# MAPUAN TYPING MANIA!

# A GUI-BASED APPLICATION (GAME) USING PYTHON PROGRAMMING LANGUAGE

****

**A CAPSTONE PROJECT DOCUMENTATION**

**Presented to**

**the College of Engineering and Architecture of Mapúa Malayan Colleges Mindanao Gen.**

**Douglas MacArthur Highway, Davao City**

**In Partial Fulfillment**

**of the Requirements for the Course of CPE003L Advanced Programming Techniques (LAB)**

# LIST OF GROUP MEMBERS

# Neil Justin Bermoy

# Tristan Angelo Buling

# Isiah Catan

# Lawrence Hao

**FEBRUARY 28 2025**

# OVERVIEW OF THE CHOSEN GAME PROGRAM

A typing game to see how good you are at typing and how fast you can type. The game will generate a random word and the player must type it as fast as they can. The game will calculate the player's typing speed and accuracy. Enter the championship and win the title of the fastest typist in Mapua University.

# ALGORITHM

# MAPUAN TYPING MANIA MAIN ALGORITHM.

1. Start game.

2. Display title screen and ask for user's name.

3. Once the user inputs their name, proceed to main menu.

4. Display main menu options: [PLAY] [LEADERBOARD] [QUIT].

- 4a. If the user chooses QUIT → Exit game.

- 4b. If the user chooses LEADERBOARD:

- Display the leaderboard.

- Prompt user to go back to the main menu.

- 4c. If the user chooses PLAY, proceed to stage selection.

5. Display stage selection menu: [TUTORIAL] [INTRO] [STAGE1] [STAGE2] [STAGE3] [ENDING] [BACK].

- 5a. If the user chooses BACK → Return to main menu.

- 5b. If the user chooses TUTORIAL → Play tutorial.

- At the end, return to stage selection or main menu.

- 5c. If the user chooses INTRO → Show introduction story.

- At the end, return to stage selection or main menu.

- 5d. If the user chooses STAGE1 → Play stage 1.

- At the end, return to stage selection or main menu.

- 5e. If the user chooses STAGE2 → Play stage 2.

- At the end, return to stage selection or main menu.

- 5f. If the user chooses STAGE3 → Play stage 3.

- At the end, return to stage selection or main menu.

- 5g. If the user chooses ENDING → Show ending story and credits.

- After credits, return to main menu.

6. Repeat steps as needed until the user chooses to quit.

# PSEUDOCODE

# *Function Main*

... Start of the game part. Initialize the necessary variables at each start of a function.

Declare String game

... Diri sugod ang duwa kung e on sa player o dili

Output "Start Game? input (on) or (no): " & ToChar(13) & "Please follow each designated instructions" & ToChar(13) & "Type in ALL CAPS when the choices are ALL CAPS"

Input game

... Check ang input

If game == "no"

... No start means wla nagsugod ang duwa

Else

... Double check

If game == "on"

... Do loop gamit, kay samantalang wala pa ang player nag ingon mo quit or no start ang game then play

Do

... username kay para sa score ug leaderboard

Declare String username

Output "TITLE SCREEN" & ToChar(13) & "MAPUAN TYPING MANIA! (LOGO)" & ToChar(13) & ToChar(13) & "Enter a username: "

Input username

... Key ang gamit sa pagstart sa dula para dili dayun diritso kung mahuman ang usename input

Output "Input any key: " & ToChar(13) & "Any key means any key on the keyboard that isnt a special function."

... call ang menu function kay naa didto ang main menu

Call mainmenu

... ask usab kung gusto ba nila e keep ang dula mag run

Output "Input (on) if u wanna keep gaming, (no) otherwise: "

Input game

Loop game == "on"

Else

... Pag dili on ang nakainput per instruction, gawas dayun sa dula kay need man necessary inputs for the game to start.

Output "Follow instructions"

End

End

... Goodbye message.

Output "No more gaeming, thnx for playing our game!"

End

Function mainmenu

... mao ni ang main menu function. Class ni siya didto sa game.

Declare String key

Input key

If key == ""

... mao "" kay wla ang user nag input ug key then molabas ka duwa.

Output "BYE BYE"

Else

... pag nakapislit ug key

... main menu choices, tulo lang pareha didto sa game

Output "MAIN MENU, SELECT ACTION:" & ToChar(13) & "" & ToChar(13) & "PLAY" & ToChar(13) & "LEADERBOARD" & ToChar(13) & "QUIT"

Declare String menuchoice

... do loop japon

Do

Input menuchoice

If menuchoice == "PLAY"

... diri ang playmenu gitawag kay subfunction siya sa main menu

Call playmenu

Else

If menuchoice == "LEADERBOARD"

... Display lang sa leaderboard tapos assume balik dayun menu, kay nagprompt man ta pero diri sa flowchart diritso nato

Output "LEADERBOARD DISPLAY"

Output "Going back to main menu"

End

End

Output "MAIN MENU, SELECT ACTION:" & ToChar(13) & "" & ToChar(13) & "PLAY" & ToChar(13) & "LEADERBOARD" & ToChar(13) & "QUIT"

Loop menuchoice != "QUIT"

... Goodbye message.

Output "Goodbye."

End

End

Function playmenu

... mao ni ang submenu, playmenu

Declare String playmenu

... mga stages nga pwede piloon sa player, wala na lock like planned kay wla na time. So ang player pwede na sila moadto miskag asa nga stage.

Output "CHOOSE A STAGE: " & ToChar(13) & "TUTORIAL" & ToChar(13) & "INTRO" & ToChar(13) & "STAGE1" & ToChar(13) & "STAGE2" & ToChar(13) & "STAGE3" & ToChar(13) & "ENDING" & ToChar(13) & ToChar(13) & "TYPE ESC TO GO BACK TO MENU"

... do loop ra japon atung logic samantalang wla mo quit jud ang user

Do

Input playmenu

... mga if na gamit kay diri nmn ang tinuod nga dula

If playmenu == "TUTORIAL"

Output "Playing tutorial" & ToChar(13) & "Assuming the player keeps playing, they go onto into intro to ending" & ToChar(13) & "once ending done, they go back to play menu." & ToChar(13) & "Otherwise, we assume the player has decided to go back."

Else

If playmenu == "INTRO"

Output "Playing introduction" & ToChar(13) & "Assuming the player keeps playing, they go onto stage 1 to ending" & ToChar(13) & "once ending done, they go back to play menu." & ToChar(13) & "Otherwise, we assume the player has decided to go back."

Else

If playmenu == "STAGE1"

Output "Playing Stage1" & ToChar(13) & "Assuming the player keeps playing, they go onto another stage to ending" & ToChar(13) & "once ending done, they go back to play menu." & ToChar(13) & "Otherwise, we assume the player has decided to go back."

Else

If playmenu == "STAGE2"

Output "Playing Stage2" & ToChar(13) & "Assuming the player keeps playing, they go onto another stage to ending" & ToChar(13) & "once ending done, they go back to play menu." & ToChar(13) & "Otherwise, we assume the player has decided to go back."

Else

If playmenu == "STAGE3"

Output "Playing Stage3" & ToChar(13) & "Assuming the player keeps playing, they go onto to the ending" & ToChar(13) & "once ending done, they go back to play menu." & ToChar(13) & "Otherwise, we assume the player has decided to go back."

Else

If playmenu == "ENDING"

Output "Playing Ending and Credits" & ToChar(13) & ToChar(13) & "Once they saw the ending and credits, they go back to play menu."

End

End

End

End

End

End

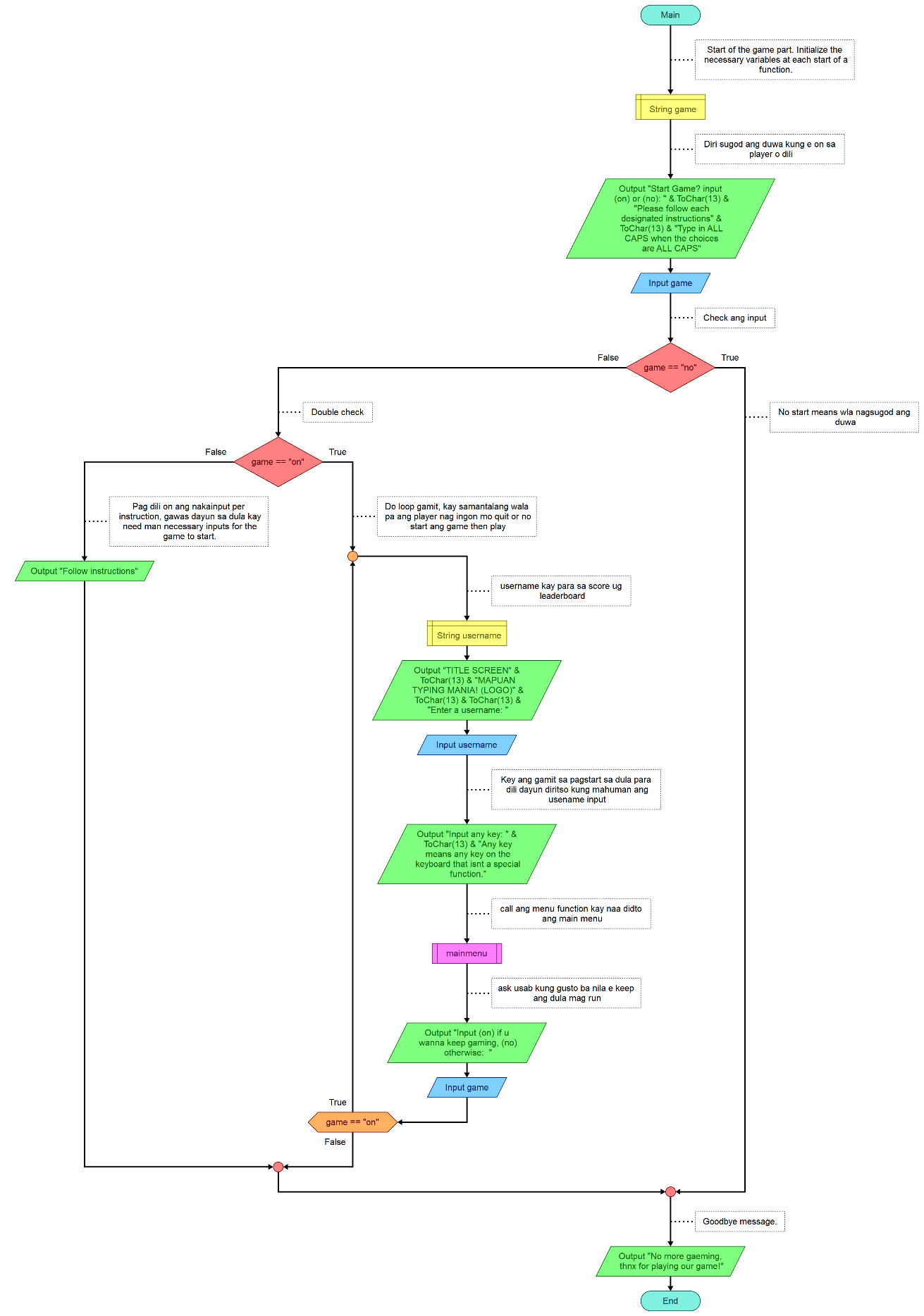
Output "CHOOSE A STAGE: " & ToChar(13) & "TUTORIAL" & ToChar(13) & "INTRO" & ToChar(13) & "STAGE1" & ToChar(13) & "STAGE2" & ToChar(13) & "STAGE3" & ToChar(13) & "ENDING" & ToChar(13) & ToChar(13) & "TYPE ESC TO GO BACK TO MENU"

Loop playmenu != "ESC"

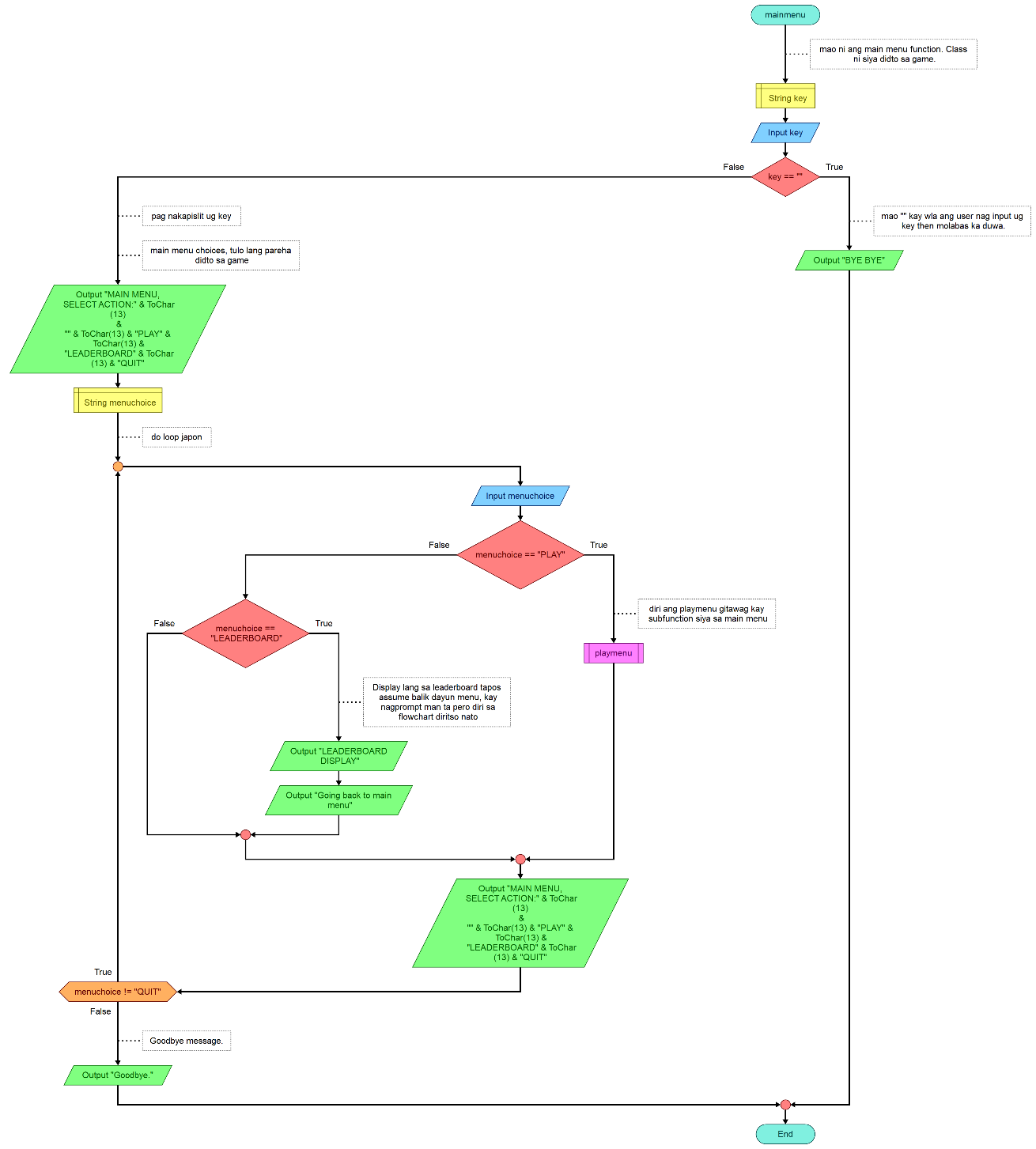
# *End*

# FLOWCHART

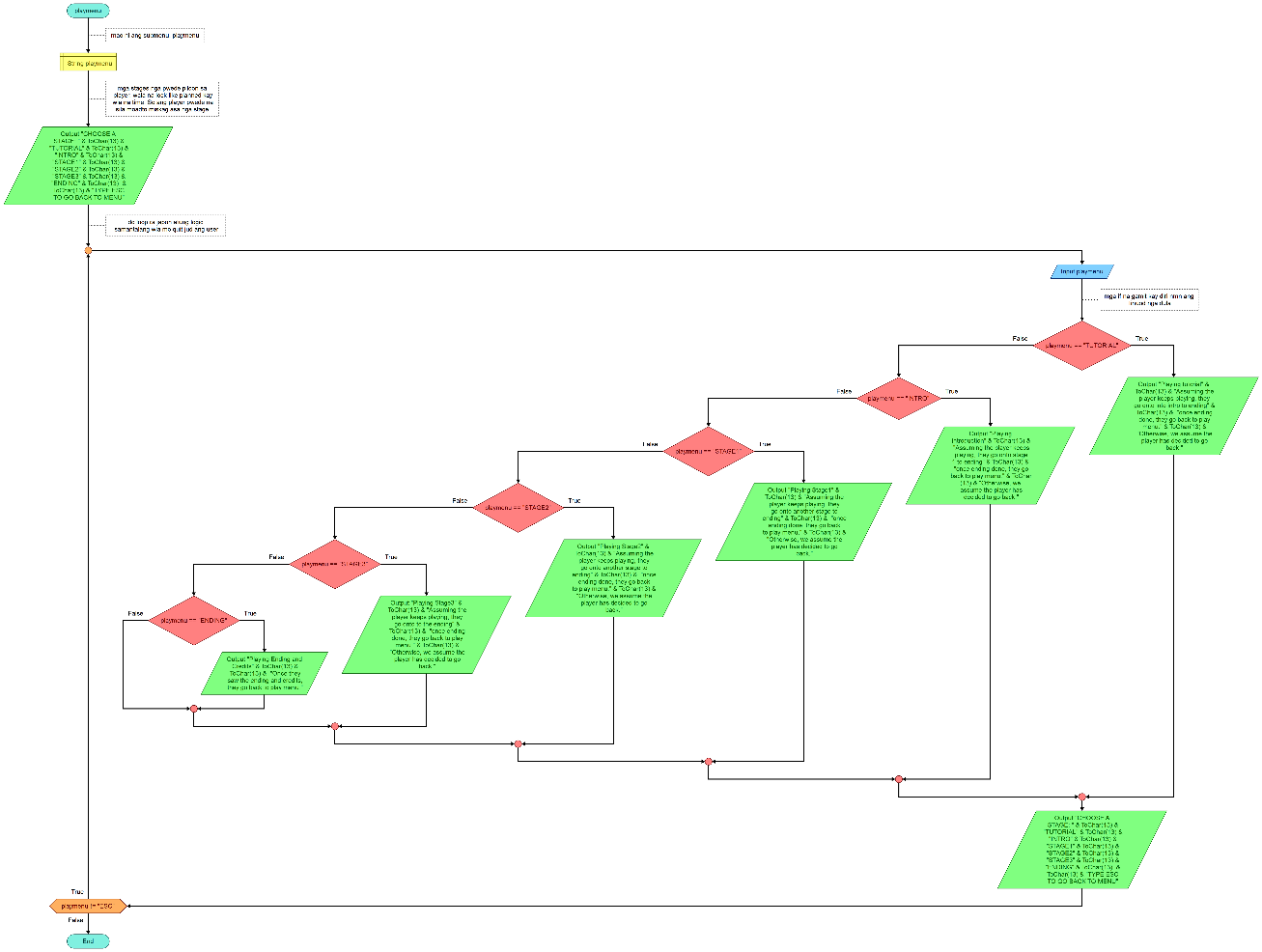
# MAIN FLOWCHART



# SUB FUNCTION FOR MAIN MENU FLOWCHART



# SUB FUNCTION FOR PLAY MENU FLOWCHART



# RESULTS and DISCUSSION

# TITLE SCREEN

A computer screen shot of a computer keyboard

AI-generated content may be incorrect.

**Function:**

Show the user the starting screen of the game and asks for a user input, that being their name. In the maximum amount of 10 letters in order to proceed.

**Objects:**

Background Image – Self explanatory

Logo – Game logo signifying the title and theme of the game

User Prompt – Self explanatory

Empty Text Box – Text field for user input

# PLAY MENU SCREEN

A screen shot of a stage

AI-generated content may be incorrect.

**Function:**

Show the user the stage selection menu.

**Objects:**

Background Image – Self explanatory

Selection choices – Leads to different screen display and the actual game itself

Esc Prompt – To prompt user to go back to main menu

# INTRO SCREEN

A screenshot of a computer

AI-generated content may be incorrect.

**Function:**

Show the user the stage introduction screen, part of each stage.

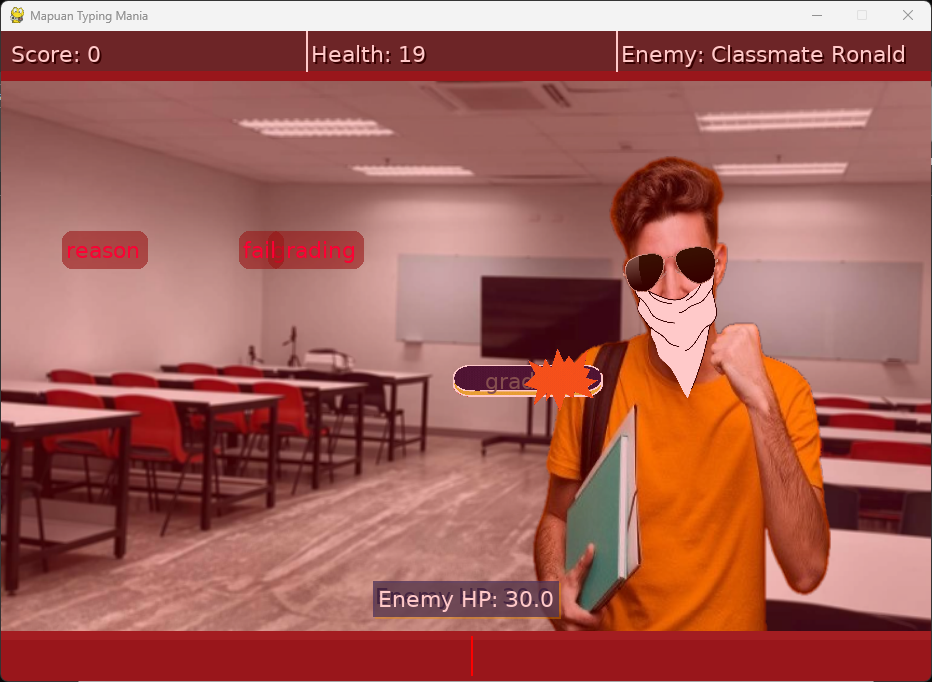
**Objects:**

Background Image – Self explanatory

Text and Colored background – Display text indicating the game’s story

Esc and continue Prompt – To prompt user to go back to main menu or continue inside the stage

# GAMEPLAY SCREEN



**Function:**

The actual typing game screen display itself.

**Objects:**

Background Image – Self explanatory

UI – User interface showing enemy hitpoints, enemy name, user’s health, and their score

Red bar with a vertical line – indicating where the words will show when the user types their input

Red words – the falling words themselves to be typed by the user and the enemy2

Enemy sprite – indicating the enemy and their hitstate

Blue rounded box – indicating what the enemy is typing to attack the player

# PAUSE SCREEN

A room with chairs and tables

AI-generated content may be incorrect.

**Function:**

Pause screen when they user to take a break from the gameplay.

**Objects:**

Background Image – Self explanatory

Prompt for resuming gameplay – Self-explanatory

Prompt for going to menu – Self-explanatory

# VICTORY SCREEN

A screenshot of a computer

AI-generated content may be incorrect.

**Function:**

When the user has finished a battle and won against an enemy, it shows their score.

**Objects:**

Background Image – Self explanatory

Text Box – Indicating Victory! And the user’s high score after they finished a stage

Prompt for continuing gameplay – Self-explanatory

Prompt for going to menu – Self-explanatory

# COMPLETE SOURCE CODE

For reasons totally valid, our group will only highlight some of the files to show that we did use most of the structures. It is a massive undertaking to highlight all our code, so I hope that it will be considered that we took such an action, if we do take the other course, it will be redundant to do so.

**Programming Structures**

1. Sequential Structures
2. Decision Structures
3. Repetition Structures
4. String Methods
5. Text File Manipulation
6. Lists and Dictionaries
7. Functions
8. Program Modularization
9. Simple Graphics and Image Processing
10. Graphical User Interfaces
11. Designing with Classes
12. Network Application and Client/Server Programming (optional)
13. Searching, Sorting, and Complexity (optional)

# MAPUANTYPINGMANIA.PY

import os  
import sys  
import numpy as np  
import pygame as pg  
import introduction  
from bgfix import stretch  
from buttons import ImageButton  
from stages import LoadingScreen, Tutorial  
from PIL import Image, ImageFilter  
import endings  
  
# Initialize Pygame and the screen  
pg.init()  
width = 930  
height = 650  
  
"""UNIVERSAL FUNCTIONS------------------------------------------------------------------------------------------------"""  
# Kuhag font gikan sa computer  
def get\_Font(size):  
 font\_path = os.path.join(os.path.dirname(\_\_file\_\_), "resources/DejaVuSans.ttf")  
 if not os.path.exists(font\_path):  
 raise FileNotFoundError(f"Font file not found: {font\_path}")  
 return pg.font.Font(font\_path, size)  
  
# Katung pa wave sa menu  
def apply\_wave\_effect(image, amplitude, frequency, phase, color\_shift):  
 try:  
 # Convert the image to a 3D array of pixels  
 arr = pg.surfarray.pixels3d(image)  
 height, width, \_ = arr.shape  
  
 # Apply the wave effect to each column of pixels  
 for x in range(width):  
 # Calculate the vertical offset for the wave effect  
 offset = int(amplitude \* np.sin(2 \* np.pi \* frequency \* x + phase))  
 arr[:, x] = np.roll(arr[:, x], offset, axis=0)  
 # Apply color shift to the pixels  
 arr[:, x] = np.clip(arr[:, x] + [color\_shift, color\_shift, color\_shift], 0, 255)  
  
 # Convert the modified array back to a surface  
 return pg.surfarray.make\_surface(arr)  
 except Exception as e:  
 print(f"Error applying wave effect: {e}")  
 return image # Return the original image in case of error  
  
# Wla ni igo ri para sa image processing  
def process\_images():  
 try:  
 # Load and blur the acadhall background image  
 pil\_acadhall\_image = Image.open('resources/backgrounds/acadhall.jpg')  
 blurred\_acadhall\_image = pil\_acadhall\_image.filter(ImageFilter.BLUR)  
 blurred\_acadhall\_image.save('resources/backgrounds/acadhall\_blurred.jpg')  
  
 # Load and blur the gym background image  
 pil\_gym\_image = Image.open('resources/backgrounds/gym.png')  
 blurred\_gym\_image = pil\_gym\_image.filter(ImageFilter.BLUR)  
 blurred\_gym\_image.save('resources/backgrounds/gym\_blurred.png')  
  
 # Load and blur the room background image  
 pil\_room\_image = Image.open('resources/backgrounds/room.jpg')  
 blurred\_room\_image = pil\_room\_image.filter(ImageFilter.BLUR)  
 blurred\_room\_image.save('resources/backgrounds/room\_blurred.jpg')  
  
 # Load and blur the bedroom background image  
 pil\_bedroom\_image = Image.open('resources/backgrounds/bedroom.jpg')  
 blurred\_bedroom\_image = pil\_bedroom\_image.filter(ImageFilter.BLUR)  
 blurred\_bedroom\_image.save('resources/backgrounds/bedrblur.jpg')  
  
 # Load and blur the plaza background image  
 pil\_plaza\_image = Image.open('resources/backgrounds/plaza.jpg')  
 blurred\_plaza\_image = pil\_plaza\_image.filter(ImageFilter.BLUR)  
 blurred\_plaza\_image.save('resources/backgrounds/plaza\_blurred.jpg')  
 except IOError as e:  
 print(f"Error processing images: {e}")  
  
"""UNIVERSAL FUNCTIONS------------------------------------------------------------------------------------------------"""  
  
"""#Sorta the whole game ============================================================================================="""  
#Menu class for the whole game  
class GameMenu(object):  
 def \_\_init\_\_(self):  
 try:  
 # Initialize the screen with resizable option  
 self.SCREEN = pg.display.set\_mode((width, height), pg.RESIZABLE)  
 self.SCREEN = pg.display.set\_mode((width, height))  
 pg.display.set\_caption("Mapuan Typing Mania")  
  
 # Load and scale the background image  
 self.BG = stretch(pg.image.load("resources/backgrounds/menu.jpg"), (width, height)).convert\_alpha()  
 self.BG = pg.transform.scale(self.BG, (width, height))  
 self.phase = 0  
  
 # Initialize and play menu music  
 pg.mixer.init()  
 self.menu\_music = "resources/sounds/songs/menu.mp3"  
 pg.mixer.music.load(self.menu\_music)  
 pg.mixer.music.play(-1)  
 except pg.error as e:  
 print(f"Error initializing game menu: {e}")  
 sys.exit()  
  
 # Pang animate sa background nga ka macolor lahi  
 def animate\_background(self):  
 amplitude = 5  
 frequency = 0.01  
 color\_shift = 50  
 self.phase += 0.05  
  
 # Calculate color transition based on phase  
 t = (np.sin(self.phase) + 1) / 2  
 r = int(255 \* (1 - t) + 128 \* t)  
 g = int(200 \* (1 - t) + 128 \* t)  
 b = int(100 \* (1 - t) + 128 \* t)  
 bg\_color = (r, g, b)  
  
 try:  
 # Apply wave effect to the background image  
 wavy\_bg = apply\_wave\_effect(self.BG.copy(), amplitude, frequency, self.phase, color\_shift)  
 wavy\_bg.fill(bg\_color, special\_flags=pg.BLEND\_RGBA\_MULT)  
 except Exception as e:  
 print(f"Error animating background: {e}")  
 return self.BG # Return the original background in case of error  
  
 return wavy\_bg  
  
 # pang resize unta sa screen pero wla matama  
 # def handle\_resize\_event(self, event):  
 # self.SCREEN = pg.display.set\_mode((event.w, event.h), pg.RESIZABLE)  
 # self.BG = stretch(pg.image.load("resources/backgrounds/menu.jpg"), (event.w, event.h)).convert\_alpha()  
 # self.BG = pg.transform.scale(self.BG, (event.w, event.h))  
  
