**Documentation for**

Warrior's Wrath

**videogame**

Under Supervision of:

Dr:Hany Aly El-Ghaish

ENG:Ahmed Adel

Done by:

Hussain Amir

Ahmad fathi

ibrahim Elgmmal

Hossam Elkhateeb

**Pygame Initialization:**

pygame.init()

This line initializes the Pygame library, allowing you to use its functionalities for game development.

**Background Images:**

FIELD\_1\_BG = pygame.image.load("assets/images/background/mk3-stage-soul-chamber00.jpg")

FIELD\_2\_BG = pygame.image.load("assets/images/background/background2.jpg") FIELD\_3\_BG = pygame.image.load("assets/images/background/background3.jpg")

These lines load three different background images for different game fields (soul chamber, background2, and background3).

**Current Background:**

CURRENT\_BACKGROUND = FIELD\_2\_BG

This sets the initial background to **FIELD\_2\_BG** (background2). The game can switch between these backgrounds based on player choices.

**Music Loading and Volume Setting:**

pygame.mixer.music.load("assets/audio/menu\_music.mp3") pygame.mixer.music.set\_volume(0.5)

This loads the game's menu music from the specified file and sets its volume to 0.5 (50%).

**Game Window Initialization:**

SCREEN = pygame.display.set\_mode((1600, 900))

This line sets up the game window with a resolution of 1600x900 pixels.

**Window Caption:**

pygame.display.set\_caption("Warrior's Wrath")

This sets the caption or title of the game window to "Warrior's Wrath."

**Background Image Loading:**

BG = pygame.image.load("assets/l6Xgvsw.jpg")

This line loads another background image for the game window.

**Fade transition between menus**

def fade\_transition(screen, duration=400):  
 fade\_surface = pygame.Surface((screen.get\_width(), screen.get\_height()))  
 fade\_surface.fill((0, 0, 0))  
 for alpha in range(0, 255, 5):  
 fade\_surface.set\_alpha(alpha)  
 screen.blit(fade\_surface, (0, 0))  
 pygame.display.flip()  
 pygame.time.delay(duration // 70) # Adjust the delay for the desired transition speed

* **Create a Black Surface:**

fade\_surface = pygame.Surface((screen.get\_width(), screen.get\_height())) fade\_surface.fill((0, 0, 0))

This creates a black surface (**fade\_surface**) with the same dimensions as the game window (**screen**). The **fill((0, 0, 0))** sets the surface to be completely black.

* **Iterate Over Alpha Values:**

for alpha in range(0, 255, 5):

It iterates over a range of alpha values from 0 to 255 (exclusive) with a step of 5. The alpha value determines the transparency of the surface, where 0 is fully transparent, and 255 is fully opaque.

* **Set Alpha Value and Blit:**

fade\_surface.set\_alpha(alpha) screen.blit(fade\_surface, (0, 0))

For each iteration, it sets the alpha value of the black surface, making it partially transparent, and then it blits (draws) this surface onto the game window (**screen**) at coordinates (0, 0).

* **Update the Display:**

pygame.display.flip()

This updates the display to reflect the changes made in the loop. Without this, you wouldn't see the gradual fade effect until the loop is complete.

* **Delay for Transition Speed:**

pygame.time.delay(duration // 70)

This introduces a delay to control the speed of the transition. The **duration // 70** determines the delay in milliseconds. Adjusting this value can control how fast or slow the fade transition occurs.

def get\_font(size):

return pygame.font.Font("assets/ChrustyRock-ORLA.ttf", size)

This function is useful for obtaining a Pygame font object with a specific size and font style, allowing you to easily create text surfaces with consistent font settings in your game.

def load\_main\_menu\_music():  
 pygame.mixer.music.load("assets/audio/menu\_music.mp3")  
 pygame.mixer.music.set\_volume(0.5)

**This function loads the main menu music from the file "assets/audio/menu\_music.mp3" using pygame.mixer.music.load.**

**It then sets the volume of the loaded music to 0.5 using pygame.mixer.music.set\_volume.**  
def play\_main\_menu\_music():  
 pygame.mixer.music.play(-1) # -1 makes the music loop indefinitely

