbl = Label(frame, text="Answer", font=("veranda", 25,
"bold"), fg="black", bg="light blue")
        securityAns.place(x=385, y=385)

        self.textsecurityans = ttk.Entry(frame,
textvariable=self.varSecurityA, font=("veranda", 25))
        self.textsecurityans.place(x=385, y=425, width=300)

        registerNowbtn = Button(frame, command=self.registerData,
text="Register Now", font=("veranda", 25, "bold"), fg="black",bg="#3895d3",
activeforeground="white", activebackground="#9e7bb5")
        registerNowbtn.place(x=160, y=500, width=170, height=50)

        goTologinBtn = Button(frame, text="Login",command=self.returnLogin,
font=("veranda", 25, "bold"), fg="black",bg="light blue",
activeforeground="white", activebackground="#3895d3")
        goTologinBtn.place(x=370, y=500, width=120, height=50)

#validates user inputs & saves inputs  to database if valid
    def registerData(self):
        #ensures no fields are left empty
        if self.varFname.get() == "" or self.varLname.get()=="" or
self.varUsername.get()=="" or self.varPassword.get()=="" or
self.varConfpassword.get()=="" or self.varSecurityQ.get()=="Select" or
self.varSecurityA.get()=="" :
            messagebox.showerror("Error", "All fields are required")
        #ensures password and confirm password match
        elif self.varPassword.get() != self.varConfpassword.get():
            messagebox.showerror("Invalid Password", "Your password and
confirmation password do not match")
        #compares user details to records already stored in database to
ensure no repeated usernames
        else:
            conn = mysql.connector.connect(host="localhost", user="root",
password="Test@123", database="mydata")
            myCursor = conn.cursor()
            query = ("select * from register where Username=%s")
            myValue = (self.varUsername.get(),)
            myCursor.execute(query, myValue)
            row = myCursor.fetchone()
            if row != None:
                messagebox.showerror("Error", "This user already exists.
Please try a different username.")
            #retrieves user details from input boxes and saves in database
if valid
            else:
                myCursor.execute("insert into register
values(%s,%s,%s,%s,%s,%s)", (
```

```
                    self.varFname.get(),
                    self.varLname.get(),
                    self.varUsername.get(),
                    self.varPassword.get(),
                    self.varSecurityQ.get(),
                    self.varSecurityA.get()
                ))
                conn.commit()
                conn.close()
                #prompts the user to return to Login window to be able to
login and play the game
                messagebox.showinfo("Success!", "You have been registered.
            Login to play.")

#destroys register/forgot password window and return to login window
    def returnLogin(self):
        self.root.destroy()

#imports programs/windows to open after successful login
from VirtualMouse import *
from SudokuGame import *
```

Sudoku Puzzle

(SudokuGame.py)

```
#imports
import pygame
import sys
import requests
from bs4 import BeautifulSoup
from tkinter import messagebox

#creates new class and initialise objects
class App:
    def __init__(self):
        pygame.init()
        self.window = pygame.display.set_mode((WIDTH, HEIGHT))
        self.running = True
        self.selected = None
        self.mousePos = None
        self.state = "playing"
        self.finished = False
        self.cellChanged = False
        self.playingButtons = [] #stores all buttons on window
        self.lockedCells = [] #stores locked cells when retrieved after
parsing
        self.incorrectCells = [] #stores incorrect cells after the 'Check
puzzle' algorithm is deployed
        self.font = pygame.font.SysFont("veranda", cellSize)
        self.grid = []
        self.getPuzzle("1")#displays an easy puzzle when window first opens
```

```
        self.load()

#initialise window; set up grid and buttons
    def run(self):
        while self.running:
            if self.state == "playing":
                self.playing_events()
                self.playing_update()
                self.playing_draw()
        pygame.quit()
        sys.exit()

###### PLAYING FUNCTIONS #####

    def playing_events(self):
        #changes self.running variable value to terminate the while loop
above, causing the window to close if the user closes the window
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False

            #selects button if user clicks on it
            if event.type == pygame.MOUSEBUTTONDOWN:
                selected = self.mouseOnGrid()
                if selected:
                    self.selected = selected
                else:
                    self.selected = None
                    for button in self.playingButtons:
                        if button.highlighted:
                            button.click()

