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"
```

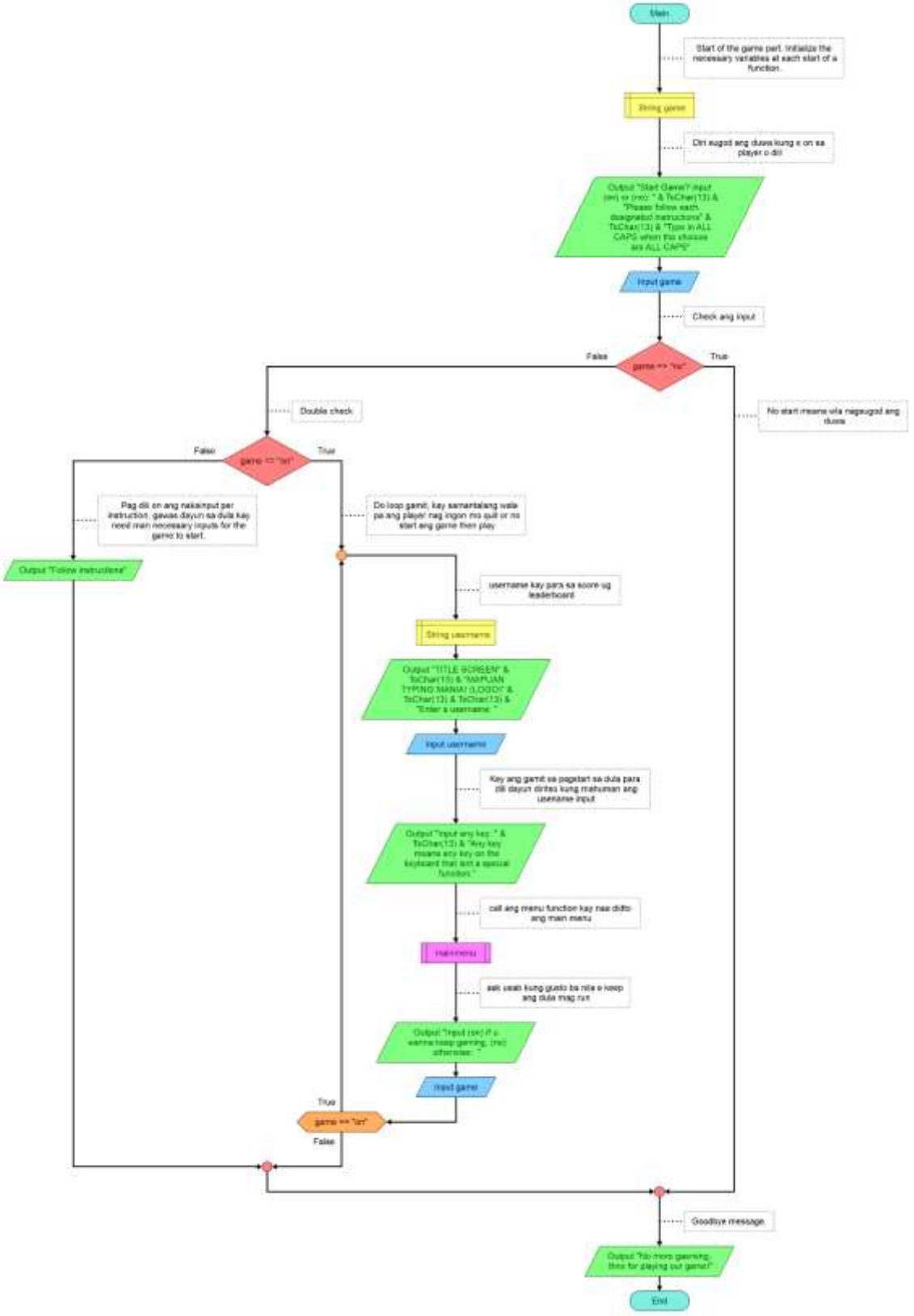
```
... do loop ra japon atung logic samantalang wla mo quit jud ang user
Do
   Input playmenu

   ... 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"
```

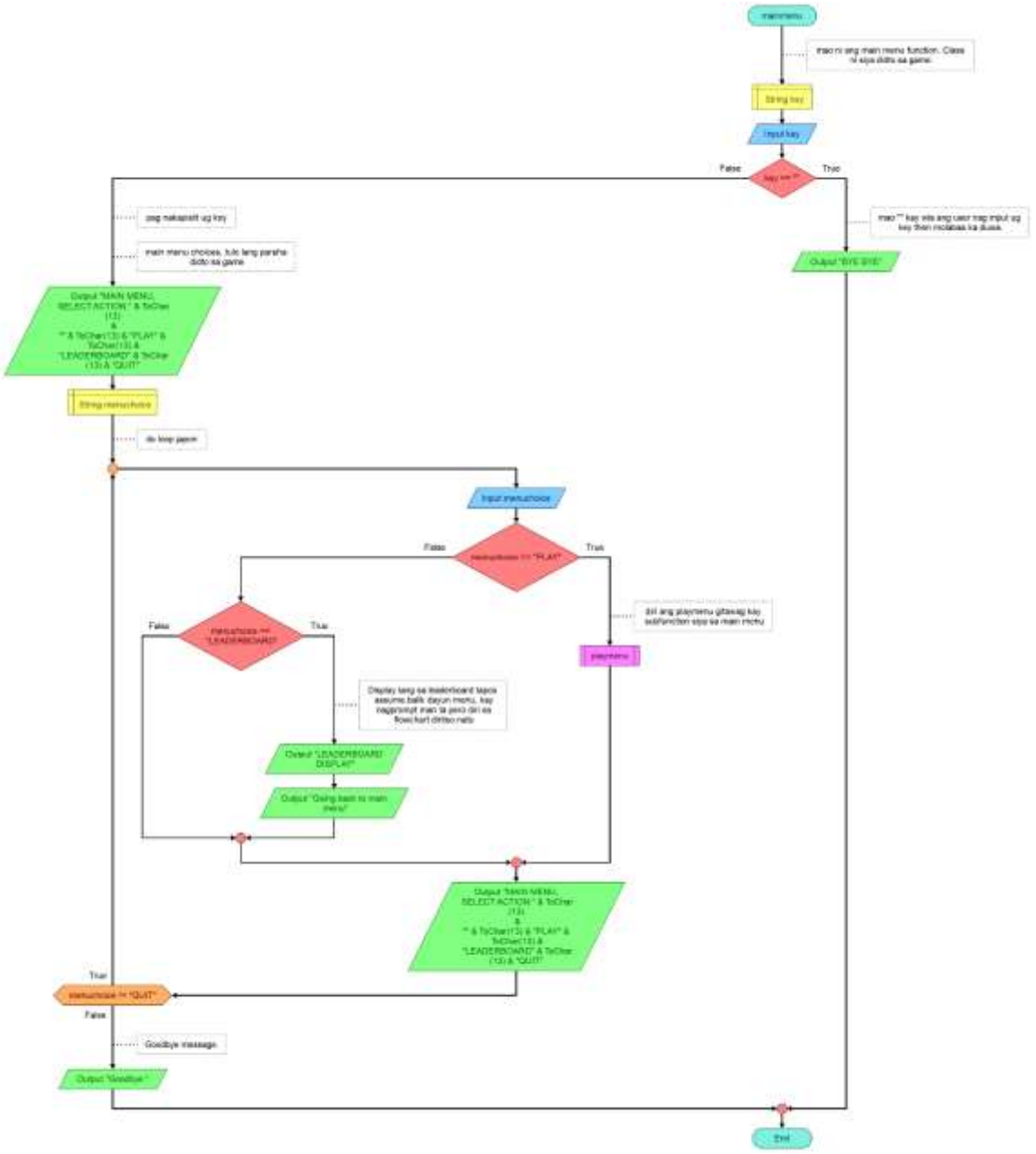
End

FLOWCHART