**This function plays the loaded main menu music in a loop indefinitely (-1 parameter) using pygame.mixer.music.play.**  
def stop\_main\_menu\_music():  
 pygame.mixer.music.stop()

**This function stops the currently playing music using pygame.mixer.music.stop**

def play\_intro\_video():  
 intro\_video\_path = "assets/gameintro1111.mp4" # Replace with the actual path to your video  
 intro\_clip = VideoFileClip(intro\_video\_path)  
 intro\_clip.preview(fps=60, audio\_fps=22050, audio\_buffersize=3000, audio\_nbytes=2, audio=True, fullscreen=True)

**intro\_video\_path:**

* + This variable holds the path to the introductory video file. It is set to "assets/gameintro1111.mp4" in the provided code. You should replace it with the actual path to your video file.

**intro\_clip = VideoFileClip(intro\_video\_path):**

* + This line creates a **VideoFileClip** object using the specified video file path. The **VideoFileClip** class is part of the MoviePy library, which is used for video editing with Python.

**intro\_clip.preview(...):**

* + This method plays a preview of the video clip. It takes several parameters to configure the preview:
    - **fps=60**: Specifies the frames per second for the preview.
    - **audio\_fps=22050**: Specifies the frames per second for the audio.
    - **audio\_buffersize=3000**: Specifies the buffer size for audio.
    - **audio\_nbytes=2**: Specifies the number of bytes per audio sample.
    - **audio=True**: Indicates that audio should be played.
    - **fullscreen=True**: Plays the video in fullscreen mode.

This function is responsible for rendering and displaying the selected fighters' names on the game screen

def display\_selected\_fighters(selected\_fighter\_1, selected\_fighter\_2):

SELECTED\_FIGHTER\_FONT = pygame.font.Font("assets/fonts/turok.ttf", 40)

# Display selected fighter for Player 1

if selected\_fighter\_1 is not None:

fighter\_1\_text = SELECTED\_FIGHTER\_FONT.render("Player 1: " + selected\_fighter\_1, True, (255, 255, 255))

fighter\_1\_rect = fighter\_1\_text.get\_rect(center=(800, 100))

SCREEN.blit(fighter\_1\_text, fighter\_1\_rect)

# Display selected fighter for Player 2

if selected\_fighter\_2 is not None:

fighter\_2\_text = SELECTED\_FIGHTER\_FONT.render("Player 2: " + selected\_fighter\_2, True, (255, 255, 0))

fighter\_2\_rect = fighter\_2\_text.get\_rect(center=(640, 100))

SCREEN.blit(fighter\_2\_text, fighter\_2\_rect)

**Font Initialization:**

* + **SELECTED\_FIGHTER\_FONT**: This line initializes a Pygame font for rendering text. It uses the Turok font file from the specified path ("assets/fonts/turok.ttf") and sets the font size to 40.

**Display Selected Fighter for Player 1:**

* + The first **if** statement checks if **selected\_fighter\_1** is not **None**. If a fighter is selected for Player 1, it proceeds to render the text.
  + **fighter\_1\_text**: This line renders the text using the selected fighter's name along with the prefix "Player 1:". The font, antialiasing (True), and text color (white) are specified.
  + **fighter\_1\_rect**: This line gets the rectangle that encloses the rendered text and centers it horizontally at (800, 100).
  + **SCREEN.blit(fighter\_1\_text, fighter\_1\_rect)**: This line blits (renders) the text surface onto the game screen at the specified rectangle position.

**Display Selected Fighter for Player 2:**

* + Similar to Player 1, this part checks if **selected\_fighter\_2** is not **None**. If a fighter is selected for Player 2, it proceeds to render the text.
  + **fighter\_2\_text**: This line renders the text for Player 2 with a different color (yellow).
  + **fighter\_2\_rect**: This line gets the rectangle for Player 2's text and centers it horizontally at (640, 100).
  + **SCREEN.blit(fighter\_2\_text, fighter\_2\_rect)**: This line blits the text surface for Player 2 onto the game screen.

