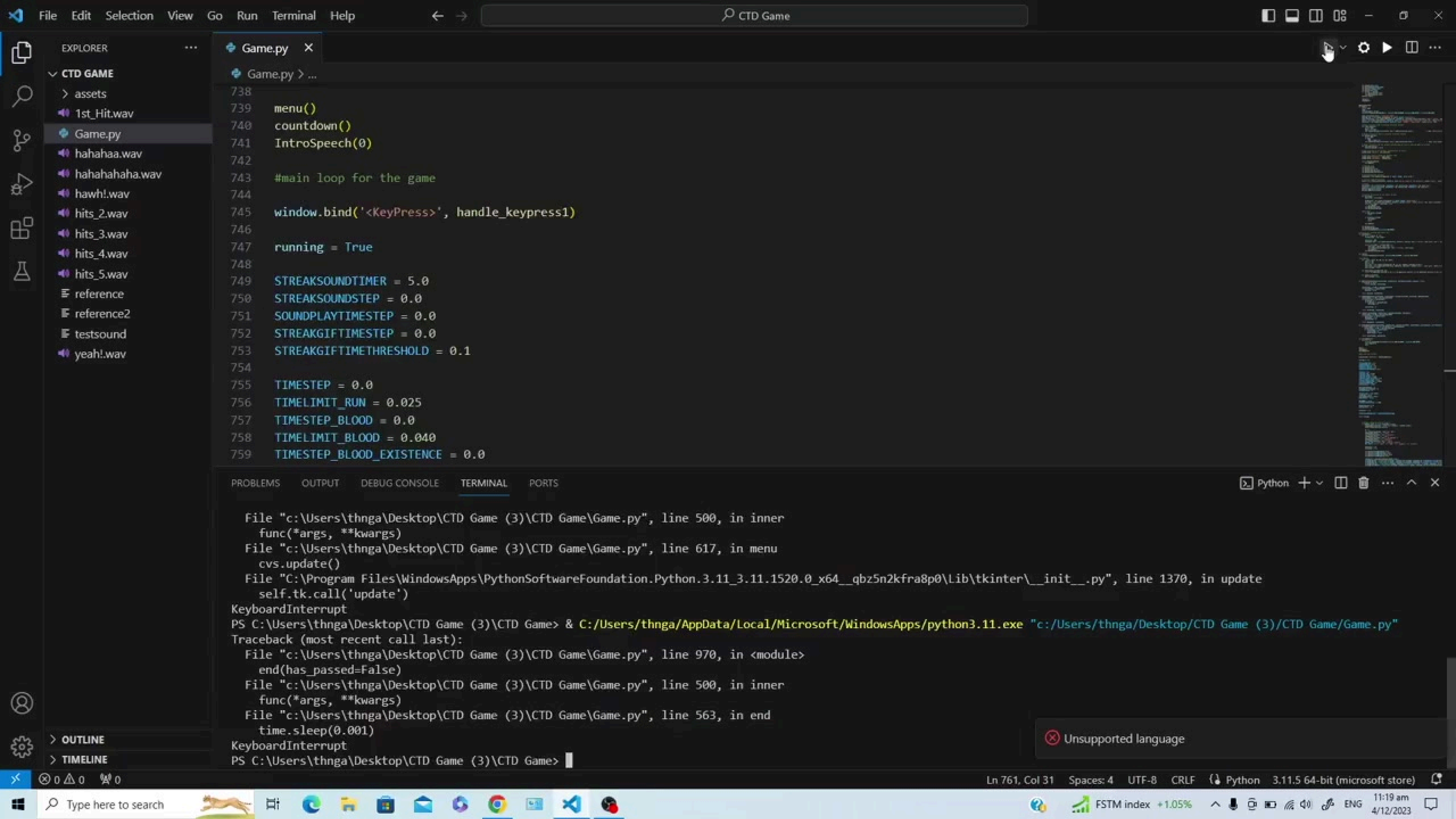
MATHEMATICAL MASSACRE

SC05 - TEAM 5A


File Edit Selection View Go Run Terminal Help
CTD Game
EXPLORER
CTD GAME
assets
1st_Hit.wav
Game.py
hahahaa.wav
hahahahaha.wav
hawh!.wav
hits_2.wav
hits_3.wav
hits_4.wav
hits_5.wav
reference
reference2
testsound
yeah!.wav
Game.py
Game.py > ...
738
739 menu()
740 countdown()
741 IntroSpeech(0)
742
743 #main loop for the game
744
745 window.bind('<KeyPress>', handle_keypress1)
746
747 running = True
748
749 STREAKSOUNDTIMER = 5.0
750 STREAKSOUNDSTEP = 0.0
751 SOUNDPLAYTIMESTEP = 0.0
752 STREAKGIFTIMESTEP = 0.0
753 STREAKGIFTIMETHRESHOLD = 0.1
754
755 TIMESTEP = 0.0
756 TIMELIMIT_RUN = 0.025
757 TIMESTEP_BLOOD = 0.0
758 TIMELIMIT_BLOOD = 0.040
759 TIMESTEP_BLOOD_EXISTENCE = 0.0
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
File "c:\Users\thnga\Desktop\CTD Game (3)\CTD Game\Game.py", line 500, in inner
func(*args, **kwargs)
File "c:\Users\thnga\Desktop\CTD Game (3)\CTD Game\Game.py", line 617, in menu
cvs.update()
File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.1520.0_x64__qbz5n2kfra8p0\Lib\tkinter__init__.py", line 1370, in update
self.tk.call('update')
KeyboardInterrupt
PS C:\Users\thnga\Desktop\CTD Game (3)\CTD Game> & C:/Users/thnga/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/thnga/Desktop/CTD Game (3)/CTD Game/Game.py"
Traceback (most recent call last):
File "c:\Users\thnga\Desktop\CTD Game (3)\CTD Game\Game.py", line 970, in <module>
end(has_passed=False)
File "c:\Users\thnga\Desktop\CTD Game (3)\CTD Game\Game.py", line 500, in inner
func(*args, **kwargs)
File "c:\Users\thnga\Desktop\CTD Game (3)\CTD Game\Game.py", line 563, in end
time.sleep(0.001)
KeyboardInterrupt
PS C:\Users\thnga\Desktop\CTD Game (3)\CTD Game>
Unsupported language
OUTLINE
TIMELINE
Ln 761, Col 31 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit (microsoft store)
Type here to search
FSTM index +1.05%
ENG
11:19 am
4/12/2023

CONTENTS

THE GAME BASE

```python
# import ...
from tkinter import *
import time

# declare the window
window = Tk()
