ALL ASSETS AND OTHER FILES IN ZIP FILE

main.py

​​import pygame # Import the pygame module

import sys # Import the sys module for system functions

from settings import \* # Import all settings

from sprites import \* # Import all sprite classes

import random # Import the random module

import webbrowser # Import the webbrowser module

from pygame import mixer # Import the mixer from pygame

pygame.init() # Initialize pygame

mixer.init() # Initialize the mixer

from pygame.locals import QUIT, MOUSEBUTTONDOWN, MOUSEBUTTONUP, MOUSEMOTION # Import specific events from pygame.locals

DEFAULT\_VOLUME = 0.5 # Set the default volume to 0.5

pygame.mixer.music.set\_volume(DEFAULT\_VOLUME) # Set the mixer music volume to default

SCREEN = pygame.display.set\_mode((1280, 720)) # Set the screen size to 1280x720

BG = pygame.image.load("assets/Background.png") # Load the background image

OPBG = pygame.image.load("assets/OptionsBG.jpg") # Load the options background image

current\_volume = 0.5 # Initialize current volume to 0.5

mixer.music.set\_volume(current\_volume) # Set the mixer music volume to current volume

def get\_font(size):

return pygame.font.Font("assets/font.ttf", size) # Load a font of a specific size

class Button():

def \_\_init\_\_(self, image, pos, text\_input, font, base\_color, hovering\_color):

self.image = image # Set button image

self.x\_pos = pos[0] # X position

self.y\_pos = pos[1] # Y position

self.font = font # Font for text

self.base\_color, self.hovering\_color = base\_color, hovering\_color # Colors

self.text\_input = text\_input # Text on button

self.text = self.font.render(self.text\_input, True, self.base\_color) # Render text

if self.image is None:

self.image = self.text # Use text as image if no image

self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos)) # Button rect

self.text\_rect = self.text.get\_rect(center=(self.x\_pos, self.y\_pos)) # Text rect

def update(self, screen):

if self.image is not None:

screen.blit(self.image, self.rect) # Draw button image

screen.blit(self.text, self.text\_rect) # Draw button text

def checkForInput(self, position):

if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):

return True # Check if clicked

return False

def changeColor(self, position):

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) # Change color on hover

else:

self.text = self.font.render(self.text\_input, True, self.base\_color) # Change back on leave

class EventHandler:

def \_\_init\_\_(self):

EventHandler.events = pygame.event.get() # Initialize events

def run():

EventHandler.events = pygame.event.get() # Update events

def clicked() -> bool:

for e in EventHandler.events:

if e.type == pygame.MOUSEBUTTONDOWN:

return True # Check if mouse clicked

return False

UNSELECTED = "red"

SELECTED = "white"

BUTTONSTATES = {

True: SELECTED,

False: UNSELECTED

}

class UI:

def init(app):

UI.font = get\_font(20) # Set small font

UI.sfont = get\_font(20) # Set medium font

UI.lfont = get\_font(40) # Set large font

UI.xlfont = get\_font(50) # Set extra-large font

UI.center = (app.screen.get\_size()[0]//2, app.screen.get\_size()[1]//2) # Center of the screen

UI.half\_width = app.screen.get\_size()[0]//2 # Half width of the screen

UI.half\_height = app.screen.get\_size()[1]//2 # Half height of the screen

UI.fonts = {

'sm': UI.sfont,

'm': UI.font,

'l': UI.lfont,

'xl': UI.xlfont

} # Font dictionary for easy access

class Slider:

def \_\_init\_\_(self, pos: tuple, size: tuple, initial\_val: float, min: int, max: int) -> None:

self.pos = pos # Position of the slider

self.size = size # Size of the slider

self.hovered = False # Hover state

self.grabbed = False # Grab state

self.slider\_left\_pos = self.pos[0] - (size[0]//2) # Left position of the slider

self.slider\_right\_pos = self.pos[0] + (size[0]//2) # Right position of the slider

self.slider\_top\_pos = self.pos[1] - (size[1]//2) # Top position of the slider

self.min = min # Minimum value

self.max = max # Maximum value

self.initial\_val = (self.slider\_right\_pos-self.slider\_left\_pos)\*initial\_val # Initial value as percentage

self.container\_rect = pygame.Rect(self.slider\_left\_pos, self.slider\_top\_pos, self.size[0], self.size[1]) # Container rectangle

self.button\_rect = pygame.Rect(self.slider\_left\_pos + self.initial\_val - 5, self.slider\_top\_pos, 10, self.size[1]) # Button rectangle

# Label

self.text = UI.fonts['m'].render(str(int(self.get\_value())), True, "white", None) # Render text

self.label\_rect = self.text.get\_rect(center = (self.pos[0], self.slider\_top\_pos - 15)) # Label rectangle

def move\_slider(self, mouse\_pos):

pos = mouse\_pos[0] # Get mouse position

if pos < self.slider\_left\_pos:

pos = self.slider\_left\_pos # Ensure within left bound

if pos > self.slider\_right\_pos:

pos = self.slider\_right\_pos # Ensure within right bound

self.button\_rect.centerx = pos # Update button position

def hover(self):

self.hovered = True # Set hover state to true

def render(self, app):

pygame.draw.rect(app.screen, "darkgray", self.container\_rect) # Draw container

pygame.draw.rect(app.screen, BUTTONSTATES[self.hovered], self.button\_rect) # Draw button

def get\_value(self):

val\_range = self.slider\_right\_pos - self.slider\_left\_pos - 1 # Calculate value range

button\_val = self.button\_rect.centerx - self.slider\_left\_pos # Calculate button value

return (button\_val/val\_range)\*(self.max-self.min)+self.min # Return scaled value

def display\_value(self, app):

self.text = UI.fonts['m'].render(str(int(self.get\_value())), True, "white", None) # Render value text

app.screen.blit(self.text, self.label\_rect) # Blit text onto screen

class NormButton:

def \_\_init\_\_(self, x, y, colour):