The choose\_fighter function is responsible for allowing the players to choose their fighters before starting the game

def choose\_fighter():  
 selected\_fighter\_1 = None  
 selected\_fighter\_2 = None  
  
 while selected\_fighter\_1 is None or selected\_fighter\_2 is None:  
 FIGHTER\_MOUSE\_POS = pygame.mouse.get\_pos()  
  
 SCREEN.fill("black")  
  
 FIGHTER\_TEXT = get\_font(45).render("Choose Your Fighters!", True, "White")  
 FIGHTER\_RECT = FIGHTER\_TEXT.get\_rect(center=(800, 160))  
 SCREEN.blit(FIGHTER\_TEXT, FIGHTER\_RECT)  
  
 WARRIOR\_BUTTON = Button(image=pygame.image.load("assets/images/warrior/warrioricon.png"), pos=(300, 300),  
 text\_input="warrior", font=get\_font(75), base\_color="#d7fcd4",  
 hovering\_color="White")  
 Samurai\_BUTTON = Button(image=pygame.image.load("assets/images/Samurai/samuraiIcon.png"), pos=(1350, 300),  
 text\_input="Samurai", font=get\_font(75), base\_color="#d7fcd4",  
 hovering\_color="White")  
  
 WomenHunter\_BUTTON = Button(image=pygame.image.load("assets/images/womenhunter/womenhuntericon.png"), pos=(300, 700),  
 text\_input="WomenHunter", font=get\_font(50), base\_color="#d7fcd4",  
 hovering\_color="White")  
  
 SUBZERO\_BUTTON = Button(image=pygame.image.load("assets/images/sub-ZERO/newSUB-zero-icon.png"), pos=(1350, 700),  
 text\_input="Sub-zero", font=get\_font(75), base\_color="#d7fcd4", hovering\_color="White")  
  
  
  
 BACK\_BUTTON = Button(image=pygame.image.load("assets/Play Rect.png"), pos=(800, 800),  
 text\_input="Back", font=get\_font(75), base\_color="#d7fcd4", hovering\_color="White")  
  
 for button in [WARRIOR\_BUTTON, Samurai\_BUTTON, BACK\_BUTTON ,WomenHunter\_BUTTON,SUBZERO\_BUTTON]:  
 button.changeColor(FIGHTER\_MOUSE\_POS)  
 button.update(SCREEN)  
  
 display\_selected\_fighters(selected\_fighter\_1, selected\_fighter\_2)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if WARRIOR\_BUTTON.checkForInput(FIGHTER\_MOUSE\_POS):  
 if selected\_fighter\_1 is None:  
 selected\_fighter\_1 = "warrior"  
 elif selected\_fighter\_2 is None:  
 selected\_fighter\_2 = "warrior"  
 elif Samurai\_BUTTON.checkForInput(FIGHTER\_MOUSE\_POS):  
 if selected\_fighter\_1 is None:  
 selected\_fighter\_1 = "Samurai"  
 elif selected\_fighter\_2 is None:  
 selected\_fighter\_2 = "Samurai"  
 elif WomenHunter\_BUTTON.checkForInput(FIGHTER\_MOUSE\_POS):  
 if selected\_fighter\_1 is None:  
 selected\_fighter\_1 = "WomenHunter"  
 elif selected\_fighter\_2 is None:  
 selected\_fighter\_2 = "WomenHunter"  
 elif SUBZERO\_BUTTON.checkForInput(FIGHTER\_MOUSE\_POS):  
 if selected\_fighter\_1 is None:  
 selected\_fighter\_1 = "Sub-zero"  
 elif selected\_fighter\_2 is None:  
 selected\_fighter\_2 = "Sub-zero"  
 elif BACK\_BUTTON.checkForInput(FIGHTER\_MOUSE\_POS):  
 fade\_transition(SCREEN)  
  
 choose\_field\_menu()  
  
 if selected\_fighter\_1 is not None and selected\_fighter\_2 is not None:  
 stop\_main\_menu\_music()  
 fade\_transition(SCREEN)  
 play\_with\_fighters(selected\_fighter\_1, selected\_fighter\_2)  
  