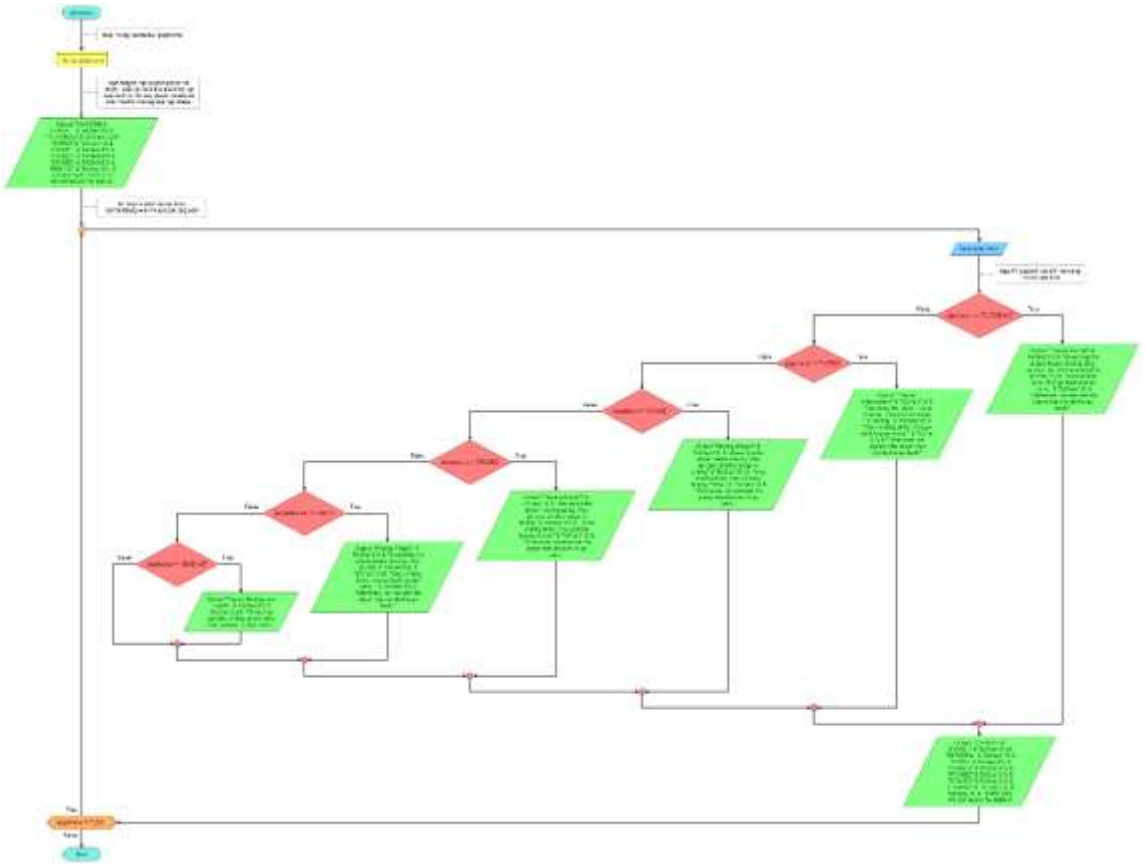
MAIN FLOWCHART

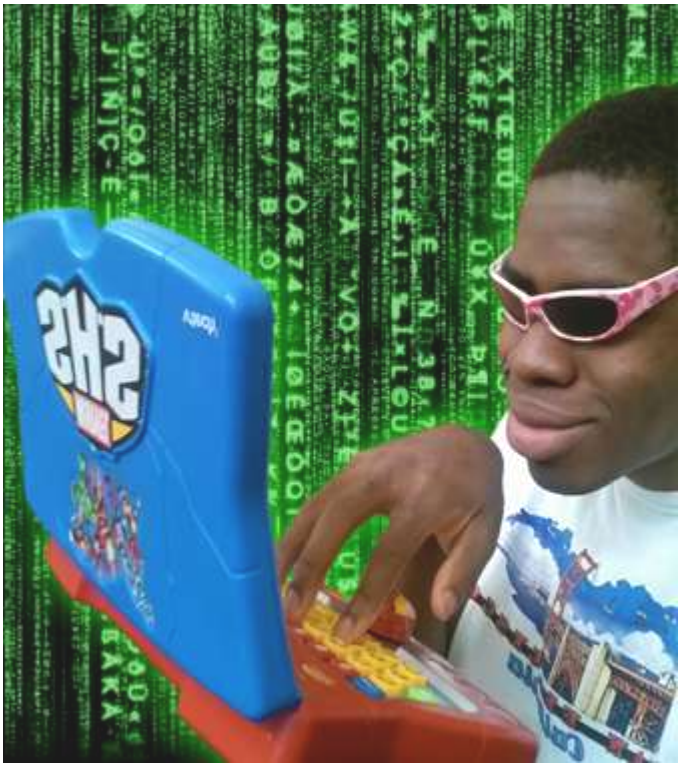


SUB FUNCTION FOR MAIN MENU FLOWCHART



SUB FUNCTION FOR PLAY MENU FLOWCHART





RESULTS and DISCUSSION

TITLE SCREEN

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

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