 # Para ni sa menu  
 def play(self):  
 while True:  
 # Get the mouse position and animate the background  
 PLAY\_MOUSE\_POS = pg.mouse.get\_pos()  
 animated\_bg = self.animate\_background()  
 self.SCREEN.blit(animated\_bg, (0, 0))  
  
 gap = 85 # Define the gap between buttons  
  
 # Render the "Choose a stage:" text with shadow  
 PLAY\_TEXT = get\_Font(30).render("Choose a stage:", True, "White")  
 PLAY\_TEXT\_SHADOW = get\_Font(30).render("Choose a stage:", True, "Black")  
 PLAY\_RECT = PLAY\_TEXT.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2 - 250))  
 self.SCREEN.blit(PLAY\_TEXT\_SHADOW, PLAY\_RECT.move(2, 2))  
 self.SCREEN.blit(PLAY\_TEXT, PLAY\_RECT)  
  
 # Render the "Press ESC" text with shadow  
 ESC\_Text = get\_Font(12).render("Press ESC to go back, "  
 "mostly works for most displays or stages too!", True, "White")  
 ESC\_Text\_Shadow = get\_Font(12).render("Press ESC to go back, "  
 "mostly works for most displays or stages too!", True, "Black")  
 ESC\_Rect = ESC\_Text.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 20))  
 self.SCREEN.blit(ESC\_Text\_Shadow, ESC\_Rect.move(2, 2))  
 self.SCREEN.blit(ESC\_Text, ESC\_Rect)  
  
 # adjust the button start in the screen so that it will go in the rightplace  
 button\_y\_start = self.SCREEN.get\_height() // 2 - 180  
  
 try:  
 # Load and position buttons  
 TUTORIAL\_BUTTON = ImageButton(pg.image.load("resources/buttons/tutorial.gif"),  
 pos=(self.SCREEN.get\_width() // 2, button\_y\_start))  
 INTRO\_BUTTON = ImageButton(pg.image.load("resources/buttons/intro.gif"),  
 pos=(self.SCREEN.get\_width() // 2, button\_y\_start + gap))  
 STAGE1\_BUTTON = ImageButton(pg.image.load("resources/buttons/stage1.gif"),  
 pos=(self.SCREEN.get\_width() // 2, button\_y\_start + 2 \* gap))  
 STAGE2\_BUTTON = ImageButton(pg.image.load("resources/buttons/stage2.gif"),  
 pos=(self.SCREEN.get\_width() // 2, button\_y\_start + 3 \* gap))  
 STAGE3\_BUTTON = ImageButton(pg.image.load("resources/buttons/stage3.gif"),  
 pos=(self.SCREEN.get\_width() // 2, button\_y\_start + 4 \* gap))  
 ENDING\_BUTTON = ImageButton(pg.image.load("resources/buttons/ending.gif"),  
 pos=(self.SCREEN.get\_width() // 2, button\_y\_start + 5 \* gap))  
 except pg.error as e:  
 print(f"Error loading button images: {e}")  
 sys.exit()  
  
 # Change button size on hover  
 TUTORIAL\_BUTTON.change\_size\_on\_hover(PLAY\_MOUSE\_POS)  
 INTRO\_BUTTON.change\_size\_on\_hover(PLAY\_MOUSE\_POS)  
 STAGE1\_BUTTON.change\_size\_on\_hover(PLAY\_MOUSE\_POS)  
 STAGE2\_BUTTON.change\_size\_on\_hover(PLAY\_MOUSE\_POS)  
 STAGE3\_BUTTON.change\_size\_on\_hover(PLAY\_MOUSE\_POS)  
 ENDING\_BUTTON.change\_size\_on\_hover(PLAY\_MOUSE\_POS)  
  
 # Update buttons on the screen  
 TUTORIAL\_BUTTON.update(self.SCREEN)  
 INTRO\_BUTTON.update(self.SCREEN)  
 STAGE1\_BUTTON.update(self.SCREEN)  
 STAGE2\_BUTTON.update(self.SCREEN)  
 STAGE3\_BUTTON.update(self.SCREEN)  
 ENDING\_BUTTON.update(self.SCREEN)  
  
 # Main game loop  
 # tanawun unsa mahibato sa game loop, if naay event or wala  
 for event in pg.event.get():  
 if event.type == pg.QUIT: #Mo quit ang game if naay event na quit  
 pg.quit()  
 sys.exit()  
 if event.type == pg.MOUSEBUTTONDOWN: #Mo check if naay mouse click  
 if TUTORIAL\_BUTTON.check\_for\_input(PLAY\_MOUSE\_POS): #Mo check if naay mouse click sa tutorial button  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 Tutorial(self.SCREEN).run()  
 elif INTRO\_BUTTON.check\_for\_input(PLAY\_MOUSE\_POS): #Mo check if naay mouse click sa intro button  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 game\_intro = introduction.Intro(self.SCREEN) #Mo run sa intro  
 game\_intro.run()  
 elif STAGE1\_BUTTON.check\_for\_input(PLAY\_MOUSE\_POS): #Mo check if naay mouse click sa stage1 button  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 stage\_intro = introduction.Stage1Intro(self.SCREEN) #Mo run sa stage1  
 stage\_intro.run()  
 elif STAGE2\_BUTTON.check\_for\_input(PLAY\_MOUSE\_POS): #Mo check if naay mouse click sa stage2 button  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 game\_intro2 = introduction.Stage2Intro(self.SCREEN)  
 game\_intro2.run()  
 elif STAGE3\_BUTTON.check\_for\_input(PLAY\_MOUSE\_POS): #Mo check if naay mouse click sa stage3 button  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 game\_intro3 = introduction.Stage3Intro(self.SCREEN)  
 game\_intro3.run()  
 elif ENDING\_BUTTON.check\_for\_input(PLAY\_MOUSE\_POS): #Mo check if naay mouse click sa ending button  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 game\_ending = endings.Ending(self.SCREEN)  
 game\_ending.run()  
 break  
 elif event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE: #Mo check if naay keyboard input na escape  
 main\_menu = GameMenu()  
 main\_menu.main\_Menu()  
  
 pg.display.update()  
  
 # Mao ni una nga menu  
 def main\_Menu(self):  
 while True:  
 MOUSE\_POS = pg.mouse.get\_pos()  
 animated\_bg = self.animate\_background()  
 self.SCREEN.blit(animated\_bg, (0, 0))  
  
 try:  
 play\_image = pg.image.load("resources/buttons/play.gif")  
 scaled\_play\_image = pg.transform.scale(play\_image, (  
 int(play\_image.get\_width() \* 1.5), int(play\_image.get\_height() \* 1.5)))  
  
 quit\_image = pg.image.load("resources/buttons/quit.gif")  
 scaled\_quit\_image = pg.transform.scale(quit\_image, (  
 int(quit\_image.get\_width() \* 1.5), int(quit\_image.get\_height() \* 1.5)))  
   
 leaderboard\_image = pg.image.load("resources/buttons/leaderboard.gif")  
 scaled\_leaderboard\_image = pg.transform.scale(leaderboard\_image, (  
 int(leaderboard\_image.get\_width() \* 1.5), int(leaderboard\_image.get\_height() \* 1.5)))  
  
 # Calculate the center coordinates and define vertical spacing  
 center\_x = self.SCREEN.get\_width() // 2  
 center\_y = self.SCREEN.get\_height() // 2  
 vertical\_spacing = 150  
  
 # Create buttons positioned relative to the center of the screen  
 PLAY\_BUTTON = ImageButton(scaled\_play\_image, pos=(center\_x, center\_y - vertical\_spacing))  
 LEADERBOARD\_BUTTON = ImageButton(scaled\_leaderboard\_image, pos=(center\_x, center\_y))  
 QUIT\_BUTTON = ImageButton(scaled\_quit\_image, pos=(center\_x, center\_y + vertical\_spacing))  
  
 except pg.error as e:  
 print(f"Error loading images: {e}")  
 sys.exit()  
  
 # OPTIONS\_BUTTON = ImageButton(pg.image.load("resources/buttons/options.gif"),  
 # pos=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2 - 50))  
  
 PLAY\_BUTTON.change\_size\_on\_hover(MOUSE\_POS)  
 # OPTIONS\_BUTTON.change\_size\_on\_hover(MOUSE\_POS)  
 LEADERBOARD\_BUTTON.change\_size\_on\_hover(MOUSE\_POS)  
 QUIT\_BUTTON.change\_size\_on\_hover(MOUSE\_POS)  
  
 PLAY\_BUTTON.update(self.SCREEN)  
 # OPTIONS\_BUTTON.update(self.SCREEN)  
 LEADERBOARD\_BUTTON.update(self.SCREEN)  
 QUIT\_BUTTON.update(self.SCREEN)  
  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 # elif event.type == pg.VIDEORESIZE:  
 # self.handle\_resize\_event(event)  
 if event.type == pg.MOUSEBUTTONDOWN:  
 if PLAY\_BUTTON.check\_for\_input(MOUSE\_POS):  
 self.play()  
 # elif OPTIONS\_BUTTON.check\_for\_input(MOUSE\_POS):  
 # self.options()  
 elif LEADERBOARD\_BUTTON.check\_for\_input(MOUSE\_POS):  
 from optional import Leaderboard  
 leaderboard = Leaderboard(self.SCREEN)  
 leaderboard.run()  
 else:  
 if QUIT\_BUTTON.check\_for\_input(MOUSE\_POS):  
 pg.quit()  
 sys.exit()  
  
 pg.display.update()  
  
 # Mao ni ang title screen  
 def title\_screen(self):  
 username = ""  
 input\_active = True  
 font = get\_Font(30)  
  
 try:  
 title\_image = pg.image.load("resources/backgrounds/title.gif").convert\_alpha()  
 title\_image = pg.transform.scale(title\_image,  
 (int(title\_image.get\_width() \* 1.2), title\_image.get\_height()))  
 except pg.error as e:  
 print(f"Error loading title image: {e}")  
 sys.exit()  
  
 # Define the position of the title image  
 title\_rect = title\_image.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2.5))  
 input\_box = pg.Rect(self.SCREEN.get\_width() // 2 - 100, title\_rect.bottom + 70, 200, 50)  
 color\_inactive = pg.Color(255, 255, 255)  
 color\_active = pg.Color('red')  
 color = color\_inactive  
  
 # Main title screen loop  
 while True:  
 MOUSE\_POS = pg.mouse.get\_pos()  
 animated\_bg = self.animate\_background()  
 self.SCREEN.blit(animated\_bg, (0, 0))  
  
 # Display the title image  
 self.SCREEN.blit(title\_image, title\_rect)  
  
 # Display the username input box  
 if input\_active:  
 prompt\_text = font.render("Enter your name (max 10 letters):", True, "White")  
 prompt\_rect = prompt\_text.get\_rect(center=(self.SCREEN.get\_width() // 2, title\_rect.bottom + 50))  
 self.SCREEN.blit(prompt\_text, prompt\_rect)  
  
 # Render the username text and input box  
 txt\_surface = font.render(username, True, color)  
 width = max(200, txt\_surface.get\_width() + 10)  
 input\_box.w = width  
 self.SCREEN.blit(txt\_surface, (input\_box.centerx - txt\_surface.get\_width() // 2, input\_box.y + 5))  
 pg.draw.rect(self.SCREEN, color, input\_box, 2)  
 else:  
 # Display the prompt to press any key to continue  
 prompt\_text = font.render("Press any key to continue", True, "White")  
 prompt\_rect = prompt\_text.get\_rect(center=(self.SCREEN.get\_width() // 2, title\_rect.bottom + 50))  
 prompt\_color = "Red" if prompt\_rect.collidepoint(MOUSE\_POS) else "White"  
 prompt\_text = font.render("Press any key to continue", True, prompt\_color)  
 self.SCREEN.blit(prompt\_text, prompt\_rect)  
  
 # Main title screen loop  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if input\_active:  
 if event.type == pg.MOUSEBUTTONDOWN:  
 if input\_box.collidepoint(event.pos):  
 color = color\_active  
 else:  
 color = color\_inactive  
 if event.type == pg.KEYDOWN:  
 if event.key == pg.K\_RETURN:  
 if 1 <= len(username) <= 10:  
 input\_active = False  
 try:  
 with open("resources/players.txt", "w") as file:  
 file.write(username + "\n")  
 except IOError as e:  
 print(f"Error writing to file: {e}")  
 else:  
 error\_text = font.render("Username must be at most 10 letters long", True, "Red")  
 error\_rect = error\_text.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, input\_box.bottom + 30))  
 self.SCREEN.blit(error\_text, error\_rect)  
 elif event.key == pg.K\_BACKSPACE:  
 username = username[:-1]  
 else:  
 if len(username) < 10:  
 username += event.unicode  
 else:  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 self.main\_Menu()  
  
 pg.display.update()  
  
"""#Sorta the whole game \*class ============================================================================================="""  
  
"""ENGINE SOUNDS VROOM VROOM ============================================================================================="""  
def main():  
 process\_images() # Run the image processing code before starting the game  
 game = GameMenu()  
 game.title\_screen()  
  
#HOLDER OF REALITY  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()  
"""ENGINE SOUNDS VROOM VROOM ============================================================================================="""

# WORDS.PY

# Dictionary for the faliing words ingame.  
tutorial\_words = {  
 3: {'CPU', 'RAM', 'ENG', 'LAB', 'APP', 'BOT', 'NET', 'CAM', 'SYS', 'OSI'},  
 4: {'Code', 'Math', 'Java', 'Data', 'Bits', 'Node', 'Chip', 'Wire', 'Term', 'Quiz', 'Byte', 'Test', 'Blog', 'Mode',  
 'Plan', 'User', 'Link', 'Role', 'Task', 'Lens'},  
 5: {'Logic', 'Debug', 'Graph', 'Cache', 'Frame', 'Stack', 'Panel', 'Table', 'Input', 'Board', 'Server', 'Binary',  
 'Module', 'Output', 'Kernel'},  
 6: {'Network'},  
 7: {'Digital'},  
 8: {'Software', 'Hardware'},  
 9: {'Project'}  
}  
  
stage1\_words = {  
 3: {'Pen', 'ink', 'Map', 'win', 'top', 'low', 'One', 'two', 'six', 'Ten', 'yet', 'try', 'log', 'sum', 'let', 'set',  
 'gap', 'aim', 'may', 'act', 'add', 'key', 'bot', 'new', 'old', 'cut', 'eat', 'see', 'sad', 'mad', 'fix', 'run',  
 'fun', 'use', 'sit', 'row', 'fig', 'out', 'ask', 'mix', 'CPU', 'RAM', 'ENG', 'LAB', 'APP', 'NET', 'CAM', 'SYS', 'OSI'},  
 4: {'exam', 'pass', 'fail', 'high', 'mark', 'book', 'test', 'time', 'comp', 'rank', 'best', 'edit', 'copy', 'drop',  
 'miss', 'list', 'quiz', 'zero', 'grad', 'late', 'note', 'page', 'goal', 'earn', 'redo', 'mean', 'term', 'read',  
 'name', 'task', 'prep', 'work', 'send', 'club', 'seat', 'plot', 'news', 'easy', 'mode', 'idea', 'Code', 'Math', 'Java', 'Data', 'Bits', 'Node', 'Chip', 'Wire', 'Term', 'Quiz', 'Byte', 'Test', 'Blog', 'Mode', 'Plan', 'User', 'Link', 'Role', 'Task', 'Lens'},  
 5: {'cheat', 'score', 'Paper', 'study', 'topic', 'learn', 'group', 'honor', 'break', 'teach', 'essay', 'grade',  
 'class', 'notes', 'math', 'write', 'check', 'focus', 'error', 'rival', 'event', 'board', 'books', 'draft',  
 'think', 'email', 'skill', 'rules', 'award', 'share', 'submit', 'report', 'module', 'review', 'school',  
 'lesson', 'winner', 'author', 'revise', 'speech', 'values', 'answer', 'online', 'typing', 'format', 'medal',  
 'effort', 'notice', 'reason', 'Effort', 'ranking', 'passing', 'failing', 'project', 'contest', 'writing',  
 'professor', 'exposure', 'evaluate', 'reference', 'scholarship', 'materials', 'critical', 'organize',  
 'challenge', 'presented', 'experience', 'excellence', 'portfolio', 'motivation', 'discussion', 'objectives',  
 'simulation', 'participation', 'leadership', 'assessment', 'university', 'principles', 'experiment',  
 'curriculum', 'competency', 'workshops', 'evaluation', 'foundation', 'submission', 'graduation', 'confidence',  
 'requirements', 'Logic', 'Debug', 'Graph', 'Cache', 'Frame', 'Stack', 'Panel', 'Table', 'Input', 'Board', 'Server', 'Binary', 'Module', 'Output', 'Kernel'},  
 6: {'Network', 'meeting', 'efforts', 'highest', 'records', 'scoring', 'courses', 'scholars', 'library', 'grading', 'entries',  
 'testing', 'authors', 'revises', 'choices', 'research', 'feedback', 'learning', 'practice', 'teamwork',  
 'magazine', 'notebook', 'elective', 'subjects', 'revising', 'lectures', 'diploma', 'creation', 'tutoring',  
 'syllabus', 'guidance', 'seminar', 'proposal', 'analysis', 'training', 'classroom', 'education', 'graduate',  
 'submitted', 'knowledge', 'plagiarism', 'qualified', 'tutorial', 'solutions', 'judgment', 'syllabary'},  
 7: {'Digital', 'coursework', 'strategies', 'methodology', 'articulation', 'proficiency', 'recommendation',  
 'accreditation', 'specialization', 'extracurricular', 'professionalism', 'comprehension', 'achievements',  
 'communication', 'interdisciplinary', 'responsibility', 'certification', 'collaboration', 'encouragement',  
 'accountability', 'apprenticeship', 'administration', 'independence', 'computational', 'technological',  
 'demonstration', 'accomplishments', 'organization'},  
 8: {'Software', 'Hardware'},  
 9: {'Project'}  
}  
  
stage2\_words = {  
 3: {'key', 'bot', 'new', 'old', 'cut', 'see', 'fix', 'run', 'fun', 'use', 'sit', 'row', 'out', 'ask', 'mix', 'CPU',  
 'RAM', 'APP', 'NET', 'SYS', 'tap', 'hit', 'win', 'tie', 'top', 'end', 'hit', 'aim', 'act', 'try', 'let', 'set',  
 'gap', 'add', 'may', 'sum', 'low', 'ten', 'bit', 'tab', 'pad', 'job', 'tag', 'key', 'ink', 'one', 'two', 'zip',  
 'max', 'hot', 'yes', 'red', 'log', 'cut', 'tip', 'box', 'sip', 'pic', 'bat', 'fan', 'pit', 'dot', 'jam', 'dig',  
 'bug', 'rid', 'cap', 'kit', 'top', 'lap', 'met', 'web'},  
  
 4: {'exam', 'pass', 'high', 'mark', 'test', 'time', 'rank', 'best', 'edit', 'copy', 'list', 'quiz', 'zero', 'redo',  
 'note', 'read', 'task', 'mode', 'idea', 'Code', 'Math', 'Java', 'Data', 'Bits', 'Byte', 'Plan', 'User', 'Link',  
 'Role', 'fast', 'slow', 'race', 'pace', 'type', 'keys', 'jump', 'move', 'goal', 'dash', 'scan', 'word', 'page',  
 'rank', 'team', 'push', 'save', 'flow', 'beat', 'best', 'note', 'grab', 'play', 'test', 'sort', 'flip', 'good',  
 'next', 'hard', 'ease', 'done', 'hold', 'show', 'case', 'load', 'time', 'stop', 'base', 'taps', 'unit', 'turn'},  
  
 5: {'score', 'Paper', 'speed', 'typing', 'focus', 'error', 'rival', 'board', 'rules', 'award', 'submit', 'report',  
 'review', 'winner', 'answer', 'online', 'format', 'medal', 'effort', 'ranking', 'contest', 'writing',  
 'logic', 'debug', 'cache', 'frame', 'stack', 'panel', 'table', 'input', 'board', 'server', 'binary', 'module',  
 'output', 'kernel', 'match', 'press', 'style', 'touch', 'quick', 'click', 'trace', 'write', 'draft', 'train',  
 'excel', 'timer', 'words', 'point', 'score', 'final', 'place', 'haste', 'event'},  
  
 6: {'Network', 'records', 'scoring', 'entries', 'testing', 'research', 'feedback', 'practice', 'guidance',  
 'analysis', 'training', 'education', 'graduate', 'knowledge', 'qualified', 'tutorial', 'solutions', 'accuracy',  
 'judgment'},  
  
 7: {'Digital', 'strategies', 'proficiency', 'recommendation', 'comprehension', 'communication', 'responsibility',  
 'certification', 'collaboration', 'encouragement', 'accountability', 'computational', 'demonstration',  
 'accomplishments'},  
  
 8: {'Software', 'Hardware', 'Keystroke', 'Leaderboard', 'WPM'},  
  
 9: {'Championship'}  
}  
  
stage3\_words = {  
 3: {'win', 'top', 'new', 'old', 'cut', 'see', 'fix', 'run', 'use', 'mix', 'CPU', 'RAM',  
 'APP', 'NET', 'SYS', 'hit',  
 'try', 'set', 'gap', 'add', 'sum', 'max', 'hot', 'log', 'box', 'tap', 'job', 'zip',  
 'bit', 'pit', 'cap', 'kit'},  
  
 4: {'high', 'test', 'rank', 'best', 'quiz', 'redo', 'task', 'mode', 'idea', 'Code',  
 'Math', 'Java', 'Data', 'Byte',  
 'Plan', 'User', 'Link', 'Role', 'fast', 'pace', 'keys', 'jump', 'move', 'goal',  
 'dash', 'scan', 'word', 'team',  
 'push', 'save', 'flow', 'play', 'sort', 'flip', 'load', 'turn'},  
  
 5: {'score', 'speed', 'focus', 'rival', 'rules', 'award', 'submit', 'winner',  
 'online', 'format', 'medal', 'effort',  
 'ranking', 'contest', 'writing', 'logic', 'debug', 'frame', 'stack', 'table',  
 'input', 'board', 'server', 'binary',  
 'module', 'output', 'kernel', 'match', 'press', 'style', 'touch', 'quick',  
 'click', 'trace', 'write', 'train',  
 'excel', 'timer', 'final', 'event'},  
  
 6: {'Network', 'records', 'scoring', 'entries', 'testing', 'research', 'feedback',  
 'practice', 'guidance', 'analysis',  
 'training', 'education', 'graduate', 'knowledge', 'qualified', 'accuracy',  
 'judgment', 'strategy', 'reaction'},  
  
 7: {'Digital', 'strategies', 'proficiency', 'competition', 'communication',  
 'responsibility', 'certification',  
 'collaboration', 'encouragement', 'accomplishments', 'demonstration',  
 'computational', 'determination'},  
  
 8: {'Software', 'Hardware', 'Keystroke', 'Leaderboard', 'WPM', 'Performance'},  
  
 9: {'Championship', 'Finalist'}  
}  
  
  
bonus\_words = {  
 3: {'UNO', 'DOS', 'EMMY', 'LAB', 'CAD', 'PCB', 'FEM', 'FFT', 'VLS'},  
 4: {'IP-GRADE', 'PASAR', 'TRES', 'CPE001', 'QUIZ', 'LABS', 'CODE', 'MATH', 'FEES',  
 'DRAW', 'LOAD', 'THESIS', 'CRAM', 'SEAT', 'WIFI', 'PASS'},  
 5: {'SINGKO', 'HUGOT', 'PUYAT', 'DONUT', 'GRACE'},  
 6: {'KABADO', 'BURNOUT', 'SABLAY', 'WALKOUT', 'VIVA'},  
 7: {'DEBARRED', 'DELOADED', 'GAMING', 'DEFENSE', 'GHOSTING', 'HAPPY T', 'HOHOL',  
 'KKB', 'BUDOL'},  
 8: {'TRANSFEREE', 'SUMMER TERM', 'MIDTERMS', 'QR CODE', 'MALAYAN', 'MATH BAGSAK',  
 'TYPHOON BREAK', 'PISO WI-FI', 'CPE LIFE',  
 'CODING HELL', 'DEBUG MODE', 'STACK OVERFLOW',  
 'SYNTAX ERROR', 'LOGIC ERROR', 'SYNTAX ERROR',  
 'INFINITE LOOP', 'LATE NIGHT CODE', 'ARDUINO',  
 'RASPBERRY PI', 'FPGA', 'CIRCUITS',  
 'RESISTANCE', 'OHM’S LAW', 'KIRCHHOFF’S LAW',  
 'MICROPROCESSOR', 'ASSEMBLY LANGUAGE', 'C++', 'PYTHON'  
 'PYTHON', 'JAVA', 'MATLAB', 'NETWORKING', 'GITHUB',  
 'MERGE CONFLICT', 'FINAL PROJECT', 'OJT', 'INTERNSHIP', 'PROGRAMMING'  
 'CAPSTONE'},  
 9: {'MAPUA MINDANAO', 'REQUILLO'}  
}

# GENWORDS.PY

# Module sa pagsulat ng mga score sa isang file at pagbabasa nila muli.  
import sys  
import random  
from string import printable  
from collections import defaultdict  
  
# Initialize the tutorial\_words dictionary  
words = defaultdict(set)  
  
# Mao ni sila mag generate us listahan nga random para gamiton sa dula  
def generate\_words\_tutorial():  
 *"""  
 Generate lists of words from the tutorial\_words and bonus\_words dictionaries.  
  
 Returns:  
 two lists of words for tutorial and bonus.  
 """* try:  
 from words import tutorial\_words, bonus\_words  
  
 tutorial\_list = [word for word\_list in tutorial\_words.values() for word in word\_list]  
 bonus\_list = [word for word\_list in bonus\_words.values() for word in word\_list]  
  
 random.shuffle(tutorial\_list)  
 random.shuffle(bonus\_list)  
  
 return tutorial\_list, bonus\_list  
 except ImportError:  
 return [], []  
  
def generate\_words\_stage1():  
 *"""  
 Generate lists of words from the stage1\_words, and bonus\_words dictionaries.  
  
 Returns:  
 two lists of words for stage1, bonus.  
 """* try:  
 from words import stage1\_words, bonus\_words  
  
 stage1\_list = [word for word\_list in stage1\_words.values() for word in word\_list]  
 bonus\_list = [word for word\_list in bonus\_words.values() for word in word\_list]  
  
 random.shuffle(stage1\_list)  
 random.shuffle(bonus\_list)  
  
 return stage1\_list, bonus\_list  
 except ImportError:  
 return [], []  
  
  
def generate\_words\_stage2():  
 *"""  
 Generate lists of words from the stage2\_words and bonus\_words dictionaries.  
  
 Returns:  
 Two lists of words for stage2 and bonus.  
 """* try:  
 from words import stage2\_words, bonus\_words  
  
 stage2\_list = [word for word\_list in stage2\_words.values() for word in word\_list]  
 bonus\_list = [word for word\_list in bonus\_words.values() for word in word\_list]  
  
 random.shuffle(stage2\_list)  
 random.shuffle(bonus\_list)  
  
 return stage2\_list, bonus\_list  
 except ImportError:  
 return [], []  
  
def generate\_words\_stage3():  
 *"""  
 Generate lists of words from the stage3\_words and bonus\_words dictionaries.  
  