# set window width and height
window.configure(width=WIDTH, height=HEIGHT)

# declare & initialize canvas for drawing shapes & texts
cvs = Canvas()
cvs.pack()


running = True

# main loop
while running:

    update_game_states()

    # update window screen
    window.update()
    # prevent screen refreshing too fast
    time.sleep(0.001)
```

Typical Tkinter setup.

Game's main loop:
For every iteration of while loop, game states (position of objects, scores, HP...) are updated.

THE GAME BASE - CLASSES

```python
class Question:
    def __init__(self, x):
        self.x = x
        self.y = 0
        self.y_vel = 2

        # from NUMBER_TYPE list, choose a type of number as answer
        NUMBER_TYPES = ['positive', 'negative']
        self.answer = random.choice(NUMBER_TYPES)

        # generate a random number corresponded to the type of answer and show it in form of text
        self.text = cvs.create_text(0, 0, text=self.generate_question(self.answer))

        # align the text to the center of the track
        width = cvs.bbox(self.text)[2] - cvs.bbox(self.text)[0]
        cvs.move(self.text, x-width, 0)
```

Simplified Question class:

Attributes:

1. x and y coordinate values.
2. y velocity.
3. answer for current question.

```python
    def generate_question(self, answer):
        if answer == 'positive':
            return random.randint(1, 100)
        elif answer == 'negative':
            return random.randint(-100, -1)
        # ...
```

Method to generate number according to answer.

```python
# width of screen
WIDTH = 1000
# x value of each tracks
START_X = [WIDTH/8, 3*WIDTH/8, 5*WIDTH/8, 7*WIDTH/8]

# initializing question object
questions = []
questions.append(Question(random.choice(START_X)))
```

Question obj is initialized on a randomized track.
&
Question obj is put into list for future tracking.

NUMBERS GENERATORS & CONDITIONS

This code takes in two parameters: number_type and PRIME_NUMBERS. This method is intended to generate different types of numbers based on the **number_type** specified. Here's what each part of the code does for each number_type:

1. **'Positive':** Generates a random positive integer between 1 and 100 (inclusive) that is not in the list PRIME_NUMBERS.
2. **'Negative':** Generates a random negative integer between -100 and -1 (inclusive).
3. **'Even':** Generates a random even integer between -100 and 100 (inclusive). If the randomly generated number is odd, it returns the previous even number.
4. **'Odd':** Generates a random odd integer between -100 and 100 (inclusive). If the randomly generated number is even, it returns the next odd number.
5. **'Prime':** Returns a random prime number from the list PRIME_NUMBERS.
6. **'Complex':** Returns a randomly generated complex number in the format of a string, consisting of two integers (real and imaginary parts) between 0 and 100.

```python
def generate_question(self, number_type, PRIME_NUMBERS):
    if number_type == 'positive':
        # if number is prime, skip
        random_number = randint(1, 100)
        while random_number in PRIME_NUMBERS:
            random_number = randint(1, 100)
        return random_number
    elif number_type == 'negative':
        return randint(-100, -1)
    #AK
    elif number_type == 'even':
       i = randint(-100,100)
       if i % 2 == 0:
          return i

       return i-1
    #AK
    elif number_type == 'odd':
        i = randint(-100,100)
        if i % 2 == 1:
            return i
        return i + 1

    elif number_type == 'prime':
        return choice(PRIME_NUMBERS)
    elif number_type == 'complex':
        return f'{randint(0, 100)} + {randint(0, 100)}j'
```

SIEVE OF ERATHOSTHENES

An algorithm that can generate prime numbers efficiently.

```python
def sieve_of_eratosthenes(upper_limit):
    PRIME_NUMBERS = []
    lp = [0] * (upper_limit + 1)
    it=2
    while(it<=upper_limit):
        if(lp[it]==0):
            lp[it]=it
            PRIME_NUMBERS.append(it)
        it2=0
        while(it*PRIME_NUMBERS[it2]<=upper_limit):
            lp[it*PRIME_NUMBERS[it2]]=PRIME_NUMBERS[it2]
            if(PRIME_NUMBERS[it2]==lp[it]):
                break
            it2+=1
        it+=1
    return PRIME_NUMBERS
```

GIFS ANIMATION - LEVEL TRANSITION

```
CURRENTFRAME = 0

def getFrames(framelist, frame_Index, gifFileName, currentFrame):
    while True:
        try:
            # Read a frame from GIF file
            part = 'gif -index {}'.format(frame_Index)
            currentFrame = PhotoImage(file=gifFileName, format=part)
        except:
            last_frame = frame_Index - 1    # Save index for last frame
            break               # Will break when GIF index is reached
        framelist.append(currentFrame)
        frame_Index += 1        # Next frame index

    return framelist, last_frame
```

```
FrameList = []
RunningimageObject, Runningframes = getFrames(FrameList, 0,Runningimage, CURRENTFRAME)
FrameList= []
BloodimageObject, BloodSpatterframes = getFrames(FrameList,0, BloodSpatterImage, CURRENTFRAME)
FrameList = []
StreakGIFObject, StreakGIFframes = getFrames(FrameList, 0, StreakImage, CURRENTFRAME)
FrameList = []
```

Return a list containing the addresses of all the different image frames for each GIF. Iterating through each element plays the GIF.

```
if SCORE // THRESHOLD > 0:
    if CURRENT_LEVEL < 3:
        CURRENT_LEVEL += 1
    elif CURRENT_LEVEL == 3:
        end(has_passed=True)
    return
```

Conditional: Increment the level when the score reaches the required amount to pass it.

KILL STREAK & PLAYING OF SOUNDS

```
if STREAKCOUNT >= STREAKSTEP and STREAKCOUNT % STREAKSTEP == 0 and isStreakExistence == False and HASSTREAKED == False:
    playSound(StreakOpeningSoundsFilenames)
    isplaying = False
    STREAKSOUNDSTEP = 0
    SOUNDPLAYTIMESTEP = 0
    if len(streakgifs) == 0:
        streakgifs.append(IMG_GIF(x=600, y=400, currentFrame=StreakGifCount, imageObject=StreakGIFObject,xdimension=250,ydimension=150,cvs = cvs))
    HASSTREAKED = True
    isStreakExistence = True
    bPLAYNEXT = False
    previousScore = SCORE
    STREAKCOUNT = 0
    STREAKGIFTIMESTEP = 0
    PLAYNEXTSOUNDCOUNTER = 0.0

HASSTREAKED, STREAKCOOLDOWNTIMER = handleHasStreaked(STREAKCOOLDOWNTIMER,STREAKCOOLDOWNDURATION,1,hasStreaked=HASSTREAKED, previousScore=previousScore,
if isStreakExistence == True:
    bPLAYNEXT, PLAYNEXTSOUNDCOUNTER = counter(counterStep=PLAYNEXTSOUNDCOUNTER,counterLimit=PLAYNEXTSOUNDINTERVAL,multiplicativeFac=2,bplaynext=bPLAYN
    if bPLAYNEXT == True:
        if isplaying == False:
            splaying, SOUNDPLAYTIMESTEP = BAnimLifetimeCounter(counterStep=SOUNDPLAYTIMESTEP, counterLimit=soundlifeLimit,multiplicativeFac=1)
            laySound(StreakRunningSoundFilenames)
            SOUNDPLAYTIMESTEP = 0
isStreakExistence,STREAKSOUNDSTEP = BAnimLifetimeCounter(counterStep=STREAKSOUNDSTEP,counterLimit=STREAKSOUNDTIMER,multiplicativeFac=1, playing=ISPLAYING)
```