self.x = x # X position

self.y = y # Y position

self.colour = colour # Button color

self.rect = pygame.Rect(x, y, BUTTON\_SIZE, BUTTON\_SIZE) # Button rectangle

def draw(self, screen):

pygame.draw.rect(screen, self.colour, self.rect) # Draw button

def clicked(self, mouse\_x, mouse\_y):

return self.rect.collidepoint(mouse\_x, mouse\_y) # Check if clicked

class EduButton:

def \_\_init\_\_(self, x, y, image\_file, width, height, flash\_image\_file, first\_audio, second\_audio):

self.x = x # X position

self.y = y # Y position

self.width = width # Width of button

self.height = height # Height of button

self.image = pygame.image.load(image\_file) # Load image

self.image = pygame.transform.scale(self.image, (width, height)) # Scale image

self.flash\_image = pygame.image.load(flash\_image\_file) # Load flash image

self.flash\_image = pygame.transform.scale(self.flash\_image, (275, 275)) # Scale flash image

self.first\_audio = pygame.mixer.Sound(first\_audio) # Load first audio

self.second\_audio = pygame.mixer.Sound(second\_audio) # Load second audio

self.hovered = False # Hover state

self.hover\_sound\_played = False # Hover sound state

def draw(self, screen):

screen.blit(self.image, (self.x, self.y)) # Draw button

def draw\_flash\_image(self, screen):

screen.blit(self.flash\_image, (self.x - 12, self.y - 12)) # Draw flash image centered

def draw\_hover\_image(self, screen):

screen.blit(self.flash\_image, (self.x - 12, self.y - 12)) # Draw hover image centered

def clicked(self, mouse\_x, mouse\_y):

return self.x <= mouse\_x <= self.x + self.width and self.y <= mouse\_y <= self.y + self.height # Check if clicked

def handle\_hover(self, mouse\_x, mouse\_y):

if self.clicked(mouse\_x, mouse\_y): # Check hover

if not self.hover\_sound\_played:

self.second\_audio.play() # Play hover sound

self.hover\_sound\_played = True # Mark sound as played

self.hovered = True # Set hover state

else:

self.hovered = False # Reset hover state

self.hover\_sound\_played = False # Reset sound state

class ChallButton:

def \_\_init\_\_(self, x, y, image, width, height, sound\_file):

self.x = x # X position

self.y = y # Y position

self.image = pygame.image.load(image).convert\_alpha() # Load and convert image

self.image = pygame.transform.scale(self.image, (width, height)) # Scale image

self.rect = self.image.get\_rect(topleft=(x, y)) # Image rectangle

self.sound = pygame.mixer.Sound(sound\_file) # Load sound

def draw(self, screen):

screen.blit(self.image, (self.x, self.y)) # Draw button

def clicked(self, mouse\_x, mouse\_y):

return self.rect.collidepoint(mouse\_x, mouse\_y) # Check if clicked

class UIElement:

def \_\_init\_\_(self, x, y, text, font\_size):

self.x = x # X position

self.y = y # Y position

self.text = text # Text to display

self.font\_size = font\_size # Font size

def draw(self, screen):

font = get\_font(self.font\_size) # Get font

text\_surface = font.render(self.text, True, WHITE) # Render text

screen.blit(text\_surface, (self.x, self.y)) # Draw text

class EduAudio:

def \_\_init\_\_(self, sound\_file: str):

self.sound = pygame.mixer.Sound(sound\_file) # Load sound file

self.current\_channel = None # Initialize audio channel

def play(self):

self.current\_channel = pygame.mixer.find\_channel(True) # Find available audio channel

self.current\_channel.play(self.sound) # Play sound

def confirm\_exit(screen, return\_to\_function):

# Create a semi-transparent overlay

overlay = pygame.Surface((WIDTH, HEIGHT))

overlay.set\_alpha(128) # Set transparency

overlay.fill((0, 0, 0)) # Fill with black color

font = get\_font(30) # Get font

message = '''

Some progress may not be saved.

Are you okay with this?''' # Confirmation message

lines = message.split('\n') # Split message into lines

yes\_button = Button(image=None, pos=(800, 475), text\_input="STAY", font=get\_font(30), base\_color="White", hovering\_color=GREEN) # Stay button

no\_button = Button(image=None, pos=(400, 475), text\_input="QUIT", font=get\_font(30), base\_color="White", hovering\_color="#800000") # Quit button

while True:

screen.blit(overlay, (0, 0)) # Draw overlay

y\_offset = 200 # Initial Y offset for text

for line in lines:

text\_surface = font.render(line, True, WHITE) # Render text line

screen.blit(text\_surface, (150, y\_offset)) # Draw text line

y\_offset += 40 # Update Y offset

yes\_button.update(screen) # Update stay button

no\_button.update(screen) # Update quit button

pygame.display.update() # Update display

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit() # Quit pygame

sys.exit() # Exit program

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos() # Get mouse position

if yes\_button.checkForInput((mouse\_x, mouse\_y)):

return\_to\_function() # Call return function

if no\_button.checkForInput((mouse\_x, mouse\_y)):

return True # Confirm exit

if event.type == pygame.MOUSEMOTION:

mouse\_x, mouse\_y = pygame.mouse.get\_pos() # Get mouse position

yes\_button.changeColor((mouse\_x, mouse\_y)) # Change stay button color

no\_button.changeColor((mouse\_x, mouse\_y)) # Change quit button color

def confirm\_clear\_high\_scores(screen):

# Create a semi-transparent overlay

overlay = pygame.Surface((WIDTH, HEIGHT))

overlay.set\_alpha(128) # Set transparency

overlay.fill((0, 0, 0)) # Fill with black color

font = get\_font(30) # Get font

message = '''