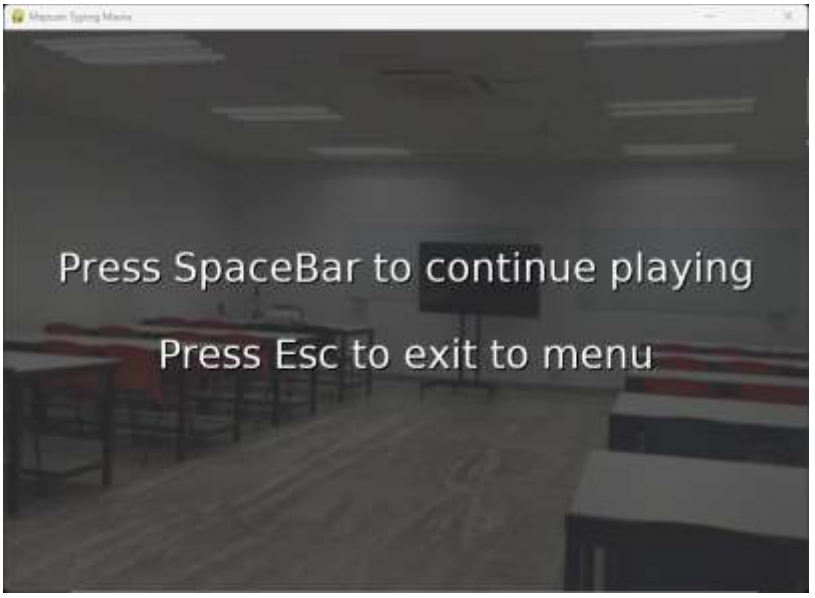
INTRO SCREEN

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

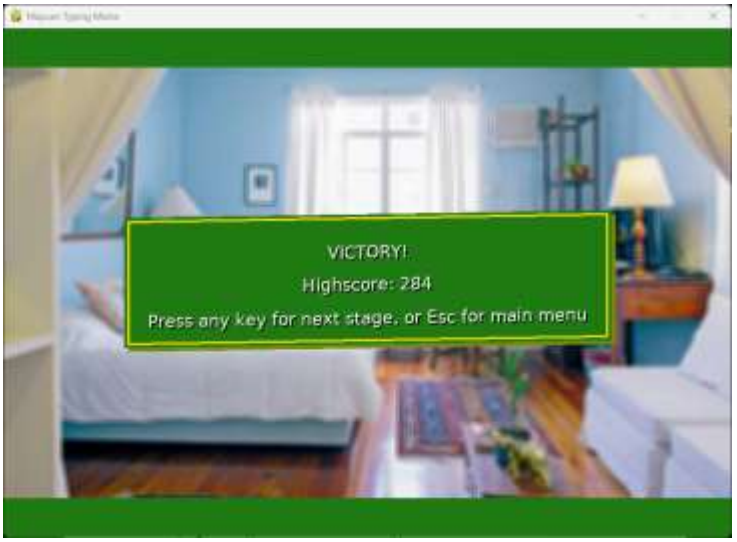
Function:
The actual typing game screen display itself.
Objects:
Background Image – Self explanatory
UI – User interface showing enemy hitpoints, enemy name, user's health, and their score
Red bar with a vertical line – indicating where the words will show when the user types their input
Red words – 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



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



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 falling 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