  
# Define the pause menu text and font  
pause\_text\_font = get\_font(100)  
pause\_text = pause\_text\_font.render("Game Paused", True, (255, 0, 0))  
pause\_text\_rect = pause\_text.get\_rect(center=(SCREEN.get\_width() // 2, SCREEN.get\_height() // 2))  
  
# Create a surface for the pause menu  
pause\_menu\_surface = pygame.Surface((SCREEN.get\_width(), SCREEN.get\_height()), pygame.SRCALPHA)  
pause\_menu\_surface.fill((0, 0, 0, 128)) # Transparent black background  
  
# Variable to keep track of the game's pause state  
is\_paused = False  
  
# Global variable to track the pause state of the music  
is\_music\_paused = False

Explanation:

**Initialization:**

* + **selected\_fighter\_1** and **selected\_fighter\_2**: These variables store the names of the fighters selected by Player 1 and Player 2, respectively.

**Main Loop:**

* + The function enters a loop that continues until both players have selected their fighters (**selected\_fighter\_1** and **selected\_fighter\_2** are not **None**).

**Screen Setup:**

* + The function clears the screen with a black background and displays the title "Choose Your Fighters!" at the center of the screen.

**Button Creation:**

* + Buttons are created for each fighter, including warrior, Samurai, WomenHunter, Sub-zero, and a Back button.

**Button Update and Display:**

* + The buttons' colors are updated based on the mouse position, and their states are updated and displayed on the screen.

**Display Selected Fighters:**

* + The **display\_selected\_fighters** function is called to display the names of the selected fighters for Player 1 and Player 2.

**Event Handling:**

* + The function listens for user events, such as quitting the game or clicking the mouse.

**Fighter Selection:**

* + If a fighter button is clicked, the corresponding fighter is assigned to the respective player (**selected\_fighter\_1** or **selected\_fighter\_2**). The function ensures that each player can only select one fighter.

**Back Button:**

* + If the Back button is clicked, a fade transition is performed, and the game transitions back to the field selection menu (**choose\_field\_menu**).

**Game Start:**

* + If both players have selected their fighters, the main menu music is stopped, and a fade transition is performed. Then, the **play\_with\_fighters** function is called to start the game with the selected fighters.

This function is responsible for handling the main game loop where the actual gameplay takes place

def play\_with\_fighters(selected\_fighter\_1, selected\_fighter\_2):  
 global is\_paused, is\_music\_paused # Declare variables as global  
  
 is\_paused = False # Variable to keep track of the game's pause state  
 pause\_menu\_surface = pygame.Surface((SCREEN.get\_width(), SCREEN.get\_height()), pygame.SRCALPHA)  
 pause\_menu\_surface.fill((0, 0, 0, 128)) # Transparent black background  
 pygame.init()  
  
 while True:  
  
  
 # create game window  
 SCREEN\_WIDTH = 1600  
 SCREEN\_HEIGHT = 900  
  
 # define timer variables  
 timer\_font = pygame.font.Font("assets/fonts/turok.ttf", 50)  
 timer\_value = 30 # initial timer value in seconds  
 timer\_cooldown = 1000 # timer cooldown in milliseconds  
 timer\_last\_update = pygame.time.get\_ticks()  
 round\_over\_delay = 3000  
  
 screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))  
 pygame.display.set\_caption("Warrior's Wrath")  
  
 # set framerate  
 clock = pygame.time.Clock()  
 FPS = 60  
  
 # define colours  
 RED = (255, 0, 0)  
 WHITE = (255, 255, 255)  
 BLACK = (0, 0, 0)  
  
 # define game variables  
 intro\_count = 0  
 last\_count\_update = pygame.time.get\_ticks()  
 score = [0, 0] # player scores. [P1, P2]  
 round\_over = False  
 #ROUND\_OVER\_COOLDOWN = 2000  
  
 # define fighter variables  
 WARRIOR\_SIZE = 250  
 WARRIOR\_SCALE = 3  
 WARRIOR\_OFFSET = [112, 100]  
 WARRIOR\_DATA = [WARRIOR\_SIZE, WARRIOR\_SCALE, WARRIOR\_OFFSET]  
  