Are you sure you want to clear

ALL High Scores?''' # Confirmation message

lines = message.split('\n') # Split message into lines

clear\_button = Button(image=None, pos=(400, 475), text\_input="CLEAR", font=get\_font(30), base\_color="White", hovering\_color="#800000") # Clear button

back\_button = Button(image=None, pos=(800, 475), text\_input="BACK", font=get\_font(30), base\_color="White", hovering\_color=GREEN) # Back button

while True:

screen.blit(overlay, (0, 0)) # Draw overlay

y\_offset = 200 # Initial Y offset for text

for line in lines:

text\_surface = font.render(line, True, WHITE) # Render text line

screen.blit(text\_surface, (150, y\_offset)) # Draw text line

y\_offset += 40 # Update Y offset

clear\_button.update(screen) # Update clear button

back\_button.update(screen) # Update back button

pygame.display.update() # Update display

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit() # Quit pygame

sys.exit() # Exit program

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos() # Get mouse position

if clear\_button.checkForInput((mouse\_x, mouse\_y)):

reset\_file\_norm = open("high\_score.txt", "w") # Open normal mode high score file

reset\_file\_norm.write("0") # Reset high score

reset\_file\_norm.close() # Close file

reset\_file\_edu = open("edu\_high\_score.txt", "w") # Open education mode high score file

reset\_file\_edu.write("0") # Reset high score

reset\_file\_edu.close() # Close file

reset\_file\_chall = open("chall\_high\_score.txt", "w") # Open challenge mode high score file

reset\_file\_chall.write("0") # Reset high score

reset\_file\_chall.close() # Close file

options() # Return to options after clearing high scores

return

if back\_button.checkForInput((mouse\_x, mouse\_y)):

options() # Return to options without clearing high scores

return

if event.type == pygame.MOUSEMOTION:

mouse\_x, mouse\_y = pygame.mouse.get\_pos() # Get mouse position

clear\_button.changeColor((mouse\_x, mouse\_y)) # Change clear button color

back\_button.changeColor((mouse\_x, mouse\_y)) # Change back button color

def game\_over\_audio():

beeps = [Audio(BEEP1), Audio(BEEP2), Audio(BEEP3), Audio(BEEP4)]

for beep in beeps:

beep.play() # Play all beats synched to get the "BEEP!" - Game over sound

def play():

while True:

PLAY\_MOUSE\_POS = pygame.mouse.get\_pos() # Get current mouse position

MENU\_MOUSE\_POS = pygame.mouse.get\_pos() # Get current mouse position

SCREEN.blit(OPBG, (0, 0)) # Draw background image

PLAY\_TEXT = get\_font(50).render("SELECT GAME MODE", True, "#b68f40") # Render game mode selection text

PLAY\_RECT = PLAY\_TEXT.get\_rect(center=(640, 50)) # Center the text

SCREEN.blit(PLAY\_TEXT, PLAY\_RECT) # Draw the text on the screen

PLAY\_BACK = Button(image=None, pos=(120, 50),

text\_input="BACK", font=get\_font(40), base\_color="White", hovering\_color="Green") # Create back button

PLAY\_BACK.changeColor(PLAY\_MOUSE\_POS) # Change color on hover

PLAY\_BACK.update(SCREEN) # Update button display

PLAY\_EDUMODE = Button(image=None, pos=(220, 600),

text\_input="EDUCATION", font=get\_font(20), base\_color=WHITE, hovering\_color=GREEN) # Create education mode button

PLAY\_EDUMODE.changeColor(PLAY\_MOUSE\_POS) # Change color on hover

PLAY\_EDUMODE.update(SCREEN) # Update button display

PLAY\_NORMMODE = Button(image=None, pos=(640, 600),

text\_input="NORMAL", font=get\_font(20), base\_color=WHITE, hovering\_color=GREEN) # Create normal mode button

PLAY\_NORMMODE.changeColor(PLAY\_MOUSE\_POS) # Change color on hover

PLAY\_NORMMODE.update(SCREEN) # Update button display

PLAY\_CHALLMMODE = Button(image=None, pos=(1060, 600),

text\_input="CHALLENGE", font=get\_font(20), base\_color=WHITE, hovering\_color=GREEN) # Create challenge mode button

PLAY\_CHALLMMODE.changeColor(PLAY\_MOUSE\_POS) # Change color on hover

PLAY\_CHALLMMODE.update(SCREEN) # Update button display

eduplayimg = pygame.image.load("assets/PlayEdu.png") # Load education mode image

eduplayimgscaled = pygame.transform.scale(eduplayimg, (180, 430)) # Scale image

SCREEN.blit(eduplayimgscaled, (130, 130)) # Draw image on screen

normplayimg = pygame.image.load("assets/PlayNorm.png") # Load normal mode image

normplayimgscaled = pygame.transform.scale(normplayimg, (200, 430)) # Scale image

SCREEN.blit(normplayimgscaled, (532, 130)) # Draw image on screen

challplayimg = pygame.image.load("assets/PlayChall.png") # Load challenge mode image

challplayimgscaled = pygame.transform.scale(challplayimg, (200, 430)) # Scale image

SCREEN.blit(challplayimgscaled, (970, 130)) # Draw image on screen

for button in [PLAY\_EDUMODE, PLAY\_NORMMODE, PLAY\_CHALLMMODE]:

button.changeColor(MENU\_MOUSE\_POS) # Change button color on hover

button.update(SCREEN) # Update button display

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit() # Quit pygame

sys.exit() # Exit program

if event.type == pygame.MOUSEBUTTONDOWN:

if PLAY\_EDUMODE.checkForInput(MENU\_MOUSE\_POS):

edumode() # Start education mode

if PLAY\_NORMMODE.checkForInput(MENU\_MOUSE\_POS):

normmode() # Start normal mode

if PLAY\_CHALLMMODE.checkForInput(MENU\_MOUSE\_POS):

challmode() # Start challenge mode

if PLAY\_BACK.checkForInput(MENU\_MOUSE\_POS):

main\_menu() # Return to main menu

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit() # Quit pygame

sys.exit() # Exit program

if event.type == pygame.MOUSEBUTTONDOWN:

if PLAY\_BACK.checkForInput(PLAY\_MOUSE\_POS):

main\_menu() # Return to main menu

pygame.display.update() # Update display

def main\_menu():

while True:

pygame.display.set\_caption(TITLE) # Set the window title

SCREEN.blit(BG, (0, 0)) # Draw the background image

MENU\_MOUSE\_POS = pygame.mouse.get\_pos() # Get the current mouse position

# Render the text onto a surface

SCIENTIFIC\_BANNER\_TEXT = get\_font(20).render("Scientifically Proven!", True, "#ff9535")