            #if the user types a key when a non-locked cell is selected
            if event.type == pygame.KEYDOWN:
                if self.selected != None and self.selected not in
self.lockedCells:
                    if self.isInt(event.unicode):
                        #the cell is updated with the new input value
                        self.grid[self.selected[1]][self.selected[0]] =
int(event.unicode)
                        self.cellChanged = True

    #if the user presses onscreen keypad when a non-locked cell is selected,
the cell is updated with the new input value
    def handleNumericButtonPress(self,value):
        if self.selected != None and self.selected not in self.lockedCells:
            self.grid[self.selected[1]][self.selected[0]] = value
            self.cellChanged = True

    #updates screen to reflect actions carried out when user 'clicks'
    def playing_update(self):
        self.mousePos = pygame.mouse.get_pos()
        for button in self.playingButtons:
```

```
            button.update(self.mousePos)

        if self.cellChanged:
            self.incorrectCells = []
             #check if board is correct if the user has completed the puzzle
              if self.allCellsDone():
                self.checkAllCells()
                #if there are no incorrect cells in the incorrectCells
array, display message congratulating the user on completing the pizzle
correctly
                if len(self.incorrectCells) == 0:
                    self.finished = True
                    messagebox.showinfo("Success!", "Congratulations! You
have solved the puzzle correctly.")

    #formats the design of the window; draws buttons and grid on window
    def playing_draw(self):
        self.window.fill(WHITE)

        for button in self.playingButtons:
            button.draw(self.window)

        if self.selected:
            self.drawSelection(self.window, self.selected)

        self.shadeLockedCells(self.window, self.lockedCells)
        self.shadeIncorrectCells(self.window, self.incorrectCells)

        self.drawNumbers(self.window)

        self.drawGrid(self.window)
        pygame.display.update()
        self.cellChanged = False

##### CHECKING FUNCTIONS #####

    #identifies if the Sudoku puzzle has been completed by the user
    def allCellsDone(self):
        for row in self.grid:
            for number in row:
                if number == 0:
                    return False
        return True

    #'Check Puzzle' function
    #checks puzzle for incorrect values (repeated digits in each row, column
and 3 by 3 grid)
    def checkAllCells(self):
        self.checkRows()
        self.checkCols()
        self.checkSmallGrid()
```

```
   #iterates through each 3 by 3 grid in the puzzle, checking for repeated
digits
   def checkSmallGrid(self):
       for x in range(3):
           for y in range(3):
               possibles = [1,2,3,4,5,6,7,8,9]
               for i in range(3):
                   for j in range(3):
                       xidx = x*3+i
                       yidx = y*3+j
                       if self.grid[yidx][xidx] in possibles:
                           possibles.remove(self.grid[yidx][xidx])
                       else:
                           if [xidx, yidx] not in self.lockedCells and
[xidx, yidx] not in self.incorrectCells:
                               self.incorrectCells.append([xidx, yidx])
#ensures first cell checked is not always marked correct
#when a cell has a value that's not in the possibles array, the program
iterates through the 3 by 3 grid once more to check if the number is
already in the grid
                           if [xidx, yidx] in self.lockedCells:
                            for k in range(3):
                                for l in range(3):
                                    xidx2 = x*3+k
                                    yidx2 = y*3+l

#if it finds a cell with the same value, that cell is appended to the
incorrectClls array instead
                                    if self.grid[yidx2][xidx2] ==
self.grid[yidx][xidx] and [xidx2, yidx2] not in self.lockedCells:

self.incorrectCells.append([xidx2, yidx2])

   #iterates through each row in the puzzle, checking for repeated digits
   def checkRows(self):
       for yidx, row in enumerate(self.grid):
           possibles = [1,2,3,4,5,6,7,8,9]
           for xidx in range(9):
               if self.grid[yidx][xidx] in possibles:
                   possibles.remove(self.grid[yidx][xidx])
               else:
                   if [xidx, yidx] not in self.lockedCells and [xidx, yidx]
not in self.incorrectCells:
                       self.incorrectCells.append([xidx, yidx])
#ensures first cell checked is not always marked correct
#when a cell has a value that's not in the possibles array, the program
iterates through the row once more to check if the number is already in the
grid
                   if [xidx, yidx] in self.lockedCells:
                       for k in range(9):
                           if self.grid[yidx][k] == self.grid[yidx][xidx]
and [k, yidx] not in self.lockedCells:
                               self.incorrectCells.append([k, yidx])
```

