```python
#Our Code Currently
# Importing the Libraries and Modules
import pygame
import random
from GameEngine.settings import *
from PlayerAndEnemies.tile import Tile
from PlayerAndEnemies.player import Player
from Weapons.weapon import Slash, Shield
from Weapons.ranged import FireBall, Arrows
from Weapons.medkit import Medkit
from Utilities.ui import UI
from PlayerAndEnemies.enemy import Enemy, Boss
from Utilities.camera import YSortCameraGroup
from Utilities.debug import debug

# Level Class
class Level:

    # Initiatization Function
    def __init__(self, levelnum = 1, player_character = "Knight"):

        # Miscilaneous Data
        self.screen = pygame.display.get_surface()
        self.world_data = read_world_data(levelnum)
        self.player_type = player_character

        # Sprite Groups
        self.visible_sprites = YSortCameraGroup()
        self.obstacle_sprites = pygame.sprite.Group()

        self.healing_sprites = pygame.sprite.Group()

        self.attack_sprites = pygame.sprite.Group()
        self.attackable_sprites = pygame.sprite.Group()

        self.enemy_attack_sprites = pygame.sprite.Group()

        # HealthBar Setup
        self.ui = UI()
```

```
        #Attacks
        self.current_attack = None

        # Create the Maps
        self.create_map()

    # Checks to see the player's attacks are Connecting with Mobs
    def player_attack_logic(self):
        if self.attack_sprites:
            for attack_sprite in self.attack_sprites:
                collided_sprites =
pygame.sprite.spritecollide(attack_sprite, self.attackable_sprites, False)
                if collided_sprites:
                    for target_sprite in collided_sprites:
                        target_sprite.get_damaged(self.player)

    # Function to Damage the Player
    def damage_player(self, amount):
        if self.player.vulnerable:
            if self.player.shield != None:
                if amount - 5 < 0:
                    amount = 0
                else:
                    amount -= 5
            self.player.hp -= amount
            self.player.vulnerable = False
            self.player.hurt_time = pygame.time.get_ticks()

    # Create the Slash Attack
    def create_attack(self):
        self.current_attack = Slash(self.player, [self.visible_sprites,
self.attack_sprites])

    # Create the Fireball Attack
    def create_fireball(self):
        self.current_attack = FireBall(self.player, [self.visible_sprites,
self.attack_sprites])

    # Create the Player's Arrow Attack
    def player_arrow(self, pos):
```

```
        distance_sorted_list = []
        for sprite in sorted(self.attackable_sprites, key = lambda sprite:
sprite.distance):
            distance_sorted_list.append(sprite)

        arrow = Arrows([self.visible_sprites, self.attack_sprites], pos,
distance_sorted_list[0], self.obstacle_sprites)
        return arrow

    # Checks to See if the Player is colliding with a healthblock or not
    def player_heal_logic(self):
        if self.healing_sprites:
            collided_sprites = pygame.sprite.spritecollide(self.player,
self.healing_sprites, False)
            if collided_sprites:
                for health_block in collided_sprites:
                    self.player.hp = self.player.data["Health"]
                    health_block.kill()

    # Create the Mob's Arrow Attack
    def mob_arrow(self, pos):
        arrow = Arrows([self.visible_sprites, self.enemy_attack_sprites],
pos, self.player, self.obstacle_sprites)
        return arrow

    # Create the Player's shield
    def shield(self):
        shield = Shield(self.player, [self.visible_sprites])
        return shield

    # Destroy the Slash Attack
    def destroy_weapon(self):
        if self.current_attack:
            self.current_attack.kill()
        self.current_attack = None

    # Checks to See if the Skeleton's arrow are colliding with the player
    def mob_attack_logic(self):
        if self.enemy_attack_sprites:
```

```
            for attack_sprite in self.enemy_attack_sprites:
                if attack_sprite.rect.colliderect(self.player.rect):
                    collided_sprites =
pygame.sprite.spritecollide(self.player, self.enemy_attack_sprites, False)
                    if collided_sprites:
                        for target_sprite in collided_sprites:
                            self.damage_player(5)
                            attack_sprite.kill()

    # Create the Maps for the Level to Include Tiles, Players, Enemy, and
Boss.
