**PYTHON PROJECT SYNOPSIS**

ANMOL PRABHAKAR

23FE10CSE00218

CSE CORE 2ND YEAR ‘L’

**TOPIC: 2D-COMBAT-GAME USING PYTHON.**

The primary goal of this project is to design and implement an interactive 2D combat game that demonstrates the application of Python programming and the Pygame module. The project aims to provide a fun and engaging gameplay experience while showcasing the capabilities of Python in game development.

This project uses Pygame, a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries, which are essential for building dynamic and interactive games.

Pygame makes it easier to manage multimedia, game loops, sprites, and user input.

Features Of Pygame module:

1. Character Controls: Players can move characters using keyboard inputs to attack, defend, and navigate the arena.
2. Graphics and Animations: The game features 2D graphics created using Pygame’s rendering capabilities, including sprites for characters and backgrounds.
3. Game Mechanics:
   * Health and damage indicators.
   * Dynamic scoring system based on player performance.
   * Multiple levels or increasing difficulty.
4. Sound Effects: Background music and combat sound effects are integrated using Pygame's audio library.
5. Collision Detection: Smooth and realistic interactions between the player and the environment/enemies.

Implementation:  
The project was developed using Python in the VS Code environment. Pygame was utilized for:

* Loading and manipulating graphics (sprites and backgrounds).
* Capturing user inputs like keystrokes.
* Managing game loops for rendering and updating frames.
* Implementing collision detection and physics for a realistic combat feel.

**Procedure I followed :**

To get started, I downloaded and installed the Pygame library in Visual Studio Code (VS Code) by running the following command in the terminal:

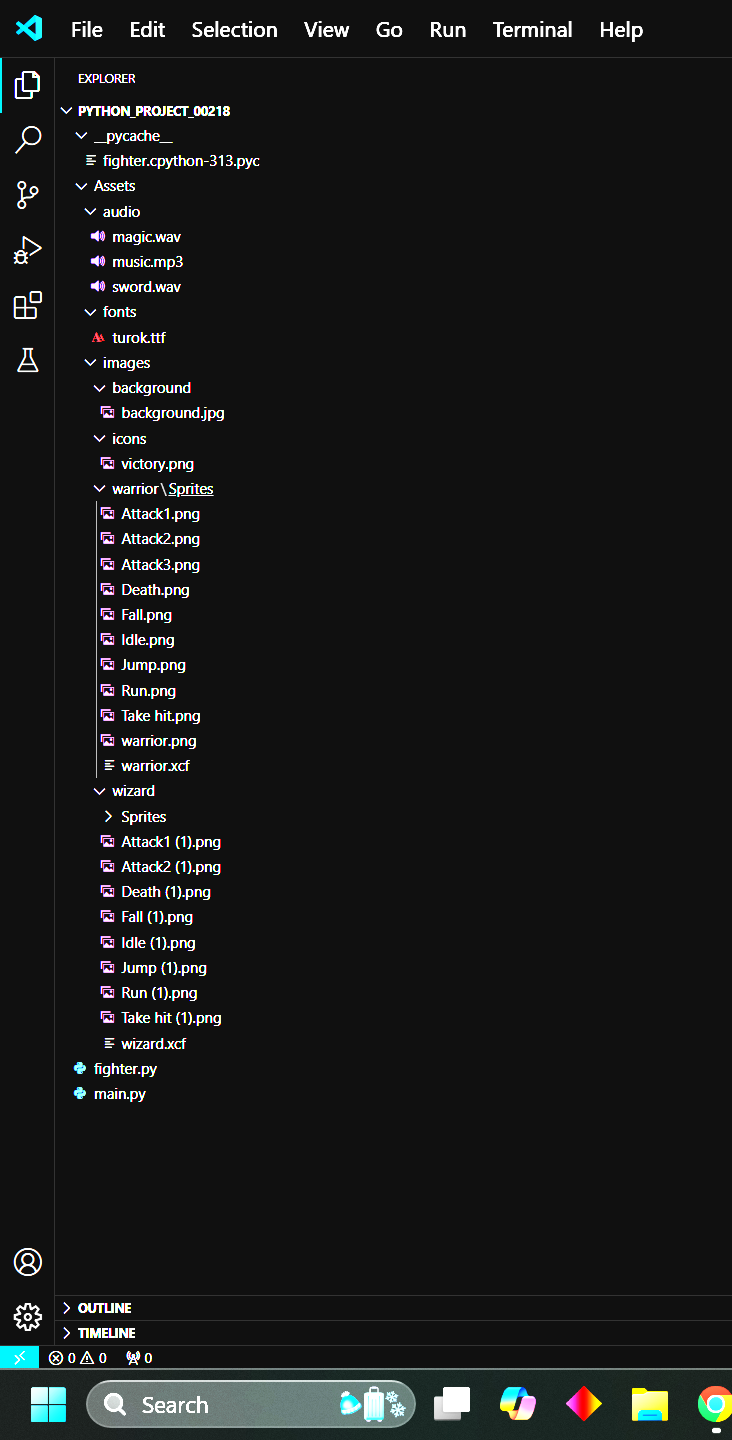
Command in terminal : pip install pygame