# Rotate the surface by 10 degrees

rotated\_banner\_text = pygame.transform.rotate(SCIENTIFIC\_BANNER\_TEXT, 10)

# Get the rectangle of the rotated surface

SCIENTIFIC\_BANNER\_RECT = rotated\_banner\_text.get\_rect(center=(300, 290))

# Blit the rotated surface onto the screen

SCREEN.blit(rotated\_banner\_text, SCIENTIFIC\_BANNER\_RECT)

MENU\_TEXT = get\_font(100).render("BRAIN BEAT", True, "#b68f40") # Render the main menu text

MENU\_RECT = MENU\_TEXT.get\_rect(center=(640, 100)) # Center the main menu text

PLAY\_BUTTON = Button(image=None, pos=(215, 450),

text\_input="PLAY", font=get\_font(40), base\_color="White", hovering\_color="#d6b4fc") # Create the Play button

OPTIONS\_BUTTON = Button(image=None, pos=(215, 550),

text\_input="OPTIONS", font=get\_font(40), base\_color="White", hovering\_color="#08787f") # Create the Options button

QUIT\_BUTTON = Button(image=None, pos=(215, 650),

text\_input="QUIT", font=get\_font(40), base\_color="White", hovering\_color="#800000") # Create the Quit button

logoimg = pygame.image.load("assets/BrainBeatNewTransparent.png") # Load the logo image

logoimgscaled = pygame.transform.scale(logoimg, (445, 445)) # Scale the logo image

SCREEN.blit(logoimgscaled, (575, 220)) # Draw the scaled logo image on the screen

SCREEN.blit(MENU\_TEXT, MENU\_RECT) # Draw the main menu text on the screen

for button in [PLAY\_BUTTON, OPTIONS\_BUTTON, QUIT\_BUTTON]:

button.changeColor(MENU\_MOUSE\_POS) # Change button color on hover

button.update(SCREEN) # Update button display

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit() # Quit pygame

sys.exit() # Exit program

if event.type == pygame.MOUSEBUTTONDOWN:

if PLAY\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

play() # Start play mode

if OPTIONS\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

options() # Go to options menu

if QUIT\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

pygame.quit() # Quit pygame

sys.exit() # Exit program

pygame.display.update() # Update the display

def options():

app = type('', (), {})() # Create an empty object to simulate the app

app.screen = SCREEN # Assign the main screen to the app

UI.init(app) # Initialize the UI with the app

slider = Slider((900, 200), (600, 40), 0.5, 0, 100) # Create a slider object

while True:

OPTIONS\_MOUSE\_POS = pygame.mouse.get\_pos() # Get the current mouse position

MOUSE\_CLICKED = pygame.mouse.get\_pressed() # Get the current mouse button state

SCREEN.blit(OPBG, (0, 0)) # Draw the options background

OPTIONS\_TEXT = get\_font(50).render("OPTIONS", True, "#b68f40") # Render the options text

OPTIONS\_RECT = OPTIONS\_TEXT.get\_rect(center=(640, 50)) # Center the options text

SCREEN.blit(OPTIONS\_TEXT, OPTIONS\_RECT) # Draw the options text on the screen

OPTIONS\_BACK = Button(image=None, pos=(120, 50),

text\_input="BACK", font=get\_font(40), base\_color="White", hovering\_color="Green") # Create the back button

OPTIONS\_BACK.changeColor(OPTIONS\_MOUSE\_POS) # Change the back button color on hover

OPTIONS\_BACK.update(SCREEN) # Update the back button display

VOLUME\_TEXT = get\_font(35).render("Volume:", True, "#08787f") # Render the volume text

VOLUME\_RECT = VOLUME\_TEXT.get\_rect(center=(325, 200)) # Center the volume text

SCREEN.blit(VOLUME\_TEXT, VOLUME\_RECT) # Draw the volume text on the screen

CLEARHS\_TEXT = get\_font(35).render("High Scores:", True, "#08787f") # Render the high scores text

CLEARHS\_RECT = CLEARHS\_TEXT.get\_rect(center=(325, 350)) # Center the high scores text

SCREEN.blit(CLEARHS\_TEXT, CLEARHS\_RECT) # Draw the high scores text on the screen

HTP\_TEXT = get\_font(35).render("How to Play:", True, "#08787f") # Render the how to play text

HTP\_RECT = HTP\_TEXT.get\_rect(center=(325, 500)) # Center the how to play text

SCREEN.blit(HTP\_TEXT, HTP\_RECT) # Draw the how to play text on the screen

HTP\_DETAIL\_TEXT = get\_font(20).render("To get the most fun out of the", True, "White") # Render the how to play detail text

HTP\_DETAIL\_RECT = HTP\_DETAIL\_TEXT.get\_rect(center=(900, 500)) # Center the how to play detail text

SCREEN.blit(HTP\_DETAIL\_TEXT, HTP\_DETAIL\_RECT) # Draw the how to play detail text on the screen

HTP\_DETAIL\_TEXT2 = get\_font(20).render("Brain Beat experience, click", True, "White") # Render the how to play detail text 2

HTP\_DETAIL\_RECT2 = HTP\_DETAIL\_TEXT2.get\_rect(center=(885, 550)) # Center the how to play detail text 2

SCREEN.blit(HTP\_DETAIL\_TEXT2, HTP\_DETAIL\_RECT2) # Draw the how to play detail text 2 on the screen

HTP\_DETAIL\_TEXT3 = get\_font(20).render("(Redirection to web page)", True, "White") # Render the how to play detail text 3

HTP\_DETAIL\_RECT3 = HTP\_DETAIL\_TEXT3.get\_rect(center=(890, 670)) # Center the how to play detail text 3