 Returns:  
 Two lists of words for stage3 and bonus.  
 """* try:  
 from words import stage3\_words, bonus\_words  
  
 stage3\_list = [word for word\_list in stage3\_words.values() for word in word\_list]  
 bonus\_list = [word for word\_list in bonus\_words.values() for word in word\_list]  
  
 random.shuffle(stage3\_list)  
 random.shuffle(bonus\_list)  
  
 return stage3\_list, bonus\_list  
 except ImportError:  
 return [], []  
  
# Holder of Reality  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) < 2:  
 print("Usage: python genwords.py <dictfile>")  
 sys.exit(1)  
  
 dictfile = sys.argv[1]  
  
 # Read and process dictionary file  
 with open(dictfile) as file:  
 wordlist = [w.strip().lower() for w in file.read().split()]  
  
 random.shuffle(wordlist)  
 wordlist = list(filter(lambda w: all(c in printable for c in w), wordlist))  
 wordlist = wordlist[:1300]  
  
 # Group tutorial\_words by length  
 for word in wordlist:  
 words[len(word)].add(word)  
  
 # Write processed tutorial\_words to tutorial\_words.py  
 with open("words.py", "w") as file:  
 file.write("tutorial\_words = {\n")  
 for length, word\_set in words.items():  
 file.write(f" {length}: {sorted(word\_set)},\n")  
 file.write("}\n")

# SCORES.PY

# Module for writing scores to a file and reading them back.  
import os  
scorefile = os.path.join(os.path.dirname(\_\_file\_\_), "resources/highscore.txt")  
playerfile = os.path.join(os.path.dirname(\_\_file\_\_), "resources/players.txt")  
  
# Magload sa skore  
def load\_score():  
 *""" Returns the highest score, or 0 if no one has scored yet """* try:  
 with open(scorefile) as file:  
 scores = sorted([int(score.strip())  
 for score in file.readlines()  
 if score.strip().isdigit()], reverse=True)  
 except IOError:  
 scores = []  
  
 return scores[0] if scores else 0  
  
def load\_score\_with\_player():  
 *""" Returns a tuple of the highest score and the corresponding player, or (0, None) if no one has scored yet """* try:  
 with open(scorefile) as file:  
 scores = {line.split(":")[0]: int(line.split(":")[1]) for line in file.readlines()}  
 if scores:  
 highest\_score\_player = max(scores, key=scores.get)  
 return scores[highest\_score\_player], highest\_score\_player  
 else:  
 return 0, None  
 except IOError:  
 return 0, None  
  
# I-save ang skore  
def get\_current\_player():  
 try:  
 with open(playerfile, "r") as file:  
 return file.readline().strip()  
 except Exception as e:  
 print(f"Error reading current player: {e}")  
 return None  
  
def write\_score(score):  
 try:  
 username = get\_current\_player()  
 if not username:  
 raise ValueError("No current player found")  
  
 # Read existing scores  
 if os.path.exists(scorefile):  
 with open(scorefile, "r") as file:  
 scores = {line.split(":")[0]: int(line.split(":")[1]) for line in file.readlines()}  
 else:  
 scores = {}  
  
 # Update the score for the given username  
 scores[username] = max(score, scores.get(username, 0))  
  
 # Write updated scores back to the file  
 with open(scorefile, "w") as file:  
 for user, score in scores.items():  
 file.write(f"{user}:{score}\n")  
 except Exception as e:  
 print(f"Error writing score: {e}")

# BGFIX.PY

# Module tig stretch sa mga images para masakto sa screen ug taro  
import pygame as pg  
# from pygame import Surface  
# import os  
# import glob  
# import random  
# from collections import namedtuple  
# ayaw hilabti ang naka comment kay para na unta sa uban nga code  
  
# mao ni siya tig stretch  
def stretch(surf, size, upscale\_factor=2):  
 width, height = size  
  
 imgw, imgh = surf.get\_rect().size  
  
 # Upscale the image by the upscale\_factor  
 imgw, imgh = int(imgw \* upscale\_factor), int(imgh \* upscale\_factor)  
 surf = pg.transform.smoothscale(surf, (imgw, imgh))  
  
 xfactor = float(width) / imgw  
 surf = pg.transform.smoothscale(surf, (int(imgw \* xfactor), int(imgh \* xfactor)))  
  
 new\_imgw, new\_imgh = surf.get\_rect().size  
  
 if new\_imgh < height:  
 yfactor = float(height) / new\_imgh  
 surf = pg.transform.smoothscale(surf, (int(new\_imgw \* yfactor), int(new\_imgh \* yfactor)))  
  
 return surf  
  
# def endswith\_any(s, \*suffixes):  
# return any(s.endswith(suffix) for suffix in suffixes)  
  
# def is\_image(fname):  
# return endswith\_any(fname, ".png", ".jpg", ".jpeg", ".bmp")  
  
# class Background(object):  
# def \_\_init\_\_(self, size):  
# self.size = size # Ensure self.size is defined before it is used  
# width, height = self.size  
#  
# self.surf = Surface(size)  
#  
# self.backgrounds = []  
#  
# files = glob.glob(os.path.join(os.path.dirname(\_\_file\_\_), "resources/backgrounds/\*"))  
#  
# bg = namedtuple("background", "image")  
# for fname in filter(is\_image, files):  
# self.backgrounds.append(  
# bg(image=stretch(pg.image.load(fname).convert(), self.size))  
# )  
#  
# random.shuffle(self.backgrounds)  
#  
# self.timer = 0  
# self.frequency = 25 # new background every N seconds  
# self.current\_bg = 0 # index of the current bg in self.backgrounds  
#  
# self.fadetime = .7  
# self.fading = 0  
# self.donefading = True  
#  
# self.set\_background()  
#  
# def update(self, timepassed):  
# old\_timer, self.timer = self.timer, (self.timer + timepassed) % self.frequency  
#  
# if self.fading < 0:  
# self.donefading = True  
# self.fading = 0  
# elif self.fading:  
# self.fading = self.fading - timepassed  
#  
# if old\_timer > self.timer:  
# old\_bg, self.current\_bg = self.current\_bg, (self.current\_bg + 1) % len(self.backgrounds)  
# if self.current\_bg != old\_bg:  
# self.fading = self.fadetime  
#  
# if self.fading:  
# self.set\_background()  
# elif self.donefading:  
# self.donefading = False  
# self.set\_background()  
#  
# def get\_current\_bg(self):  
# return self.backgrounds[self.current\_bg]  
#  
# def set\_background(self):  
# if self.fading:  
# old\_bg = (self.current\_bg - 1) % len(self.backgrounds)  
# new = self.get\_current\_bg().image  
# old = self.backgrounds[old\_bg].image.copy()  
# old.set\_alpha(self.fading \* 255 / self.fadetime)  
#  
# self.blit(new)  
# self.blit(old)  
# else:  
# self.blit(self.get\_current\_bg().image)  
#  
# def browse(self, direction):  
# dirs = {'forward': 1, 'backward': -1}  
# self.current\_bg = (self.current\_bg + dirs[direction]) % len(self.backgrounds)  
# self.set\_background()  
# self.timer = 0  
#  
# def blit(self, surf):  
# self.surf.blit(surf, surf.get\_rect(centerx=self.surf.get\_rect().centerx,  
# centery=self.surf.get\_rect().centery))

# BUTTONS.PY

# Module for the buttons of the game  
import pygame  
  
# Kato ni daan na button wala na gamit karun  
class Button():  
 def \_\_init\_\_(self, image, pos, text\_input, font, base\_color, hovering\_color):  
 try:  
 self.image = image  
 self.x\_pos = pos[0]  
 self.y\_pos = pos[1]  
 self.font = font  
 self.base\_color, self.hovering\_color = base\_color, hovering\_color  
 self.text\_input = text\_input  
 self.text = self.font.render(self.text\_input, True, self.base\_color)  
 if self.image is None:  
 self.image = self.text  
 self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos))  
 self.text\_rect = self.text.get\_rect(center=(self.x\_pos, self.y\_pos))  
 except Exception as e:  
 print(f"Error initializing Button: {e}")  
  
 def update(self, screen):  
 try:  
 if self.image is not None:  
 screen.blit(self.image, self.rect)  
 screen.blit(self.text, self.text\_rect)  
 except Exception as e:  
 print(f"Error updating Button: {e}")  
  
 def checkForInput(self, position):  
 try:  
 if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):  
 return True  
 return False  
 except Exception as e:  
 print(f"Error checking for input: {e}")  
 return False  
  
 def changeColor(self, position):  
 try:  
 if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):  
 self.text = self.font.render(self.text\_input, True, self.hovering\_color)  
 else:  
 self.text = self.font.render(self.text\_input, True, self.base\_color)  
 except Exception as e:  
 print(f"Error changing color: {e}")  
  
# Kato ni na button gamit karun  
class ImageButton:  
 def \_\_init\_\_(self, image, pos):  
 self.image = image  
 self.original\_image = image  
 self.x\_pos = pos[0]  
 self.y\_pos = pos[1]  
 self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos))  
  
 def update(self, screen):  
 screen.blit(self.image, self.rect)  
  
 def check\_for\_input(self, position):  
 return self.rect.collidepoint(position)  
  
 def change\_size\_on\_hover(self, position):  
 if self.rect.collidepoint(position):  
 self.image = pygame.transform.scale(self.original\_image,  
 (int(self.original\_image.get\_width() \* 0.95), int(self.original\_image.get\_height() \* 0.95)))  
 else:  
 self.image = self.original\_image  
 self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos))

# ENDINGS.PY

import sys  
import threading  
  
import pygame as pg  
from bgfix import stretch  
from introduction import DynamicText  
import introduction  
  
"""ENDING START---------------------------------------------------------------------------------------------------"""  
  
class Ending:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.background = stretch(pg.image.load("resources/backgrounds/bedrblur.jpg"), (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 20)  
 self.text = introduction.import\_text("resources/ending.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
 self.fade\_alpha = 0  
 self.fading = False  
  
 # Start a new thread to load and play ending music  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.music = "resources/sounds/songs/ending.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 credits = Credits(self.SCREEN)  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if self.text\_fully\_displayed:  
 self.fading = True  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 if self.fading:  
 self.fade\_alpha = min(self.fade\_alpha + 5, 255)  
 fade\_surface = pg.Surface(self.SCREEN.get\_size())  
 fade\_surface.fill((0, 0, 0))  
 fade\_surface.set\_alpha(self.fade\_alpha)  
 self.SCREEN.blit(fade\_surface, (0, 0))  
 if self.fade\_alpha == 255:  
 pg.mixer\_music.stop()  
 credits.run()  
  
  
 else:  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
"""ENDINGS END ----------------------------------------------------------------------------------------------------  
  
Credits START----------------------------------------------------------------------------------------------------"""  
class Credits:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.backgrounds = [  
 stretch(pg.image.load(f"resources/backgrounds/{img}"), (width, height)).convert\_alpha()  
 for img in ["gym\_blurred.png", "acadhall\_blurred.jpg", "bedrblur.jpg", "plaza\_blurred.jpg", "room\_blurred.jpg"]  
 ]  
 self.current\_bg\_index = 0  
 self.fade\_alpha = 0  
 self.fade\_in = True  
 self.font = pg.font.Font(None, 24)  
 self.text = self.import\_text("resources/credits.txt").split('\n')  
 self.skip\_prompt = self.font.render("Press ESC to exit", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press ESC to exit", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.last\_switch\_time = pg.time.get\_ticks()  
  
 def run(self):  
 pg.mixer.init()  
 music = "resources/sounds/songs/credits.mp3"  
 pg.mixer\_music.load(music)  
 pg.mixer.music.play(-1)  
 pg.time.set\_timer(pg.USEREVENT, 100)  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 pass  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 return # Exit the credits loop  
  
 self.update\_background()  
 self.SCREEN.blit(self.backgrounds[self.current\_bg\_index], (0, 0))  
 self.draw\_text()  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
 def update\_background(self):  
 current\_time = pg.time.get\_ticks()  
 if current\_time - self.last\_switch\_time > 5000: # Switch every 5 seconds  
 self.last\_switch\_time = current\_time  
 self.fade\_in = not self.fade\_in  
 if not self.fade\_in:  
 self.current\_bg\_index = (self.current\_bg\_index + 1) % len(self.backgrounds)  
  
 if self.fade\_in:  
 self.fade\_alpha = min(self.fade\_alpha + 5, 255)  
 else:  
 self.fade\_alpha = max(self.fade\_alpha - 5, 0)  
  
 self.backgrounds[self.current\_bg\_index].set\_alpha(self.fade\_alpha)  
  
 def draw\_text(self):  
 screen\_width = self.SCREEN.get\_width()  
 screen\_height = self.SCREEN.get\_height()  
 line\_height = self.font.get\_height()  
 y\_offset = 50  
 shadow\_offset = (2, 2)  
  
 # Draw gradient background  
 text\_block\_rect = pg.Rect(15, y\_offset - 8, screen\_width - 30, screen\_height - y\_offset)  
 self.draw\_gradient\_rect(self.SCREEN, text\_block\_rect, (168, 0, 0, 150), (38, 19, 94, 150))  
  
 # Draw the first four sentences in the center  
 for i in range(4):  
 sentence = self.text[i]  
 text\_surface = self.font.render(sentence, True, (255, 255, 255))  
 shadow\_surface = self.font.render(sentence, True, (0, 0, 0))  
 text\_rect = text\_surface.get\_rect(center=(screen\_width // 2, y\_offset))  
 shadow\_rect = text\_rect.move(\*shadow\_offset)  
 self.SCREEN.blit(shadow\_surface, shadow\_rect)  
 self.SCREEN.blit(text\_surface, text\_rect)  
 y\_offset += line\_height + 10  
  
 # Draw the remaining sentences in three columns  
 col1\_x = screen\_width // 6  
 col2\_x = screen\_width // 2  
 col3\_x = 5 \* screen\_width // 6  
 col1\_y\_offset = y\_offset  
 col2\_y\_offset = y\_offset  
 col3\_y\_offset = y\_offset  
  
 # Donut man and Cold in the first column  
 for i in range(4, 17):  
 sentence = self.text[i]  
 text\_surface = self.font.render(sentence, True, (255, 255, 255))  
 shadow\_surface = self.font.render(sentence, True, (0, 0, 0))  
 text\_rect = text\_surface.get\_rect(center=(col1\_x, col1\_y\_offset))  
 shadow\_rect = text\_rect.move(\*shadow\_offset)  
 self.SCREEN.blit(shadow\_surface, shadow\_rect)  
 self.SCREEN.blit(text\_surface, text\_rect)  
 col1\_y\_offset += line\_height + 10  
  
 # Ma'am mmy in the middle column  
 for i in range(17, 21):  
 sentence = self.text[i]  
 text\_surface = self.font.render(sentence, True, (255, 255, 255))  
 shadow\_surface = self.font.render(sentence, True, (0, 0, 0))  
 text\_rect = text\_surface.get\_rect(center=(col2\_x, col2\_y\_offset))  
 shadow\_rect = text\_rect.move(\*shadow\_offset)  
 self.SCREEN.blit(shadow\_surface, shadow\_rect)  
 self.SCREEN.blit(text\_surface, text\_rect)  
 col2\_y\_offset += line\_height + 10  
  
 # Tan and Hao in the third column  
 for i in range(22, len(self.text)):  
 sentence = self.text[i]  
 text\_surface = self.font.render(sentence, True, (255, 255, 255))  
 shadow\_surface = self.font.render(sentence, True, (0, 0, 0))  
 text\_rect = text\_surface.get\_rect(center=(col3\_x, col3\_y\_offset))  
 shadow\_rect = text\_rect.move(\*shadow\_offset)  
 self.SCREEN.blit(shadow\_surface, shadow\_rect)  
 self.SCREEN.blit(text\_surface, text\_rect)  
 col3\_y\_offset += line\_height + 10  
  
 def draw\_gradient\_rect(self, surface, rect, color1, color2):  
 *"""Draw a vertical gradient rectangle with rounded corners."""* color1 = pg.Color(\*color1)  
 color2 = pg.Color(\*color2)  
 height = rect.height  
 width = rect.width  
 radius = 20 # Radius for rounded corners  
  
 # Create a surface with per-pixel alpha  
 gradient\_surface = pg.Surface((width, height), pg.SRCALPHA)  
  
 # Draw the gradient  
 for y in range(height):  
 color = color1.lerp(color2, y / height)  
 pg.draw.line(gradient\_surface, color, (0, y), (width, y))  
  
 # Create a mask for rounded corners  
 mask = pg.Surface((width, height), pg.SRCALPHA)  
 pg.draw.rect(mask, (255, 255, 255, 255), (0, 0, width, height), border\_radius=radius)  
  
 # Apply the mask to the gradient surface  
 gradient\_surface.blit(mask, (0, 0), special\_flags=pg.BLEND\_RGBA\_MIN)  
  
 # Blit the gradient surface onto the target surface  
 surface.blit(gradient\_surface, rect.topleft)  
  
 def import\_text(self, filename):  
 with open(filename, 'r') as file:  
 return file.read()  
"""Credits END----------------------------------------------------------------------------------------------------"""

# INTRODUCTION.PY

import sys  
import threading  
import pygame as pg  
from bgfix import stretch  
from stages import LoadingScreen, Stage1  
  
"""UNIVERSAL FUNCTIONS------------------------------------------------------------------------------------------------"""  
# load mixer  
pg.mixer.init()  
  
# Import text  
def import\_text(file\_path):  
 try:  
 with open(file\_path, "r", encoding="utf-8") as file:  
 return file.read()  
 except IOError as e:  
 print(f"Error reading file {file\_path}: {e}")  
 return "" # Return an empty string in case of error  
  
# Text generator  
def text\_generator(text):  
 tmp = ''  
 for letter in text:  
 tmp += letter  
 yield tmp # Yield the current state of the text  
  
# Mao ni nagahimo atung mga linya sa mga intro ug outro  
class DynamicText:  
 def \_\_init\_\_(self, font, text, pos, speed=20, autoreset=False, line\_spacing=0.5):  
 self.done = False  
 self.font = font  
 self.text = text.split('\n')  
 self.pos = pos  
 self.speed = speed  
 self.autoreset = autoreset  
 self.line\_spacing = line\_spacing  
 self.current\_sentence = 0  
 self.\_gen = text\_generator(self.text[self.current\_sentence])  
 self.rendered\_sentences = []  
 self.current\_text = ''  
 self.update()  
  
 def reset(self):  
 try:  
 self.\_gen = text\_generator(self.text[self.current\_sentence])  
 self.done = False  
 self.rendered\_sentences = []  
 self.current\_text = ''  
 self.update()  
 except Exception as e:  
 print(f"Error resetting text: {e}")  
  
 def update(self):  
 if not self.done:  
 try:  
 self.current\_text = next(self.\_gen)  
 except StopIteration:  
 self.rendered\_sentences.append(self.font.render(self.current\_text, True, (255, 255, 255)))  
 self.current\_sentence += 1  
 if self.current\_sentence < len(self.text):  
 self.\_gen = text\_generator(self.text[self.current\_sentence])  
 self.current\_text = ''  
 else:  
 self.done = True  
 if self.autoreset:  
 self.current\_sentence = 0  
 self.reset()  
 except Exception as e:  
 print(f"Error updating text: {e}")  
  
 def draw\_gradient\_rect(self, screen, rect, color1, color2, color3):  
 *"""Draw a vertical gradient rectangle with corner cuts."""* try:  
 color1 = pg.Color(\*color1)  
 color2 = pg.Color(\*color2)  
 color3 = pg.Color(\*color3)  
 height = rect.height  
 width = rect.width  
 cut\_size = 10 # Size of the corner cuts  
  
 # Create a surface with per-pixel alpha  
 gradient\_surface = pg.Surface((width, height), pg.SRCALPHA)  
  
 half\_height = height // 2  
  
 for y in range(height):  
 if y < half\_height:  
 ratio = y / half\_height  
 r = int(color1.r \* (1 - ratio) + color2.r \* ratio)  
 g = int(color1.g \* (1 - ratio) + color2.g \* ratio)  
 b = int(color1.b \* (1 - ratio) + color2.b \* ratio)  
 a = int(color1.a \* (1 - ratio) + color2.a \* ratio)  
 else:  
 ratio = (y - half\_height) / half\_height  
 r = int(color2.r \* (1 - ratio) + color3.r \* ratio)  
 g = int(color2.g \* (1 - ratio) + color3.g \* ratio)  
 b = int(color2.b \* (1 - ratio) + color3.b \* ratio)  
 a = int(color2.a \* (1 - ratio) + color3.a \* ratio)  
  
 color = (r, g, b, a)  
  
 if y < cut\_size:  
 pg.draw.line(gradient\_surface, color, (cut\_size - y, y), (width - cut\_size + y, y))  
 elif y > height - cut\_size:  
 pg.draw.line(gradient\_surface, color, (y - (height - cut\_size), y),  
 (width - y + (height - cut\_size), y))  
 else:  
 pg.draw.line(gradient\_surface, color, (0, y), (width, y))  
  
 # Blit the gradient surface onto the screen  
 screen.blit(gradient\_surface, rect.topleft)  
 except Exception as e:  
 print(f"Error drawing gradient rectangle: {e}")  
  
 def draw(self, screen):  
 y\_offset = 0  
 line\_height = self.font.get\_height()  
 screen\_width = screen.get\_width()  
 shadow\_offset = (1.8, 1.8) # Offset for the text shadow  
  
 try:  
 # Calculate the total height of the text block  
 total\_height = len(self.rendered\_sentences) \* (line\_height + int(line\_height \* self.line\_spacing))  
 if not self.done:  
 total\_height += line\_height + int(line\_height \* self.line\_spacing)  
  
 # Draw the gradient background for the entire text block  
 text\_block\_rect = pg.Rect(15, self.pos[1] - 8, screen\_width - 60, total\_height + 10)  
 self.draw\_gradient\_rect(screen, text\_block\_rect, (190, 32, 17, 210), (25, 38, 50, 175),  
 (225, 187, 182, 150))  
  
 for sentence in self.rendered\_sentences:  
 # Render each sentence with a shadow  
 text\_rect = sentence.get\_rect(center=(screen\_width // 2, self.pos[1] + y\_offset))  
 shadow\_rect = text\_rect.move(\*shadow\_offset)  
 shadow\_surface = self.font.render(self.text[self.rendered\_sentences.index(sentence)], True, (0, 0, 0))  
  
 screen.blit(shadow\_surface, shadow\_rect)  
 screen.blit(sentence, text\_rect)  
  
 y\_offset += line\_height + int(line\_height \* self.line\_spacing)  
  
 if not self.done:  
 # Render the current text with a shadow  
 current\_render = self.font.render(self.current\_text, True, (255, 255, 255))  
 text\_rect = current\_render.get\_rect(center=(screen\_width // 2, self.pos[1] + y\_offset))  
 shadow\_rect = text\_rect.move(\*shadow\_offset)  
 shadow\_surface = self.font.render(self.current\_text, True, (0, 0, 0))  
  
 screen.blit(shadow\_surface, shadow\_rect)  
 screen.blit(current\_render, text\_rect)  
 except Exception as e:  
 print(f"Error drawing text: {e}")  
  
"""UNIVERSAL FUNCTIONS END ----------------------------------------------------------------------------------------------------  
  
INTRODUCTION START----------------------------------------------------------------------------------------------------"""  
class Intro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 try:  
 self.background = stretch(pg.image.load("resources/backgrounds/acadhall\_blurred.jpg"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 22)  
 self.text = import\_text("resources/introtext.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
 except Exception as e:  
 print(f"Error initializing Intro: {e}")  
 sys.exit()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.intro\_music = "resources/sounds/songs/intro.mp3"  
 pg.mixer.music.load(self.intro\_music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 try:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 LoadingScreen(self.SCREEN).run()  
 pg.mixer.music.stop()  
 Stage1Intro(self.SCREEN).run()  
 except Exception as e:  
 print(e)  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
 except Exception as e:  
 print(f"Error running intro: {e}")  
"""INTRODUCTION END --------------------------------------------------------------------------------------------------------  
  
STAGE 1 INTRO-OUTRO START----------------------------------------------------------------------------------------------------"""  
class Stage1Intro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 try:  
 self.background = stretch(pg.image.load("resources/backgrounds/room\_blurred.jpg"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 22)  
 self.text = import\_text("resources/stage1intro.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
 except Exception as e:  
 print(f"Error initializing Stage1Intro: {e}")  
 sys.exit()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.music = "resources/sounds/songs/s1\_inout.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 try:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 pg.mixer.music.stop()  
 LoadingScreen(self.SCREEN).run()  
 Stage1(self.SCREEN, 1).run()  
 return  
 except Exception as e:  
 print(f"Error loading next stage {e}")  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
 except Exception as e:  
 print(f"Error running Stage1Intro: {e}")  
  
class Stage1Outro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 try:  
 self.background = stretch(pg.image.load("resources/backgrounds/room\_blurred.jpg"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 22)  
 self.text = import\_text("resources/stage1outro.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
 except Exception as e:  
 print(f"Error initializing Stage1Outro: {e}")  
 sys.exit()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play outro song  
 self.music = "resources/sounds/songs/s1\_inout.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 try:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 LoadingScreen(self.SCREEN).run()  
 pg.mixer.music.stop()  
 Stage2Intro(self.SCREEN).run()  
 except Exception as e:  
 print(e)  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
 except Exception as e:  
 print(f"Error running Stage1Outro: {e}")  
"""STAGE 1 INTRO-OUTRO END ----------------------------------------------------------------------------------------------------  
  
STAGE 2 INTRO-OUTRO START----------------------------------------------------------------------------------------------------"""  
  
  
class Stage2Intro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.background = stretch(pg.image.load("resources/backgrounds/gym\_blurred.png"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 20)  
 self.text = import\_text("resources/stage2intro.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.music = "resources/sounds/songs/s1\_inout.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 LoadingScreen(self.SCREEN).run()  
 pg.mixer.music.stop()  
 from stages import Stage2  
 Stage2(self.SCREEN, 2).run()  
 return  
 except Exception as e:  
 print(e)  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
  
class Stage2Outro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.background = stretch(pg.image.load("resources/backgrounds/gym\_blurred.png"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 22)  
 self.text = import\_text("resources/stage2outro.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.music = "resources/sounds/songs/s1\_inout.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 LoadingScreen(self.SCREEN).run()  
 pg.mixer.music.stop()  
 Stage3Intro(self.SCREEN).run()  
 # from stages import Stage3  
 # Stage3(self.SCREEN, 3).run()  
 return  
 except Exception as e:  
 print(e)  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
  
"""STAGE 2 INTRO-OUTRO END ---------------------------------------------------------------------------------------------  
  
STAGE 3 INTRO-OUTRO START--------------------------------------------------------------------------------------------"""  
class Stage3Intro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.background = stretch(pg.image.load("resources/backgrounds/plaza\_blurred.jpg"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 24)  
 self.text = import\_text("resources/stage3intro.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 pg.mixer.init()  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.music = "resources/sounds/songs/s1\_inout.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 LoadingScreen(self.SCREEN).run()  
 from stages import Stage3  
 pg.mixer.music.stop()  
 Stage3(self.SCREEN, 3).run()  
 return  
 except Exception as e:  
 print(e)  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
  
class Stage3Outro:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.background = stretch(pg.image.load("resources/backgrounds/plaza\_blurred.jpg"),  
 (width, height)).convert\_alpha()  
 self.background = pg.transform.smoothscale(self.background, self.SCREEN.get\_size())  
 self.font = pg.font.Font(None, 24)  
 self.text = import\_text("resources/stage3outro.txt")  
 self.message = DynamicText(self.font, self.text, (50, 50), speed=20, autoreset=False)  
 self.skip\_prompt = self.font.render("Press any key to skip", True, (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render("Press any key to skip", True, (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
 self.text\_fully\_displayed = False  
  
 # Start a new thread to load and play intro music  
 pg.mixer.init()  
 self.music\_thread = threading.Thread(target=self.load\_and\_play\_music)  
 self.music\_thread.start()  
  
 def load\_and\_play\_music(self):  
 try:  
 # Load and play intro song  
 self.music = "resources/sounds/songs/s1\_inout.mp3"  
 pg.mixer.music.load(self.music)  
 pg.mixer.music.play(-1)  
 except Exception as e:  
 print(f"Error loading and playing music: {e}")  
  
 def run(self):  
 pg.time.set\_timer(pg.USEREVENT, self.message.speed)  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 if event.type == pg.USEREVENT:  
 self.message.update()  
 if event.type == pg.KEYDOWN or event.type == pg.MOUSEBUTTONDOWN:  
 if event.type == pg.KEYDOWN and event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 if self.text\_fully\_displayed:  
 try:  
 if event.type == pg.KEYDOWN:  
 LoadingScreen(self.SCREEN).run()  
 pg.mixer.music.stop()  
 from endings import Ending  
 Ending(self.SCREEN).run()  
 return  
 except Exception as e:  
 print(e)  
 else:  
 self.text\_fully\_displayed = True  
 self.message.done = True  
 self.message.rendered\_sentences = [self.font.render(line, True, (255, 255, 255))  
 for line in self.text.split('\n')]  
 self.skip\_prompt = self.font.render("Press any key to continue or press esc to go back",  
 True,  
 (255, 255, 255))  
 self.skip\_prompt\_shadow = self.font.render(  
 "Press any key to continue or press esc to go back",  
 True,  
 (0, 0, 0))  
 self.skip\_rect = self.skip\_prompt.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 50))  
  
 self.SCREEN.blit(self.background, (0, 0))  
 self.message.draw(self.SCREEN)  
 self.SCREEN.blit(self.skip\_prompt\_shadow, self.skip\_rect.move(2, 2))  
 self.SCREEN.blit(self.skip\_prompt, self.skip\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
"""STAGE 3 INTRO-OUTRO END ---------------------------------------------------------------------------------------------"""

# STAGES.PY

import time as time  
import pygame as pg  
import sys  
import math  
import random  
from PIL import Image, ImageFilter  
from PIL.ImageChops import offset  
  
from bgfix import stretch  
from scores import load\_score, write\_score  
import numpy as np  
from genwords import generate\_words\_tutorial, generate\_words\_stage1, generate\_words\_stage2, generate\_words\_stage3  
import threading  
  