```
   #iterates through each column in the puzzle, checking for repeated
digits
   def checkCols(self):
       for xidx in range(9):
           possibles = [1,2,3,4,5,6,7,8,9]
           for yidx, row in enumerate(self.grid):
               if self.grid[yidx][xidx] in possibles:
                   possibles.remove(self.grid[yidx][xidx])
               else:
                   if [xidx, yidx] not in self.lockedCells and [xidx, yidx]
not in self.incorrectCells:
                       self.incorrectCells.append([xidx, yidx])
#ensures first cell checked is not always marked correct
#when a cell has a value that's not in the possibles array, the program
iterates through the column once more to check if the number is already in
the grid
                   if [xidx, yidx] in self.lockedCells:
                       for k, row in enumerate(self.grid):
                           if self.grid[k][xidx] == self.grid[yidx][xidx]
and [xidx, k] not in self.lockedCells:
                               self.incorrectCells.append([xidx, k])

##### HELPER FUNCTIONS #####

  #scrapes puzzle from HTML website
   def getPuzzle(self, difficulty):
       #sends a request to website for its data
       #sorts content scraped based on difficluty
       #difficulty passed in as string (1-3)based on button clicked by user
       html_doc =
requests.get("https://nine.websudoku.com/?level={}".format(difficulty)).con
tent
       soup = BeautifulSoup(html_doc)

#array of cell IDs so program can pick out appropriate cells in source code
when parsing
       ids = ['f00', 'f01', 'f02', 'f03', 'f04', 'f05', 'f06', 'f07',
'f08', 'f10', 'f11',
       'f12', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f20', 'f21',
'f22', 'f23',
       'f24', 'f25', 'f26', 'f27', 'f28', 'f30', 'f31', 'f32', 'f33',
'f34', 'f35',
       'f36', 'f37', 'f38', 'f40', 'f41', 'f42', 'f43', 'f44', 'f45',
'f46', 'f47',
       'f48', 'f50', 'f51', 'f52', 'f53', 'f54', 'f55', 'f56', 'f57',
'f58', 'f60',
       'f61', 'f62', 'f63', 'f64', 'f65', 'f66', 'f67', 'f68', 'f70',
'f71', 'f72',
       'f73', 'f74', 'f75', 'f76', 'f77', 'f78', 'f80', 'f81', 'f82',
'f83', 'f84',
       'f85', 'f86', 'f87', 'f88']
```

```
#stores the values of the Sudoku grid after parsing
        data = []

#searches for cells in Sudoku grid and appends their values to the data
array
        for cid in ids:
            data.append(soup.find('input', id=cid))
        board = [[0 for x in range(9)] for x in range(9)]
        for index, cell in enumerate(data):
            try:
                board[index//9][index%9] = int(cell['value'])
            except:
                pass

#makes board created the Sudoku grid for the game and reloads window to
have it displayed to the user
        self.grid = board
        self.load()

    #shades incorrect cells red when 'Check Puzzle' function is called
    def shadeIncorrectCells(self, window, incorrect):
        for cell in incorrect:
            pygame.draw.rect(window, INCORRECTCELLCOLOUR,
(cell[0]*cellSize+gridPos[0], cell[1]*cellSize+gridPos[1], cellSize,
cellSize))

    #shades locked cells grey to indicate to the user that they are not able
to change those numbers on the puzzle
    def shadeLockedCells(self, window, locked):
        for cell in locked:
            pygame.draw.rect(window, LOCKEDCELLCOLOUR,
(cell[0]*cellSize+gridPos[0], cell[1]*cellSize+gridPos[1], cellSize,
cellSize))

    #formats numbers input into the puzzle
    def drawNumbers(self, window):
        for yidx, row in enumerate(self.grid):
            for xidx, num in enumerate(row):
                if num != 0:
                    pos = [(xidx*cellSize)+gridPos[0],
(yidx*cellSize)+gridPos[1]]
                    self.textToScreen(window, str(num), pos)