 Samurai\_SIZE = 250  
 Samurai\_SCALE = 3  
 Samurai\_OFFSET = [112, 100]  
 Samurai\_DATA = [Samurai\_SIZE, Samurai\_SCALE, Samurai\_OFFSET]  
  
 [subzero\_SIZE = 250](#size)  
 subzero\_SCALE = 1.75  
 subzero\_OFFSET = [112, 50]  
 subzero\_DATA = [subzero\_SIZE, subzero\_SCALE, subzero\_OFFSET]  
  
 WomenHunter\_SIZE = 250  
 WomenHunter\_SCALE = 3  
 WomenHunter\_OFFSET = [112, 95]  
 WomenHunter\_DATA = [WomenHunter\_SIZE, WomenHunter\_SCALE, WomenHunter\_OFFSET]  
  
 #HUNTRESS\_SIZE = 162  
 #HUNTRESS\_SCALE = 4  
 #HUNTRESS\_OFFSET = [112, 59]  
 #HUNTRESS\_DATA = [HUNTRESS\_SIZE, HUNTRESS\_SCALE, HUNTRESS\_OFFSET]  
  
 # load music and sounds  
 pygame.mixer.music.load("assets/audio/game\_music.mp3")  
 pygame.mixer.music.set\_volume(0.5)  
 pygame.mixer.music.play(-1, 0.0, 5000)  
  
 sword\_fx = pygame.mixer.Sound("assets/audio/sword.wav") #for warrior  
 sword\_fx.set\_volume(0.5)  
 magic\_fx = pygame.mixer.Sound("assets/images/Samurai/samurai-sword.mp3")  
 magic\_fx.set\_volume(0.75)  
 spear\_fx = pygame.mixer.Sound("assets/images/womenhunter/heavy-spear-hammer-large-[AudioTrimmer.com].wav")  
 spear\_fx.set\_volume(0.75)  
 punch\_fx = pygame.mixer.Sound("assets/images/sub-ZERO/punch-sound-effects-28649-[AudioTrimmer.com].mp3")  
 punch\_fx.set\_volume(0.75)  
  
 victory\_sound = pygame.mixer.Sound("assets/audio/victory sound.mp3")  
 victory\_sound.set\_volume(0.5)  
 # load background image  
  
  
 # load spritesheets  
 warrior\_sheet = pygame.image.load("assets/images/warrior/Sprites/warriorNEW.png").convert\_alpha()  
 Samurai\_sheet = pygame.image.load("assets/images/Samurai/Sprites/Samurai.png").convert\_alpha()  
 subzero\_sheet = pygame.image.load("assets/images/sub-ZERO/FINALSUBZERO.png").convert\_alpha()  
 WomenHunter\_sheet = pygame.image.load("assets/images/Samurai/Sprites/hunter.png").convert\_alpha()  
 #Huntress\_sheet = pygame.image.load("assets/images/Huntress/Sprites/Character/Huntressv4.png").convert\_alpha()  
  
 # load vicory image  
 victory\_img = pygame.image.load("assets/images/icons/victory.png").convert\_alpha()  
  
 # define number of steps in each animation  
 WARRIOR\_ANIMATION\_STEPS = [4, 8, 2, 4, 4, 3, 7]  
 Samurai\_ANIMATION\_STEPS = [8, 6, 2, 4, 4, 3, 8]  
 subzero\_ANIMATION\_STEPS = [8, 8, 4, 6, 8, 3, 8]  
 WomenHunter\_ANIMATION\_STEPS = [8, 8, 2, 5, 5, 3, 8]  
 #Huntress\_ANIMATION\_STEPS = [10, 8 ,1, 6, 6, 3, 10]  
  
 # define font  
 count\_font = pygame.font.Font("assets/fonts/turok.ttf", 80)  
 score\_font = pygame.font.Font("assets/fonts/turok.ttf", 30)

Explanation

**Initialization:**

* + **is\_paused**: Variable to keep track of the game's pause state.
  + **pause\_menu\_surface**: A transparent black surface used for creating a pause menu overlay.
  + The **pygame.init()** function is called to initialize Pygame.