"""UNIVERSAL FUNCTIONS---------------------------------------------------------------------------------------------------"""  
# load mixer  
pg.mixer.init()  
  
def pause(screen, background):  
 paused = True  
 overlay = pg.Surface(screen.get\_size(), pg.SRCALPHA)  
 overlay.fill((50, 50, 50, 200))  
  
 font = pg.font.Font("resources/DejaVuSans.ttf", 45)  
  
 resume\_text = "Press SpaceBar to continue playing"  
 resume\_text\_surf = font.render(resume\_text, True, pg.Color("white"))  
 resume\_text\_shadow = font.render(resume\_text, True, pg.Color("black"))  
 resume\_rect = resume\_text\_surf.get\_rect(center=(screen.get\_width() // 2, screen.get\_height() // 2 - 50))  
 menu\_text = "Press Esc to exit to menu"  
 menu\_text\_surf = font.render(menu\_text, True, pg.Color("white"))  
 menu\_text\_shadow = font.render(menu\_text, True, pg.Color("black"))  
 menu\_rect = menu\_text\_surf.get\_rect(center=(screen.get\_width() // 2, screen.get\_height() // 2 + 50))  
  
 while paused:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_SPACE:  
 paused = False  
 elif event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
  
 screen.blit(background, (0, 0)) # Draw the background  
 screen.blit(overlay, (0, 0)) # Draw the overlay  
 screen.blit(resume\_text\_shadow, resume\_rect.move(2, 2))  
 screen.blit(resume\_text\_surf, resume\_rect)  
 screen.blit(menu\_text\_shadow, menu\_rect.move(2, 2))  
 screen.blit(menu\_text\_surf, menu\_rect)  
 pg.display.flip()  
 pg.time.Clock().tick(60)  
  
def transform\_color(color, changes, max\_=255, min\_=0, step=1):  
 *""" Return an RGB triplet which has changed slightly from the color taken as input """* assert max\_ < 256 and min\_ >= 0 and max\_ >= min\_  
 red, green, blue = color  
  
 result = []  
 for color in (red, green, blue):  
 highest = min(color + changes, max\_)  
 lowest = max(color - changes, min\_)  
  
 if lowest >= highest:  
 highest = lowest + 1  
  
 result.append(random.randrange(lowest, highest))  
  
 return tuple(result)  
  
  
def apply\_wave\_effect(image, amplitude, frequency, phase, color\_shift):  
 arr = pg.surfarray.pixels3d(image)  
 height, width, \_ = arr.shape  
 for x in range(width):  
 offset = int(amplitude \* np.sin(2 \* np.pi \* frequency \* x + phase))  
 arr[:, x] = np.roll(arr[:, x], offset, axis=0)  
 arr[:, x] = np.clip(arr[:, x] + [color\_shift, color\_shift, color\_shift], 0, 255)  
 return pg.surfarray.make\_surface(arr)  
  
def handle\_explosion\_effect(screen, font, sprite\_rect, completed\_word, explosions):  
 # Compute enemy text box dimensions similar to those in the draw method  
 total\_width = font.size(completed\_word)[0]  
 text\_height = font.size(completed\_word)[1]  
 scaled\_width = int(total\_width \* 1.5) + 20  
 scaled\_height = int(text\_height \* 1.5) + 10  
 # Calculate the enemy text box rect at midright of the enemy sprite  
 word\_box\_rect = pg.Rect(0, 0, scaled\_width, scaled\_height)  
 word\_box\_rect.midright = (sprite\_rect.left - 20, sprite\_rect.centery)  
 # Load and scale the explosion image  
 explosion\_image = pg.image.load(f'resources/transparent/boom-{random.randint(1, 3)}.gif').convert\_alpha()  
 scale\_factor = 0.20 # Adjust explosion size as needed  
 new\_width = int(explosion\_image.get\_width() \* scale\_factor)  
 new\_height = int(explosion\_image.get\_height() \* scale\_factor)  
 explosion\_image = pg.transform.scale(explosion\_image, (new\_width, new\_height))  
 # Position explosion so its left edge touches the text box's right edge  
 explosion\_rect = explosion\_image.get\_rect(midleft=(word\_box\_rect.right, word\_box\_rect.centery))  
 explosions.append((explosion\_image, explosion\_rect, pg.time.get\_ticks()))  
  
# for thread later  
# def play\_loading\_music(music\_file):  
# pg.mixer.music.load("resources/sounds/songs/)  
  
  
"""UNIVERSAL FUNCTIONS---------------------------------------------------------------------------------------------------  
  
LOADING SCREEN START -------------------------------------------------------------------------------------------------"""  
  
class LoadingScreen:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.BG = stretch(pg.image.load("resources/backgrounds/menu.jpg"), (width, height)).convert\_alpha()  
 self.BG = pg.transform.scale(self.BG, (width, height))  
 self.font = pg.font.Font(None, 22)  
 self.text\_prompt = self.font.render("LOADING NEXT...", True, (255, 255, 255))  
 self.text\_prompt\_rect = self.text\_prompt.get\_rect(center=(width // 2, height // 2 + 150))  
 self.phase = 0  
  
  
 def animate\_background(self):  
 amplitude = 5  
 frequency = 0.01  
 color\_shift = 50  
 self.phase += 0.05  
  
 t = (np.sin(self.phase) + 1) / 2  
 r = int(255 \* (1 - t) + 128 \* t)  
 g = int(200 \* (1 - t) + 128 \* t)  
 b = int(100 \* (1 - t) + 128 \* t)  
 bg\_color = (r, g, b)  
  
 wavy\_bg = apply\_wave\_effect(self.BG.copy(), amplitude, frequency, self.phase, color\_shift)  
 wavy\_bg.fill(bg\_color, special\_flags=pg.BLEND\_RGBA\_MULT)  
 return wavy\_bg  
  
 def run(self):  
 title\_image = pg.image.load("resources/backgrounds/title.gif").convert\_alpha()  
 title\_image = pg.transform.scale(title\_image,  
 (int(title\_image.get\_width() \* 1.2), title\_image.get\_height()))  
 title\_rect = title\_image.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2.5))  
 clock = pg.time.Clock()  
 start\_time = pg.time.get\_ticks()  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
  
 current\_time = pg.time.get\_ticks()  
 if current\_time - start\_time > 1500: # 1000 milliseconds = 1 seconds  
 return  
  
 animated\_bg = self.animate\_background()  
 self.SCREEN.blit(animated\_bg, (0, 0))  
 self.SCREEN.blit(title\_image, title\_rect)  
 self.SCREEN.blit(self.text\_prompt, self.text\_prompt\_rect)  
 pg.display.update()  
 clock.tick(60)  
  
"""LOADING SCREEN END ---------------------------------------------------------------------------------------------------  
  
TUTORIAL START -------------------------------------------------------------------------------------------------------"""  
class Tutorial:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 22)  
 self.BG = stretch(pg.image.load("resources/backgrounds/bedrblur.jpg").convert\_alpha(), (width, height))  
 self.phase = 0  
  
 pg.key.set\_repeat(250, 30)  
  
 self.clock = pg.time.Clock()  
 self.words, self.bonus\_words = generate\_words\_tutorial()  
 self.current\_words = {}  
 self.word\_timer = 0  
 self.word\_frequency = 2.5  
 self.level = 1  
 self.score = 0  
 self.health = 10  
 self.prompt\_content = ''  
 self.word\_speed = 50  
 self.word\_widths = {}  
 self.highscore = load\_score()  
 self.enemy = TutorialEnemy(screen, self.level)  
 self.enemy.talking = True  
 self.fade\_alpha = 0  
 self.fade\_direction = 1  
 self.damage\_flash\_alpha = 0  
  
 # Load background music  
 pg.mixer.init()  
 self.tutorial\_prebattle\_music = "resources/sounds/songs/tutorial\_prebattle.mp3"  
 self.inbattle\_music = "resources/sounds/songs/tutorial\_inbattle.mp3"  
  
 # Load sound effects  
 self.enemyhit\_sfx = pg.mixer.Sound("resources/sounds/sfx/enemyhit.mp3")  
 self.win\_sfx = pg.mixer.Sound("resources/sounds/sfx/win.mp3")  
 self.wordcomplete\_sfx = pg.mixer.Sound("resources/sounds/sfx/wordcomplete.mp3")  
  
 self.explosions = []  
  
 def run(self):  
 width, height = self.SCREEN.get\_size()  
 battle\_started = False  
  
 self.help\_display()  
 self.before\_battle\_display()  
 battle\_started = True  
  
 pg.mixer.music.load(self.inbattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 timepassed = self.clock.tick(60) / 1000.0  
  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 if battle\_started:  
 pause(self.SCREEN, self.BG)  
 else:  
 return  
 if battle\_started:  
 if event.unicode.isprintable():  
 self.prompt\_content += event.unicode  
 elif event.key == pg.K\_BACKSPACE:  
 self.prompt\_content = self.prompt\_content[:-1]  
 elif event.key == pg.K\_RETURN:  
 self.prompt\_content = ''  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 if self.health <= 0:  
 self.display\_game\_over()  
 return  
  
 if not battle\_started:  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_rect = prompt\_surf.get\_rect(center=(width // 2, height // 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
 else:  
 if self.fade\_alpha < 255:  
 self.apply\_fade\_effect()  
 else:  
 self.word\_timer += timepassed  
 if self.word\_timer > self.word\_frequency and len(self.current\_words) < len(self.words):  
 self.add\_word(width)  
 self.word\_timer = 0  
  
 while len(self.current\_words) < 3:  
 self.add\_word(width)  
  
 for word, meta in list(self.current\_words.items()):  
 meta[1] += timepassed  
 y = (meta[1] \* self.word\_speed) + abs(math.cos(meta[1] \* 3) \* 10)  
 word\_rect = pg.Rect(meta[0], y, self.font.size(word)[0], self.font.size(word)[1])  
 if y > height:  
 del self.current\_words[word]  
 self.health -= 1  
 self.damage\_flash\_alpha = 150  
 elif word == self.prompt\_content:  
 del self.current\_words[word]  
 if word in self.bonus\_words:  
 self.health = self.health + 1  
 self.score += len(word) \* 5 # Higher score for bonus words  
 else:  
 self.score += len(word) \* 2 # Regular score for normal words  
 self.prompt\_content = ""  
 self.wordcomplete\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 if word == self.enemy.current\_word:  
 damage = len(word) \* 0.2  
 self.enemy.hitpoints = max(0, self.enemy.hitpoints - damage)  
 self.enemy.is\_hit = True  
 self.enemy.reset\_word(self.current\_words)  
 self.enemyhit\_sfx.play()  
 else:  
 self.enemy.is\_hit = False  
 else:  
 word\_surf = self.create\_word\_surf(word, meta[2])  
 word\_rect = word\_surf.get\_rect(center=(meta[0], y))  
 enemy\_rect = self.enemy.sprite\_rect  
 if word\_rect.colliderect(enemy\_rect):  
 if enemy\_rect.left - word\_rect.width - 10 >= 0:  
 word\_rect.right = enemy\_rect.left - 10  
 else:  
 word\_rect.left = enemy\_rect.right + 10  
 self.SCREEN.blit(word\_surf, word\_rect)  
  
 if self.current\_words:  
 if self.enemy.update(timepassed, self.prompt\_content, self.current\_words):  
 self.health -= 1  
 self.damage\_flash\_alpha = 150  
  
 if self.enemy.hitpoints <= 0:  
 self.win\_sfx.play()  
 pg.mixer.music.stop()  
 self.defeat\_display()  
 return  
  
 self.enemy.draw()  
 self.SCREEN.blit(self.generate\_prompt\_surf(), (0, height - 50))  
 self.draw\_ui()  
 self.draw\_enemy\_hitpoints()  
  
 if self.damage\_flash\_alpha > 0:  
 flash\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 flash\_surf.fill((255, 0, 0, self.damage\_flash\_alpha))  
 self.SCREEN.blit(flash\_surf, (0, 0))  
 self.damage\_flash\_alpha = max(0, self.damage\_flash\_alpha - 8)  
  
 # Draw and manage explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions if  
 current\_time - start\_time < 500]  
 for img, rect, start\_time in self.explosions:  
 self.SCREEN.blit(img, rect)  
  
 pg.display.flip()  
  
 def handle\_explosion\_effect(self, word\_rect):  
 explosion\_image = pg.image.load(f'resources/transparent/boom-{random.randint(1, 3)}.gif').convert\_alpha()  
 scale\_factor = 0.20 # Adjust this factor to make the explosion image larger  
 new\_width = int(explosion\_image.get\_width() \* scale\_factor)  
 new\_height = int(explosion\_image.get\_height() \* scale\_factor)  
 explosion\_image = pg.transform.scale(explosion\_image, (new\_width, new\_height))  
 explosion\_rect = explosion\_image.get\_rect(center=word\_rect.center)  
 self.explosions.append((explosion\_image, explosion\_rect, pg.time.get\_ticks()))  
  
 def help\_display(self):  
 # Play pre-battle music  
 pg.mixer.music.load(self.tutorial\_prebattle\_music)  
 pg.mixer.music.play(-1)  
  
 help\_images = [stretch(pg.image.load(f"resources/help/help-{i}.png").convert\_alpha(), self.SCREEN.get\_size())  
 for i in range(1, 7)]  
 current\_image\_index = 0  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 elif event.key == pg.K\_LEFT and current\_image\_index > 0:  
 current\_image\_index -= 1  
 elif event.key == pg.K\_RIGHT and current\_image\_index < len(help\_images) - 1:  
 current\_image\_index += 1  
 elif event.key == pg.K\_RETURN and current\_image\_index == len(help\_images) - 1:  
 self.before\_battle\_display()  
 return  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Display the current help image  
 help\_image = help\_images[current\_image\_index]  
 self.SCREEN.blit(help\_image, (0, 0))  
  
 # Create a smaller font  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 18)  
  
 # Draw the left arrow prompt if not on the first image  
 if current\_image\_index > 0:  
 left\_prompt\_text = "Press Left Arrow to go back"  
 left\_prompt\_surf = small\_font.render(left\_prompt\_text, True, pg.Color("white"))  
 left\_prompt\_shadow = small\_font.render(left\_prompt\_text, True, pg.Color("black"))  
 left\_prompt\_rect = left\_prompt\_surf.get\_rect(  
 bottomright=(self.SCREEN.get\_width() - 10, self.SCREEN.get\_height() - 60))  
 self.SCREEN.blit(left\_prompt\_shadow, left\_prompt\_rect.move(2, 2))  
 self.SCREEN.blit(left\_prompt\_surf, left\_prompt\_rect)  
  
 # Draw the right arrow prompt if not on the last image  
 if current\_image\_index < len(help\_images) - 1:  
 right\_prompt\_text = "Press Right Arrow to continue"  
 right\_prompt\_surf = small\_font.render(right\_prompt\_text, True, pg.Color("white"))  
 right\_prompt\_shadow = small\_font.render(right\_prompt\_text, True, pg.Color("black"))  
 right\_prompt\_rect = right\_prompt\_surf.get\_rect(  
 bottomright=(self.SCREEN.get\_width() - 10, self.SCREEN.get\_height() - 40))  
 self.SCREEN.blit(right\_prompt\_shadow, right\_prompt\_rect.move(2, 2))  
 self.SCREEN.blit(right\_prompt\_surf, right\_prompt\_rect)  
  
 # Draw the prompt to continue at the last image  
 if current\_image\_index == len(help\_images) - 1:  
 continue\_prompt\_text = "Press Enter to go to the battle"  
 continue\_prompt\_surf = small\_font.render(continue\_prompt\_text, True, pg.Color("white"))  
 continue\_prompt\_shadow = small\_font.render(continue\_prompt\_text, True, pg.Color("black"))  
 continue\_prompt\_rect = continue\_prompt\_surf.get\_rect(  
 bottomright=(self.SCREEN.get\_width() - 10, self.SCREEN.get\_height() - 40))  
 self.SCREEN.blit(continue\_prompt\_shadow, continue\_prompt\_rect.move(2, 2))  
 self.SCREEN.blit(continue\_prompt\_surf, continue\_prompt\_rect)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def before\_battle\_display(self):  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 return # Exit the display and start the battle  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the enemy sprite talking  
 talk\_sprite = self.enemy.talk\_sprite  
 talk\_sprite\_rect = talk\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(talk\_sprite, talk\_sprite\_rect)  
  
 # Draw the dialogue box with shadow  
 dialogue\_text = "\"Take it easy champ!\""  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 25)  
 dlg\_surf = small\_font.render(dialogue\_text, True, pg.Color("white"))  
 dlg\_shadow = small\_font.render(dialogue\_text, True, pg.Color("black"))  
  
 box\_width = int(dlg\_surf.get\_width() \* 1.5) + 20  
 box\_height = int(dlg\_surf.get\_height() \* 1.5) + 20  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (talk\_sprite\_rect.left - 100, talk\_sprite\_rect.centery - box\_height // 2),  
 (talk\_sprite\_rect.left - 100 + box\_width, talk\_sprite\_rect.centery - box\_height // 2 - offset),  
 (talk\_sprite\_rect.left - 100 + box\_width, talk\_sprite\_rect.centery + box\_height // 2 - offset),  
 (talk\_sprite\_rect.left - 100, talk\_sprite\_rect.centery + box\_height // 2)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (114, 141, 17, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (32, 122, 19), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, box\_width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
 dlg\_shadow = pg.transform.rotate(dlg\_shadow, angle)  
  
 # Blit the shadow text first, then the main text  
 dlg\_box\_rect = pg.Rect(talk\_sprite\_rect.left - 100, talk\_sprite\_rect.centery - box\_height // 2, box\_width,  
 box\_height)  
 self.SCREEN.blit(dlg\_shadow, dlg\_surf.get\_rect(center=dlg\_box\_rect.center).move(2, 2))  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_box\_rect.center))  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (32, 122, 19)  
 shadow\_color = (0, 0, 0)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 # Draw the prompt to continue  
 prompt\_text = "Press Enter to go to the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2 + 300))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def defeat\_display(self):  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 self.display\_victory()  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu  
 game.main\_Menu(self.SCREEN)  
 return  
 elif event.key == pg.K\_RETURN:  
 return # Exit the display and start the battle  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the enemy sprite talking  
 defeat\_sprite = self.enemy.defeat\_sprite  
 defeat\_sprite\_rect = defeat\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(defeat\_sprite, defeat\_sprite\_rect)  
  
 # Draw the dialogue box with shadow  
 dialogue\_text = "\"Nice one man!...press any key to go victory screen\""  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 25)  
 dlg\_surf = small\_font.render(dialogue\_text, True, pg.Color("white"))  
 dlg\_shadow = small\_font.render(dialogue\_text, True, pg.Color("black"))  
  
 box\_width = int(dlg\_surf.get\_width())  
 box\_height = int(dlg\_surf.get\_height())  
  
 # Define the points for the parallelogram shape  
 offset = 3  
 box\_x = (self.SCREEN.get\_width() - box\_width) // 2  
 box\_y = (self.SCREEN.get\_height() - box\_height) // 2 + 30  
 box\_points = [  
 (box\_x, box\_y),  
 (box\_x + box\_width, box\_y - offset),  
 (box\_x + box\_width, box\_y + box\_height - offset),  
 (box\_x, box\_y + box\_height)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (114, 141, 17, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (32, 122, 19), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, box\_width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
 dlg\_shadow = pg.transform.rotate(dlg\_shadow, angle)  
  
 # Blit the shadow text first, then the main text  
 dlg\_box\_rect = pg.Rect(box\_x, box\_y, box\_width, box\_height)  
 self.SCREEN.blit(dlg\_shadow, dlg\_surf.get\_rect(center=dlg\_box\_rect.center).move(2, 2))  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_box\_rect.center))  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (32, 122, 19)  
 shadow\_color = (0, 0, 0)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def display\_victory(self):  
 if self.score > self.highscore:  
 self.highscore = self.score  
 write\_score(self.highscore)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 else:  
 from introduction import Intro  
 LoadingScreen(self.SCREEN).run()  
 Intro(self.SCREEN).run()  
  
 # Prepare text surfaces and their positions  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 victory\_surf = self.font.render("VICTORY!", True, pg.Color("white"))  
 victory\_shadow = self.font.render("VICTORY!", True, pg.Color("black"))  
 highscore\_text = f"Highscore: {self.highscore}"  
 highscore\_surf = self.font.render(highscore\_text, True, pg.Color("white"))  
 highscore\_shadow = self.font.render(highscore\_text, True, pg.Color("black"))  
 prompt\_text = "Press any key for next stage, or Esc for main menu"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
  
 victory\_rect = victory\_surf.get\_rect(center=(center[0], center[1] - 40))  
 hs\_rect = highscore\_surf.get\_rect(center=center)  
 prompt\_rect = prompt\_surf.get\_rect(center=(center[0], center[1] + 40))  
  
 # Calculate the bounding rectangle of all text surfaces and add padding  
 union\_rect = victory\_rect.union(hs\_rect).union(prompt\_rect)  
 padding = 10  
 dlg\_rect = pg.Rect(  
 union\_rect.left - padding,  
 union\_rect.top - padding,  
 union\_rect.width + 6 \* padding,  
 union\_rect.height + 6 \* padding  
 )  
  
 # Center the dialog box on the screen  
 dlg\_rect.center = center  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Create the dialog box surface with an opaque yellow red color  
 dlg\_box = pg.Surface((dlg\_rect.width, dlg\_rect.height))  
 dlg\_box.fill((255, 193, 33))  
  
 # Draw background and dialog box  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (114, 141, 17, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (32, 122, 19), box\_points)  
  
 # Draw border as a parallelogram  
 border\_padding = 5  
 border\_points = [  
 (dlg\_rect.left + border\_padding, dlg\_rect.top + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.top - offset + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.bottom - offset - border\_padding),  
 (dlg\_rect.left + border\_padding, dlg\_rect.bottom - border\_padding)  
 ]  
 pg.draw.polygon(self.SCREEN, (255, 213, 0), border\_points, 3)  
  
 # Rotate the text surfaces to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 victory\_surf = pg.transform.rotate(victory\_surf, angle)  
 victory\_shadow = pg.transform.rotate(victory\_shadow, angle)  
 highscore\_surf = pg.transform.rotate(highscore\_surf, angle)  
 highscore\_shadow = pg.transform.rotate(highscore\_shadow, angle)  
 prompt\_surf = pg.transform.rotate(prompt\_surf, angle)  
 prompt\_shadow = pg.transform.rotate(prompt\_shadow, angle)  
  
 # Blit each text surface centered at their respective positions  
 self.SCREEN.blit(victory\_shadow, victory\_rect.move(2, 2))  
 self.SCREEN.blit(victory\_surf, victory\_rect)  
 self.SCREEN.blit(highscore\_shadow, hs\_rect.move(2, 2))  
 self.SCREEN.blit(highscore\_surf, hs\_rect)  
 self.SCREEN.blit(prompt\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (32, 122, 19)  
 shadow\_color = (0, 0, 0)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def add\_word(self, width):  
 found\_word = False  
 while not found\_word and len(self.current\_words) < len(self.words):  
 if random.random() < 0.3: # 50% chance to add a bonus word  
 selected = random.choice(self.bonus\_words)  
 else:  
 selected = random.choice(self.words)  
  
 if all(not w.startswith(selected[0]) for w in self.current\_words):  
 if selected not in self.word\_widths:  
 self.word\_widths[selected] = self.font.size(selected)[0]  
 w\_width = self.word\_widths[selected]  
 x = random.randrange(45, width - w\_width - 10) # Ensure the word does not overlap the screen edges  
 # Ensure the word does not overlap with the enemy sprite and other tutorial\_words  
 if not (self.enemy.sprite\_rect.left < x < self.enemy.sprite\_rect.right) and \  
 all(abs(x - meta[0]) > w\_width + 15 for meta in self.current\_words.values()):  
 self.current\_words[selected] = [x, 0, (150, 150, 150)]  
 found\_word = True  
  
 def create\_word\_surf(self, word, color):  
 w, h = self.font.size(word)  
 w += 12 # Increase width for padding  
 h += 12 # Increase height for padding  
 Surf = pg.Surface((w, h), pg.SRCALPHA, 32)  
  
 # Determine if the word is a bonus word  
 is\_bonus\_word = word in self.bonus\_words  
  
 # Create a rounded rectangle background with a different color for bonus words  
 if is\_bonus\_word:  
 bg\_color = (255, 215, 0, 200) # Gold color for bonus words with some opacity  
 else:  
 bg\_color = (77, 120, 77, 200) # Constant background color with some opacity  
  
 pg.draw.rect(Surf, bg\_color, Surf.get\_rect(), border\_radius=10)  
  
 being\_written = self.prompt\_content and word.startswith(self.prompt\_content)  
 start\_text = self.prompt\_content if being\_written else ''  
 end\_text = word[len(self.prompt\_content):] if being\_written else word  
 start\_surf = self.font.render(start\_text, True, pg.Color("black"))  
  
 # Apply transform\_color to the end\_text color for more vibrancy  
 transformed\_color = transform\_color(color, 100)  
 end\_surf = self.font.render(end\_text, True, transformed\_color)  
  
 Surf.blit(start\_surf, (8, 8))  
 Surf.blit(end\_surf, end\_surf.get\_rect(right=w - 8, centery=h // 2))  
 return Surf  
  
 def generate\_prompt\_surf(self):  
 width = self.SCREEN.get\_width()  
 surf = pg.Surface((width, 50), pg.SRCALPHA)  
 shadow\_surf = pg.Surface((width, 10), pg.SRCALPHA)  
  
 # Create shadow  
 shadow\_surf.fill((197, 136, 0, 100))  
 surf.fill((32, 122, 19))  
 surf.set\_alpha(255)  
  
 self.SCREEN.blit(surf, (0, 0))  
 surf.blit(shadow\_surf, (0, -1))  
  
  
 color = pg.Color("yellow") if any(w.startswith(self.prompt\_content) for w in self.current\_words) else pg.Color(  
 "white")  
 rendered = self.font.render(self.prompt\_content, True, color)  
  
 # Create shadow text  
 shadow\_rendered = self.font.render(self.prompt\_content, True, pg.Color("black"))  
  
 # Center the prompt text horizontally on the surface  
 rect = rendered.get\_rect(centerx=width // 2, centery=25)  
 shadow\_rect = shadow\_rendered.get\_rect(centerx=width // 2 - 2, centery=25 - 2) # Offset for shadow effect  
  
 # Blit shadow first, then main text  
 surf.blit(shadow\_rendered, shadow\_rect)  
 surf.blit(rendered, rect)  
  
 # Draw a bar to indicate the position  
 bar\_width = 2  
 bar\_height = 40  
 bar\_x = rect.right + 5  
 bar\_y = 5  
 pg.draw.rect(surf, pg.Color("red"), (bar\_x, bar\_y, bar\_width, bar\_height))  
  
 return surf  
  
 def draw\_enemy\_hitpoints(self):  
 hp\_text = f"Enemy HP: {self.enemy.hitpoints:.1f}"  
 hp\_text\_shadow = self.font.render(hp\_text, True, pg.Color("black"))  
 hp\_surf = self.font.render(hp\_text, True, (255, 255, 255))  
 hp\_box = pg.Surface((hp\_surf.get\_width() + 10, hp\_surf.get\_height() + 10), pg.SRCALPHA)  
 hp\_box.fill((114, 141, 17, 190))  
  
 # Initialize and update fade alpha for enemy hitpoints  
 if not hasattr(self, 'hp\_alpha'):  
 self.hp\_alpha = 0  
 if self.hp\_alpha < 255:  
 self.hp\_alpha += 5 # Adjust increment as needed for smoother or faster fade  
 hp\_box.set\_alpha(self.hp\_alpha)  
  
 hp\_box\_rect = hp\_box.get\_rect(midtop=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 100))  
  
 # Create shadow of box  
 shadow\_offset = 2  
 shadow\_box = pg.Surface((hp\_box.get\_width(), hp\_box.get\_height()), pg.SRCALPHA)  
 shadow\_box.fill((32, 122, 19, 100)) # Darker color for shadow  
 shadow\_box\_rect = hp\_box\_rect.move(shadow\_offset, shadow\_offset)  
  
 # Blit shadow first, then the hitpoint box  
 self.SCREEN.blit(shadow\_box, shadow\_box\_rect)  
 self.SCREEN.blit(hp\_text\_shadow, hp\_box\_rect.move(2,2))  
 self.SCREEN.blit(hp\_box, hp\_box\_rect)  
 self.SCREEN.blit(hp\_surf, hp\_surf.get\_rect(center=hp\_box\_rect.center))  
  
 def draw\_ui(self):  
 top\_box = pg.Surface((self.SCREEN.get\_width(), 40), pg.SRCALPHA)  
 top\_box.fill(("#364216"))  
 top\_box\_rect = top\_box.get\_rect()  
 if not hasattr(self, 'ui\_alpha'):  
 self.ui\_alpha = 0  
  
 if self.ui\_alpha < 255:  
 self.ui\_alpha += 1 # Adjust the increment value as needed  
  
 top\_box.set\_alpha(self.ui\_alpha)  
 self.SCREEN.blit(top\_box, top\_box\_rect)  
  
 # Render the main text and its shadow  
 score\_surf = self.font.render(f"Score: {self.score}", True, (255, 255, 255))  
 health\_surf = self.font.render(f"Health: {self.health}", True, (255, 255, 255))  
 enemy\_name = self.font.render(f"Enemy: Tutorial Guy", True, (255, 255, 255))  
 score\_shadow = self.font.render(f"Score: {self.score}", True, (0, 0, 0))  
 health\_shadow = self.font.render(f"Health: {self.health}", True, (0, 0, 0))  
 enemy\_shadow = self.font.render(f"Enemy: Tutorial Guy", True, (0, 0, 0))  
  
 # Calculate positions for the text  
 screen\_width = self.SCREEN.get\_width()  
 score\_pos = (10, 10)  
 health\_pos = (screen\_width // 3, 10)  
 enemy\_pos = (2 \* screen\_width // 3, 10)  
  