SCREEN.blit(HTP\_DETAIL\_TEXT3, HTP\_DETAIL\_RECT3) # Draw the how to play detail text 3 on the screen

# Create the How to Play Button

HTP\_BUTTON = Button(image=None, pos=(900, 610),

text\_input="HERE", font=get\_font(40), base\_color="White", hovering\_color="#fc4665")

HTP\_BUTTON.changeColor(OPTIONS\_MOUSE\_POS) # Change the How to Play button color on hover

HTP\_BUTTON.update(SCREEN) # Update the How to Play button display

CLEARHS\_BUTTON = Button(image=None, pos=(880, 350),

text\_input="CLEAR ALL", font=get\_font(30), base\_color="White", hovering\_color="#fc4665")

CLEARHS\_BUTTON.changeColor(OPTIONS\_MOUSE\_POS) # Change the Clear High Scores button color on hover

CLEARHS\_BUTTON.update(SCREEN) # Update the Clear High Scores button display

mouse\_pos = pygame.mouse.get\_pos() # Get the current mouse position

mouse = pygame.mouse.get\_pressed() # Get the current mouse button state

# Handle the slider

if slider.container\_rect.collidepoint(mouse\_pos):

if mouse[0]:

slider.grabbed = True # Check if the slider is grabbed

if not mouse[0]:

slider.grabbed = False # Release the slider if the mouse button is not pressed

if slider.button\_rect.collidepoint(mouse\_pos):

slider.hover() # Change the slider state to hovered

if slider.grabbed:

slider.move\_slider(mouse\_pos) # Move the slider

slider.hover() # Change the slider state to hovered

else:

slider.hovered = False # Change the slider state to not hovered

slider.render(app) # Render the slider

slider.display\_value(app) # Display the slider value

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit() # Quit pygame

sys.exit() # Exit the program

if event.type == pygame.MOUSEBUTTONDOWN:

if OPTIONS\_BACK.checkForInput(OPTIONS\_MOUSE\_POS):

main\_menu() # Go back to the main menu

if HTP\_BUTTON.checkForInput(OPTIONS\_MOUSE\_POS):

webbrowser.open("https://online.publuu.com/554222/1249336") # Open the How to Play link

if CLEARHS\_BUTTON.checkForInput(pygame.mouse.get\_pos()):

if pygame.mouse.get\_pressed()[0]:

confirm\_clear\_high\_scores(SCREEN) # Confirm clearing high scores

pygame.display.update() # Update the display

def edumode():

pygame.init()

screen = pygame.display.set\_mode((WIDTH, HEIGHT)) # Initialize pygame screen with specified dimensions

pygame.display.set\_caption(TITLE) # Set the window title of the game

clock = pygame.time.Clock() # Initialize pygame clock for managing frame rate

# Load animal images and corresponding sounds

ANIMAL\_IMAGES = ["EduMode/Pig.png", "EduMode/Snake.png", "EduMode/Cow.png", "EduMode/Lion.png"]

FIRST\_ANIMAL\_SOUNDS = ["EduMode/PigAudio.mp3", "EduMode/SnakeAudio.mp3", "EduMode/CowAudio.mp3", "EduMode/LionAudio.mp3"]

SECOND\_ANIMAL\_SOUNDS = ["EduMode/FinalPigOink.mp3", "EduMode/FinalSnakeHiss.mp3", "EduMode/FinalCowMoo.mp3", "EduMode/FinalLionRoar.mp3"]

FLASH\_IMAGES = ["EduMode/PigFlash.png", "EduMode/SnakeFlash.png", "EduMode/CowFlash.png", "EduMode/LionFlash.png"]

# Create instances of EduButton with adjusted coordinates, sizes, and associated files

animals = [EduButton(250, 100, ANIMAL\_IMAGES[0], width=250, height=250, flash\_image\_file=FLASH\_IMAGES[0], first\_audio=FIRST\_ANIMAL\_SOUNDS[0], second\_audio=SECOND\_ANIMAL\_SOUNDS[0]),

EduButton(600, 100, ANIMAL\_IMAGES[1], width=250, height=250, flash\_image\_file=FLASH\_IMAGES[1], first\_audio=FIRST\_ANIMAL\_SOUNDS[1], second\_audio=SECOND\_ANIMAL\_SOUNDS[1]),

EduButton(250, 400, ANIMAL\_IMAGES[2], width=250, height=250, flash\_image\_file=FLASH\_IMAGES[2], first\_audio=FIRST\_ANIMAL\_SOUNDS[2], second\_audio=SECOND\_ANIMAL\_SOUNDS[2]),

EduButton(600, 400, ANIMAL\_IMAGES[3], width=250, height=250, flash\_image\_file=FLASH\_IMAGES[3], first\_audio=FIRST\_ANIMAL\_SOUNDS[3], second\_audio=SECOND\_ANIMAL\_SOUNDS[3])]

back\_button = Button(image=None, pos=(120, 50), text\_input="BACK", font=get\_font(40), base\_color="White", hovering\_color="#b68f40")

OPTIONS\_TEXT = get\_font(50).render("EDUCATION", True, "#b68f40") # Render the text for the game mode title

OPTIONS\_RECT = OPTIONS\_TEXT.get\_rect(center=(640, 50)) # Get the rectangle of the rendered text for positioning

# Function to retrieve the highest score from the file

def get\_high\_score():

with open("edu\_high\_score.txt", "r") as file:

score = file.read()

return int(score)

# Function to save the score to the file if it's higher than the existing high score

def save\_score(score, high\_score):

with open("edu\_high\_score.txt", "w") as file:

if score > high\_score:

file.write(str(score))

else:

file.write(str(high\_score))

# Function to initialize a new game session by resetting variables

def new\_game():

nonlocal waiting\_input, pattern, current\_step, score, high\_score

waiting\_input = False

pattern = []

current\_step = 0

score = 0

high\_score = get\_high\_score()