**Game Loop:**

* + The function enters a game loop (**while True:**) that continues indefinitely until manually exited.

**Window Setup:**

* + The game window is set up with a specific width and height.

**Timer Variables:**

* + Variables related to the in-game timer are defined, including the font, initial timer value, cooldown, and last update time.

**Framerate Setup:**

* + The Pygame clock is set up to control the framerate.

**Color Definitions:**

* + Colors (RED, WHITE, BLACK) are defined for later use.

**Game Variables:**

* + Variables like **intro\_count**, **last\_count\_update**, **score**, and **round\_over** are initialized.

**Fighter Variables:**

* + Size, scale, and offset values for each fighter are defined.

**Music and Sound Loading:**

* + Background music (**game\_music.mp3**) and various sound effects for different fighters are loaded.

**Spritesheets and Victory Image Loading:**

* + Spritesheets for each fighter and a victory image are loaded.

**Animation Steps:**

* + The number of animation steps for each fighter is defined.

**Font Definitions:**

* + Fonts for the in-game count and score are defined.

**determine the “fighter”\_SIZE value**

2000 pixel



1750 pixel

2000 pixel



For each animation picture which is 8 in this sprite sheet

So 2000÷8= 250

So subzero size will be 250 pixel width and height

This function is designed to draw a simple health bar on the game screen.

def draw\_health\_bar(health, x, y):  
 ratio = health / 100  
 pygame.draw.rect(screen, WHITE, (x - 2, y - 2, 404, 34))  
 pygame.draw.rect(screen, BLACK, (x, y, 400, 30))  
 pygame.draw.rect(screen, RED, (x, y, 400 \* ratio, 30))

**Ratio Calculation:**

* + **ratio = health / 100**: Calculates the ratio of the current health to the maximum health. This ratio is used to determine the width of the filled portion of the health bar.

**Drawing the Health Bar:**

* + **pygame.draw.rect(screen, WHITE, (x - 2, y - 2, 404, 34))**: Draws a white rectangle as the outline of the health bar, slightly larger than the actual bar for a border effect.
  + **pygame.draw.rect(screen, BLACK, (x, y, 400, 30))**: Draws a black rectangle as the background of the health bar. This rectangle represents the total health.
  + **pygame.draw.rect(screen, RED, (x, y, 400 \* ratio, 30))**: Draws a red rectangle representing the actual health based on the calculated ratio. The width of this rectangle is proportional to the health percentage.

**Parameters:**

* + **health**: The current health value.
  + **x, y**: The coordinates where the top-left corner of the health bar should be drawn on the screen.

This part is using conditional expressions (also known as ternary operators) to create instances of the **Fighter** class based on the selected fighters (**selected\_fighter\_1** and **selected\_fighter\_2**)

fighter\_1 = Fighter(1, 200, 700, False, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx) \  
 if selected\_fighter\_1 == "warrior" else Fighter(1, 200, 700, False, Samurai\_DATA, Samurai\_sheet,  
 Samurai\_ANIMATION\_STEPS, magic\_fx) \  
 if selected\_fighter\_1 == "Samurai" else Fighter(1, 200, 700, False, WomenHunter\_DATA, WomenHunter\_sheet,  
 WomenHunter\_ANIMATION\_STEPS, spear\_fx) \  
 if selected\_fighter\_1 == "WomenHunter" else Fighter(1, 200, 700, False, subzero\_DATA, subzero\_sheet,  
 subzero\_ANIMATION\_STEPS, punch\_fx)  
  
fighter\_2 = Fighter(2, 1400, 700, True, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx) \  
 if selected\_fighter\_2 == "warrior" else Fighter(2, 1400, 700, True, Samurai\_DATA, Samurai\_sheet,  
 Samurai\_ANIMATION\_STEPS, magic\_fx) \  
 if selected\_fighter\_2 == "Samurai" else Fighter(2, 1400, 700, True, WomenHunter\_DATA, WomenHunter\_sheet,  
 WomenHunter\_ANIMATION\_STEPS, spear\_fx) \  
 if selected\_fighter\_2 == "WomenHunter" else Fighter(2, 1400, 700, True, subzero\_DATA, subzero\_sheet,  
 subzero\_ANIMATION\_STEPS, punch\_fx)