 # Offset for the shadow effect  
 shadow\_offset = (2, 2)  
  
 # Blit the shadow first, then the main text  
 self.SCREEN.blit(score\_shadow, (score\_pos[0] + shadow\_offset[0], score\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(health\_shadow, (health\_pos[0] + shadow\_offset[0], health\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(enemy\_shadow, (enemy\_pos[0] + shadow\_offset[0], enemy\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(score\_surf, score\_pos)  
 self.SCREEN.blit(health\_surf, health\_pos)  
 self.SCREEN.blit(enemy\_name, enemy\_pos)  
  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (screen\_width // 3 - 5, 0),  
 (screen\_width // 3 - 5, 40), 2)  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (2 \* screen\_width // 3 - 5, 0),  
 (2 \* screen\_width // 3 - 5, 40), 2)  
  
 def display\_game\_over(self):  
 write\_score(self.score)  
 game\_over = self.font.render("GAME OVER", True, (255, 0, 0))  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 self.SCREEN.blit(game\_over, game\_over.get\_rect(center=center))  
 pg.display.flip()  
 pg.time.wait(2000)  
  
 def apply\_fade\_effect(self):  
 if self.fade\_direction != 0:  
 self.fade\_alpha += self.fade\_direction \* 10  
 if self.fade\_alpha >= 255:  
 self.fade\_alpha = 255  
 self.fade\_direction = 0  
 elif self.fade\_alpha <= 0:  
 self.fade\_alpha = 0  
 self.fade\_direction = 0  
 fade\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 fade\_surf.fill((255, 0, 0, self.fade\_alpha))  
 self.SCREEN.blit(fade\_surf, (0, 0))  
  
class TutorialEnemy:  
 def \_\_init\_\_(self, screen, level):  
 self.screen = screen  
 self.width, self.height = self.screen.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 36)  
 self.hitpoints = 10 + level \* 5  
 self.word\_speed = 2  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
 self.is\_hit = False  
 self.sprite\_alpha = 0  
  
 self.normal\_sprite = pg.image.load("resources/sprites/neil-fight.png").convert\_alpha()  
 self.hit\_sprite = pg.image.load("resources/sprites/neil-hit-color.gif").convert\_alpha()  
 self.talk\_sprite = pg.image.load("resources/sprites/neil-talk.png").convert\_alpha()  
 self.defeat\_sprite = pg.image.load("resources/sprites/neil-defeat.png").convert\_alpha()  
 self.normal\_sprite = pg.transform.scale(self.normal\_sprite, (450, 650))  
 self.hit\_sprite = pg.transform.scale(self.hit\_sprite, (450, 650))  
 self.talk\_sprite = pg.transform.scale(self.talk\_sprite, (450, 650))  
 self.defeat\_sprite = pg.transform.scale(self.defeat\_sprite, (450, 650))  
 self.sprite\_rect = self.normal\_sprite.get\_rect()  
 self.sprite\_rect.centerx = self.width - 250  
 self.sprite\_rect.centery = self.height // 2  
 self.word\_bg\_image = pg.image.load("resources/transparent/tutorial.gif").convert\_alpha()  
 self.explosions = []  
  
 def reset\_word(self, current\_words):  
 if self.current\_word in current\_words:  
 del current\_words[self.current\_word]  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 def update(self, timepassed, player\_input, current\_words):  
 if self.sprite\_alpha < 255:  
 self.sprite\_alpha += 5  
  
 if self.hitpoints <= 0:  
 return False  
  
 if not self.current\_word and current\_words:  
 self.current\_word = random.choice(list(current\_words.keys()))  
 self.word\_progress = 0  
  
 if self.current\_word and (self.current\_word not in current\_words):  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 if self.start\_timer > 0:  
 self.start\_timer -= timepassed  
 return False  
  
 if self.current\_word:  
 self.word\_progress += timepassed \* self.word\_speed  
 meta = current\_words[self.current\_word]  
 # Use the updated meta data for y-position  
 word\_x = meta[0]  
 meta\_y = meta[1]  
 y = (meta\_y \* self.word\_speed) + abs(math.cos(meta\_y \* 3) \* 10)  
 word\_rect = pg.Rect(word\_x, y, self.font.size(self.current\_word)[0],  
 self.font.size(self.current\_word)[1])  
 if self.word\_progress >= len(self.current\_word):  
 # Store the completed word before resetting  
 completed\_word = self.current\_word  
 handle\_explosion\_effect(self.screen, self.font, self.sprite\_rect, completed\_word, self.explosions)  
 if self.current\_word in current\_words:  
 current\_words.pop(self.current\_word)  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.0  
 return True  
  
 return False  
  
 def draw(self):  
 if self.hitpoints <= 0:  
 current\_sprite = self.defeat\_sprite  
 else:  
 current\_sprite = self.hit\_sprite if self.is\_hit else self.normal\_sprite  
  
 sprite\_with\_alpha = current\_sprite.copy()  
 sprite\_with\_alpha.set\_alpha(self.sprite\_alpha)  
 self.screen.blit(sprite\_with\_alpha, self.sprite\_rect)  
  
 if self.hitpoints > 0 and self.current\_word:  
 # Render the typed and remaining portions of the word  
 typed = self.current\_word[:int(self.word\_progress)]  
 remaining = self.current\_word[int(self.word\_progress):]  
 typed\_surf = self.font.render(typed, True, ("#569612"))  
 remaining\_surf = self.font.render(remaining, True, (100, 100, 100))  
  
 total\_width = typed\_surf.get\_width() + remaining\_surf.get\_width()  
 text\_height = typed\_surf.get\_height()  
  
 # Define the text box size based on the text dimensions with extra margin  
 box\_width = int(total\_width \* 1.5) + 20  
 box\_height = int(text\_height \* 1.5) + 25  
  
 # Scale the background image for the word box  
 word\_bg\_image\_scaled = pg.transform.scale(self.word\_bg\_image, (box\_width, box\_height))  
  
 # Position the text box with a negative x-coordinate to overlay over the sprite  
 word\_box\_rect = word\_bg\_image\_scaled.get\_rect(  
 midright=(self.sprite\_rect.left - 20, self.sprite\_rect.centery))  
 word\_box\_rect.x += 100 # Adjust this value as needed to overlay the text box  
  
 # Calculate centered text position within the text box  
 text\_x = word\_box\_rect.left + (box\_width - total\_width) // 2  
 text\_y = word\_box\_rect.top + (box\_height - text\_height) // 2  
  
 # Blit the text box and then the text centered in it  
 self.screen.blit(word\_bg\_image\_scaled, word\_box\_rect)  
 self.screen.blit(typed\_surf, (text\_x, text\_y))  
 self.screen.blit(remaining\_surf, (text\_x + typed\_surf.get\_width(), text\_y))  
  
 # Draw any active explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions  
 if current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.screen.blit(img, rect)  
  
"""TUTORIAL END ---------------------------------------------------------------------------------------------------------  
  
STAGE 1 START --------------------------------------------------------------------------------------------------------"""  
class Stage1:  
 def \_\_init\_\_(self, screen, level):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 22)  
 self.BG = stretch(pg.image.load("resources/backgrounds/roomblur.jpg").convert\_alpha(), (width, height))  
 self.phase = 0  
  
 pg.key.set\_repeat(250, 30)  
  
 self.clock = pg.time.Clock()  
 self.stage1\_words, self.bonus\_words = generate\_words\_stage1()  
 self.current\_words = {}  
 self.word\_timer = 0  
 self.word\_frequency = 5  
 self.level = level  
 self.score = 0  
 self.health = 20 \* (level)  
 self.prompt\_content = ''  
 self.word\_speed = 40  
 self.word\_widths = {}  
 self.highscore = load\_score()  
 self.enemies = [Minion1(screen, self.level), Minion2(screen, self.level), Minion3(screen, self.level),  
 Boss(screen, self.level)]  
 self.current\_enemy\_index = 0  
 self.enemy = self.enemies[self.current\_enemy\_index]  
 self.enemy.talking = True  
 self.fade\_alpha = 0  
 self.fade\_direction = 1  
 self.damage\_flash\_alpha = 0  
 self.bonus\_word\_counter = 0  
  
 # Load background music  
 self.inbattle\_music = "resources/sounds/songs/s1minion.mp3"  
 self.prebattle\_music = "resources/sounds/songs/s1\_prebattle.mp3"  
 self.victory\_music = "resources/sounds/songs/s1victory.mp3"  
  
 # Load sound effects  
 self.enemyhit\_sfx = pg.mixer.Sound("resources/sounds/sfx/enemyhit.mp3")  
 self.win\_sfx = pg.mixer.Sound("resources/sounds/sfx/win.mp3")  
 self.wordcomplete\_sfx = pg.mixer.Sound("resources/sounds/sfx/wordcomplete.mp3")  
  
 self.explosions = []  
 self.bossfight\_pause\_timer = 0  
 self.falling\_words\_pause\_timer = 0  
 self.last\_bonus\_action = 'damage'  
  
 def run(self):  
 width, height = self.SCREEN.get\_size()  
 battle\_started = False  
 hue = 0  
  
 # self.defeat\_display(Boss(self.SCREEN, self.level))  
  
 while self.current\_enemy\_index < len(self.enemies):  
 self.enemy = self.enemies[self.current\_enemy\_index]  
 self.before\_battle\_display(self.enemy)  
 battle\_started = True  
  
 # self.current\_enemy\_index = 3  
 # if self.current\_enemy\_index == 3:  
 # self.enemy = self.enemies[self.current\_enemy\_index]  
 # self.before\_battle\_display(self.enemy)  
 # battle\_started = True  
  
 pg.mixer.music.load(self.inbattle\_music)  
 pg.mixer.music.play(-1)  
  
 if self.enemy == self.enemies[3]:  
 self.inbattle\_music = "resources/sounds/songs/s1boss.mp3"  
 pg.mixer.music.load(self.inbattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 timepassed = self.clock.tick(60) / 1000.0  
  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 if battle\_started:  
 pause(self.SCREEN, self.BG)  
 else:  
 return  
 if battle\_started:  
 if event.unicode.isprintable():  
 self.prompt\_content += event.unicode  
 elif event.key == pg.K\_BACKSPACE:  
 self.prompt\_content = self.prompt\_content[:-1]  
 elif event.key == pg.K\_RETURN:  
 self.prompt\_content = ''  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 if self.health <= 0:  
 self.display\_game\_over()  
 return  
  
 if not battle\_started:  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_rect = prompt\_surf.get\_rect(center=(width // 2, height // 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
 else:  
 if self.fade\_alpha < 255:  
 self.apply\_fade\_effect()  
 else:  
 self.word\_timer += timepassed  
 if self.word\_timer > self.word\_frequency and len(self.current\_words) < len(self.stage1\_words):  
 # Add normal word  
 self.add\_word(width, self.stage1\_words, 'stage1', self.enemy)  
 self.word\_timer = 0  
  
 # Check for bonus words  
 if self.bonus\_word\_counter >= 5:  
 self.add\_word(width, self.bonus\_words, 'bonus', self.enemy)  
  
 # Keep minimum number of words  
 while len(self.current\_words) < 4:  
 self.add\_word(width, self.stage1\_words, 'stage1', self.enemy)  
  
 for word, meta in list(self.current\_words.items()):  
 meta[1] += timepassed  
 y = (meta[1] \* self.word\_speed) + abs(math.cos(meta[1] \* 3) \* 10)  
 word\_rect = pg.Rect(meta[0], y, self.font.size(word)[0], self.font.size(word)[1])  
 if y > height:  
 del self.current\_words[word]  
 self.health -= 1  
 self.damage\_flash\_alpha = 150  
 elif word == self.prompt\_content:  
 del self.current\_words[word]  
 self.score += len(word) \* 2  
 self.prompt\_content = ""  
 self.wordcomplete\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 if word == self.enemy.current\_word:  
 self.apply\_damage(1, word)  
 self.handle\_explosion\_effect(word\_rect)  
 elif word in self.bonus\_words:  
 if self.last\_bonus\_action == 'damage':  
 self.apply\_damage(3, word)  
 self.last\_bonus\_action = 'health'  
 self.handle\_explosion\_effect(word\_rect)  
 else:  
 self.health = min(self.health + 1.5, 50)  
 self.last\_bonus\_action = 'damage'  
 self.enemy.reset\_word(self.current\_words)  
 self.enemy.is\_hit = True  
 self.enemyhit\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 else:  
 self.enemy.is\_hit = False  
  
 else:  
 word\_surf = self.create\_word\_surf(word, meta[2], hue, meta[3])  
 word\_rect = word\_surf.get\_rect(center=(meta[0], y))  
 enemy\_rect = self.enemy.sprite\_rect  
 if word\_rect.colliderect(enemy\_rect):  
 if enemy\_rect.left - word\_rect.width - 10 >= 0:  
 word\_rect.right = enemy\_rect.left - 10  
 else:  
 word\_rect.left = enemy\_rect.right + 10  
 self.SCREEN.blit(word\_surf, word\_rect)  
  
 if self.current\_words:  
 if self.enemy.update(timepassed, self.prompt\_content, self.current\_words):  
 if isinstance(self.enemy, Boss):  
 self.health -= 2 \* self.level # Boss deals twice the damage  
 else:  
 self.health -= self.level # Minions deal damage based on the level  
 self.damage\_flash\_alpha = 150  
  
 if self.enemy.hitpoints <= 0:  
 self.win\_sfx.play()  
 pg.mixer.music.stop()  
 self.defeat\_display(self.enemy)  
 self.current\_enemy\_index += 1  
 break  
  
 self.enemy.draw()  
 self.SCREEN.blit(self.generate\_prompt\_surf(), (0, height - 50))  
 self.draw\_ui()  
 self.draw\_enemy\_hitpoints()  
  
 if self.damage\_flash\_alpha > 0:  
 flash\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 flash\_surf.fill((255, 0, 0, self.damage\_flash\_alpha))  
 self.SCREEN.blit(flash\_surf, (0, 0))  
 self.damage\_flash\_alpha = max(0, self.damage\_flash\_alpha - 8)  
  
 # Draw and manage explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions if  
 current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.SCREEN.blit(img, rect)  
  
 pg.display.flip()  
 hue = (hue + 1) % 360 # Update hue for the next frame  
  
  
 self.display\_victory()  
 pg.mixer.music.stop()  
 from introduction import Stage1Outro  
 outro = Stage1Outro(self.SCREEN)  
 outro.run()  
  
 def apply\_damage(self, damage, word, reset\_word=True, play\_sound=True):  
 self.enemy.hitpoints = max(0, self.enemy.hitpoints - ((damage \* len(word)) \* 0.25))  
 self.enemy.is\_hit = True  
 if reset\_word:  
 self.enemy.reset\_word(self.current\_words)  
 if play\_sound:  
 self.enemyhit\_sfx.play()  
  
 def handle\_explosion\_effect(self, word\_rect):  
 explosion\_image = pg.image.load(f'resources/transparent/boom-{random.randint(1, 3)}.gif').convert\_alpha()  
 scale\_factor = 0.22 # Adjust this factor to make the explosion image larger  
 new\_width = int(explosion\_image.get\_width() \* scale\_factor)  
 new\_height = int(explosion\_image.get\_height() \* scale\_factor)  
 explosion\_image = pg.transform.scale(explosion\_image, (new\_width, new\_height))  
 explosion\_rect = explosion\_image.get\_rect(center=word\_rect.center)  
 self.explosions.append((explosion\_image, explosion\_rect, pg.time.get\_ticks()))  
  
 def before\_battle\_display(self, minion):  
 fade\_duration = 1.0 # Duration of the fade-in effect in seconds  
 fade\_alpha = 0 # Initial alpha value for fade-in effect  
 fade\_increment = 255 / (fade\_duration \* 60) # Increment per frame (assuming 60 FPS)  
  
 pg.mixer.music.load(self.prebattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 pg.mixer.music.stop()  
 return # Exit the display and start the battle  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the minion sprite talking with fade-in effect  
 talk\_sprite = minion.talk\_sprite.copy()  
 talk\_sprite.set\_alpha(fade\_alpha)  
 talk\_sprite\_rect = talk\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(talk\_sprite, talk\_sprite\_rect)  
  
 # Draw the dialogue box with a parallelogram shape and shadow  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 20)  
 dlg\_surf = small\_font.render(minion.dialogue\_text, True, pg.Color(250, 250, 250))  
 dlg\_surf.set\_alpha(fade\_alpha)  
  
 # Calculate the bounding rectangle of the text surface and add padding  
 padding = 10  
 dlg\_rect = pg.Rect(  
 talk\_sprite\_rect.left - dlg\_surf.get\_width() - padding \* 2 - 20,  
 talk\_sprite\_rect.centery - dlg\_surf.get\_height() // 2 - padding,  
 dlg\_surf.get\_width() + padding \* 2,  
 dlg\_surf.get\_height() + padding \* 2  
 )  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
  
 # Blit the text surface centered at its position  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_rect.center))  
  
 # Draw the top bar with fade-in effect  
 top\_bar\_height = 50  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), top\_bar\_height), pg.SRCALPHA)  
 top\_bar.fill((167, 57, 57, fade\_alpha))  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), top\_bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill((125, 28, 28, fade\_alpha))  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Draw the bottom bar with fade-in effect  
 bottom\_bar\_height = 100  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bottom\_bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill((167, 57, 57, fade\_alpha))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bottom\_bar\_height))  
  
 # Draw the prompt to continue with fade-in effect  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - bottom\_bar\_height // 2))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Apply fade-in effect  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def defeat\_display(self, minion):  
 fade\_duration = 1.0  
 fade\_alpha = 0  
 fade\_increment = 255 / (fade\_duration \* 60)  
  
 pg.mixer.music.load(self.victory\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 pg.mixer.music.stop()  
 return  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 defeat\_sprite = minion.defeat\_sprite.copy()  
 defeat\_sprite.set\_alpha(fade\_alpha)  
 defeat\_sprite\_rect = defeat\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(defeat\_sprite, defeat\_sprite\_rect)  
  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 20)  
 dlg\_surf = small\_font.render(minion.defeat\_text, True, pg.Color(250, 250, 250))  
 dlg\_surf.set\_alpha(fade\_alpha)  
  
 # Calculate the bounding rectangle of the text surface and add padding  
 padding = 10  
 dlg\_rect = pg.Rect(  
 0, 0,  
 dlg\_surf.get\_width() + padding \* 2,  
 dlg\_surf.get\_height() + padding \* 2  
 )  
 dlg\_rect.centerx = self.SCREEN.get\_width() // 2  
 dlg\_rect.centery = defeat\_sprite\_rect.centery  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
  
 # Blit the text surface centered at its position  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_rect.center))  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (167, 57, 57)  
 shadow\_color = (125, 28, 28)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar\_y = self.SCREEN.get\_height() - bar\_height  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, bottom\_bar\_y))  
 self.SCREEN.blit(bottom\_bar, (0, bottom\_bar\_y))  
  
 prompt\_text = "Press Enter to continue"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, bottom\_bar\_y + bar\_height // 2))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def display\_victory(self):  
 if self.score > self.highscore:  
 self.highscore = self.score  
 write\_score(self.highscore)  
  
 pg.mixer.music.load(self.victory\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 else:  
 LoadingScreen(self.SCREEN).run()  
 from introduction import Stage1Outro  
 outro = Stage1Outro(self.SCREEN)  
 outro.run()  
  
 # Prepare text surfaces and their positions  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 victory\_surf = self.font.render("VICTORY!", True, pg.Color("white"))  
 victory\_shadow = self.font.render("VICTORY!", True, pg.Color("black"))  
 highscore\_text = f"Highscore: {self.highscore}"  
 highscore\_surf = self.font.render(highscore\_text, True, pg.Color("white"))  
 highscore\_shadow = self.font.render(highscore\_text, True, pg.Color("black"))  
 prompt\_text = "Press any key for next stage, or Esc for main menu"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
  
 victory\_rect = victory\_surf.get\_rect(center=(center[0], center[1] - 40))  
 hs\_rect = highscore\_surf.get\_rect(center=center)  
 prompt\_rect = prompt\_surf.get\_rect(center=(center[0], center[1] + 40))  
  
 # Calculate the bounding rectangle of all text surfaces and add padding  
 union\_rect = victory\_rect.union(hs\_rect).union(prompt\_rect)  
 padding = 10  
 dlg\_rect = pg.Rect(  
 union\_rect.left - padding,  
 union\_rect.top - padding,  
 union\_rect.width + 6 \* padding,  
 union\_rect.height + 6 \* padding  
 )  
  
 # Center the dialog box on the screen  
 dlg\_rect.center = center  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Create the dialog box surface with an opaque yellow red color  
 dlg\_box = pg.Surface((dlg\_rect.width, dlg\_rect.height))  
 dlg\_box.fill((255, 193, 33))  
  
 # Draw background and dialog box  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Draw border as a parallelogram  
 border\_padding = 5  
 border\_points = [  
 (dlg\_rect.left + border\_padding, dlg\_rect.top + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.top - offset + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.bottom - offset - border\_padding),  
 (dlg\_rect.left + border\_padding, dlg\_rect.bottom - border\_padding)  
 ]  
 pg.draw.polygon(self.SCREEN, (211, 200, 74), border\_points, 3)  
  
 # Rotate the text surfaces to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 victory\_surf = pg.transform.rotate(victory\_surf, angle)  
 victory\_shadow = pg.transform.rotate(victory\_shadow, angle)  
 highscore\_surf = pg.transform.rotate(highscore\_surf, angle)  
 highscore\_shadow = pg.transform.rotate(highscore\_shadow, angle)  
 prompt\_surf = pg.transform.rotate(prompt\_surf, angle)  
 prompt\_shadow = pg.transform.rotate(prompt\_shadow, angle)  
  
 # Blit each text surface centered at their respective positions  
 self.SCREEN.blit(victory\_shadow, victory\_rect.move(2, 2))  
 self.SCREEN.blit(victory\_surf, victory\_rect)  
 self.SCREEN.blit(highscore\_shadow, hs\_rect.move(2, 2))  
 self.SCREEN.blit(highscore\_surf, hs\_rect)  
 self.SCREEN.blit(prompt\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (167, 57, 57)  
 shadow\_color = (125, 28, 28)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def rainbow(self, hue):  
 color = pg.Color("#1e5294")  
 hue = (hue + 1) % 360  
 color.hsva = (hue, 100, 100, 100)  
 return color  
  
 def add\_word(self, width, words, word\_type, enemy):  
 found\_word = False  
 while not found\_word and len(self.current\_words) < len(words):  
 if word\_type == 'bonus' and self.bonus\_word\_counter >= 5:  
 selected = random.choice(self.bonus\_words)  
 self.bonus\_word\_counter = 0 # Reset counter after adding bonus word  
 # For normal words  
 else:  
 # Adjust selection logic to balance word lengths  
 word\_lengths = [len(word) for word in words]  
 current\_lengths = [len(word) for word in self.current\_words.keys()]  
 length\_counts = {length: current\_lengths.count(length) for length in set(word\_lengths)}  
  
 # Calculate weights to balance word lengths  
 weights = []  
 for length in word\_lengths:  
 if length\_counts.get(length, 0) < 2: # Prefer lengths not yet on screen  
 weights.append(1)  
 else:  
 weights.append(0.1)  
  
 selected = random.choices(words, weights=weights, k=1)[0]  
 if word\_type == 'stage1': # Only increment for normal words  
 self.bonus\_word\_counter += 1  
  
 # Skip if word is already on screen or starts with same letter  
 if selected not in self.current\_words and \  
 all(not w.startswith(selected[0]) for w in self.current\_words):  
 if selected not in self.word\_widths:  
 self.word\_widths[selected] = self.font.size(selected)[0]  
 w\_width = self.word\_widths[selected]  
 x = random.randrange(45, width - w\_width - 10)  
  
 # Check for overlaps  
 if not (enemy.sprite\_rect.left < x < enemy.sprite\_rect.right) and \  
 all(abs(x - meta[0]) > w\_width + 15 for meta in self.current\_words.values()):  
 self.current\_words[selected] = [x, 0, (150, 150, 150), word\_type]  
 found\_word = True  
  
 # Adjust word frequency based on word type  
 if word\_type == 'bonus':  
 self.word\_frequency = max(2.0, self.word\_frequency - 0.1)  
 else:  
 self.word\_frequency = min(5.0, self.word\_frequency + 0.1)  
  
 def create\_word\_surf(self, word, color, hue, word\_type):  
 w, h = self.font.size(word)  
 w += 12 # Increase width for padding  
 h += 12 # Increase height for padding  
 Surf = pg.Surface((w, h), pg.SRCALPHA, 32)  
  
 pg.draw.rect(Surf, (125, 28, 28, 150), Surf.get\_rect(), border\_radius=10)  
  
 being\_written = self.prompt\_content and word.startswith(self.prompt\_content)  
 start\_text = self.prompt\_content if being\_written else ''  
 end\_text = word[len(self.prompt\_content):] if being\_written else word  
 start\_surf = self.font.render(start\_text, True, pg.Color("black"))  
  
 # Set constant colors for bonus and bossfight word types  
 if word in self.bonus\_words:  
 transformed\_color = pg.Color("#ff3300")  
 # print("bonus")  
 else:  
 transformed\_color = self.rainbow(hue)  
 # print("normal")  
  
 end\_surf = self.font.render(end\_text, True, transformed\_color)  
 Surf.blit(start\_surf, (8, 8))  
 Surf.blit(end\_surf, end\_surf.get\_rect(right=w - 8, centery=h // 2))  
 return Surf  
  
 def generate\_prompt\_surf(self):  
 width = self.SCREEN.get\_width()  
 surf = pg.Surface((width, 50), pg.SRCALPHA)  
 shadow\_surf = pg.Surface((width, 10), pg.SRCALPHA)  
  
 # Create shadow  
 shadow\_surf.fill((167, 57, 57, 79))  
 surf.fill((125, 28, 35))  
 surf.set\_alpha(255)  
  
 self.SCREEN.blit(surf, (0, 0))  
 surf.blit(shadow\_surf, (0, -1))  
  
 color = pg.Color("#ff6600") if any(w.startswith(self.prompt\_content) for w in self.current\_words) else pg.Color(  
 "#ffffff")  
 rendered = self.font.render(self.prompt\_content, True, color)  
  
 # Create shadow text  
 shadow\_rendered = self.font.render(self.prompt\_content, True, pg.Color("black"))  
  
 # Center the prompt text horizontally on the surface  
 rect = rendered.get\_rect(centerx=width // 2, centery=25)  
 shadow\_rect = shadow\_rendered.get\_rect(centerx=width // 2 - 2, centery=25 - 2) # Offset for shadow effect  
  
 # Blit shadow first, then main text  
 surf.blit(shadow\_rendered, shadow\_rect)  
 surf.blit(rendered, rect)  
  
 # Draw a bar to indicate the position  
 bar\_width = 2  
 bar\_height = 40  
 bar\_x = rect.right + 5  
 bar\_y = 5  
 pg.draw.rect(surf, pg.Color("red"), (bar\_x, bar\_y, bar\_width, bar\_height))  
  
 return surf  
  
 def draw\_enemy\_hitpoints(self):  
 hp\_text = f"Enemy HP: {self.enemy.hitpoints:.1f}"  
 hp\_text\_shadow = self.font.render(hp\_text, True, pg.Color("black"))  
 hp\_surf = self.font.render(hp\_text, True, (255, 255, 255))  
 hp\_box = pg.Surface((hp\_surf.get\_width() + 10, hp\_surf.get\_height() + 10), pg.SRCALPHA)  
 hp\_box.fill((26, 62, 112, 190))  
  
 # Initialize and update fade alpha for enemy hitpoints  
 if not hasattr(self, 'hp\_alpha'):  
 self.hp\_alpha = 0  
 if self.hp\_alpha < 255:  
 self.hp\_alpha += 5 # Adjust increment as needed for smoother or faster fade  
 hp\_box.set\_alpha(self.hp\_alpha)  
  
 hp\_box\_rect = hp\_box.get\_rect(midtop=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 100))  
  
 # Create shadow of box  
 shadow\_offset = 2  
 shadow\_box = pg.Surface((hp\_box.get\_width(), hp\_box.get\_height()), pg.SRCALPHA)  
 shadow\_box.fill((224, 180, 0, 100)) # Darker color for shadow  
 shadow\_box\_rect = hp\_box\_rect.move(shadow\_offset, shadow\_offset)  
  
 # Blit shadow first, then the hitpoint box  
 self.SCREEN.blit(shadow\_box, shadow\_box\_rect)  
 self.SCREEN.blit(hp\_text\_shadow, hp\_box\_rect.move(2,2))  
 self.SCREEN.blit(hp\_box, hp\_box\_rect)  
 self.SCREEN.blit(hp\_surf, hp\_surf.get\_rect(center=hp\_box\_rect.center))  
  
 def draw\_ui(self):  
 top\_box = pg.Surface((self.SCREEN.get\_width(), 40), pg.SRCALPHA)  
 top\_box.fill((54, 54, 54, 200)) # Adjusted background color with opacity  
 top\_box\_rect = top\_box.get\_rect()  
 if not hasattr(self, 'ui\_alpha'):  
 self.ui\_alpha = 0  
  
 if self.ui\_alpha < 255:  
 self.ui\_alpha += 1 # Adjust the increment value as needed  
  
 top\_box.set\_alpha(self.ui\_alpha)  
 self.SCREEN.blit(top\_box, top\_box\_rect)  
  
 # Render the main text and its shadow  
 score\_surf = self.font.render(f"Score: {self.score}", True, (255, 255, 255))  
 health\_surf = self.font.render(f"Health: {self.health}", True, (255, 255, 255))  
 enemy\_name = self.font.render(f"Enemy: {self.enemy.name}", True, (255, 255, 255))  
 score\_shadow = self.font.render(f"Score: {self.score}", True, (0, 0, 0))  
 health\_shadow = self.font.render(f"Health: {self.health}", True, (0, 0, 0))  
 enemy\_shadow = self.font.render(f"Enemy: {self.enemy.name}", True, (0, 0, 0))  
  
 # Calculate positions for the text  
 screen\_width = self.SCREEN.get\_width()  
 score\_pos = (10, 10)  
 health\_pos = (screen\_width // 3, 10)  
 enemy\_pos = (2 \* screen\_width // 3, 10)  
  