    def create_map(self):
        for yi, row in enumerate(self.world_data):
            for xi, col in enumerate(row):
                x = xi * TILE_SIZE
                y = yi * TILE_SIZE

                if int(col) >= 0 and int(col) != 15 and int(col) != 16 and
int(col) != 19 and int(col) != 10:
                    Tile((x,y), [self.obstacle_sprites,
self.visible_sprites], int(col))
                if int(col) == 19:
                    Medkit(x, y, [self.visible_sprites,
self.healing_sprites])
                if int(col) == 10:
                    Boss((x, y), [self.visible_sprites,
self.attackable_sprites], self.obstacle_sprites, self.damage_player)
                if int(col) == 16:
                    monster_types = ["Zombie", "Skeleton", "Slime"]
                    Enemy(random.choice(monster_types), (x, y),
[self.visible_sprites, self.attackable_sprites]
                          , self.obstacle_sprites, self.damage_player,
self.mob_arrow)
                if int(col) == 15:
                    self.player = Player((x, y), [self.visible_sprites],
self.obstacle_sprites,
                                         self.player_type,
self.create_attack,
                                         self.destroy_weapon, self.shield,
```

```
                                        self.create_fireball,
self.player_arrow)

    # Displays the Level
    def run(self):
        self.visible_sprites.custom_draw(self.player)
        self.visible_sprites.update()
        self.visible_sprites.enemy_update(self.player)
        self.player_attack_logic()
        self.mob_attack_logic()
        self.player_heal_logic()
        self.ui.display(self.player)

# Import Libraries
import csv
import pygame

# Initialize Font
pygame.font.init()

#Use Tiled
color_constants = {"Black": (0, 0, 0),
    "White": (255, 255, 255),
    "Red": (255, 0, 0),
    "Green": (0, 255, 0),
    "Blue": (0, 0, 255),
    "Yellow": (255, 255, 0),
    "Purple": (128, 0, 128),
    "Orange": (255, 165, 0)}

# General Info
FPS = 60
SCREEN_WIDTH, SCREEN_HEIGHT = (1000, 800)
TILE_SIZE = 1000 // 16

#UI Info
BAR_HEIGHT = 20
HEALTH_BAR_WIDTH = 200
UI_FONT = "Arial"
UI_FONT_SIZE = 18
```

```
HEALTH_COLOR = 'red'
UI_BG_COLOR = "#222222"
TEXT_COLOR = "#EEEEEE"
UI_BORDER_COLOR = "#111111"
WATER_COLOR = "#71ddee"

# Credit Font and Text Label Creation
credit_font = pygame.font.SysFont("Arial", 35)
def blit_text(surface, width, text, pos, font,
color=pygame.Color('black')):
    words = [word.split(' ') for word in text.splitlines()]  # 2D array
where each row is a list of words.
    space = font.size(' ')[0]  # The width of a space.
    x, y = pos
    for line in words:
        for word in line:
            word_surface = font.render(word, 0, color)
            word_width, word_height = word_surface.get_size()
            if x + word_width >= width:
                x = pos[0]  # Reset the x.
                y += word_height  # Start on a new row.
            surface.blit(word_surface, (x, y))
            x += word_width + space
        x = pos[0]  # Reset the x.
        y += word_height  # Start on a new row.