    #shades selected unlocked cells light blue
    def drawSelection(self, window, pos):
        pygame.draw.rect(window, LIGHTBLUE, ((pos[0]*cellSize)+gridPos[0],
(pos[1]*cellSize)+gridPos[1], cellSize, cellSize))

    #draws Sudoku puzzle gird
    def drawGrid(self, window):
        pygame.draw.rect(window, BLACK, (gridPos[0], gridPos[1], WIDTH-404,
HEIGHT-154), 2)
        for x in range(9):
```

```
            pygame.draw.line(window, BLACK, (gridPos[0]+(x*cellSize),
gridPos[1]), (gridPos[0]+(x*cellSize), gridPos[1]+495), 2 if x % 3 == 0
else 1)
            pygame.draw.line(window, BLACK, (gridPos[0],
gridPos[1]+(x*cellSize)), (gridPos[0]+495, gridPos[1]++(x*cellSize)), 2 if
x % 3 == 0 else 1)

    #identifies the location of the mouse
    def mouseOnGrid(self):
        if self.mousePos[0] < gridPos[0] or self.mousePos[1] < gridPos[1]:
            return False
        if self.mousePos[0] > gridPos[0]+gridSize or self.mousePos[1] >
gridPos[1]+gridSize:
            return False
        return ((self.mousePos[0]-gridPos[0])//cellSize,
(self.mousePos[1]-gridPos[1])//cellSize)

    #creates buttons on the window, appends them to playingButtons array and
assigns them functions for when clicked
    def loadButtons(self):
        self.playingButtons.append(Button(660, 475, WIDTH//7, 40,
                                          function=self.checkAllCells,
                                          colour=(27,142,207),
                                          text="Check"))

        self.playingButtons.append(Button(50, 40, WIDTH//7, 40,
                                          colour=(117,172,112),
                                          function=self.getPuzzle,
                                          params="1",
                                          text="Easy"))
        self.playingButtons.append(Button(190, 40, WIDTH//7, 40,
                                          colour=(204,197,110),
                                          function=self.getPuzzle,
                                          params="2",
                                          text="Medium"))
        self.playingButtons.append(Button(330, 40, WIDTH//7, 40,
                                          colour=(199,129,48),
                                          function=self.getPuzzle,
                                          params="3",
                                          text="Hard"))


        self.playingButtons.append(Button(600, 200, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=1,
                                        text="1"))
        self.playingButtons.append(Button(685, 200, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=2,
```

```
                                        text="2"))
        self.playingButtons.append(Button(770, 200, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=3,
                                        text="3"))
        self.playingButtons.append(Button(600, 285, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=4,
                                        text="4"))
        self.playingButtons.append(Button(685, 285, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=5,
                                        text="5"))
        self.playingButtons.append(Button(770, 285, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=6,
                                        text="6"))
        self.playingButtons.append(Button(600, 370, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=7,
                                        text="7"))
        self.playingButtons.append(Button(685, 370, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=8,
                                        text="8"))
        self.playingButtons.append(Button(770, 370, 75, 75,
                                        colour=(208, 208, 208),

function=self.handleNumericButtonPress,
                                        params=9,
                                        text="9"))

    #formats onscreen text
    def textToScreen(self, window, text, pos):
        font = self.font.render(text, False, BLACK)
        fontWidth = font.get_width()
        fontHeight = font.get_height()
        pos[0] += (cellSize-fontWidth)//2
        pos[1] += (cellSize-fontHeight)//2
        window.blit(font, pos)
```

```
    #loads components on window every time the window refreshes (eg. a new
Sudoku puzzle is generated)
    def load(self):
        self.playingButtons = []
        self.loadButtons()
        self.lockedCells = []
        self.incorrectCells = []
        self.finished = False

        #sets locked cells from original board
        for yidx, row in enumerate(self.grid):
            for xidx, num in enumerate(row):
                if num != 0:
                    self.lockedCells.append([xidx, yidx])

    #ensures input in selected cell is an integer
    def isInt(self, string):
        try:
            int(string)
            return True
        except:
            return False