 # Offset for the shadow effect  
 shadow\_offset = (2, 2)  
  
 # Blit the shadow first, then the main text  
 self.SCREEN.blit(score\_shadow, (score\_pos[0] + shadow\_offset[0], score\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(health\_shadow, (health\_pos[0] + shadow\_offset[0], health\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(enemy\_shadow, (enemy\_pos[0] + shadow\_offset[0], enemy\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(score\_surf, score\_pos)  
 self.SCREEN.blit(health\_surf, health\_pos)  
 self.SCREEN.blit(enemy\_name, enemy\_pos)  
  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (screen\_width // 3 - 5, 0),  
 (screen\_width // 3 - 5, 40), 2)  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (2 \* screen\_width // 3 - 5, 0),  
 (2 \* screen\_width // 3 - 5, 40), 2)  
  
 def display\_game\_over(self):  
 write\_score(self.score)  
 game\_over = self.font.render("GAME OVER", True, (255, 0, 0))  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 self.SCREEN.blit(game\_over, game\_over.get\_rect(center=center))  
 pg.display.flip()  
 pg.time.wait(2000)  
  
 def apply\_fade\_effect(self):  
 if self.fade\_direction != 0:  
 self.fade\_alpha += self.fade\_direction \* 10  
 if self.fade\_alpha >= 255:  
 self.fade\_alpha = 255  
 self.fade\_direction = 0  
 elif self.fade\_alpha <= 0:  
 self.fade\_alpha = 0  
 self.fade\_direction = 0  
 fade\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 fade\_surf.fill((255, 0, 0, self.fade\_alpha))  
 self.SCREEN.blit(fade\_surf, (0, 0))  
  
class Stage1Enemies:  
 def \_\_init\_\_(self, screen, level, normal\_sprite\_path, hit\_sprite\_path):  
 self.screen = screen  
 self.width, self.height = self.screen.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 36)  
 self.hitpoints = 25 + level \* 5  
 self.word\_speed = 1  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
 self.is\_hit = False  
 self.sprite\_alpha = 0  
  
 self.normal\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.hit\_sprite = pg.image.load(hit\_sprite\_path).convert\_alpha()  
 self.talk\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.defeat\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.normal\_sprite = pg.transform.scale(self.normal\_sprite, (300, 500))  
 self.hit\_sprite = pg.transform.scale(self.hit\_sprite, (300, 500))  
 self.talk\_sprite = pg.transform.scale(self.talk\_sprite, (300, 500))  
 self.defeat\_sprite = pg.transform.scale(self.defeat\_sprite, (300, 500))  
 self.sprite\_rect = self.normal\_sprite.get\_rect()  
 self.sprite\_rect.centerx = self.width - 250  
 self.sprite\_rect.centery = self.height - 300 # Adjusted to align with the prompt surf  
 self.word\_bg\_image = pg.image.load("resources/transparent/tristan.gif").convert\_alpha()  
 self.explosions = []  
  
 def reset\_word(self, current\_words):  
 if self.current\_word in current\_words:  
 del current\_words[self.current\_word]  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 def update(self, timepassed, player\_input, current\_words):  
 if self.sprite\_alpha < 255:  
 self.sprite\_alpha += 5  
  
 if self.hitpoints <= 0:  
 return False  
  
 if not self.current\_word and current\_words:  
 self.current\_word = random.choice(list(current\_words.keys()))  
 self.word\_progress = 0  
  
 if self.current\_word and (self.current\_word not in current\_words):  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 if self.start\_timer > 0:  
 self.start\_timer -= timepassed  
 return False  
  
 if self.current\_word:  
 self.word\_progress += timepassed \* self.word\_speed  
 meta = current\_words[self.current\_word]  
 # Use the updated meta data for y-position  
 word\_x = meta[0]  
 meta\_y = meta[1]  
 y = (meta\_y \* self.word\_speed) + abs(math.cos(meta\_y \* 3) \* 10)  
 word\_rect = pg.Rect(word\_x, y, self.font.size(self.current\_word)[0],  
 self.font.size(self.current\_word)[1])  
 if self.word\_progress >= len(self.current\_word):  
 # Store the completed word before resetting  
 completed\_word = self.current\_word  
 handle\_explosion\_effect(self.screen, self.font, self.sprite\_rect, completed\_word, self.explosions)  
 if self.current\_word in current\_words:  
 current\_words.pop(self.current\_word)  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.0  
 return True  
  
 return False  
  
 def get\_font\_size(self, word\_length):  
 if word\_length > 5:  
 return 24 # Smaller font size for words longer than 5 letters  
 else:  
 return 28 # Default font size  
  
 def draw(self):  
 if self.hitpoints <= 0:  
 current\_sprite = self.defeat\_sprite  
 else:  
 current\_sprite = self.hit\_sprite if self.is\_hit else self.normal\_sprite  
  
 sprite\_with\_alpha = current\_sprite.copy()  
 sprite\_with\_alpha.set\_alpha(self.sprite\_alpha)  
 self.screen.blit(sprite\_with\_alpha, self.sprite\_rect)  
  
 if self.hitpoints > 0 and self.current\_word:  
 # Render the typed and remaining portions of the word  
 typed = self.current\_word[:int(self.word\_progress)]  
 remaining = self.current\_word[int(self.word\_progress):]  
  
 # Get appropriate font size based on word length  
 font\_size = self.get\_font\_size(len(self.current\_word))  
 if len(self.current\_word) > 6:  
 font\_size -= 2  
 font = pg.font.Font("resources/DejaVuSans.ttf", font\_size)  
  
 typed\_surf = font.render(typed, True, (255, 0, 0))  
 remaining\_surf = font.render(remaining, True, (100, 100, 100))  
  
 total\_width = typed\_surf.get\_width() + remaining\_surf.get\_width()  
 text\_height = typed\_surf.get\_height()  
  
 # Define the text box size based on the text dimensions with extra margin  
 box\_width = int(total\_width \* 1.75) + 20  
 box\_height = int(text\_height \* 1.5) + 10  
  
 # Scale the background image for the word box  
 word\_bg\_image\_scaled = pg.transform.scale(self.word\_bg\_image, (box\_width, box\_height))  
  
 # Position the text box with a negative x-coordinate to overlay over the sprite  
 word\_box\_rect = word\_bg\_image\_scaled.get\_rect(  
 midright=(self.sprite\_rect.left - 20, self.sprite\_rect.centery))  
 word\_box\_rect.x += 100 # Adjust this value as needed to overlay the text box  
  
 # Calculate centered text position within the text box  
 text\_x = word\_box\_rect.left + (box\_width - total\_width) // 2  
 text\_y = word\_box\_rect.top + (box\_height - text\_height) // 2  
  
 # Blit the text box and then the text centered in it  
 self.screen.blit(word\_bg\_image\_scaled, word\_box\_rect)  
 self.screen.blit(typed\_surf, (text\_x, text\_y))  
 self.screen.blit(remaining\_surf, (text\_x + typed\_surf.get\_width(), text\_y))  
  
 # Draw any active explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions  
 if current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.screen.blit(img, rect)  
  
 def draw\_before\_battle(self):  
 self.screen.blit(self.normal\_sprite, self.sprite\_rect)  
  
class Minion1(Stage1Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/White-1.png", "resources/sprites/White-1-hit.gif")  
 self.name = "Classmate Ronald"  
 self.dialogue\_text = "\"Haha! I am going to sabotage your projects!\""  
 self.defeat\_text = "\"You still got a high score?\""  
 self.word\_speed = 1.2  
  
class Minion2(Stage1Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/White-2.png",  
 "resources/sprites/White-2-hit.gif")  
 self.name = "Classmate Igni"  
 self.dialogue\_text = "\"You know, you can back down now?\""  
 self.defeat\_text = "\"Ok, I give up...\""  
 self.word\_speed = 1.4  
  
class Minion3(Stage1Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/White-3.png",  
 "resources/sprites/White-3-hit.gif")  
 self.name = "Classmate Hans"  
 self.dialogue\_text = "\"You cant be that good?! Time to pound\""  
 self.defeat\_text = "\"No Friggin WAY!\""  
 self.word\_speed = 1.6  
  
class Boss(Stage1Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/tan-fight.png", "resources/sprites/tan-hit.gif")  
 self.name = "THE RIVAL"  
 self.dialogue\_text = "\" This is your final test, Good Luck!\""  
 self.defeat\_text = "\"Congratulations! You got me, have to concede...!\""  
 self.defeat\_sprite = pg.image.load("resources/sprites/tan-defeat.png").convert\_alpha()  
 self.talk\_sprite = pg.image.load("resources/sprites/tan-talk.png").convert\_alpha()  
 self.normal\_sprite = pg.transform.smoothscale(self.normal\_sprite, (450, 650))  
 self.hit\_sprite = pg.transform.smoothscale(self.hit\_sprite, (450, 650))  
 self.talk\_sprite = pg.transform.smoothscale(self.talk\_sprite, (450, 650))  
 self.defeat\_sprite = pg.transform.smoothscale(self.defeat\_sprite, (450, 650))  
 self.hitpoints = 50 + level \* 10 # Boss has more hitpoints  
 self.word\_speed = 2.5 # Boss has a faster word speed  
 self.sprite\_rect.centery = self.height // 2 # Adjusted to align with the prompt surf  
  
  
"""STAGE 1 END----------------------------------------------------------------------------------------------------------  
  
STAGE 2 START--------------------------------------------------------------------------------------------------------"""  
class Stage2:  
 def \_\_init\_\_(self, screen, level):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 22)  
 self.BG = stretch(pg.image.load("resources/backgrounds/gym\_blurred.png").convert\_alpha(), (width, height))  
 self.phase = 0  
  
 pg.key.set\_repeat(250, 30)  
  
 self.clock = pg.time.Clock()  
 self.stage2\_words, self.bonus\_words = generate\_words\_stage2()  
 self.current\_words = {}  
 self.word\_timer = 0  
 self.word\_frequency = 10  
 self.level = level  
 self.score = 0  
 self.health = 20 \* (level)  
 self.prompt\_content = ''  
 self.word\_speed = 50  
 self.word\_widths = {}  
 self.highscore = load\_score()  
 self.enemies = [Minion1STwo(screen, self.level), Minion2STwo(screen, self.level), Minion3STwo(screen, self.level),  
 BossSTwo(screen, self.level)]  
 self.current\_enemy\_index = 0  
 self.enemy = self.enemies[self.current\_enemy\_index]  
 self.enemy.talking = True  
 self.fade\_alpha = 0  
 self.fade\_direction = 1  
 self.damage\_flash\_alpha = 0  
 self.bonus\_word\_counter = 0  
  
 # Load background music  
 self.inbattle\_music = "resources/sounds/songs/s2minion.mp3"  
 self.prebattle\_music = "resources/sounds/songs/s2\_prebattle.mp3"  
 self.victory\_music = "resources/sounds/songs/s1victory.mp3"  
  
 # Load sound effects  
 self.enemyhit\_sfx = pg.mixer.Sound("resources/sounds/sfx/enemyhit.mp3")  
 self.win\_sfx = pg.mixer.Sound("resources/sounds/sfx/win.mp3")  
 self.wordcomplete\_sfx = pg.mixer.Sound("resources/sounds/sfx/wordcomplete.mp3")  
  
 self.explosions = []  
 self.bossfight\_pause\_timer = 0  
 self.falling\_words\_pause\_timer = 0  
 self.last\_bonus\_action = 'damage'  
  
 def run(self):  
 width, height = self.SCREEN.get\_size()  
 battle\_started = False  
 hue = 0  
  
 self.defeat\_display(BossSTwo(self.SCREEN, self.level))  
  
 # while self.current\_enemy\_index < len(self.enemies):  
 # self.enemy = self.enemies[self.current\_enemy\_index]  
 # self.before\_battle\_display(self.enemy)  
 # battle\_started = True  
 #  
 # pg.mixer.music.load(self.inbattle\_music)  
 # pg.mixer.music.play(-1)  
  
 self.current\_enemy\_index = 3  
 if self.current\_enemy\_index == 3:  
 self.enemy = self.enemies[self.current\_enemy\_index]  
 self.before\_battle\_display(self.enemy)  
 battle\_started = True  
  
 if self.enemy == self.enemies[3]:  
 self.inbattle\_music = "resources/sounds/songs/s2boss.mp3"  
 pg.mixer.music.load(self.inbattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 timepassed = self.clock.tick(60) / 1000.0  
  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 if battle\_started:  
 pause(self.SCREEN, self.BG)  
 else:  
 return  
 if battle\_started:  
 if event.unicode.isprintable():  
 self.prompt\_content += event.unicode  
 elif event.key == pg.K\_BACKSPACE:  
 self.prompt\_content = self.prompt\_content[:-1]  
 elif event.key == pg.K\_RETURN:  
 self.prompt\_content = ''  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 if isinstance(self.enemy, BossSTwo):  
 max\_health = self.enemy.get\_max\_health()  
 self.enemy.hitpoints = min(self.enemy.hitpoints + 0.1 \* timepassed, max\_health)  
  
 if self.health <= 0:  
 self.display\_game\_over()  
 return  
  
 if not battle\_started:  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_rect = prompt\_surf.get\_rect(center=(width // 2, height // 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
 else:  
 if self.fade\_alpha < 255:  
 self.apply\_fade\_effect()  
 else:  
 self.word\_timer += timepassed  
 if self.word\_timer > self.word\_frequency and len(self.current\_words) < len(self.stage2\_words):  
 # Add normal word  
 self.add\_word(width, self.stage2\_words, 'stage2', self.enemy)  
 self.word\_timer = 0  
  
 # Check for bonus words  
 if self.bonus\_word\_counter >= 5:  
 self.add\_word(width, self.bonus\_words, 'bonus', self.enemy)  
  
 # Keep minimum number of words  
 while len(self.current\_words) < 5:  
 self.add\_word(width, self.stage2\_words, 'stage2', self.enemy)  
  
 for word, meta in list(self.current\_words.items()):  
 meta[1] += timepassed  
 y = (meta[1] \* self.word\_speed) + abs(math.cos(meta[1] \* 3) \* 10)  
 word\_rect = pg.Rect(meta[0], y, self.font.size(word)[0], self.font.size(word)[1])  
 if y > height:  
 del self.current\_words[word]  
 self.health -= 1  
 self.damage\_flash\_alpha = 150  
 elif word == self.prompt\_content:  
 del self.current\_words[word]  
 self.score += len(word) \* 2  
 self.prompt\_content = ""  
 self.wordcomplete\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 if word == self.enemy.current\_word:  
 self.apply\_damage(1, word)  
 self.handle\_explosion\_effect(word\_rect)  
 elif word in self.bonus\_words:  
 if self.last\_bonus\_action == 'damage':  
 self.apply\_damage(3, word)  
 self.last\_bonus\_action = 'health'  
 self.handle\_explosion\_effect(word\_rect)  
 else:  
 self.health = min(self.health + 1.5, 50)  
 self.last\_bonus\_action = 'damage'  
 self.enemy.reset\_word(self.current\_words)  
 self.enemy.is\_hit = True  
 self.enemyhit\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 else:  
 self.enemy.is\_hit = False  
  
 else:  
 word\_surf = self.create\_word\_surf(word, meta[2], hue, meta[3])  
 word\_rect = word\_surf.get\_rect(center=(meta[0], y))  
 enemy\_rect = self.enemy.sprite\_rect  
 if word\_rect.colliderect(enemy\_rect):  
 if enemy\_rect.left - word\_rect.width - 10 >= 0:  
 word\_rect.right = enemy\_rect.left - 10  
 else:  
 word\_rect.left = enemy\_rect.right + 10  
 self.SCREEN.blit(word\_surf, word\_rect)  
  
 if self.current\_words:  
 if self.enemy.update(timepassed, self.prompt\_content, self.current\_words):  
 if isinstance(self.enemy, Boss):  
 self.health -= 1.25 \* self.level # Boss deals near twice the damage  
 else:  
 self.health -= self.level # Minions deal damage based on the level  
 self.damage\_flash\_alpha = 150  
  
 if self.enemy.hitpoints <= 0:  
 self.win\_sfx.play()  
 pg.mixer.music.stop()  
 self.defeat\_display(self.enemy)  
 self.current\_enemy\_index += 1  
 break  
  
 self.enemy.draw()  
 self.SCREEN.blit(self.generate\_prompt\_surf(), (0, height - 50))  
 self.draw\_ui()  
 self.draw\_enemy\_hitpoints()  
  
 if self.damage\_flash\_alpha > 0:  
 flash\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 flash\_surf.fill((255, 0, 0, self.damage\_flash\_alpha))  
 self.SCREEN.blit(flash\_surf, (0, 0))  
 self.damage\_flash\_alpha = max(0, self.damage\_flash\_alpha - 8)  
  
 # Draw and manage explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions if  
 current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.SCREEN.blit(img, rect)  
  
 pg.display.flip()  
 hue = (hue + 1) % 360 # Update hue for the next frame  
  
 self.display\_victory()  
 from introduction import Stage2Outro  
 outro = Stage2Outro(self.SCREEN)  
 outro.run()  
  
 def apply\_damage(self, damage, word, reset\_word=True, play\_sound=True):  
 self.enemy.hitpoints = max(0, self.enemy.hitpoints - ((damage \* len(word)) \* 0.35))  
 self.enemy.is\_hit = True  
 if reset\_word:  
 self.enemy.reset\_word(self.current\_words)  
 if play\_sound:  
 self.enemyhit\_sfx.play()  
  
 def handle\_explosion\_effect(self, word\_rect):  
 explosion\_image = pg.image.load(f'resources/transparent/boom-{random.randint(1, 3)}.gif').convert\_alpha()  
 scale\_factor = 0.20 # Adjust this factor to make the explosion image larger  
 new\_width = int(explosion\_image.get\_width() \* scale\_factor)  
 new\_height = int(explosion\_image.get\_height() \* scale\_factor)  
 explosion\_image = pg.transform.scale(explosion\_image, (new\_width, new\_height))  
 explosion\_rect = explosion\_image.get\_rect(center=word\_rect.center)  
 self.explosions.append((explosion\_image, explosion\_rect, pg.time.get\_ticks()))  
  
 def before\_battle\_display(self, minion):  
 fade\_duration = 1.0 # Duration of the fade-in effect in seconds  
 fade\_alpha = 0 # Initial alpha value for fade-in effect  
 fade\_increment = 255 / (fade\_duration \* 60) # Increment per frame (assuming 60 FPS)  
  
 pg.mixer.music.load(self.prebattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 pg.mixer.music.stop()  
 return # Exit the display and start the battle  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the minion sprite talking with fade-in effect  
 talk\_sprite = minion.talk\_sprite.copy()  
 talk\_sprite.set\_alpha(fade\_alpha)  
 talk\_sprite\_rect = talk\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(talk\_sprite, talk\_sprite\_rect)  
  
 # Adjust font size based on the length of the dialogue text  
 dialogue\_text = minion.dialogue\_text  
 words = dialogue\_text.split()  
 font\_size = 20 if len(words) <= 7 else 15  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", font\_size)  
 dlg\_surf = small\_font.render(dialogue\_text, True, pg.Color(250, 250, 250))  
 dlg\_surf.set\_alpha(fade\_alpha)  
  
 # Calculate the bounding rectangle of the text surface and add padding  
 padding = 10  
 dlg\_rect = pg.Rect(  
 self.SCREEN.get\_width() // 2 - dlg\_surf.get\_width() // 2 - padding,  
 self.SCREEN.get\_height() // 2 - dlg\_surf.get\_height() // 2 - padding,  
 dlg\_surf.get\_width() + padding \* 2,  
 dlg\_surf.get\_height() + padding \* 2  
 )  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
  
 # Blit the text surface centered at its position  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_rect.center))  
  
 # Draw the top bar with fade-in effect  
 top\_bar\_height = 50  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), top\_bar\_height), pg.SRCALPHA)  
 top\_bar.fill((167, 57, 57, fade\_alpha))  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), top\_bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill((125, 28, 28, fade\_alpha))  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Draw the bottom bar with fade-in effect  
 bottom\_bar\_height = 100  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bottom\_bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill((167, 57, 57, fade\_alpha))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bottom\_bar\_height))  
  
 # Draw the prompt to continue with fade-in effect  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - bottom\_bar\_height // 2))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Apply fade-in effect  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def defeat\_display(self, minion):  
 fade\_duration = 1.0  
 fade\_alpha = 0  
 fade\_increment = 255 / (fade\_duration \* 60)  
  
 pg.mixer.music.load(self.victory\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 pg.mixer.music.stop()  
 return  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 defeat\_sprite = minion.defeat\_sprite.copy()  
 defeat\_sprite.set\_alpha(fade\_alpha)  
 defeat\_sprite\_rect = defeat\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(defeat\_sprite, defeat\_sprite\_rect)  
  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 20)  
 dlg\_surf = small\_font.render(minion.defeat\_text, True, pg.Color(250, 250, 250))  
 dlg\_surf.set\_alpha(fade\_alpha)  
  
 # Calculate the bounding rectangle of the text surface and add padding  
 padding = 10  
 dlg\_rect = pg.Rect(  
 0, 0,  
 dlg\_surf.get\_width() + padding \* 2,  
 dlg\_surf.get\_height() + padding \* 2  
 )  
 dlg\_rect.centerx = self.SCREEN.get\_width() // 2  
 dlg\_rect.centery = defeat\_sprite\_rect.centery  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
  
 # Blit the text surface centered at its position  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_rect.center))  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (167, 57, 57)  
 shadow\_color = (125, 28, 28)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar\_y = self.SCREEN.get\_height() - bar\_height  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, bottom\_bar\_y))  
 self.SCREEN.blit(bottom\_bar, (0, bottom\_bar\_y))  
  
 prompt\_text = "Press Enter to continue"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, bottom\_bar\_y + bar\_height // 2))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def display\_victory(self):  
 if self.score > self.highscore:  
 self.highscore = self.score  
 write\_score(self.highscore)  
  
 pg.mixer.music.load(self.victory\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 else:  
 LoadingScreen(self.SCREEN).run()  
 from introduction import Stage1Outro  
 outro = Stage1Outro(self.SCREEN)  
 outro.run()  
  
 # Prepare text surfaces and their positions  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 victory\_surf = self.font.render("VICTORY!", True, pg.Color("white"))  
 victory\_shadow = self.font.render("VICTORY!", True, pg.Color("black"))  
 highscore\_text = f"Highscore: {self.highscore}"  
 highscore\_surf = self.font.render(highscore\_text, True, pg.Color("white"))  
 highscore\_shadow = self.font.render(highscore\_text, True, pg.Color("black"))  
 prompt\_text = "Press any key for next stage, or Esc for main menu"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
  
 victory\_rect = victory\_surf.get\_rect(center=(center[0], center[1] - 40))  
 hs\_rect = highscore\_surf.get\_rect(center=center)  
 prompt\_rect = prompt\_surf.get\_rect(center=(center[0], center[1] + 40))  
  
 # Calculate the bounding rectangle of all text surfaces and add padding  
 union\_rect = victory\_rect.union(hs\_rect).union(prompt\_rect)  
 padding = 10  
 dlg\_rect = pg.Rect(  
 union\_rect.left - padding,  
 union\_rect.top - padding,  
 union\_rect.width + 6 \* padding,  
 union\_rect.height + 6 \* padding  
 )  
  
 # Center the dialog box on the screen  
 dlg\_rect.center = center  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Create the dialog box surface with an opaque yellow red color  
 dlg\_box = pg.Surface((dlg\_rect.width, dlg\_rect.height))  
 dlg\_box.fill((255, 193, 33))  
  
 # Draw background and dialog box  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Draw border as a parallelogram  
 border\_padding = 5  
 border\_points = [  
 (dlg\_rect.left + border\_padding, dlg\_rect.top + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.top - offset + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.bottom - offset - border\_padding),  
 (dlg\_rect.left + border\_padding, dlg\_rect.bottom - border\_padding)  
 ]  
 pg.draw.polygon(self.SCREEN, (211, 200, 74), border\_points, 3)  
  
 # Rotate the text surfaces to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 victory\_surf = pg.transform.rotate(victory\_surf, angle)  
 victory\_shadow = pg.transform.rotate(victory\_shadow, angle)  
 highscore\_surf = pg.transform.rotate(highscore\_surf, angle)  
 highscore\_shadow = pg.transform.rotate(highscore\_shadow, angle)  
 prompt\_surf = pg.transform.rotate(prompt\_surf, angle)  
 prompt\_shadow = pg.transform.rotate(prompt\_shadow, angle)  
  
 # Blit each text surface centered at their respective positions  
 self.SCREEN.blit(victory\_shadow, victory\_rect.move(2, 2))  
 self.SCREEN.blit(victory\_surf, victory\_rect)  
 self.SCREEN.blit(highscore\_shadow, hs\_rect.move(2, 2))  
 self.SCREEN.blit(highscore\_surf, hs\_rect)  
 self.SCREEN.blit(prompt\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (167, 57, 57)  
 shadow\_color = (125, 28, 28)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def rainbow(self, hue):  
 color = pg.Color("white")  
 hue = (hue + 1) % 360  
 color.hsva = (hue, 100, 100, 100)  
 return color  
  
 def add\_word(self, width, words, word\_type, enemy):  
 found\_word = False  
 while not found\_word and len(self.current\_words) < len(words):  
 if word\_type == 'bonus' and self.bonus\_word\_counter >= 5:  
 selected = random.choice(self.bonus\_words)  
 self.bonus\_word\_counter = 0 # Reset counter after adding bonus word  
 # For normal words  
 else:  
 # Adjust selection logic to balance word lengths  
 word\_lengths = [len(word) for word in words]  
 current\_lengths = [len(word) for word in self.current\_words.keys()]  
 length\_counts = {length: current\_lengths.count(length) for length in set(word\_lengths)}  
  
 # Calculate weights to balance word lengths  
 weights = []  
 for length in word\_lengths:  
 if length\_counts.get(length, 0) < 2: # Prefer lengths not yet on screen  
 weights.append(1)  
 else:  
 weights.append(0.1)  
  
 selected = random.choices(words, weights=weights, k=1)[0]  
 if word\_type == 'stage2': # Only increment for normal words  
 self.bonus\_word\_counter += 1  
  
 # Skip if word is already on screen or starts with same letter  
 if selected not in self.current\_words and \  
 all(not w.startswith(selected[0]) for w in self.current\_words):  
 if selected not in self.word\_widths:  
 self.word\_widths[selected] = self.font.size(selected)[0]  
 w\_width = self.word\_widths[selected]  
 x = random.randrange(45, width - w\_width - 10)  
  
 # Check for overlaps  
 if not (enemy.sprite\_rect.left < x < enemy.sprite\_rect.right) and \  
 all(abs(x - meta[0]) > w\_width + 15 for meta in self.current\_words.values()):  
 self.current\_words[selected] = [x, 0, (150, 150, 150), word\_type]  
 found\_word = True  
  
 # Adjust word frequency based on word type  
 if word\_type == 'bonus':  
 self.word\_frequency = max(2.0, self.word\_frequency - 0.1)  
 else:  
 self.word\_frequency = min(5.0, self.word\_frequency + 0.1)  
  
 def create\_word\_surf(self, word, color, hue, word\_type):  
 w, h = self.font.size(word)  
 w += 12 # Increase width for padding  
 h += 12 # Increase height for padding  
 Surf = pg.Surface((w, h), pg.SRCALPHA, 32)  
  
 pg.draw.rect(Surf, (222, 153, 0, 200), Surf.get\_rect(), border\_radius=10)  
  
 being\_written = self.prompt\_content and word.startswith(self.prompt\_content)  
 start\_text = self.prompt\_content if being\_written else ''  
 end\_text = word[len(self.prompt\_content):] if being\_written else word  
 start\_surf = self.font.render(start\_text, True, pg.Color("black"))  
  
 # Set constant colors for bonus and bossfight word types  
 if word in self.bonus\_words:  
 transformed\_color = pg.Color("gold")  
 # print("bonus")  
 else:  
 transformed\_color = self.rainbow(hue)  
 # print("normal")  
  
 end\_surf = self.font.render(end\_text, True, transformed\_color)  
 Surf.blit(start\_surf, (8, 8))  
 Surf.blit(end\_surf, end\_surf.get\_rect(right=w - 8, centery=h // 2))  
 return Surf  
  
 def generate\_prompt\_surf(self):  
 width = self.SCREEN.get\_width()  
 surf = pg.Surface((width, 50), pg.SRCALPHA)  
 shadow\_surf = pg.Surface((width, 10), pg.SRCALPHA)  
  
 # Create shadow  
 shadow\_surf.fill((167, 57, 57, 79))  
 surf.fill((125, 28, 35))  
 surf.set\_alpha(255)  
  
 self.SCREEN.blit(surf, (0, 0))  
 surf.blit(shadow\_surf, (0, -1))  
  
 color = pg.Color("#ff6600") if any(w.startswith(self.prompt\_content) for w in self.current\_words) else pg.Color(  
 "#ffffff")  
 rendered = self.font.render(self.prompt\_content, True, color)  
  
 # Create shadow text  
 shadow\_rendered = self.font.render(self.prompt\_content, True, pg.Color("black"))  
  
 # Center the prompt text horizontally on the surface  
 rect = rendered.get\_rect(centerx=width // 2, centery=25)  
 shadow\_rect = shadow\_rendered.get\_rect(centerx=width // 2 - 2, centery=25 - 2) # Offset for shadow effect  
  
 # Blit shadow first, then main text  
 surf.blit(shadow\_rendered, shadow\_rect)  
 surf.blit(rendered, rect)  
  
 # Draw a bar to indicate the position  
 bar\_width = 2  
 bar\_height = 40  
 bar\_x = rect.right + 5  
 bar\_y = 5  
 pg.draw.rect(surf, pg.Color("red"), (bar\_x, bar\_y, bar\_width, bar\_height))  
  
 return surf  
  
 def draw\_enemy\_hitpoints(self):  
 hp\_text = f"Enemy HP: {self.enemy.hitpoints:.1f}"  
 hp\_text\_shadow = self.font.render(hp\_text, True, pg.Color("black"))  
 hp\_surf = self.font.render(hp\_text, True, (255, 255, 255))  
 hp\_box = pg.Surface((hp\_surf.get\_width() + 10, hp\_surf.get\_height() + 10), pg.SRCALPHA)  
 hp\_box.fill((26, 62, 112, 190))  
  
 # Initialize and update fade alpha for enemy hitpoints  
 if not hasattr(self, 'hp\_alpha'):  
 self.hp\_alpha = 0  
 if self.hp\_alpha < 255:  
 self.hp\_alpha += 5 # Adjust increment as needed for smoother or faster fade  
 hp\_box.set\_alpha(self.hp\_alpha)  
  
 hp\_box\_rect = hp\_box.get\_rect(midtop=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 100))  
  