# Level Creation and Level Num
levelnum = 0

def read_world_data(levelnum):
    world_data = []
    with open(f"OtherAssets/Levels/Level{levelnum}data.csv") as file:
        reader = csv.reader(file)
        for row in reader:
            world_data.append(row)
    return world_data

world_data = read_world_data(levelnum)
```

```python
# Importing Libraries
import pygame
import math
import json
from GameEngine.settings import *
from PlayerAndEnemies.entity import Entity
from PlayerAndEnemies.spritesheet import SpriteSheet

# Enemy Classes
class Enemy(Entity):
    def __init__(self, monster_type, pos, groups, obstacle_sprites,
damage_player, skeleton_shot):

        # Sprites Setup
        super().__init__(groups)
        self.monster_type = monster_type
        with open(f"MonsterAssets/{monster_type}/CharacterInfo.json") as
file:
            self.data = json.load(file)
        self.sprite_type = "Mob"

        # Projectile Group
        self.arrows = pygame.sprite.Group()

        # Stats Variable
        self.speed = self.data["VEL"]
        self.hp = self.data["Health"]
        self.damage = self.data["AttackDamage"]
        self.resistance = self.data["Resistance"]

        # Notice and Attack Radius
        self.notice_radius = 750
        if self.monster_type == "Skeleton" :
            self.attack_radius = 500
        else:
            self.attack_radius = 60

        # Attack Variables
        self.can_attack = True
        self.attack_time = 0
```

```
        self.damage_player = damage_player
        self.skeleton_shot = skeleton_shot

        if self.monster_type != "Skeleton":
            self.cooldown_time = 400
        else:
            self.cooldown_time = 800

        # Vulnerability Variables
        self.invisibility_duration = 300
        self.vulnerable = True
        self.hit_time = None

        # States and Positioning
        self.previous_state = None
        self.state = "Idle"
        self.image = pygame.Surface((self.data["Width"],
self.data["Height"]))
        self.rect = self.image.get_rect(topleft = pos)
        self.hitbox = self.rect.inflate(-30, -30)
        self.walls = obstacle_sprites

    # Enemy Animation
    def animate(self):

        #Loop over to next frame index
        self.frame_index += self.animation_speed
        if self.frame_index >= self.data["AnimationSteps"]:
            if self.state == "Attack":
                self.frame_index = 0
            self.frame_index = 0

        # Set the Image
        if self.state != self.previous_state:
            self.previous_state = self.state
            self.sprite_sheet = SpriteSheet(self.data[self.state])
            self.frame_index = 0

        # Image Size
        height = self.data["Height"]
```

```
        width = self.data["Width"]

        self.image = self.sprite_sheet.get_image(int(self.frame_index),
width, height, 1, (0, 0, 0))

        #Flicker Logic
        if not self.vulnerable:
            alpha = self.wave_value()
            self.image.set_alpha(alpha)
        else:
            self.image.set_alpha(255)

    # Player Tracking
    def get_player_distance_direction(self, player):

        # Finding Coordinate Differences
        enemy_vec = pygame.math.Vector2(self.rect.center)
        player_vec = pygame.math.Vector2(player.rect.center)
        diff = (player_vec - enemy_vec)
        distance = diff.magnitude()

        # Finding the Direction
        if distance > 0:
            direction = diff.normalize()
        else:
            direction = pygame.math.Vector2()

        return (distance, direction)

    # Change the State of the Mob
    def get_status(self, player):

        self.distance = self.get_player_distance_direction(player)[0]

        if self.distance <= self.attack_radius and self.can_attack:
            self.state = "Attack"
        elif self.distance <= self.notice_radius:
            self.state = "Move"
        else:
            self.state = "Idle"
```

```
    # Use the Mob's State to Perform Different Actions
    def actions(self, player):

        if self.state == 'Attack':
            self.attack_time = pygame.time.get_ticks()
            self.direction = pygame.math.Vector2()
            self.animation_speed = 0.15

            if self.monster_type != "Skeleton":
                if self.rect.colliderect(player.rect):
                    self.damage_player(self.damage)
                    self.can_attack = False
            else:
                if len(self.arrows) <= 4:
                    arrow = self.skeleton_shot(self.rect.center)
                    self.can_attack = False
                    self.arrows.add(arrow)

        elif self.state == "Move":
            self.direction = self.get_player_distance_direction(player)[1]
            self.animation_speed = 0.15
        else:
            self.direction = pygame.math.Vector2()
            self.animation_speed = 0

    # Cooldowns for Attack and Vulnerability Periods
    def cooldown(self):

        current_time = pygame.time.get_ticks()
        if not self.can_attack:
            if current_time - self.attack_time > self.cooldown_time:
                self.can_attack = True