**This part of the code represents the main game loop, which is the core structure of the game that repeats continuously, updating the game state, handling user input, and rendering the graphics.**

# game loop  
run = True  
while True:  
  
 clock.tick(FPS)  
  
 # draw background  
 draw\_bg()  
  
 # show player stats  
 draw\_health\_bar(fighter\_1.health, 50, 30)  
 draw\_health\_bar(fighter\_2.health, 1140, 30)  
 draw\_text("P1: " + str(score[0]), score\_font, RED, 50, 60)  
 draw\_text("P2: " + str(score[1]), score\_font, RED, 1090, 60)  
  
 # draw the "Back to Menu" button  
 BACK\_TO\_MENU\_BUTTON.changeColor(pygame.mouse.get\_pos())  
 BACK\_TO\_MENU\_BUTTON.update(screen)  
  
 # display countdown timer  
 draw\_text(str(timer\_value), timer\_font, RED, 800,100)  
  
 # update countdown  
 current\_time = pygame.time.get\_ticks()  
  
 if not is\_paused: # Only update timer and fighters if the game is not paused  
 if not round\_over and timer\_value > 0:  
 if current\_time - timer\_last\_update >= timer\_cooldown:  
 timer\_value -= 1  
 timer\_last\_update = current\_time  
  
 if timer\_value <= 0:  
 if not round\_over:  
 round\_over = True  
 round\_over\_start\_time = pygame.time.get\_ticks()  
 if fighter\_1.health < fighter\_2.health:  
 fighter\_1.health = 0  
 else:  
 fighter\_2.health = 0  
  
 if round\_over:  
 if current\_time - round\_over\_start\_time >= round\_over\_delay:  
 round\_over = False  
 intro\_count = 3  
  
 choose\_fighter()  
  
  
 # draw fighters  
 fighter\_2.draw(SCREEN)  
 fighter\_1.draw(screen)  
  
 # check for player defeat  
 if not round\_over:  
 if not fighter\_1.alive:  
 score[1] += 1  
 round\_over = True  
 round\_over\_start\_time = pygame.time.get\_ticks()  
 elif not fighter\_2.alive:  
 score[0] += 1  
 round\_over = True  
 round\_over\_start\_time = pygame.time.get\_ticks()  
 else:  
 if not victory\_sound.play(0, 3000):  
 # display victory image  
 screen.blit(victory\_img, (650, 150))  
  
 # event handler  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False  
 pygame.quit()  
 sys.exit()  
 elif event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_ESCAPE:  
 is\_paused = not is\_paused  
 if is\_paused:  
 pygame.mixer.music.pause()  
 is\_music\_paused = True  
  
 else:  
 pygame.mixer.music.unpause()  
 is\_music\_paused = False  
  
 elif event.type == pygame.MOUSEBUTTONDOWN:  
 if BACK\_TO\_MENU\_BUTTON.checkForInput(pygame.mouse.get\_pos()):  
 play\_main\_menu\_music()  
 fade\_transition(SCREEN)  
 choose\_fighter()  
  
 if is\_paused:  
 # Draw the pause menu  
 SCREEN.blit(pause\_menu\_surface, (0, 0))  
 SCREEN.blit(pause\_text, pause\_text\_rect)  
 else:  
 if intro\_count <= 0:  
 # move fighters  
 fighter\_1.move(SCREEN\_WIDTH, SCREEN\_HEIGHT, screen, fighter\_2, round\_over)  
 fighter\_2.move(SCREEN\_WIDTH, SCREEN\_HEIGHT, screen, fighter\_1, round\_over)  
 else:  
 # display count timer  
 draw\_text(str(intro\_count), count\_font, RED, SCREEN\_WIDTH / 2, SCREEN\_HEIGHT / 3)  
 # update count timer  
 if (pygame.time.get\_ticks() - last\_count\_update) >= 1000:  
 intro\_count -= 1  
 last\_count\_update = pygame.time.get\_ticks()  
  
 # update fighters  
 fighter\_1.update()  
 fighter\_2.update()  
  
  
 # update display  
 pygame.display.update()  
  
# exit pygame  
pygame.quit()

Explanation:

**Clock Tick:**

* + **clock.tick(FPS)** controls the frame rate, making the game run at a specific frames-per-second rate (**FPS**).