 # Create shadow of box  
 shadow\_offset = 2  
 shadow\_box = pg.Surface((hp\_box.get\_width(), hp\_box.get\_height()), pg.SRCALPHA)  
 shadow\_box.fill((224, 180, 0, 100)) # Darker color for shadow  
 shadow\_box\_rect = hp\_box\_rect.move(shadow\_offset, shadow\_offset)  
  
 # Blit shadow first, then the hitpoint box  
 self.SCREEN.blit(shadow\_box, shadow\_box\_rect)  
 self.SCREEN.blit(hp\_text\_shadow, hp\_box\_rect.move(2,2))  
 self.SCREEN.blit(hp\_box, hp\_box\_rect)  
 self.SCREEN.blit(hp\_surf, hp\_surf.get\_rect(center=hp\_box\_rect.center))  
  
 def draw\_ui(self):  
 top\_box = pg.Surface((self.SCREEN.get\_width(), 40), pg.SRCALPHA)  
 top\_box.fill((54, 54, 54, 200)) # Adjusted background color with opacity  
 top\_box\_rect = top\_box.get\_rect()  
 if not hasattr(self, 'ui\_alpha'):  
 self.ui\_alpha = 0  
  
 if self.ui\_alpha < 255:  
 self.ui\_alpha += 1 # Adjust the increment value as needed  
  
 top\_box.set\_alpha(self.ui\_alpha)  
 self.SCREEN.blit(top\_box, top\_box\_rect)  
  
 # Render the main text and its shadow  
 score\_surf = self.font.render(f"Score: {self.score}", True, (255, 255, 255))  
 health\_surf = self.font.render(f"Health: {self.health}", True, (255, 255, 255))  
 enemy\_name = self.font.render(f"Enemy: {self.enemy.name}", True, (255, 255, 255))  
 score\_shadow = self.font.render(f"Score: {self.score}", True, (0, 0, 0))  
 health\_shadow = self.font.render(f"Health: {self.health}", True, (0, 0, 0))  
 enemy\_shadow = self.font.render(f"Enemy: {self.enemy.name}", True, (0, 0, 0))  
  
 # Calculate positions for the text  
 screen\_width = self.SCREEN.get\_width()  
 score\_pos = (10, 10)  
 health\_pos = (screen\_width // 3, 10)  
 enemy\_pos = (2 \* screen\_width // 3, 10)  
  
 # Offset for the shadow effect  
 shadow\_offset = (2, 2)  
  
 # Blit the shadow first, then the main text  
 self.SCREEN.blit(score\_shadow, (score\_pos[0] + shadow\_offset[0], score\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(health\_shadow, (health\_pos[0] + shadow\_offset[0], health\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(enemy\_shadow, (enemy\_pos[0] + shadow\_offset[0], enemy\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(score\_surf, score\_pos)  
 self.SCREEN.blit(health\_surf, health\_pos)  
 self.SCREEN.blit(enemy\_name, enemy\_pos)  
  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (screen\_width // 3 - 5, 0),  
 (screen\_width // 3 - 5, 40), 2)  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (2 \* screen\_width // 3 - 5, 0),  
 (2 \* screen\_width // 3 - 5, 40), 2)  
  
 def display\_game\_over(self):  
 write\_score(self.score)  
 game\_over = self.font.render("GAME OVER", True, (255, 0, 0))  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 self.SCREEN.blit(game\_over, game\_over.get\_rect(center=center))  
 pg.display.flip()  
 pg.time.wait(2000)  
  
 def apply\_fade\_effect(self):  
 if self.fade\_direction != 0:  
 self.fade\_alpha += self.fade\_direction \* 10  
 if self.fade\_alpha >= 255:  
 self.fade\_alpha = 255  
 self.fade\_direction = 0  
 elif self.fade\_alpha <= 0:  
 self.fade\_alpha = 0  
 self.fade\_direction = 0  
 fade\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 fade\_surf.fill((255, 0, 0, self.fade\_alpha))  
 self.SCREEN.blit(fade\_surf, (0, 0))  
  
class Stage2Enemies:  
 def \_\_init\_\_(self, screen, level, normal\_sprite\_path, hit\_sprite\_path):  
 self.screen = screen  
 self.width, self.height = self.screen.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 36)  
 self.hitpoints = 25 + level \* 5  
 self.word\_speed = 1  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
 self.is\_hit = False  
 self.sprite\_alpha = 0  
 self.name = ""  
  
 self.normal\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.hit\_sprite = pg.image.load(hit\_sprite\_path).convert\_alpha()  
 self.talk\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.defeat\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.normal\_sprite = pg.transform.scale(self.normal\_sprite, (300, 500))  
 self.hit\_sprite = pg.transform.scale(self.hit\_sprite, (300, 500))  
 self.talk\_sprite = pg.transform.scale(self.talk\_sprite, (300, 500))  
 self.defeat\_sprite = pg.transform.scale(self.defeat\_sprite, (300, 500))  
 self.sprite\_rect = self.normal\_sprite.get\_rect()  
 self.sprite\_rect.centerx = self.width - 250  
 self.sprite\_rect.centery = self.height - 300 # Adjusted to align with the prompt surf  
 self.word\_bg\_image = pg.image.load("resources/transparent/hao.gif").convert\_alpha()  
 self.explosions = []  
  
 def reset\_word(self, current\_words):  
 if self.current\_word in current\_words:  
 del current\_words[self.current\_word]  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 def update(self, timepassed, player\_input, current\_words):  
 if self.sprite\_alpha < 255:  
 self.sprite\_alpha += 5  
  
 if self.hitpoints <= 0:  
 return False  
  
 if not self.current\_word and current\_words:  
 self.current\_word = random.choice(list(current\_words.keys()))  
 self.word\_progress = 0  
  
 if self.current\_word and (self.current\_word not in current\_words):  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 if self.start\_timer > 0:  
 self.start\_timer -= timepassed  
 return False  
  
 if self.current\_word:  
 self.word\_progress += timepassed \* self.word\_speed  
 meta = current\_words[self.current\_word]  
 # Use the updated meta data for y-position  
 word\_x = meta[0]  
 meta\_y = meta[1]  
 y = (meta\_y \* self.word\_speed) + abs(math.cos(meta\_y \* 3) \* 10)  
 word\_rect = pg.Rect(word\_x, y, self.font.size(self.current\_word)[0],  
 self.font.size(self.current\_word)[1])  
 if self.word\_progress >= len(self.current\_word):  
 # Store the completed word before resetting  
 completed\_word = self.current\_word  
 handle\_explosion\_effect(self.screen, self.font, self.sprite\_rect, completed\_word, self.explosions)  
 if self.current\_word in current\_words:  
 current\_words.pop(self.current\_word)  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.0  
 return True  
  
 return False  
  
 def get\_font\_size(self, word\_length):  
 if word\_length > 5:  
 return 24 # Smaller font size for words longer than 5 letters  
 else:  
 return 28 # Default font size  
  
 def draw(self):  
 if self.hitpoints <= 0:  
 current\_sprite = self.defeat\_sprite  
 else:  
 current\_sprite = self.hit\_sprite if self.is\_hit else self.normal\_sprite  
  
 sprite\_with\_alpha = current\_sprite.copy()  
 sprite\_with\_alpha.set\_alpha(self.sprite\_alpha)  
 self.screen.blit(sprite\_with\_alpha, self.sprite\_rect)  
  
 if self.hitpoints > 0 and self.current\_word:  
 # Render the typed and remaining portions of the word  
 typed = self.current\_word[:int(self.word\_progress)]  
 remaining = self.current\_word[int(self.word\_progress):]  
  
 # Get appropriate font size based on word length  
 font\_size = self.get\_font\_size(len(self.current\_word))  
 font = pg.font.Font("resources/DejaVuSans.ttf", font\_size)  
  
 typed\_surf = font.render(typed, True, (255, 0, 0))  
 remaining\_surf = font.render(remaining, True, (100, 100, 100))  
  
 total\_width = typed\_surf.get\_width() + remaining\_surf.get\_width()  
 text\_height = typed\_surf.get\_height()  
  
 # Define the text box size based on the text dimensions with extra margin  
 box\_width = int(total\_width \* 1.5) + 20  
 box\_height = int(text\_height \* 1.5) + 10  
  
 # Scale the background image for the word box  
 word\_bg\_image\_scaled = pg.transform.scale(self.word\_bg\_image, (box\_width, box\_height))  
  
 # Position the text box with a negative x-coordinate to overlay over the sprite  
 word\_box\_rect = word\_bg\_image\_scaled.get\_rect(  
 midright=(self.sprite\_rect.left - 20, self.sprite\_rect.centery))  
 word\_box\_rect.x += 100 # Adjust this value as needed to overlay the text box  
  
 # Calculate centered text position within the text box  
 text\_x = word\_box\_rect.left + (box\_width - total\_width) // 2  
 text\_y = word\_box\_rect.top + (box\_height - text\_height) // 2  
  
 # Blit the text box and then the text centered in it  
 self.screen.blit(word\_bg\_image\_scaled, word\_box\_rect)  
 self.screen.blit(typed\_surf, (text\_x, text\_y))  
 self.screen.blit(remaining\_surf, (text\_x + typed\_surf.get\_width(), text\_y))  
  
 # Draw any active explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions  
 if current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.screen.blit(img, rect)  
  
 def draw\_before\_battle(self):  
 self.screen.blit(self.normal\_sprite, self.sprite\_rect)  
  
class Minion1STwo(Stage2Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/Yellow-1.png", "resources/sprites/Yellow-1-hit.gif")  
 self.dialogue\_text = "\"Prepare yourself for the battle!\""  
 self.defeat\_text = "\"Congrats!\""  
 self.word\_speed = 1.4  
 self.name = "Contestant Bogart"  
  
class Minion2STwo(Stage2Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/Yellow-2.png",  
 "resources/sprites/Yellow-2-hit.gif")  
 self.dialogue\_text = "\"Prepare yourself for the battle!\""  
 self.defeat\_text = "\"Congrats!\""  
 self.word\_speed = 1.6  
 self.name = "Contestant Pedro"  
  
class Minion3STwo(Stage2Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/Yellow-3.png",  
 "resources/sprites/Yellow-3-hit.gif")  
 self.dialogue\_text = "\"Prepare yourself for the battle!\""  
 self.defeat\_text = "\"Congrats!\""  
 self.word\_speed = 1.8  
 self.name = "Contestant Farje"  
  
class BossSTwo(Stage2Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/hao-fight.png", "resources/sprites/hao-hit.gif")  
 self.defeat\_sprite = pg.image.load("resources/sprites/hao-defeat.png").convert\_alpha()  
 self.talk\_sprite = pg.image.load("resources/sprites/hao-talk.png").convert\_alpha()  
 self.normal\_sprite = pg.transform.smoothscale(self.normal\_sprite, (450, 650))  
 self.hit\_sprite = pg.transform.smoothscale(self.hit\_sprite, (450, 650))  
 self.talk\_sprite = pg.transform.smoothscale(self.talk\_sprite, (450, 650))  
 self.defeat\_sprite = pg.transform.smoothscale(self.defeat\_sprite, (450, 650))  
 self.dialogue\_text = "\"I am kind of a slow typer so please go easy on me!\""  
 self.defeat\_text = "\"Damn you're fast! Can you teach me how to type fast?\""  
 self.hitpoints = 50 + level \* 10 # Boss has more hitpoints  
 self.max\_health = self.hitpoints # Store the initial maximum health  
 self.word\_speed = 3.0 # Boss has a faster word speed  
 self.sprite\_rect.centery = self.height // 2 # Adjusted to align with the prompt surf  
 self.name = "THE INTELLECTUAL"  
  
 def get\_max\_health(self):  
 return self.max\_health  
  
"""STAGE 2 END----------------------------------------------------------------------------------------------------------  
  
STAGE 3 START--------------------------------------------------------------------------------------------------------"""  
class Stage3:  
 def \_\_init\_\_(self, screen, level):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 22)  
 self.BG = stretch(pg.image.load("resources/backgrounds/plaza\_blurred.jpg").convert\_alpha(), (width, height))  
 self.phase = 0  
  
 pg.key.set\_repeat(250, 30)  
  
 self.clock = pg.time.Clock()  
 self.stage3\_words, self.bonus\_words = generate\_words\_stage3()  
 self.current\_words = {}  
 self.word\_timer = 0  
 self.word\_frequency = 15  
 self.level = level  
 self.score = 0  
 self.health = 20 \* (level)  
 self.prompt\_content = ''  
 self.word\_speed = 58  
 self.word\_widths = {}  
 self.highscore = load\_score()  
 self.enemies = [Minion1SThree(screen, self.level), Minion2SThree(screen, self.level), Minion3SThree(screen, self.level),  
 BossSThree(screen, self.level)]  
 self.current\_enemy\_index = 0  
 self.enemy = self.enemies[self.current\_enemy\_index]  
 self.enemy.talking = True  
 self.fade\_alpha = 0  
 self.fade\_direction = 1  
 self.damage\_flash\_alpha = 0  
 self.bonus\_word\_counter = 0  
  
 # Load background music  
 self.inbattle\_music = "resources/sounds/songs/s3minion.mp3"  
 self.prebattle\_music = "resources/sounds/songs/s3\_prebattle.mp3"  
 self.victory\_music = "resources/sounds/songs/s1victory.mp3"  
 self.mamemmy\_music = "resources/sounds/songs/mamemmy.mp3"  
  
 # Load sound effects  
 self.enemyhit\_sfx = pg.mixer.Sound("resources/sounds/sfx/enemyhit.mp3")  
 self.win\_sfx = pg.mixer.Sound("resources/sounds/sfx/win.mp3")  
 self.wordcomplete\_sfx = pg.mixer.Sound("resources/sounds/sfx/wordcomplete.mp3")  
  
 self.explosions = []  
 self.bossfight\_pause\_timer = 0  
 self.falling\_words\_pause\_timer = 0  
 self.last\_bonus\_action = 'damage'  
  
 def run(self):  
 width, height = self.SCREEN.get\_size()  
 battle\_started = False  
 hue = 0  
  
 self.maam\_emmy\_display()  
  
 while self.current\_enemy\_index < len(self.enemies):  
 self.enemy = self.enemies[self.current\_enemy\_index]  
 self.before\_battle\_display(self.enemy)  
 battle\_started = True  
  
 # self.current\_enemy\_index = 3  
 # if self.current\_enemy\_index == 3:  
 # self.enemy = self.enemies[self.current\_enemy\_index]  
 # self.maam\_emmy\_display()  
 # self.before\_battle\_display(self.enemy)  
 # battle\_started = True  
  
 pg.mixer.music.load(self.inbattle\_music)  
 pg.mixer.music.play(-1)  
  
  
 if self.enemy == self.enemies[3]:  
 self.inbattle\_music = "resources/sounds/songs/s3boss.mp3"  
 pg.mixer.music.load(self.inbattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 timepassed = self.clock.tick(60) / 1000.0  
  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 if battle\_started:  
 pause(self.SCREEN, self.BG)  
 else:  
 return  
 if battle\_started:  
 if event.unicode.isprintable():  
 self.prompt\_content += event.unicode  
 elif event.key == pg.K\_BACKSPACE:  
 self.prompt\_content = self.prompt\_content[:-1]  
 elif event.key == pg.K\_RETURN:  
 self.prompt\_content = ''  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 if isinstance(self.enemy, BossSTwo):  
 max\_health = self.enemy.get\_max\_health()  
 self.enemy.hitpoints = min(self.enemy.hitpoints + 0.02 \* timepassed, max\_health)  
  
 if self.health <= 0:  
 self.display\_game\_over()  
 return  
  
 if not battle\_started:  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_rect = prompt\_surf.get\_rect(center=(width // 2, height // 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
 else:  
 if self.fade\_alpha < 255:  
 self.apply\_fade\_effect()  
 else:  
 self.word\_timer += timepassed  
 if self.word\_timer > self.word\_frequency and len(self.current\_words) < len(self.stage3\_words):  
 # Add normal word  
 self.add\_word(width, self.stage3\_words, 'stage3', self.enemy)  
 self.word\_timer = 0  
  
 # Check for bonus words  
 if self.bonus\_word\_counter >= 5:  
 self.add\_word(width, self.bonus\_words, 'bonus', self.enemy)  
  
 # Keep minimum number of words  
 while len(self.current\_words) < 5:  
 self.add\_word(width, self.stage3\_words, 'stage3', self.enemy)  
  
 for word, meta in list(self.current\_words.items()):  
 meta[1] += timepassed  
 y = (meta[1] \* self.word\_speed) + abs(math.cos(meta[1] \* 3) \* 10)  
 word\_rect = pg.Rect(meta[0], y, self.font.size(word)[0], self.font.size(word)[1])  
 if y > height:  
 del self.current\_words[word]  
 self.health -= 1  
 self.damage\_flash\_alpha = 150  
 elif word == self.prompt\_content:  
 del self.current\_words[word]  
 self.score += len(word) \* 2  
 self.prompt\_content = ""  
 self.wordcomplete\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 if word == self.enemy.current\_word:  
 self.apply\_damage(1, word)  
 self.handle\_explosion\_effect(word\_rect)  
 elif word in self.bonus\_words:  
 if self.last\_bonus\_action == 'damage':  
 self.apply\_damage(3, word)  
 self.last\_bonus\_action = 'health'  
 self.handle\_explosion\_effect(word\_rect)  
 else:  
 self.health = min(self.health + 1.5, 50)  
 self.last\_bonus\_action = 'damage'  
 self.enemy.reset\_word(self.current\_words)  
 self.enemy.is\_hit = True  
 self.enemyhit\_sfx.play()  
 self.handle\_explosion\_effect(word\_rect)  
 else:  
 self.enemy.is\_hit = False  
  
 else:  
 word\_surf = self.create\_word\_surf(word, meta[2], hue, meta[3])  
 word\_rect = word\_surf.get\_rect(center=(meta[0], y))  
 enemy\_rect = self.enemy.sprite\_rect  
 if word\_rect.colliderect(enemy\_rect):  
 if enemy\_rect.left - word\_rect.width - 10 >= 0:  
 word\_rect.right = enemy\_rect.left - 10  
 else:  
 word\_rect.left = enemy\_rect.right + 10  
 self.SCREEN.blit(word\_surf, word\_rect)  
  
 if self.current\_words:  
 if self.enemy.update(timepassed, self.prompt\_content, self.current\_words):  
 if isinstance(self.enemy, Boss):  
 self.health -= 1.25 \* self.level # Boss deals near twice the damage  
 else:  
 self.health -= self.level # Minions deal damage based on the level  
 self.damage\_flash\_alpha = 150  
  
 if self.enemy.hitpoints <= 0:  
 self.win\_sfx.play()  
 pg.mixer.music.stop()  
 self.defeat\_display(self.enemy)  
 self.current\_enemy\_index += 1  
 break  
  
 self.enemy.draw()  
 self.SCREEN.blit(self.generate\_prompt\_surf(), (0, height - 50))  
 self.draw\_ui()  
 self.draw\_enemy\_hitpoints()  
  
 if self.damage\_flash\_alpha > 0:  
 flash\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 flash\_surf.fill((255, 0, 0, self.damage\_flash\_alpha))  
 self.SCREEN.blit(flash\_surf, (0, 0))  
 self.damage\_flash\_alpha = max(0, self.damage\_flash\_alpha - 8)  
  
 # Draw and manage explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions if  
 current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.SCREEN.blit(img, rect)  
  
 pg.display.flip()  
 hue = (hue + 1) % 360 # Update hue for the next frame  
  
 self.display\_victory()  
 from endings import Ending  
 ending = Ending(self.SCREEN)  
 ending.run()  
  
 def apply\_damage(self, damage, word, reset\_word=True, play\_sound=True):  
 self.enemy.hitpoints = max(0, self.enemy.hitpoints - ((damage \* len(word)) \* 0.35))  
 self.enemy.is\_hit = True  
 if reset\_word:  
 self.enemy.reset\_word(self.current\_words)  
 if play\_sound:  
 self.enemyhit\_sfx.play()  
  
 def handle\_explosion\_effect(self, word\_rect):  
 explosion\_image = pg.image.load(f'resources/transparent/boom-{random.randint(1, 3)}.gif').convert\_alpha()  
 scale\_factor = 0.20 # Adjust this factor to make the explosion image larger  
 new\_width = int(explosion\_image.get\_width() \* scale\_factor)  
 new\_height = int(explosion\_image.get\_height() \* scale\_factor)  
 explosion\_image = pg.transform.scale(explosion\_image, (new\_width, new\_height))  
 explosion\_rect = explosion\_image.get\_rect(center=word\_rect.center)  
 self.explosions.append((explosion\_image, explosion\_rect, pg.time.get\_ticks()))  
  
 def maam\_emmy\_display(self):  
 fade\_alpha = 0  
 fade\_increment = 5 # Adjust this value to control the speed of the fade-in effect  
  
 pg.mixer.music.load(self.mamemmy\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 return # Exit the display and start the battle  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the enemy sprite talking  
 talk\_sprite = pg.image.load("resources/sprites/mamemmy.png").convert\_alpha()  
 # Scale down the sprite to fit the screen  
 sprite\_scale\_factor = 0.3 # Adjust this factor as needed  
 talk\_sprite = pg.transform.scale(talk\_sprite, (int(talk\_sprite.get\_width() \* sprite\_scale\_factor),  
 int(talk\_sprite.get\_height() \* sprite\_scale\_factor)))  
 talk\_sprite.set\_alpha(fade\_alpha)  
 talk\_sprite\_rect = talk\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(talk\_sprite, talk\_sprite\_rect)  
  
 # Draw the dialogue box with shadow  
 dialogue\_text = ("\"If you quit right now, you can't have anything you want. "  
 "So do your best\"")  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 22)  
 dlg\_surf = small\_font.render(dialogue\_text, True, pg.Color("white"))  
 dlg\_shadow = small\_font.render(dialogue\_text, True, pg.Color("black"))  
  
 box\_width = int(dlg\_surf.get\_width() \* 1.5) + 5  
 box\_height = int(dlg\_surf.get\_height() \* 1.5) + 5  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (self.SCREEN.get\_width() // 2 - box\_width // 2, self.SCREEN.get\_height() // 2 - box\_height // 2),  
 (self.SCREEN.get\_width() // 2 + box\_width // 2,  
 self.SCREEN.get\_height() // 2 - box\_height // 2 - offset),  
 (self.SCREEN.get\_width() // 2 + box\_width // 2,  
 self.SCREEN.get\_height() // 2 + box\_height // 2 - offset),  
 (self.SCREEN.get\_width() // 2 - box\_width // 2, self.SCREEN.get\_height() // 2 + box\_height // 2)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (114, 141, 17, fade\_alpha), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (32, 122, 19, fade\_alpha), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, box\_width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
 dlg\_shadow = pg.transform.rotate(dlg\_shadow, angle)  
  
 # Blit the shadow text first, then the main text  
 dlg\_box\_rect = pg.Rect(self.SCREEN.get\_width() // 2 - box\_width // 2,  
 self.SCREEN.get\_height() // 2 - box\_height // 2, box\_width, box\_height)  
 dlg\_shadow.set\_alpha(fade\_alpha)  
 dlg\_surf.set\_alpha(fade\_alpha)  
 self.SCREEN.blit(dlg\_shadow, dlg\_surf.get\_rect(center=dlg\_box\_rect.center).move(2, 2))  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_box\_rect.center))  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (32, 122, 19)  
 shadow\_color = (0, 0, 0)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar.set\_alpha(fade\_alpha)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 top\_bar\_shadow.set\_alpha(fade\_alpha)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar.set\_alpha(fade\_alpha)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 bottom\_bar\_shadow.set\_alpha(fade\_alpha)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 # Draw the prompt to continue  
 prompt\_text = "Press Enter to go to the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2 + 300))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def before\_battle\_display(self, minion):  
 fade\_duration = 1.0 # Duration of the fade-in effect in seconds  
 fade\_alpha = 0 # Initial alpha value for fade-in effect  
 fade\_increment = 255 / (fade\_duration \* 60) # Increment per frame (assuming 60 FPS)  
  
 pg.mixer.music.load(self.prebattle\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 pg.mixer.music.stop()  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 pg.mixer.music.stop()  
 return # Exit the display and start the battle  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the minion sprite talking with fade-in effect  
 talk\_sprite = minion.talk\_sprite.copy()  
 talk\_sprite.set\_alpha(fade\_alpha)  
 talk\_sprite\_rect = talk\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(talk\_sprite, talk\_sprite\_rect)  
  
 # Adjust font size based on the length of the dialogue text  
 dialogue\_text = minion.dialogue\_text  
 words = dialogue\_text.split()  
 font\_size = 20 if len(words) <= 7 else 15  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", font\_size)  
 dlg\_surf = small\_font.render(dialogue\_text, True, pg.Color(250, 250, 250))  
 dlg\_surf.set\_alpha(fade\_alpha)  
  
 # Calculate the bounding rectangle of the text surface and add padding  
 padding = 10  
 dlg\_rect = pg.Rect(  
 self.SCREEN.get\_width() // 2 - dlg\_surf.get\_width() // 2 - padding,  
 self.SCREEN.get\_height() // 2 - dlg\_surf.get\_height() // 2 - padding,  
 dlg\_surf.get\_width() + padding \* 2,  
 dlg\_surf.get\_height() + padding \* 2  
 )  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
  
 # Blit the text surface centered at its position  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_rect.center))  
  
 # Draw the top bar with fade-in effect  
 top\_bar\_height = 50  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), top\_bar\_height), pg.SRCALPHA)  
 top\_bar.fill((167, 57, 57, fade\_alpha))  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), top\_bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill((125, 28, 28, fade\_alpha))  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Draw the bottom bar with fade-in effect  
 bottom\_bar\_height = 100  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bottom\_bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill((167, 57, 57, fade\_alpha))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bottom\_bar\_height))  
  
 # Draw the prompt to continue with fade-in effect  
 prompt\_text = "Press Enter to start the battle"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - bottom\_bar\_height // 2))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Apply fade-in effect  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def defeat\_display(self, minion):  
 fade\_duration = 1.0  
 fade\_alpha = 0  
 fade\_increment = 255 / (fade\_duration \* 60)  
  
 pg.mixer.music.load(self.victory\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.main\_Menu()  
 return  
 elif event.key == pg.K\_RETURN:  
 pg.mixer.music.stop()  
 return  
  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 defeat\_sprite = minion.defeat\_sprite.copy()  
 defeat\_sprite.set\_alpha(fade\_alpha)  
 defeat\_sprite\_rect = defeat\_sprite.get\_rect(  
 center=(self.SCREEN.get\_width() - 250, self.SCREEN.get\_height() // 2))  
 self.SCREEN.blit(defeat\_sprite, defeat\_sprite\_rect)  
  
 small\_font = pg.font.Font("resources/DejaVuSans.ttf", 20)  
 dlg\_surf = small\_font.render(minion.defeat\_text, True, pg.Color(250, 250, 250))  
 dlg\_surf.set\_alpha(fade\_alpha)  
  
 # Calculate the bounding rectangle of the text surface and add padding  
 padding = 10  
 dlg\_rect = pg.Rect(  
 0, 0,  
 dlg\_surf.get\_width() + padding \* 2,  
 dlg\_surf.get\_height() + padding \* 2  
 )  
 dlg\_rect.centerx = self.SCREEN.get\_width() // 2  
 dlg\_rect.centery = defeat\_sprite\_rect.centery  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Rotate the text surface to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 dlg\_surf = pg.transform.rotate(dlg\_surf, angle)  
  
 # Blit the text surface centered at its position  
 self.SCREEN.blit(dlg\_surf, dlg\_surf.get\_rect(center=dlg\_rect.center))  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (167, 57, 57)  
 shadow\_color = (125, 28, 28)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar\_y = self.SCREEN.get\_height() - bar\_height  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, bottom\_bar\_y))  
 self.SCREEN.blit(bottom\_bar, (0, bottom\_bar\_y))  
  
 prompt\_text = "Press Enter to continue"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_surf\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
 prompt\_surf.set\_alpha(fade\_alpha)  
 prompt\_surf\_shadow.set\_alpha(fade\_alpha)  
 prompt\_rect = prompt\_surf.get\_rect(  
 center=(self.SCREEN.get\_width() // 2, bottom\_bar\_y + bar\_height // 2))  
 self.SCREEN.blit(prompt\_surf\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 if fade\_alpha < 255:  
 fade\_alpha = min(255, fade\_alpha + fade\_increment)  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def display\_victory(self):  
 if self.score > self.highscore:  
 self.highscore = self.score  
 write\_score(self.highscore)  
  
 pg.mixer.music.load(self.victory\_music)  
 pg.mixer.music.play(-1)  
  
 while True:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 sys.exit()  
 elif event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 from mapuantypingmania import GameMenu  
 game = GameMenu()  
 game.play()  
 else:  
 LoadingScreen(self.SCREEN).run()  
 from introduction import Stage1Outro  
 outro = Stage1Outro(self.SCREEN)  
 outro.run()  
  
 # Prepare text surfaces and their positions  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 victory\_surf = self.font.render("VICTORY!", True, pg.Color("white"))  
 victory\_shadow = self.font.render("VICTORY!", True, pg.Color("black"))  
 highscore\_text = f"Highscore: {self.highscore}"  
 highscore\_surf = self.font.render(highscore\_text, True, pg.Color("white"))  
 highscore\_shadow = self.font.render(highscore\_text, True, pg.Color("black"))  
 prompt\_text = "Press any key for next stage, or Esc for main menu"  
 prompt\_surf = self.font.render(prompt\_text, True, pg.Color("white"))  
 prompt\_shadow = self.font.render(prompt\_text, True, pg.Color("black"))  
  
 victory\_rect = victory\_surf.get\_rect(center=(center[0], center[1] - 40))  
 hs\_rect = highscore\_surf.get\_rect(center=center)  
 prompt\_rect = prompt\_surf.get\_rect(center=(center[0], center[1] + 40))  
  