# Function to run the main game loop

def run\_game():

nonlocal playing, clicked\_button

playing = True

while playing:

clock.tick(FPS)

clicked\_button = None

events()

draw()

update()

# Function to update the game state based on user input and game logic

def update():

nonlocal waiting\_input, current\_step, playing, score

if not waiting\_input:

pygame.time.wait(1000)

pattern.append(random.choice(animals))

for button in pattern:

button\_animation(button)

pygame.time.wait(3200) # Adjusted delay to accommodate audio length

waiting\_input = True

else:

if clicked\_button and clicked\_button == pattern[current\_step]:

# Delay to ensure clear feedback for correct click

pygame.time.wait(500)

clicked\_button.second\_audio.play() # Play the second audio when user clicks correctly

user\_clicked\_button\_animation(clicked\_button)

current\_step += 1

if current\_step == len(pattern):

score += 1

waiting\_input = False

current\_step = 0

elif clicked\_button and clicked\_button != pattern[current\_step]:

game\_over\_animation()

save\_score(score, high\_score)

playing = False

# Function to animate button flash effect during pattern presentation

def button\_animation(button):

sound = button.first\_audio

original\_surface = screen.copy()

flash\_surface = pygame.Surface((250, 250)) # Adjusted flash surface size

flash\_surface = flash\_surface.convert\_alpha()

r, g, b, a = (255, 255, 255, 0) # Using white for flash

sound.play()

for start, end, step in ((0, 255, 1), (255, 0, -1)):

for alpha in range(start, end, ANIMATION\_SPEED \* step):

screen.blit(original\_surface, (0, 0))

flash\_surface.fill((r, g, b, alpha))

screen.blit(flash\_surface, (button.x, button.y)) # Adjusted position to match button location

pygame.display.update()

clock.tick(FPS)

screen.blit(original\_surface, (0, 0))

# Function to animate the flash effect when the user clicks on a button

def user\_clicked\_button\_animation(button):

button.draw\_flash\_image(screen)

# Function to animate game over effect when the user makes a mistake

def game\_over\_animation():

original\_surface = screen.copy()

flash\_surface = pygame.Surface((screen.get\_size()))

flash\_surface = flash\_surface.convert\_alpha()

game\_over\_audio()

r, g, b = WHITE

for \_ in range(3):

for start, end, step in ((0, 255, 1), (255, 0, -1)):

for alpha in range(start, end, ANIMATION\_SPEED \* step):

screen.blit(original\_surface, (0, 0))

flash\_surface.fill((r, g, b, alpha))

screen.blit(flash\_surface, (0, 0))

pygame.display.update()

clock.tick(FPS)

# Function to draw elements onto the screen during the game

def draw():

screen.fill(BGCOLOUR)

UIElement(900, 300, f"Score: {str(score)}", 25).draw(screen)

UIElement(900, 380, f"High score: {str(high\_score)}", 25).draw(screen)

for button in animals:

if button.hovered:

button.draw\_hover\_image(screen)

else:

button.draw(screen)

back\_button.update(screen)

screen.blit(OPTIONS\_TEXT, OPTIONS\_RECT)

pygame.display.update()

# Function to handle various events during the game, such as mouse clicks and movements

def events():

nonlocal clicked\_button, playing

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

for button in animals:

if button.clicked(mouse\_x, mouse\_y):

clicked\_button = button

user\_clicked\_button\_animation(clicked\_button)

if back\_button.checkForInput((mouse\_x, mouse\_y)):

confirm\_exit(screen, edumode)

play()

return

if event.type == pygame.MOUSEMOTION:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

for button in animals:

button.handle\_hover(mouse\_x, mouse\_y)

back\_button.changeColor((mouse\_x, mouse\_y))

# Initialize game variables and start the main game loop

waiting\_input = False

pattern = []

current\_step = 0

score = 0

high\_score = get\_high\_score()

playing = True

clicked\_button = None

while True:

new\_game()

run\_game()

def normmode():

pygame.init()

screen = pygame.display.set\_mode((WIDTH, HEIGHT)) # Initialize pygame screen with specified dimensions

pygame.display.set\_caption(TITLE) # Set the window title of the game

clock = pygame.time.Clock() # Initialize pygame clock for managing frame rate

beeps = [Audio(BEEP1), Audio(BEEP2), Audio(BEEP3), Audio(BEEP4)]

flash\_colours = [YELLOW, BLUE, RED, GREEN]

colours = [DARKYELLOW, DARKBLUE, DARKRED, DARKGREEN]

buttons = [

NormButton(410, 150, DARKYELLOW), # Create a button with dark yellow color at specified position

NormButton(630, 150, DARKBLUE), # Create a button with dark blue color at specified position

NormButton(410, 370, DARKRED), # Create a button with dark red color at specified position

NormButton(630, 370, DARKGREEN), # Create a button with dark green color at specified position

]

back\_button = Button(image=None, pos=(120, 50), text\_input="BACK", font=get\_font(40), base\_color="White", hovering\_color="#b68f40")

OPTIONS\_TEXT = get\_font(50).render("NORMAL", True, "#b68f40") # Render the text for the game mode title

OPTIONS\_RECT = OPTIONS\_TEXT.get\_rect(center=(640, 50)) # Get the rectangle of the rendered text for positioning

# Function to retrieve the highest score from the file

def get\_high\_score():

with open("high\_score.txt", "r") as file:

score = file.read()

return int(score)

# Function to save the score to the file if it's higher than the existing high score

def save\_score(score, high\_score):

with open("high\_score.txt", "w") as file:

if score > high\_score:

file.write(str(score))

else:

file.write(str(high\_score))

# Function to initialize a new game session by resetting variables

def new\_game():

nonlocal waiting\_input, pattern, current\_step, score, high\_score

waiting\_input = False

pattern = []

current\_step = 0

score = 0

high\_score = get\_high\_score()

# Function to run the main game loop

def run\_game():

nonlocal playing, clicked\_button

playing = True

while playing:

clock.tick(FPS)

clicked\_button = None

events()

draw()

update()