#creates new class to deal with buttons and clicks
class gameButton:
    def __init__(self, x, y, width, height, text=None, colour=(73, 73, 73),
highlightedColour=(189, 189, 189),
                 function=None, params=None):
        self.image = pygame.Surface((width, height))
        self.pos = (x, y)
        self.rect = self.image.get_rect()
        self.rect.topleft = self.pos
        self.text = text
        self.colour = colour
        self.highlightedColour = highlightedColour
        self.function = function
        self.params = params
        self.highlighted = False
        self.width = width
        self.height = height

    #identifies when the mouse is over a button and changes the colour of
the button
    def update(self, mouse):
        if self.rect.collidepoint(mouse):
            self.highlighted = True
        else:
            self.highlighted = False

#draws the widgets on the window in the appropriate locations and displays
it on the window using the blit()method
    def draw(self, window):
```

```
        self.image.fill(self.highlightedColour if self.highlighted else
self.colour)
        if self.text:
            self.drawText(self.text)
        window.blit(self.image, self.pos)

#tells the program to run the function associated with the button if it is
clicked
#includes the parameters of the function if there are any
    def click(self):
        if self.params:
            self.function(self.params)
        else:
            self.function()

#draws text that is to be displayed on the window in a specified format and
displays it on the window using the blit()method
    def drawText(self, text):
        font = pygame.font.SysFont("arial", 20, "bold")
        text = font.render(text, False, (0, 0, 0))
        width, height = text.get_size()
        x = (self.width - width) // 2
        y = (self.height - height) // 2
        self.image.blit(text, (x, y))

#defines variables WIDTH and HEIGHT (dimensions of window)
WIDTH = 900
HEIGHT = 650

#defines the colour (RGB) code for all variables representing colours
referenced in the program
WHITE = (255,255,255)
BLACK = (0,0,0)
LIGHTBLUE = (173, 216, 230)
LOCKEDCELLCOLOUR = (189,189,189)
INCORRECTCELLCOLOUR = (195,121,121)

#defines position and size of grid and its cells
gridPos = (50, 100)
cellSize = 55
gridSize = cellSize*9
```

Virtual Mouse

(HandTrackingModule.py)

```
#imports
import cv2
import mediapipe as mp
import time
import math
import numpy as np
```

```
#creates new class and initialises object attributes
class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=False,
trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,
self.detectionCon, self.trackCon)

        self.mpDraw = mp.solutions.drawing_utils

        self.tipIds = [4, 8, 12, 16, 20]#numbers correspond to the tips of
all 5 fingers

    #identifies hand and hand landmarks in webcam feed
    def findHands(self, img, draw=True):
        img = cv2.resize(img, None, fx=0.5, fy=0.5,
interpolation=cv2.INTER_AREA)
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)#converts webcam image
to RGB colour space
        self.results = self.hands.process(imgRGB)

       #overlays drawing on hand landmarks
        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
self.mpHands.HAND_CONNECTIONS)
        return img

    #finds the coordinates of the hand
    def findPosition(self, img, handNo=0, draw=True):
        xList = []#stores x coordinates
        yList = []#stores y coordinates
        bbox = []#stores coordinates of the corners of the boundary box
surrounding the detected hand
        self.lmList = []#2D array stores coordinates for tracked hand
landmoarks (ie. tip of index finger)

        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
                h, w, c = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                xList.append(cx)
                yList.append(cy)
                self.lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 8, (255, 0, 0), cv2.FILLED)
```

```
            xmin, xmax = min(xList), max(xList)#finds minimum and maximum
coordinates on the x axis
            ymin, ymax = min(yList), max(yList)#finds minimum and maximum
coordinates on the y axis
          bbox = xmin, ymin, xmax, ymax #identifies coordinates of corners
of border box around the hand based on x and y coordinates found above


            if draw:
                cv2.rectangle(img, (bbox[0]-20, bbox[1]-20), (bbox[2]+20,
bbox[3]+20), (0, 255, 0), 2)#draws the boundary box around detected hand

        return self.lmList, bbox

    #identifies when fingers are up
    def fingersUp(self):
        fingers = []#array stores fingers that are raised

#checks if thumb is raised; appends to fingers array if so
        if self.lmList[self.tipIds[0]][1] >
self.lmList[self.tipIds[0]-1][1]:
            fingers.append(1)
        else:
            fingers.append(0)