Upon Downloading the pip[Preffered installer Programs] of pygame , I started to gather the assests such as :

* BackGround Image.
* Player 1 [ warrior ] Animation.
* Player 2 [ Wizard] Animation.
* Animations of Run , Jump , Attack , Death.
* Sound effect for weapons.
* Background Music

Then I organised the assests , named the file for easier access in programs and started to code.

[ As pygame module includes New and unique functions specific for Gaming , I took help from Resources such as Youtube , Pygame Tutorial , and AI for efficient learing ]

ORGANIZATION OF ASSESTS AND FILES :  
  


Code in main.py :  
import pygame # Import the Pygame library for graphics and game development.

from pygame import mixer # Import the mixer module for handling sounds.

from fighter import Fighter # Import the custom Fighter class.

# Initialize the mixer (for sounds) and Pygame library.

mixer.init()

pygame.init()

# Create the game window.

SCREEN\_WIDTH = 1000 # Width of the game window in pixels.

SCREEN\_HEIGHT = 600 # Height of the game window in pixels.

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT)) # Initialize the game display.

pygame.display.set\_caption("ANMOL\_PRABHAKAR : 23FE10CSE00218") # Set the title of the game window.

# Set the framerate of the game.

clock = pygame.time.Clock() # Create a clock object to manage game updates.

FPS = 60 # Define the game frame rate (frames per second).

# Define some colors using RGB format.

RED = (255, 0, 0) # Color for health bar background.

YELLOW = (255, 255, 0) # Color for the health indicator.

WHITE = (255, 255, 255) # Border color.

# Define game variables.

intro\_count = 3 # Countdown before the match starts.

last\_count\_update = pygame.time.get\_ticks() # Time of the last countdown update.

score = [0, 0] # Player scores: [Player 1, Player 2].

round\_over = False # Flag to check if the round has ended.

ROUND\_OVER\_COOLDOWN = 2000 # Time (in ms) to reset the round.

# Define fighter attributes for scaling and positioning.

WARRIOR\_SIZE = 162 # Original sprite size for the Warrior.

WARRIOR\_SCALE = 4 # Scaling factor for the Warrior sprite.

WARRIOR\_OFFSET = [72, 56] # Positional offset for centering the Warrior sprite.

WARRIOR\_DATA = [WARRIOR\_SIZE, WARRIOR\_SCALE, WARRIOR\_OFFSET] # Bundle Warrior's attributes.

WIZARD\_SIZE = 250 # Original sprite size for the Wizard.

WIZARD\_SCALE = 3 # Scaling factor for the Wizard sprite.

WIZARD\_OFFSET = [112, 107] # Positional offset for centering the Wizard sprite.

WIZARD\_DATA = [WIZARD\_SIZE, WIZARD\_SCALE, WIZARD\_OFFSET] # Bundle Wizard's attributes.

# Load background music and sound effects.

pygame.mixer.music.load("assets/audio/music.mp3") # Load background music.

pygame.mixer.music.set\_volume(1.0) # Set volume for the background music.

pygame.mixer.music.play(-1, 0.0, 5000) # Loop the music indefinitely with a delay.

sword\_fx = pygame.mixer.Sound("assets/audio/sword.wav") # Load sword attack sound effect.

sword\_fx.set\_volume(2.0) # Set volume for the sword sound effect.

magic\_fx = pygame.mixer.Sound("assets/audio/magic.wav") # Load magic attack sound effect.

magic\_fx.set\_volume(2.0) # Set volume for the magic sound effect.