 # Calculate the bounding rectangle of all text surfaces and add padding  
 union\_rect = victory\_rect.union(hs\_rect).union(prompt\_rect)  
 padding = 10  
 dlg\_rect = pg.Rect(  
 union\_rect.left - padding,  
 union\_rect.top - padding,  
 union\_rect.width + 6 \* padding,  
 union\_rect.height + 6 \* padding  
 )  
  
 # Center the dialog box on the screen  
 dlg\_rect.center = center  
  
 # Define the points for the parallelogram shape  
 offset = 10  
 box\_points = [  
 (dlg\_rect.left, dlg\_rect.top),  
 (dlg\_rect.right, dlg\_rect.top - offset),  
 (dlg\_rect.right, dlg\_rect.bottom - offset),  
 (dlg\_rect.left, dlg\_rect.bottom)  
 ]  
  
 shadow\_points = [(x + 5, y + 5) for x, y in box\_points]  
  
 # Create the dialog box surface with an opaque yellow red color  
 dlg\_box = pg.Surface((dlg\_rect.width, dlg\_rect.height))  
 dlg\_box.fill((255, 193, 33))  
  
 # Draw background and dialog box  
 self.SCREEN.blit(self.BG, (0, 0))  
  
 # Draw the shadow first  
 pg.draw.polygon(self.SCREEN, (255, 204, 0, 150), shadow\_points)  
  
 # Draw the main box  
 pg.draw.polygon(self.SCREEN, (26, 62, 112), box\_points)  
  
 # Draw border as a parallelogram  
 border\_padding = 5  
 border\_points = [  
 (dlg\_rect.left + border\_padding, dlg\_rect.top + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.top - offset + border\_padding),  
 (dlg\_rect.right - border\_padding, dlg\_rect.bottom - offset - border\_padding),  
 (dlg\_rect.left + border\_padding, dlg\_rect.bottom - border\_padding)  
 ]  
 pg.draw.polygon(self.SCREEN, (211, 200, 74), border\_points, 3)  
  
 # Rotate the text surfaces to match the angle of the parallelogram  
 angle = math.degrees(math.atan2(offset, dlg\_rect.width))  
 victory\_surf = pg.transform.rotate(victory\_surf, angle)  
 victory\_shadow = pg.transform.rotate(victory\_shadow, angle)  
 highscore\_surf = pg.transform.rotate(highscore\_surf, angle)  
 highscore\_shadow = pg.transform.rotate(highscore\_shadow, angle)  
 prompt\_surf = pg.transform.rotate(prompt\_surf, angle)  
 prompt\_shadow = pg.transform.rotate(prompt\_shadow, angle)  
  
 # Blit each text surface centered at their respective positions  
 self.SCREEN.blit(victory\_shadow, victory\_rect.move(2, 2))  
 self.SCREEN.blit(victory\_surf, victory\_rect)  
 self.SCREEN.blit(highscore\_shadow, hs\_rect.move(2, 2))  
 self.SCREEN.blit(highscore\_surf, hs\_rect)  
 self.SCREEN.blit(prompt\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_surf, prompt\_rect)  
  
 # Draw the top and bottom bars with shadows  
 bar\_height = 50  
 bar\_color = (167, 57, 57)  
 shadow\_color = (125, 28, 28)  
  
 # Top bar  
 top\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar.fill(bar\_color)  
 top\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 top\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(top\_bar\_shadow, (0, 0))  
 self.SCREEN.blit(top\_bar, (0, 0))  
  
 # Bottom bar  
 bottom\_bar = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar.fill(bar\_color)  
 bottom\_bar\_shadow = pg.Surface((self.SCREEN.get\_width(), bar\_height), pg.SRCALPHA)  
 bottom\_bar\_shadow.fill(shadow\_color)  
 self.SCREEN.blit(bottom\_bar\_shadow, (0, self.SCREEN.get\_height() - bar\_height))  
 self.SCREEN.blit(bottom\_bar, (0, self.SCREEN.get\_height() - bar\_height))  
  
 pg.display.flip()  
 self.clock.tick(60)  
  
 def rainbow(self, hue):  
 color = pg.Color("white")  
 hue = (hue + 1) % 360  
 color.hsva = (hue, 100, 100, 100)  
 return color  
  
 def add\_word(self, width, words, word\_type, enemy):  
 found\_word = False  
 while not found\_word and len(self.current\_words) < len(words):  
 if word\_type == 'bonus' and self.bonus\_word\_counter >= 5:  
 selected = random.choice(self.bonus\_words)  
 self.bonus\_word\_counter = 0 # Reset counter after adding bonus word  
 # For normal words  
 else:  
 # Adjust selection logic to balance word lengths  
 word\_lengths = [len(word) for word in words]  
 current\_lengths = [len(word) for word in self.current\_words.keys()]  
 length\_counts = {length: current\_lengths.count(length) for length in set(word\_lengths)}  
  
 # Calculate weights to balance word lengths  
 weights = []  
 for length in word\_lengths:  
 if length\_counts.get(length, 0) < 2: # Prefer lengths not yet on screen  
 weights.append(1)  
 else:  
 weights.append(0.1)  
  
 selected = random.choices(words, weights=weights, k=1)[0]  
 if word\_type == 'stage3': # Only increment for normal words  
 self.bonus\_word\_counter += 1  
  
 # Skip if word is already on screen or starts with same letter  
 if selected not in self.current\_words and \  
 all(not w.startswith(selected[0]) for w in self.current\_words):  
 if selected not in self.word\_widths:  
 self.word\_widths[selected] = self.font.size(selected)[0]  
 w\_width = self.word\_widths[selected]  
 x = random.randrange(45, width - w\_width - 10)  
  
 # Check for overlaps  
 if not (enemy.sprite\_rect.left < x < enemy.sprite\_rect.right) and \  
 all(abs(x - meta[0]) > w\_width + 15 for meta in self.current\_words.values()):  
 self.current\_words[selected] = [x, 0, (150, 150, 150), word\_type]  
 found\_word = True  
  
 # Adjust word frequency based on word type  
 if word\_type == 'bonus':  
 self.word\_frequency = max(2.0, self.word\_frequency - 0.1)  
 else:  
 self.word\_frequency = min(5.0, self.word\_frequency + 0.1)  
  
 def create\_word\_surf(self, word, color, hue, word\_type):  
 w, h = self.font.size(word)  
 w += 12 # Increase width for padding  
 h += 12 # Increase height for padding  
 Surf = pg.Surface((w, h), pg.SRCALPHA, 32)  
  
 pg.draw.rect(Surf, (222, 153, 0, 200), Surf.get\_rect(), border\_radius=10)  
  
 being\_written = self.prompt\_content and word.startswith(self.prompt\_content)  
 start\_text = self.prompt\_content if being\_written else ''  
 end\_text = word[len(self.prompt\_content):] if being\_written else word  
 start\_surf = self.font.render(start\_text, True, pg.Color("black"))  
  
 # Set constant colors for bonus and bossfight word types  
 if word in self.bonus\_words:  
 transformed\_color = pg.Color("gold")  
 # print("bonus")  
 else:  
 transformed\_color = self.rainbow(hue)  
 # print("normal")  
  
 end\_surf = self.font.render(end\_text, True, transformed\_color)  
 Surf.blit(start\_surf, (8, 8))  
 Surf.blit(end\_surf, end\_surf.get\_rect(right=w - 8, centery=h // 2))  
 return Surf  
  
 def generate\_prompt\_surf(self):  
 width = self.SCREEN.get\_width()  
 surf = pg.Surface((width, 50), pg.SRCALPHA)  
 shadow\_surf = pg.Surface((width, 10), pg.SRCALPHA)  
  
 # Create shadow  
 shadow\_surf.fill((167, 57, 57, 79))  
 surf.fill((125, 28, 35))  
 surf.set\_alpha(255)  
  
 self.SCREEN.blit(surf, (0, 0))  
 surf.blit(shadow\_surf, (0, -1))  
  
 color = pg.Color("#ff6600") if any(w.startswith(self.prompt\_content) for w in self.current\_words) else pg.Color(  
 "#ffffff")  
 rendered = self.font.render(self.prompt\_content, True, color)  
  
 # Create shadow text  
 shadow\_rendered = self.font.render(self.prompt\_content, True, pg.Color("black"))  
  
 # Center the prompt text horizontally on the surface  
 rect = rendered.get\_rect(centerx=width // 2, centery=25)  
 shadow\_rect = shadow\_rendered.get\_rect(centerx=width // 2 - 2, centery=25 - 2) # Offset for shadow effect  
  
 # Blit shadow first, then main text  
 surf.blit(shadow\_rendered, shadow\_rect)  
 surf.blit(rendered, rect)  
  
 # Draw a bar to indicate the position  
 bar\_width = 2  
 bar\_height = 40  
 bar\_x = rect.right + 5  
 bar\_y = 5  
 pg.draw.rect(surf, pg.Color("red"), (bar\_x, bar\_y, bar\_width, bar\_height))  
  
 return surf  
  
 def draw\_enemy\_hitpoints(self):  
 hp\_text = f"Enemy HP: {self.enemy.hitpoints:.1f}"  
 hp\_text\_shadow = self.font.render(hp\_text, True, pg.Color("black"))  
 hp\_surf = self.font.render(hp\_text, True, (255, 255, 255))  
 hp\_box = pg.Surface((hp\_surf.get\_width() + 10, hp\_surf.get\_height() + 10), pg.SRCALPHA)  
 hp\_box.fill((26, 62, 112, 190))  
  
 # Initialize and update fade alpha for enemy hitpoints  
 if not hasattr(self, 'hp\_alpha'):  
 self.hp\_alpha = 0  
 if self.hp\_alpha < 255:  
 self.hp\_alpha += 5 # Adjust increment as needed for smoother or faster fade  
 hp\_box.set\_alpha(self.hp\_alpha)  
  
 hp\_box\_rect = hp\_box.get\_rect(midtop=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 100))  
  
 # Create shadow of box  
 shadow\_offset = 2  
 shadow\_box = pg.Surface((hp\_box.get\_width(), hp\_box.get\_height()), pg.SRCALPHA)  
 shadow\_box.fill((224, 180, 0, 100)) # Darker color for shadow  
 shadow\_box\_rect = hp\_box\_rect.move(shadow\_offset, shadow\_offset)  
  
 # Blit shadow first, then the hitpoint box  
 self.SCREEN.blit(shadow\_box, shadow\_box\_rect)  
 self.SCREEN.blit(hp\_text\_shadow, hp\_box\_rect.move(2,2))  
 self.SCREEN.blit(hp\_box, hp\_box\_rect)  
 self.SCREEN.blit(hp\_surf, hp\_surf.get\_rect(center=hp\_box\_rect.center))  
  
 def draw\_ui(self):  
 top\_box = pg.Surface((self.SCREEN.get\_width(), 40), pg.SRCALPHA)  
 top\_box.fill((54, 54, 54, 200)) # Adjusted background color with opacity  
 top\_box\_rect = top\_box.get\_rect()  
 if not hasattr(self, 'ui\_alpha'):  
 self.ui\_alpha = 0  
  
 if self.ui\_alpha < 255:  
 self.ui\_alpha += 1 # Adjust the increment value as needed  
  
 top\_box.set\_alpha(self.ui\_alpha)  
 self.SCREEN.blit(top\_box, top\_box\_rect)  
  
 # Render the main text and its shadow  
 score\_surf = self.font.render(f"Score: {self.score}", True, (255, 255, 255))  
 health\_surf = self.font.render(f"Health: {self.health}", True, (255, 255, 255))  
 enemy\_name = self.font.render(f"Enemy: {self.enemy.name}", True, (255, 255, 255))  
 score\_shadow = self.font.render(f"Score: {self.score}", True, (0, 0, 0))  
 health\_shadow = self.font.render(f"Health: {self.health}", True, (0, 0, 0))  
 enemy\_shadow = self.font.render(f"Enemy: {self.enemy.name}", True, (0, 0, 0))  
  
 # Calculate positions for the text  
 screen\_width = self.SCREEN.get\_width()  
 score\_pos = (10, 10)  
 health\_pos = (screen\_width // 3, 10)  
 enemy\_pos = (2 \* screen\_width // 3, 10)  
  
 # Offset for the shadow effect  
 shadow\_offset = (2, 2)  
  
 # Blit the shadow first, then the main text  
 self.SCREEN.blit(score\_shadow, (score\_pos[0] + shadow\_offset[0], score\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(health\_shadow, (health\_pos[0] + shadow\_offset[0], health\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(enemy\_shadow, (enemy\_pos[0] + shadow\_offset[0], enemy\_pos[1] + shadow\_offset[1]))  
 self.SCREEN.blit(score\_surf, score\_pos)  
 self.SCREEN.blit(health\_surf, health\_pos)  
 self.SCREEN.blit(enemy\_name, enemy\_pos)  
  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (screen\_width // 3 - 5, 0),  
 (screen\_width // 3 - 5, 40), 2)  
 pg.draw.line(self.SCREEN, (255, 255, 255),  
 (2 \* screen\_width // 3 - 5, 0),  
 (2 \* screen\_width // 3 - 5, 40), 2)  
  
 def display\_game\_over(self):  
 write\_score(self.score)  
 game\_over = self.font.render("GAME OVER", True, (255, 0, 0))  
 center = (self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() // 2)  
 self.SCREEN.blit(game\_over, game\_over.get\_rect(center=center))  
 pg.display.flip()  
 pg.time.wait(2000)  
  
 def apply\_fade\_effect(self):  
 if self.fade\_direction != 0:  
 self.fade\_alpha += self.fade\_direction \* 10  
 if self.fade\_alpha >= 255:  
 self.fade\_alpha = 255  
 self.fade\_direction = 0  
 elif self.fade\_alpha <= 0:  
 self.fade\_alpha = 0  
 self.fade\_direction = 0  
 fade\_surf = pg.Surface(self.SCREEN.get\_size(), pg.SRCALPHA)  
 fade\_surf.fill((255, 0, 0, self.fade\_alpha))  
 self.SCREEN.blit(fade\_surf, (0, 0))  
  
class Stage3Enemies:  
 def \_\_init\_\_(self, screen, level, normal\_sprite\_path, hit\_sprite\_path):  
 self.screen = screen  
 self.width, self.height = self.screen.get\_size()  
 self.font = pg.font.Font("resources/DejaVuSans.ttf", 36)  
 self.hitpoints = 25 + level \* 5  
 self.word\_speed = 1  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
 self.is\_hit = False  
 self.sprite\_alpha = 0  
  
 self.normal\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.hit\_sprite = pg.image.load(hit\_sprite\_path).convert\_alpha()  
 self.talk\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.defeat\_sprite = pg.image.load(normal\_sprite\_path).convert\_alpha()  
 self.normal\_sprite = pg.transform.scale(self.normal\_sprite, (300, 500))  
 self.hit\_sprite = pg.transform.scale(self.hit\_sprite, (300, 500))  
 self.talk\_sprite = pg.transform.scale(self.talk\_sprite, (300, 500))  
 self.defeat\_sprite = pg.transform.scale(self.defeat\_sprite, (300, 500))  
 self.sprite\_rect = self.normal\_sprite.get\_rect()  
 self.sprite\_rect.centerx = self.width - 250  
 self.sprite\_rect.centery = self.height - 300 # Adjusted to align with the prompt surf  
 self.word\_bg\_image = pg.image.load("resources/transparent/ice.gif").convert\_alpha()  
 self.explosions = []  
  
 def reset\_word(self, current\_words):  
 if self.current\_word in current\_words:  
 del current\_words[self.current\_word]  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 def update(self, timepassed, player\_input, current\_words):  
 if self.sprite\_alpha < 255:  
 self.sprite\_alpha += 5  
  
 if self.hitpoints <= 0:  
 return False  
  
 if not self.current\_word and current\_words:  
 self.current\_word = random.choice(list(current\_words.keys()))  
 self.word\_progress = 0  
  
 if self.current\_word and (self.current\_word not in current\_words):  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
 if self.start\_timer > 0:  
 self.start\_timer -= timepassed  
 return False  
  
 if self.current\_word:  
 self.word\_progress += timepassed \* self.word\_speed  
 meta = current\_words[self.current\_word]  
 # Use the updated meta data for y-position  
 word\_x = meta[0]  
 meta\_y = meta[1]  
 y = (meta\_y \* self.word\_speed) + abs(math.cos(meta\_y \* 3) \* 10)  
 word\_rect = pg.Rect(word\_x, y, self.font.size(self.current\_word)[0],  
 self.font.size(self.current\_word)[1])  
 if self.word\_progress >= len(self.current\_word):  
 # Store the completed word before resetting  
 completed\_word = self.current\_word  
 handle\_explosion\_effect(self.screen, self.font, self.sprite\_rect, completed\_word, self.explosions)  
 if self.current\_word in current\_words:  
 current\_words.pop(self.current\_word)  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.0  
 return True  
  
 return False  
  
 def get\_font\_size(self, word\_length):  
 if word\_length > 5:  
 return 24 # Smaller font size for words longer than 5 letters  
 else:  
 return 28 # Default font size  
  
 def draw(self):  
 if self.hitpoints <= 0:  
 current\_sprite = self.defeat\_sprite  
 else:  
 current\_sprite = self.hit\_sprite if self.is\_hit else self.normal\_sprite  
  
 sprite\_with\_alpha = current\_sprite.copy()  
 sprite\_with\_alpha.set\_alpha(self.sprite\_alpha)  
 self.screen.blit(sprite\_with\_alpha, self.sprite\_rect)  
  
 if self.hitpoints > 0 and self.current\_word:  
 # Render the typed and remaining portions of the word  
 typed = self.current\_word[:int(self.word\_progress)]  
 remaining = self.current\_word[int(self.word\_progress):]  
  
 # Get appropriate font size based on word length  
 font\_size = self.get\_font\_size(len(self.current\_word))  
 font = pg.font.Font("resources/DejaVuSans.ttf", font\_size)  
  
 typed\_surf = font.render(typed, True, (255, 0, 0))  
 remaining\_surf = font.render(remaining, True, (100, 100, 100))  
  
 total\_width = typed\_surf.get\_width() + remaining\_surf.get\_width()  
 text\_height = typed\_surf.get\_height()  
  
 # Define the text box size based on the text dimensions with extra margin  
 box\_width = int(total\_width \* 1.5) + 20  
 box\_height = int(text\_height \* 1.5) + 10  
  
 # Scale the background image for the word box  
 word\_bg\_image\_scaled = pg.transform.scale(self.word\_bg\_image, (box\_width, box\_height))  
  
 # Position the text box with a negative x-coordinate to overlay over the sprite  
 word\_box\_rect = word\_bg\_image\_scaled.get\_rect(  
 midright=(self.sprite\_rect.left - 20, self.sprite\_rect.centery))  
 word\_box\_rect.x += 100 # Adjust this value as needed to overlay the text box  
  
 # Calculate centered text position within the text box  
 text\_x = word\_box\_rect.left + (box\_width - total\_width) // 2  
 text\_y = word\_box\_rect.top + (box\_height - text\_height) // 2  
  
 # Blit the text box and then the text centered in it  
 self.screen.blit(word\_bg\_image\_scaled, word\_box\_rect)  
 self.screen.blit(typed\_surf, (text\_x, text\_y))  
 self.screen.blit(remaining\_surf, (text\_x + typed\_surf.get\_width(), text\_y))  
  
 # Draw any active explosions  
 current\_time = pg.time.get\_ticks()  
 self.explosions = [(img, rect, start\_time) for img, rect, start\_time in self.explosions  
 if current\_time - start\_time < 500]  
 for img, rect, \_ in self.explosions:  
 self.screen.blit(img, rect)  
  
 def draw\_before\_battle(self):  
 self.screen.blit(self.normal\_sprite, self.sprite\_rect)  
  
class Minion1SThree(Stage3Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/Red-1.png", "resources/sprites/Red-1-hit.gif")  
 self.dialogue\_text = "\"Prepare yourself for the battle!\""  
 self.defeat\_text = "\"Congrats!\""  
 self.word\_speed = 1.6  
 self.name = "Finalist Havier"  
  
class Minion2SThree(Stage3Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/Red-2.png",  
 "resources/sprites/Red-2-hit.gif")  
 self.dialogue\_text = "\"Prepare yourself for the battle!\""  
 self.defeat\_text = "\"Congrats!\""  
 self.word\_speed = 1.8  
 self.name = "Finalist Vade"  
  
class Minion3SThree(Stage3Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/Red-3.png",  
 "resources/sprites/Red-3-hit.gif")  
 self.dialogue\_text = "\"Prepare yourself for the battle!\""  
 self.defeat\_text = "\"Congrats!\""  
 self.word\_speed = 2.0  
 self.name = "Finalist Kris"  
  
class BossSThree(Stage3Enemies):  
 def \_\_init\_\_(self, screen, level):  
 super().\_\_init\_\_(screen, level, "resources/sprites/ice-fight.png", "resources/sprites/ice-hit-color.gif")  
 self.defeat\_sprite = pg.image.load("resources/sprites/ice-defeat.png").convert\_alpha()  
 self.talk\_sprite = pg.image.load("resources/sprites/ice-talk.png").convert\_alpha()  
 self.normal\_sprite = pg.transform.smoothscale(self.normal\_sprite, (450, 650))  
 self.hit\_sprite = pg.transform.smoothscale(self.hit\_sprite, (450, 650))  
 self.talk\_sprite = pg.transform.smoothscale(self.talk\_sprite, (450, 650))  
 self.defeat\_sprite = pg.transform.smoothscale(self.defeat\_sprite, (450, 650))  
 self.dialogue\_text = ("\"Let's see who is cooler, Me or You?"  
 "This game will show it all so give it your best!\"")  
 self.defeat\_text = "\"Damn I feel cold! You are too cool that I am freezing!\""  
 self.hitpoints = 50 + level \* 10 # Boss has more hitpoints  
 self.max\_health = self.hitpoints # Store the initial maximum health  
 self.word\_speed = 4.4 # Boss has a faster word speed  
 self.sprite\_rect.centery = self.height // 2 # Adjusted to align with the prompt surf  
 self.name = "THE ICE KING"  
  
 def get\_max\_health(self):  
 return self.max\_health  
  
 def reset\_word(self, current\_words):  
 self.current\_word = ""  
 self.word\_progress = 0  
 self.start\_timer = 2.5  
  
  
  
"""STAGE 3 END----------------------------------------------------------------------------------------------------------"""

# OPTIONAL.PY

import pygame as pg  
import random  
from mapuantypingmania import apply\_wave\_effect  
import numpy as np  
  
class Leaderboard:  
 def \_\_init\_\_(self, screen):  
 self.SCREEN = screen  
 width, height = self.SCREEN.get\_size()  
 self.font = pg.font.Font(None, 30)  
 self.title\_font = pg.font.Font(None, 40)  
 self.prompt\_font = pg.font.Font(None, 25)  
 self.background\_path = "resources/backgrounds/menu.jpg"  
 self.scorefile\_path = "resources/highscore.txt"  
 self.max\_leaders = 6  
 try:  
 self.background = pg.image.load(self.background\_path)  
 self.background = pg.transform.scale(self.background, (width, height))  
 except Exception as e:  
 print(f"Error loading background: {e}")  
 self.background = pg.Surface((width, height))  
 self.phase = 0  
  
 def quick\_sort(self, arr):  
 if len(arr) <= 1:  
 return arr  
 pivot = arr[0]  
 left = [item for item in arr[1:] if item[1] > pivot[1]]  
 right = [item for item in arr[1:] if item[1] <= pivot[1]]  
 return self.quick\_sort(left) + [pivot] + self.quick\_sort(right)  
  
 def load\_scores(self):  
 try:  
 with open(self.scorefile\_path, "r") as file:  
 scores = {}  
 for line in file.readlines():  
 if ':' in line:  
 name, score\_str = line.strip().split(":", 1)  
 try:  
 score = int(score\_str.strip())  
 except ValueError:  
 score = 0  
 scores[name.strip()] = score  
 items = list(scores.items())  
 sorted\_scores = self.quick\_sort(items)  
 return sorted\_scores[:self.max\_leaders]  
 except IOError:  
 return []  
  
 def animate\_background(self):  
 amplitude = 5  
 frequency = 0.01  
 color\_shift = 50  
 self.phase += 0.05  
  
 # Calculate color transition based on phase  
 t = (np.sin(self.phase) + 1) / 2  
 r = int(255 \* (1 - t) + 128 \* t)  
 g = int(200 \* (1 - t) + 128 \* t)  
 b = int(100 \* (1 - t) + 128 \* t)  
 bg\_color = (r, g, b)  
  
 try:  
 # Apply wave effect to the background image  
 wavy\_bg = apply\_wave\_effect(self.background.copy(), amplitude, frequency, self.phase, color\_shift)  
 wavy\_bg.fill(bg\_color, special\_flags=pg.BLEND\_RGBA\_MULT)  
 except Exception as e:  
 print(f"Error animating background: {e}")  
 return self.background # Return the original background in case of error  
  
 return wavy\_bg  
  
 def run(self):  
 clock = pg.time.Clock()  
 while True:  
 animated\_bg = self.animate\_background()  
 try:  
 for event in pg.event.get():  
 if event.type == pg.QUIT:  
 pg.quit()  
 return  
 if event.type == pg.KEYDOWN:  
 if event.key == pg.K\_ESCAPE:  
 return  
  
 self.SCREEN.blit(animated\_bg, (0, 0))  
  
 # Calculate the vertical starting position as 1/4 of the screen's height  
 quarter\_height = self.SCREEN.get\_height() // 4  
  
 # Draw centered title at 1/4 down the screen  
 title\_text = self.title\_font.render("HIGHSCORES:", True, (255, 215, 0))  
 title\_shadow = self.title\_font.render("HIGHSCORES:", True, (0, 0, 0))  
 title\_rect = title\_text.get\_rect(center=(self.SCREEN.get\_width() // 2, quarter\_height))  
 self.SCREEN.blit(title\_shadow, title\_rect.move(2, 2))  
 self.SCREEN.blit(title\_text, title\_rect)  
  
 # Define an offset so that the scores start a few pixels below the title  
 score\_start\_y = quarter\_height + 50  
  
 # Draw centered scores starting below the title  
 scores = self.load\_scores()  
 for i, (name, score) in enumerate(scores):  
 line = f"{name} {score}"  
 text = self.font.render(line, True, (255, 255, 255))  
 text\_shadow = self.font.render(line, True, (0, 0, 0))  
 text\_rect = text.get\_rect(center=(self.SCREEN.get\_width() // 2, score\_start\_y + i \* 35))  
 self.SCREEN.blit(text\_shadow, text\_rect.move(2, 2))  
 self.SCREEN.blit(text, text\_rect)  
  
 # Draw shadowed prompt centered at bottom  
 prompt = "Press ESC to go back to main menu"  
 prompt\_shadow = self.prompt\_font.render(prompt, True, (0, 0, 0))  
 prompt\_text = self.prompt\_font.render(prompt, True, (255, 255, 255))  
 prompt\_rect = prompt\_text.get\_rect(center=(self.SCREEN.get\_width() // 2, self.SCREEN.get\_height() - 30))  
 self.SCREEN.blit(prompt\_shadow, prompt\_rect.move(2, 2))  
 self.SCREEN.blit(prompt\_text, prompt\_rect)  
  
 pg.display.flip()  
 clock.tick(30)  
 except Exception as e:  
 print(f"Error running Leaderboard: {e}")

# CONCLUSION AND REALIZATION

While our group was developing this game, it an overall, a rewarding experience. From idea to implementation (our scope was so much), we also gained valuable insights into game development, programming, and problem-solving. This report summarizes our key learnings and experiences.

**Key Learnings**

1. **Game Development with Python**
   * Learned to use Python and Pygame to build a game, using pygame as the foundation.
   * Understood game design concepts like user input, rendering, and game loops, game loops is a core design that should be better prioritized than most stuff in the game.
2. **Challenges in Development**
   * Transforming ideas into a working game was challenging, having it visioned and then trying to implement it was nerve-wracking and sanity reducing.
   * Debugging and optimizing code required patience and persistence, with that much code who wouldn’t need those two traits.
3. **Coding and Problem-Solving**
   * Explored different coding approaches to achieve the same result, coding is fun in a way that u can discover many things to do one thing.
   * Improved debugging and logical thinking skills, using printouts and logically thinking the flow of code and the game was a game changer for of all us.
4. **Skill Development**
   * Enhanced problem-solving, patience, and creativity, with doing stuff comes gaining stuff and so we gained some of those traits and skills while doing this game.
   * Gained experience in structuring and managing game projects, who knew it would be a massive undertaking even for one simple game.
5. **Appreciation for Game Development**
   * Realized the effort needed to create engaging games, ideas and vision and imagination along with skills needs to come together for this.
   * Learned the importance of planning, iteration, and continuous learning, never stop reading and solving problems.

In conclusion, this project was a fun and educational journey for our. We developed a game using Python, improved problem-solving skills, and gained a deeper appreciation for game development. Despite challenges, it was a valuable learning experience and GREAT TIME FOR ALL.

# REFERENCES

List all the sources you have used to make this project. (List of websites, books or etc.)

Kenneth A. Lambert - Fundamentals of Python\_ First Programs, 2nd Edition-Cengage

<https://www.pygame.org> and related python coding websites

<https://github.com/>

<https://www.reddit.com/>

<https://www.youtube.com/>

onedrive.com

https://malayanmindanao.blackboard.com/ultra/courses/\_23325\_1/outline/edit/document/\_1379207\_1?courseId=\_23325\_1&view=content

Pycharm community edition

Flowgorithm

Pixel art drawing program

Photoshop

Yt – dlp youtube downloader