#checks if any of the other 4 fingers are raised; appends to fingers array
if so
        for id in range(1, 5):
            if self.lmList[self.tipIds[id]][2] <
self.lmList[self.tipIds[id]-2][2]:
                fingers.append(1)
            else:
                fingers.append(0)

        return fingers

    #finds the distance between fingers
    def findDistance(self, p1, p2, img, draw=True, r=5, t=2):
        x1, y1 = self.lmList[p1][1], self.lmList[p1][2]
        x2, y2 = self.lmList[p2][1], self.lmList[p2][2]
        cx, cy = (x1+x2)//2, (y1+y2)//2

#overlays drawing to show user the distance between the two fingers, the
tips of index and middle fingers and a point exactly in the middle of them
        if draw:
            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
            cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
            cv2.circle(img, (cx, cy), r, (255, 0, 255), cv2.FILLED)

        length = math.hypot(x2-x1, y2-y1)

        return length, img, [x1, y1, x2, y2, cx, cy]
```

(VirtualMouse.py)

```
#imports
import cv2
import numpy as np
import HandTrackingModule as htm
import time
import autopy

class Mouse:
    #initialises window
    def __init__(self, root):
        self.root = root
        self.root.title("Virtual Mouse")

    wCam, hCam = 850, 800 #dimensions of camera window
    frameR = 50 #frame reduction
    smoothening = 7 #smoothness of mouse movement

    cap = cv2.VideoCapture(0) #creates a video capture object and return the video from the first webcam of the computer
    cap.set(3,wCam) #sets width of camera feed
    cap.set(4,hCam) #sets height of camera feed

    #resets window
    pTime = 0
    plocX, plocY = 0, 0
    clocX, clocY = 0, 0
    detector = htm.handDetector(maxHands=1)#only identifies 1 hand
    wScr, hScr = autopy.screen.size()#identifies size of screen and stores width and height in variables

    while True:
        success, img = cap.read()#captures image from camera
        img = detector.findHands(img)#identifies hands if they are visible
        lmList, bbox = detector.findPosition(img)#finds position of hand landmarks and, subsequently, the border box that surrounds them

        if len(lmList) != 0:
            x1, y1 = lmList[8][1:]#find coordinates of tip and top knuckle of index finger
            x2, y2 = lmList[12][1:]#find coordinates of tip and top knuckle of middle finger

            fingers = detector.fingersUp()#idendity if fingers are up based on coordinates retieved

            cv2.rectangle(img, (frameR, frameR), (wCam-(6*frameR), hCam-(10*frameR)), (225, 0, 225), 2)#adjust dimensions of border box to suit dimensions of user's webcam
```

```
#if only the index finger is up, draw a circle over the fingers and move
the mouse corresponding to the movement of the finger
      if fingers[1] == 1 and fingers[2] == 0:

                x3 = np.interp(x1, (frameR, wCam-(6*frameR)), (0, wScr))
                y3 = np.interp(y1, (frameR, hCam-(10*frameR)), (0, hScr))

                clocX = plocX + (x3-plocX)/smoothening
                clocY = plocY + (y3 - plocY) /smoothening

                autopy.mouse.move(wScr-clocX, clocY)
                cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
                plocX, plocY = clocX, clocY

#if index and middle fingers are up, draw a circle over the 2 fingers and a
line between them
#if the two fingers are within a specific distance to each other treat it
as a mouse click
            if fingers[1] == 1 and fingers[2] == 1:
                length, img, lineInfo = detector.findDistance(8, 12, img)
                if length <40:
                    cv2.circle(img, (lineInfo[4], lineInfo[5]), 15, (0, 255,
0), cv2.FILLED)
                    autopy.mouse.click()


        cTime = time.time() #returns the time as a floating point number
        pTime = cTime
      fps = 1 / (cTime - pTime) #calculates frames per seconds using
current and previous time
        cv2.putText(img, str(int(fps)), (10, 40), cv2.FONT_HERSHEY_DUPLEX,
1, (255, 0, 255), 2)#displays frames per second of user's webcam

        cv2.imshow("input", img)#launches virtual mouse window; displays
webcam image
        cv2.waitKey(1)#wait for 1ms before refreshing the window, appears as
a live video feed
```