# Load the background image.

bg\_image = pygame.image.load("assets/images/background/background.jpg").convert\_alpha() # Background image.

# Load sprite sheets for fighters.

warrior\_sheet = pygame.image.load("assets/images/warrior/Sprites/warrior.png").convert\_alpha() # Warrior sprite sheet.

wizard\_sheet = pygame.image.load("assets/images/wizard/Sprites/wizard.png").convert\_alpha() # Wizard sprite sheet.

# Load victory image for the end of the round.

victory\_img = pygame.image.load("assets/images/icons/victory.png").convert\_alpha()

# Define animation steps for each action of both fighters.

WARRIOR\_ANIMATION\_STEPS = [10, 8, 1, 7, 7, 3, 7] # Warrior's animation frames for [idle, run, jump, attacks, etc.].

WIZARD\_ANIMATION\_STEPS = [8, 8, 1, 8, 8, 3, 7] # Wizard's animation frames.

# Define fonts for text rendering.

count\_font = pygame.font.Font("assets/fonts/turok.ttf", 80) # Large font for countdown.

score\_font = pygame.font.Font("assets/fonts/turok.ttf", 30) # Smaller font for scores.

# Function to draw text on the screen.

def draw\_text(text, font, text\_col, x, y):

img = font.render(text, True, text\_col) # Render text with anti-aliasing.

screen.blit(img, (x, y)) # Display the text at the specified position.

# Function to draw the background.

def draw\_bg():

scaled\_bg = pygame.transform.scale(bg\_image, (SCREEN\_WIDTH, SCREEN\_HEIGHT)) # Scale the background to fit the screen.

screen.blit(scaled\_bg, (0, 0)) # Draw the scaled background.

# Function to draw health bars for fighters.

def draw\_health\_bar(health, x, y):

ratio = health / 100 # Calculate the health-to-bar ratio.

pygame.draw.rect(screen, WHITE, (x - 2, y - 2, 404, 34)) # Draw white border around the health bar.

pygame.draw.rect(screen, RED, (x, y, 400, 30)) # Draw the background (red) health bar.

pygame.draw.rect(screen, YELLOW, (x, y, 400 \* ratio, 30)) # Draw the filled (yellow) health bar.

# Create two fighter instances with their respective attributes and sprites.

fighter\_1 = Fighter(1, 200, 310, False, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx) # Player 1.

fighter\_2 = Fighter(2, 700, 310, True, WIZARD\_DATA, wizard\_sheet, WIZARD\_ANIMATION\_STEPS, magic\_fx) # Player 2.

# Main game loop.

run = True

while run:

clock.tick(FPS) # Control the game's frame rate.

draw\_bg() # Draw the background.

# Display health bars and scores for both players.

draw\_health\_bar(fighter\_1.health, 20, 20)

draw\_health\_bar(fighter\_2.health, 580, 20)

draw\_text("P1: " + str(score[0]), score\_font, RED, 20, 60)

draw\_text("P2: " + str(score[1]), score\_font, RED, 580, 60)

# Handle pre-match countdown.

if intro\_count <= 0:

# Allow fighters to move if the countdown is over.

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 the countdown timer.

draw\_text(str(intro\_count), count\_font, RED, SCREEN\_WIDTH / 2, SCREEN\_HEIGHT / 3)

# Update the countdown every second.

if (pygame.time.get\_ticks() - last\_count\_update) >= 1000:

intro\_count -= 1

last\_count\_update = pygame.time.get\_ticks()

# Update and draw fighters.

fighter\_1.update()

fighter\_2.update()

fighter\_1.draw(screen)

fighter\_2.draw(screen)

# Check if a player is defeated and handle round logic.

if not round\_over:

if not fighter\_1.alive: # Player 1 is defeated.

score[1] += 1

round\_over = True

round\_over\_time = pygame.time.get\_ticks()

elif not fighter\_2.alive: # Player 2 is defeated.

score[0] += 1

round\_over = True

round\_over\_time = pygame.time.get\_ticks()

else:

# Display victory image if the round is over.

screen.blit(victory\_img, (360, 150))