# Function to update the game state based on user input and game logic

def update():

nonlocal waiting\_input, current\_step, playing, score

if not waiting\_input:

pygame.time.wait(1000)

pattern.append(random.choice(colours))

for button in pattern:

button\_animation(button)

pygame.time.wait(200)

waiting\_input = True

else:

if clicked\_button and clicked\_button == pattern[current\_step]:

button\_animation(clicked\_button)

current\_step += 1

if current\_step == len(pattern):

score += 1

waiting\_input = False

current\_step = 0

elif clicked\_button and clicked\_button != pattern[current\_step]:

game\_over\_animation()

save\_score(score, high\_score)

playing = False

# Function to animate button flash effect during pattern presentation

def button\_animation(colour):

for i in range(len(colours)):

if colours[i] == colour:

sound = beeps[i]

flash\_colour = flash\_colours[i]

button = buttons[i]

original\_surface = screen.copy()

flash\_surface = pygame.Surface((BUTTON\_SIZE, BUTTON\_SIZE))

flash\_surface = flash\_surface.convert\_alpha()

r, g, b = flash\_colour

sound.play()

for start, end, step in ((0, 255, 1), (255, 0, -1)):

for alpha in range(start, end, ANIMATION\_SPEED \* step):

screen.blit(original\_surface, (0, 0))

flash\_surface.fill((r, g, b, alpha))

screen.blit(flash\_surface, (button.x, button.y)) # Adjusted position to match button location

pygame.display.update()

clock.tick(FPS)

screen.blit(original\_surface, (0, 0))

# Function to animate game over effect when the user makes a mistake

def game\_over\_animation():

original\_surface = screen.copy()

flash\_surface = pygame.Surface((screen.get\_size()))

flash\_surface = flash\_surface.convert\_alpha()

game\_over\_audio()

r, g, b = WHITE

for \_ in range(3):

for start, end, step in ((0, 255, 1), (255, 0, -1)):

for alpha in range(start, end, ANIMATION\_SPEED \* step):

screen.blit(original\_surface, (0, 0))

flash\_surface.fill((r, g, b, alpha))

screen.blit(flash\_surface, (0, 0))

pygame.display.update()

clock.tick(FPS)

# Function to draw elements onto the screen during the game

def draw():

screen.fill(BGCOLOUR)

UIElement(900, 300, f"Score: {str(score)}", 25).draw(screen)

UIElement(900, 380, f"High score: {str(high\_score)}", 25).draw(screen)

for button in buttons:

button.draw(screen)

# Draw the BACK button

back\_button.update(screen)

# Draw the OPTIONS text

screen.blit(OPTIONS\_TEXT, OPTIONS\_RECT)

pygame.display.update()

# Function to handle various events during the game, such as mouse clicks and movements

def events():

nonlocal clicked\_button, playing

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

for button in buttons:

if button.clicked(mouse\_x, mouse\_y):

clicked\_button = button.colour

# Check if the BACK button is clicked

if back\_button.checkForInput((mouse\_x, mouse\_y)):

confirm\_exit(screen, normmode)

play() # Call the play() function

return # Exit the current normmode function

if event.type == pygame.MOUSEMOTION:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

back\_button.changeColor((mouse\_x, mouse\_y))

# Initialize game variables and start the main game loop

waiting\_input = False

pattern = []

current\_step = 0

score = 0

high\_score = get\_high\_score()

playing = True

clicked\_button = None

while True:

new\_game()

run\_game()

def challmode():

pygame.init()

screen = pygame.display.set\_mode((WIDTH, HEIGHT)) # Initialize pygame screen with specified dimensions

pygame.display.set\_caption(TITLE) # Set the window title of the game

clock = pygame.time.Clock() # Initialize pygame clock for managing frame rate

# Load images and sounds for challenge mode

HARD\_IMAGES = ["ChallMode/Fire.png", "ChallMode/Joker.png", "ChallMode/PurpSkull.png", "ChallMode/Skull.png"]

HARD\_SOUNDS = ["ChallMode/FireAudio.mp3", "ChallMode/JokerAudio.mp3", "ChallMode/PurpSkullAudio.mp3", "ChallMode/SkullAudio.mp3"]

challimg = [ChallButton(250, 150, HARD\_IMAGES[0], width=260, height=260, sound\_file=HARD\_SOUNDS[0]), # Create challenge buttons with specified parameters

ChallButton(560, 160, HARD\_IMAGES[1], width=250, height=250, sound\_file=HARD\_SOUNDS[1]),

ChallButton(250, 450, HARD\_IMAGES[2], width=250, height=200, sound\_file=HARD\_SOUNDS[2]),

ChallButton(590, 460, HARD\_IMAGES[3], width=200, height=200, sound\_file=HARD\_SOUNDS[3])]

back\_button = Button(image=None, pos=(120, 50), text\_input="BACK", font=get\_font(40), base\_color="White", hovering\_color="#b68f40")

OPTIONS\_TEXT = get\_font(50).render("CHALLENGE", True, "#b68f40") # Render the text for the challenge mode title

OPTIONS\_RECT = OPTIONS\_TEXT.get\_rect(center=(640, 50)) # Get the rectangle of the rendered text for positioning

# Function to retrieve the highest score from the file, handling potential file absence

def get\_high\_score():

try:

with open("chall\_high\_score.txt", "r") as file:

score = int(file.read())

except FileNotFoundError:

score = 0

return score

# Function to save the score to the file if it's higher than the existing high score

def save\_score(score, high\_score):

with open("chall\_high\_score.txt", "w") as file:

file.write(str(max(score, high\_score)))

# Function to initialize a new game session by resetting variables

def new\_game():

nonlocal waiting\_input, pattern, current\_step, score, high\_score, start\_ticks

waiting\_input = False

pattern = []

current\_step = 0

score = 0

high\_score = get\_high\_score()

start\_ticks = pygame.time.get\_ticks()

# Function to run the main game loop

def run\_game():

nonlocal playing

playing = True

while playing:

clock.tick(FPS)

events()

draw()

update()