        if not self.vulnerable:
            if current_time - self.hit_time > self.invisibility_duration:
                self.vulnerable = True

    # The function used to Damage the Mob
    def get_damaged(self, player):
        if self.vulnerable:
```

```
            self.direction = self.get_player_distance_direction(player)[1]
            self.hp -= player.damage
            self.check_death()

            self.vulnerable = False
            self.hit_time = pygame.time.get_ticks()

    # Function to Calculate the Knockback Value of the Mob
    def knockback(self):
        if not self.vulnerable:
            self.direction *= -self.resistance

    # Checks for the Death Condition of the Mob
    def check_death(self):
        if self.hp < 0:
            self.player.score += 1
            self.kill()

    # Update the Mob
    def update(self):
        self.knockback()
        self.cooldown()
        self.animate()
        self.move(self.speed)

    # Update the Mob's Tracking for the Player
    def enemy_update(self, player):

        self.player = player
        if player.rect.x > self.rect.x:
            self.image = pygame.transform.flip(self.image.convert_alpha(),
True, False)
        else:
            self.image = pygame.transform.flip(self.image.convert_alpha(),
False, False)

        self.get_status(player)
        self.actions(player)

# Boss Class, just a scaled up enemy class
```

```
class Boss(Enemy):
    def __init__(self, pos, groups, obstacle_sprites, damage_player):
        super().__init__("Cyclops", pos, groups, obstacle_sprites,
damage_player, None)

    def check_death(self):
        if self.hp < 0:
            self.player.score += 20
            self.kill()

    def update(self):
        super().update()
        self.scale(2)

# Importing the Libraries
import pygame
import math

# Entity Class
class Entity(pygame.sprite.Sprite):
    # Initialization Function
    def __init__(self, groups):
        super().__init__(groups)
        self.frame_index = 0
        self.animation_speed = 0.15
        self.data = None
        self.projectiles = pygame.sprite.Group()
        self.direction = pygame.math.Vector2()

    # Scales up the image
    def scale(self, value):
        width = self.data["Width"] * value
        height = self.data["Height"] * value

        self.image = pygame.transform.scale(self.image, (width, height))
        self.rect = self.image.get_rect(center = self.rect.center)

    # Gets the collision condition for the direction
    def collision(self, direction):
        if direction == "horizontal":
```

```
            for sprite in self.walls:
                if sprite.rect.colliderect(self.hitbox):
                    if self.direction.x > 0:
                        self.hitbox.right = sprite.rect.left
                    if self.direction.x < 0:
                        self.hitbox.left = sprite.rect.right
        if direction == "vertical":
            for sprite in self.walls:
                if sprite.rect.colliderect(self.hitbox):
                    if self.direction.y > 0:
                        self.hitbox.bottom = sprite.rect.top
                    if self.direction.y < 0:
                        self.hitbox.top = sprite.rect.bottom

    # Checks to see if the alpha value is 255 or 0
    def wave_value(self):
        value = math.sin(pygame.time.get_ticks())
        if value >= 0:
            return 255
        else:
            return 0

    # Moves the Entity sprite based on the speed and direction
    def move(self, speed):
        if self.direction.magnitude() != 0:
            self.direction = self.direction.normalize()

        self.hitbox.x += self.direction.x * speed
        self.collision("horizontal")
        self.hitbox.y += self.direction.y * speed
        self.collision("vertical")
        self.rect.center = self.hitbox.center

import json
import pygame
from GameEngine.settings import *
from PlayerAndEnemies.entity import Entity
from PlayerAndEnemies.spritesheet import SpriteSheet
```

```
class Player(Entity):
    def __init__(self, position, groups, wall, player_type, create_attack,
destroy_weapon, create_shield, wizard_attack, ranger_attack):

        super().__init__(groups)
        self.player_type = player_type
        self.sprite_type = "Player"
        self.score = 0

        # Animation Data

        self.previous_state = None

        #Image and Rect

        self.image =
pygame.transform.scale(pygame.image.load(f"CharacterAssets/{player_type}/{
player_type}Single.png").convert_alpha(), (64, 64))
        self.rect = self.image.get_rect(topleft = position)
        self.hitbox = self.rect.inflate(-50, -30)

        # Obstacles

        self.walls = wall

        # Shield

        self.shield = None
        self.shield_on = False
        self.shield_cooldown_start = 0
        self.shield_cooldown = False
        self.shield_available = True
        self.create_shield = create_shield

        #Attacks

        ## General ##
        self.attacking = False
        self.attack_time = None
```

```
        ## Knight ##
        self.create_attack = create_attack