**Drawing:**

* + **draw\_bg()** draws the game background.
  + **draw\_health\_bar()** displays the health bars for both players.
  + **draw\_text()** displays text for player scores and the countdown timer.
  + The "Back to Menu" button is drawn and updated.

**Countdown Timer:**

* + The countdown timer (**timer\_value**) is displayed and updated.

**Game Logic Updates:**

* + The countdown timer is updated based on the elapsed time.
  + The game logic updates only if the game is not paused, checking for round over conditions and player defeats.

**Rendering Fighters:**

* + The fighters are drawn on the screen.

**Event Handling:**

* + The game handles events such as quitting, key presses (e.g., the escape key to toggle pause), and mouse clicks on the "Back to Menu" button.

**Pause Handling:**

* + If the game is paused, a pause menu is displayed.

**Fighter Movement and Update:**

* + If the countdown is over, the fighters' movement and updates are processed.

**Update Display:**

* + The display is updated, reflecting the changes made during the frame.

**Game Exit:**

* + The game loop continues indefinitely until the user quits the game, and then it exits pygame.

This part of the code defines a function **how\_to\_play()** that represents the screen providing information on how to play the game

def how\_to\_play():  
 while True:  
 how\_to\_play\_MOUSE\_POS = pygame.mouse.get\_pos()  
  
 how\_to\_play\_TEXT = get\_font(45).render("This is the how\_to\_play screen.", True, "Black")  
 how\_to\_play\_RECT = how\_to\_play\_TEXT.get\_rect(center=(640, 260))  
 SCREEN.blit(how\_to\_play\_TEXT, how\_to\_play\_RECT)  
  
  
  
 # Load the "How to play" image  
 how\_to\_play\_image = pygame.image.load("assets/how-to-play00000.png")  
 screen\_rect = SCREEN.get\_rect()  
 image\_rect = how\_to\_play\_image.get\_rect(center=(screen\_rect.centerx, screen\_rect.centery))  
 SCREEN.blit(how\_to\_play\_image, image\_rect)  
  
 how\_to\_play\_BACK = Button(image=None, pos=(800, 850),  
 text\_input="BACK", font=get\_font(75), base\_color="white", hovering\_color="Green")  
  
 how\_to\_play\_BACK.changeColor(how\_to\_play\_MOUSE\_POS)  
 how\_to\_play\_BACK.update(SCREEN)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 fade\_transition(SCREEN)  
 pygame.quit()  
 sys.exit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 if how\_to\_play\_BACK.checkForInput(how\_to\_play\_MOUSE\_POS):  
 fade\_transition(SCREEN)  
 main\_menu()  
  
 pygame.display.update()

Explanation:

**Mouse Position:**

* + **how\_to\_play\_MOUSE\_POS = pygame.mouse.get\_pos()** retrieves the current mouse position.

**Render Text:**

* + **how\_to\_play\_TEXT** renders a text message, and **how\_to\_play\_RECT** gets its rectangular dimensions.
  + The text is displayed at the center of the screen using **SCREEN.blit()**.

**Load and Display Image:**

* + An image ("How to play") is loaded and displayed at the center of the screen.
  + The image is loaded using **pygame.image.load()**, and its position is set to the center of the screen.

**Create and Display Button:**

* + A "BACK" button is created using the **Button** class, and its appearance is updated based on mouse hover.
  + The button is displayed using **SCREEN.blit()**.

**Event Handling:**

* + The function handles events such as quitting the game and mouse clicks.
  + If the "BACK" button is clicked, the **main\_menu()** function is called, and a fade transition is initiated.

**Update Display:**

* + **pygame.display.update()** updates the display to reflect the changes made during the frame.