# Reset fighters and round after the cooldown.

if pygame.time.get\_ticks() - round\_over\_time > ROUND\_OVER\_COOLDOWN:

round\_over = False

intro\_count = 3

fighter\_1 = Fighter(1, 200, 310, False, WARRIOR\_DATA, warrior\_sheet, WARRIOR\_ANIMATION\_STEPS, sword\_fx)

fighter\_2 = Fighter(2, 700, 310, True, WIZARD\_DATA, wizard\_sheet, WIZARD\_ANIMATION\_STEPS, magic\_fx)

# Handle quit events.

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

pygame.display.update() # Refresh the game display.

# Quit Pygame when the game loop ends.pygame.quit()

Code in fighter.py :  
import pygame # Import Pygame library for handling graphics and animations.

# Fighter class to manage player behavior, animations, and interactions.

class Fighter():

def \_\_init\_\_(self, player, x, y, flip, data, sprite\_sheet, animation\_steps, sound):

"""

Initializes a fighter with given attributes and data.

Parameters:

player (int): Player number (1 or 2) to distinguish controls.

x, y (int): Initial position of the fighter on the screen.

flip (bool): Whether to horizontally flip the sprite.

data (list): Fighter-specific data [sprite size, scale, offset].

sprite\_sheet (Surface): The sprite sheet containing all animations.

animation\_steps (list): Number of frames for each animation.

sound (Sound): Attack sound effect for the fighter.

"""

self.player = player

self.size = data[0] # Size of each sprite in the sheet.

self.image\_scale = data[1] # Scaling factor for the sprite.

self.offset = data[2] # Positional offset for centering sprites.

self.flip = flip # Whether the sprite should face the opposite direction.

self.animation\_list = self.load\_images(sprite\_sheet, animation\_steps) # Preload animations.

self.action = 0 # Current action (0: idle, 1: run, 2: jump, etc.).

self.frame\_index = 0 # Index of the current animation frame.

self.image = self.animation\_list[self.action][self.frame\_index] # Current sprite image.

self.update\_time = pygame.time.get\_ticks() # Time since last animation update.

self.rect = pygame.Rect((x, y, 80, 180)) # Collision box for the fighter.

self.vel\_y = 0 # Vertical velocity for jump physics.

self.running = False # Is the fighter running?

self.jump = False # Is the fighter mid-jump?

self.attacking = False # Is the fighter attacking?

self.attack\_type = 0 # Type of attack (1 or 2).

self.attack\_cooldown = 0 # Cooldown before next attack.

self.attack\_sound = sound # Attack sound effect.

self.hit = False # Has the fighter been hit?

self.health = 100 # Fighter's health.

self.alive = True # Is the fighter still alive?

def load\_images(self, sprite\_sheet, animation\_steps):

"""

Extracts animation frames from a sprite sheet.

Parameters:

sprite\_sheet (Surface): The sprite sheet containing animations.

animation\_steps (list): Number of frames in each animation row.

Returns:

list: Nested list containing all animations as scaled images.

"""

animation\_list = []

for y, animation in enumerate(animation\_steps):

temp\_img\_list = []

for x in range(animation):

temp\_img = sprite\_sheet.subsurface(

x \* self.size, y \* self.size, self.size, self.size

) # Extract frame.

temp\_img\_list.append(

pygame.transform.scale(

temp\_img,

(self.size \* self.image\_scale, self.size \* self.image\_scale),

)

) # Scale the image.

animation\_list.append(temp\_img\_list) # Append animation frames.

return animation\_list

def move(self, screen\_width, screen\_height, surface, target, round\_over):

"""

Handles movement, jumping, and attacking logic for the fighter.

"""

SPEED = 10 # Movement speed.

GRAVITY = 2 # Gravity for jump mechanics.

dx = 0 # Change in x-position.

dy = 0 # Change in y-position.

self.running = False # Reset running state.

self.attack\_type = 0 # Reset attack type.