External Timer

Play Sound Cue

Internal Timer

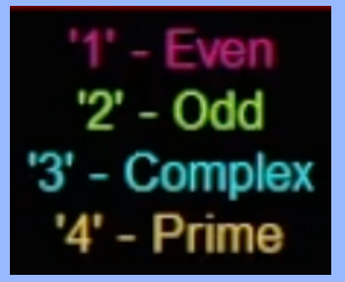
Play Sound Cue

Internal Timer

TEXT & SCOREBOARD

TKINTER COLOR

```
text1 = cvs.create_text(80, HEIGHT - 80, text= "'1' - Positive", fill='turquoise1', font=('Purisa', 13))     # 1: Positive OR Odd
text2 = cvs.create_text(80, HEIGHT - 60, text="'2' - Negative", fill='SeaGreen1', font=('Purisa', 13))       # 2: Negative OR Even
text3 = cvs.create_text(80, HEIGHT - 40, text='', fill="deep pink", font=('Purisa', 13))                     # 3: Prime
text4 = cvs.create_text(80, HEIGHT - 20, text='', fill="goldenrod1", font=('Purisa', 13))                    # 4: Complex
hpText = cvs.create_text(WIDTH-70, HEIGHT-62, text=f"HP: {HP}", font=('Purisa', 15), fill="red3")
scoreText = cvs.create_text(WIDTH-70, HEIGHT-39, text=f"Scores: {SCORE}", font=('Purisa', 15), fill="RoyalBlue3")
levelText = cvs.create_text(WIDTH-295, HEIGHT-20, text=f"LEVEL {CURRENT_LEVEL}", font=('Purisa', 18), fill="yellow")
```

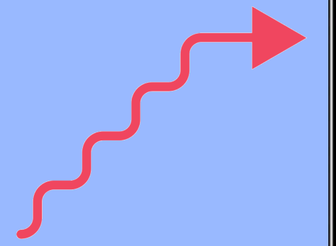


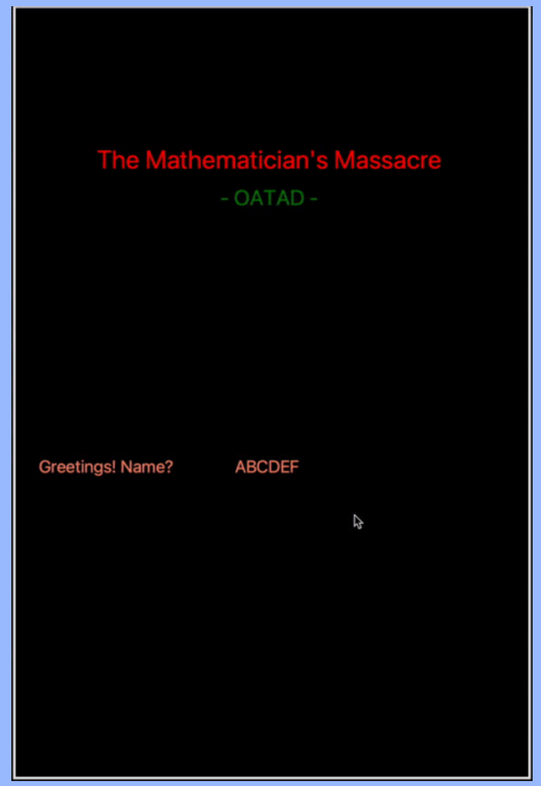

```
THRESHOLD = 20
POINTS = {
    'positive': 1,
    'negative': 1,
    'even': 2,
    'odd': 2,
    'complex': 3,
    'prime': 5
}
```

```
SCORE += POINTS[questions[0].answer]
cvs.itemconfig(scoreText, text=f"Scores: {SCORE}")
```

```
# helper function to add a character from name variable
def input_name(ev):
    global name
    name += ev.char
    cvs.itemconfig(name_instruction, text = name_instruction_title + '              ' + name, fill='salmon1')

# helper function to remove a character from name variable
def delete_char(ev):
    global name
    if name:
        name = name[:-1]
    cvs.itemconfig(name_instruction, text = name_instruction_title + '              ' +  name, fill='dodger blue')
```





THANK YOU FOR LISTENING!