# Function to update the game state based on user input and game logic

def update():

nonlocal waiting\_input, current\_step, playing, score, start\_ticks

if not waiting\_input:

pygame.time.wait(1000)

pattern.append(random.choice(challimg))

for button in pattern:

button\_animation(button, play\_sound=True)

pygame.time.wait(1500) # Adjusted delay to 1.5 seconds between each button

waiting\_input = True

start\_ticks = pygame.time.get\_ticks()

else:

if pygame.time.get\_ticks() - start\_ticks >= 3000:

game\_over\_animation()

save\_score(score, high\_score)

playing = False

# Function to animate button flash effect during pattern presentation

def button\_animation(button, play\_sound=False):

original\_surface = screen.copy()

button\_rect = button.image.get\_rect(topleft=(button.x, button.y)) # Get the rect of the button's image

flash\_surface = pygame.Surface((button\_rect.width, button\_rect.height)) # Use image dimensions

flash\_surface = flash\_surface.convert\_alpha()

r, g, b, a = (255, 255, 255, 0) # Using white for flash

if play\_sound:

button.sound.play()

for start, end, step in ((0, 255, 1), (255, 0, -1)):

for alpha in range(start, end, ANIMATION\_SPEED \* step):

screen.blit(original\_surface, (0, 0))

flash\_surface.fill((r, g, b, alpha))

screen.blit(flash\_surface, button\_rect.topleft) # Position adjusted to button's top-left corner

pygame.display.update()

clock.tick(FPS)

screen.blit(original\_surface, (0, 0))

# Function to animate game over effect when the user makes a mistake

def game\_over\_animation():

original\_surface = screen.copy()

flash\_surface = pygame.Surface((screen.get\_size()))

flash\_surface = flash\_surface.convert\_alpha()

game\_over\_audio()

r, g, b = WHITE

for \_ in range(3):

for start, end, step in ((0, 255, 1), (255, 0, -1)):

for alpha in range(start, end, ANIMATION\_SPEED \* step):

screen.blit(original\_surface, (0, 0))

flash\_surface.fill((r, g, b, alpha))

screen.blit(flash\_surface, (0, 0))

pygame.display.update()

clock.tick(FPS)

# Function to draw elements onto the screen during the game

def draw():

screen.fill(BGCOLOUR)

UIElement(900, 300, f"Score: {str(score)}", 25).draw(screen)

UIElement(900, 380, f"High score: {str(high\_score)}", 25).draw(screen)

for button in challimg:

button.draw(screen)

# Draw the BACK button

back\_button.update(screen)

# Draw the OPTIONS text

screen.blit(OPTIONS\_TEXT, OPTIONS\_RECT)

pygame.display.update()

# Function to handle various events during the game, such as mouse clicks and movements

def events():

nonlocal playing, current\_step, waiting\_input, start\_ticks, score

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

for button in challimg:

if button.clicked(mouse\_x, mouse\_y):

if button == pattern[current\_step]:

button\_animation(button) # Call button\_animation without sound

current\_step += 1

start\_ticks = pygame.time.get\_ticks() # Reset the timer

if current\_step == len(pattern):

score += 1

waiting\_input = False

current\_step = 0

return

else:

game\_over\_animation()

save\_score(score, high\_score)

playing = False

return

if back\_button.checkForInput((mouse\_x, mouse\_y)):

confirm\_exit(screen, challmode)

play() # Call the play() function

return

if event.type == pygame.MOUSEMOTION:

mouse\_x, mouse\_y = pygame.mouse.get\_pos()

back\_button.changeColor((mouse\_x, mouse\_y))

waiting\_input = False

pattern = []

current\_step = 0

score = 0

high\_score = get\_high\_score()

playing = True

start\_ticks = pygame.time.get\_ticks()

while True:

new\_game()

run\_game()

main\_menu() #DO NOT CHANGE

settings.py

# COLORS (r, g, b)

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

DARKGREY = (40, 40, 40)

LIGHTGREY = (100, 100, 100)

GREEN = (0, 255, 0)

DARKGREEN = (0, 155, 0)

BLUE = (0, 0, 255)

DARKBLUE = (0, 0, 155)

RED = (255, 0, 0)

DARKRED = (155, 0, 0)

YELLOW = (255, 255, 0)

DARKYELLOW = (155, 155, 0)

BGCOLOUR = BLACK

# game settings

WIDTH = 1280 #LOCK IN

HEIGHT = 720 #LOCK IN

FPS = 60

TITLE = "Brain Beat"

BUTTON\_SIZE = 200

ANIMATION\_SPEED = 20

BEEP1 = 880

BEEP2 = 659

BEEP3 = 554

BEEP4 = 440

sprites.py

import pygame

from settings import \*

import math

import numpy

pygame.mixer.init()

class Audio:

def \_\_init\_\_(self, frequency: int):

duration = 0.5

bits = 16

sample\_rate = 44100

total\_samples = int(round(duration \* sample\_rate))

data = numpy.zeros((total\_samples, 2), dtype=numpy.int16)

max\_sample = 2 \*\* (bits - 1) - 1

for sample in range(total\_samples):

sample\_time = float(sample) / sample\_rate

for channel in range(2):

data[sample][channel] = int(round(max\_sample \* math.sin(2 \* math.pi \* frequency \* sample\_time)))

self.sound = pygame.sndarray.make\_sound(data)

self.current\_channel = None

def play(self):

self.current\_channel = pygame.mixer.find\_channel(True)

self.current\_channel.play(self.sound)

def set\_volume(self, volume):

self.sound.set\_volume(volume)

button.py

import pygame

from settings import \*

from sprites import \*

import random

class Button():

def \_\_init\_\_(self, image, pos, text\_input, font, base\_color, hovering\_color):

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))

def update(self, screen):

if self.image is not None:

screen.blit(self.image, self.rect)

screen.blit(self.text, self.text\_rect)

def checkForInput(self, position):

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

def changeColor(self, position):

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)