# Get key inputs for player controls.

key = pygame.key.get\_pressed()

# Only allow movement and actions if not attacking, alive, and round is active.

if self.attacking == False and self.alive == True and round\_over == False:

if self.player == 1: # Player 1 controls.

if key[pygame.K\_a]: # Move left.

dx = -SPEED

self.running = True

if key[pygame.K\_d]: # Move right.

dx = SPEED

self.running = True

if key[pygame.K\_w] and not self.jump: # Jump.

self.vel\_y = -30

self.jump = True

if key[pygame.K\_r] or key[pygame.K\_t]: # Attack.

self.attack(target)

self.attack\_type = 1 if key[pygame.K\_r] else 2

elif self.player == 2: # Player 2 controls.

if key[pygame.K\_LEFT]:

dx = -SPEED

self.running = True

if key[pygame.K\_RIGHT]:

dx = SPEED

self.running = True

if key[pygame.K\_UP] and not self.jump:

self.vel\_y = -30

self.jump = True

if key[pygame.K\_KP1] or key[pygame.K\_KP2]:

self.attack(target)

self.attack\_type = 1 if key[pygame.K\_KP1] else 2

# Apply gravity.

self.vel\_y += GRAVITY

dy += self.vel\_y

# Ensure the fighter stays within screen bounds.

if self.rect.left + dx < 0:

dx = -self.rect.left

if self.rect.right + dx > screen\_width:

dx = screen\_width - self.rect.right

if self.rect.bottom + dy > screen\_height - 110:

self.vel\_y = 0

self.jump = False

dy = screen\_height - 110 - self.rect.bottom

# Ensure the fighter faces their opponent.

self.flip = target.rect.centerx < self.rect.centerx

# Apply attack cooldown.

if self.attack\_cooldown > 0:

self.attack\_cooldown -= 1

# Update position.

self.rect.x += dx

self.rect.y += dy

def update(self):

"""

Updates the fighter's animation based on their current state.

"""

if self.health <= 0:

self.health = 0

self.alive = False

self.update\_action(6) # Death animation.

elif self.hit:

self.update\_action(5) # Hit animation.

elif self.attacking:

self.update\_action(3 if self.attack\_type == 1 else 4) # Attack animations.

elif self.jump:

self.update\_action(2) # Jump animation.

elif self.running:

self.update\_action(1) # Run animation.

else:

self.update\_action(0) # Idle animation.

animation\_cooldown = 50 # Time between frame updates (ms).

self.image = self.animation\_list[self.action][self.frame\_index] # Update image.

# Update frame index if cooldown has elapsed.

if pygame.time.get\_ticks() - self.update\_time > animation\_cooldown:

self.frame\_index += 1

self.update\_time = pygame.time.get\_ticks()

# Reset animation if it finishes.

if self.frame\_index >= len(self.animation\_list[self.action]):

if not self.alive:

self.frame\_index = len(self.animation\_list[self.action]) - 1

else:

self.frame\_index = 0

if self.action in {3, 4}: # Reset attack state.

self.attacking = False

self.attack\_cooldown = 20

if self.action == 5: # Reset hit state.

self.hit = False

self.attacking = False

self.attack\_cooldown = 20

def attack(self, target):

"""

Handles attacking logic, including hit detection.

"""

if self.attack\_cooldown == 0:

self.attacking = True

self.attack\_sound.play()

attacking\_rect = pygame.Rect(

self.rect.centerx - (2 \* self.rect.width \* self.flip),

self.rect.y,

2 \* self.rect.width,

self.rect.height,

) # Attack hitbox.

if attacking\_rect.colliderect(target.rect):

target.health -= 10

target.hit = True

def update\_action(self, new\_action):

"""

Changes the fighter's current action if it's different from the previous one.

"""

if new\_action != self.action:

self.action = new\_action

self.frame\_index = 0

self.update\_time = pygame.time.get\_ticks()

def draw(self, surface):

"""

Draws the fighter's current sprite on the screen.

"""

img = pygame.transform.flip(self.image, self.flip, False) # Flip sprite if needed.

surface.blit(img, (self.rect.x - (self.offset[0] \* self.image\_scale), self.rect.y - (self.offset[1] \* self.image\_scale)))

**CONCLUSION**

This mini-Python-Project introduced me to the vastness of python programming language and how it can be used to create interactive games using a wide set of libraries under different modules.  
I learned how to make animations using the combination of several images , how to mix sounds , manage interactions , Learned how to organize resources , downloading necessary assets , and most important exercise and improve my programming skills.