        self.destroy_weapon = destroy_weapon

        ## Wizard ##
        self.wizard_attack = wizard_attack
        self.fireball = None

        ## Ranger ##
        self.ranger_attack = ranger_attack
        self.arrows = pygame.sprite.Group()

        # Damage to Player

        self.vulnerable = True
        self.hurt_time = None
        self.invulnerability_duration = 500

        #Graphics Setup

        self.state = "Down"
        self.import_player_assets()

    # Animation

    def animate(self):

        #loop over to next frame index

        self.frame_index += self.animation_speed
        if not self.attacking:
            if self.frame_index >= self.data["AnimationSteps"]:
                self.frame_index = 0
        else:
            if self.frame_index >= self.data["AttackAnimationSteps"]:
                self.frame_index = 0

        # Set the Image

        if self.state != self.previous_state:
```

```
            self.sprite_sheet = SpriteSheet(self.data[self.state])
            self.previous_state = self.state
            if "Idle" in self.state:
                self.animation_speed = 0
            elif "Attack" in self.state:
                if self.player_type == "Knight":
                    self.animation_speed = 0.25
                elif self.player_type == "Wizard":
                    self.animation_speed = 0.09
                else:
                    self.animation_speed = 0.15
            else:
                self.animation_speed = 0.15

        if self.attacking:
            height = self.data["AttackHeight"]
            width = self.data["AttackWidth"]
        else:
            height = self.data["Height"]
            width = self.data["Width"]

        self.image = self.sprite_sheet.get_image(int(self.frame_index),
width, height, 1, (0, 0, 0))
        self.rect = self.image.get_rect(center = self.hitbox.center)

        # Flicker Logic

        if not self.vulnerable:
            alpha = self.wave_value()
            self.image.set_alpha(alpha)
        else:
            self.image.set_alpha(255)

    # Import Player JSON

    def import_player_assets(self):
        with
open(f"CharacterAssets/{self.player_type}/CharacterInfo.json") as file:
            self.data = json.load(file)
```

```
        self.hp = self.data["Health"]
        self.damage = self.data["AttackDamage"]
        self.speed = self.data["VEL"]

        if self.player_type == "Knight":
            self.attack_cooldown = 750
        elif self.player_type == "Ranger":
            self.attack_cooldown = 550
        else:
            self.attack_cooldown = 3000

    # Change Player Status

    def get_status(self):

        # Idle
        if self.direction.x == 0 and self.direction.y == 0:
            if not "Idle" in self.state and not "Attack" in self.state:
                self.state = self.state + "_Idle"

        if self.attacking:
            self.direction.x = 0
            self.direction.y = 0
            if not "Attack" in self.state:
                if "Idle" in self.state:
                    self.state = self.state.replace("Idle", "Attack")
                else:
                    self.state = self.state + "_Attack"
        else:
            if "Attack" in self.state:
                self.state = self.state.replace("_Attack", "")

    # Input Options

    def input(self):
        if not self.attacking:
            keys = pygame.key.get_pressed()

            #Movement Input
```

```
            if keys[pygame.K_w]:
                self.direction.y = -1
                self.state = "Up"
            elif keys[pygame.K_s]:
                self.direction.y = 1
                self.state = "Down"
            else:
                self.direction.y = 0

            if keys[pygame.K_d]:
                self.direction.x = 1
                self.state = "Right"
            elif keys[pygame.K_a]:
                self.direction.x = -1
                self.state = "Left"
            else:
                self.direction.x = 0

            #Attack Input
            if keys[pygame.K_RALT]:
                self.attacking = True
                self.attack_time = pygame.time.get_ticks()
                if self.player_type == "Knight":
                    self.create_attack()
                elif self.player_type == "Wizard":
                    self.wizard_attack()
                else:
                    if len(self.arrows) <= 5:
                        arrow = self.ranger_attack(self.rect.center)
                        self.arrows.add(arrow)

            # Shield Input
            if self.shield_available:
                if keys[pygame.K_RCTRL]:
                    if self.shield == None:
                        self.shield = self.create_shield()
                        self.shield_available = False

    # Attack Cooldown
```

```
    def cooldown(self):
        current_time = pygame.time.get_ticks()
        if self.attacking:
            if current_time - self.attack_time > self.attack_cooldown:
                self.attacking = False
                self.destroy_weapon()
        if not self.vulnerable:
            if current_time - self.hurt_time >
self.invulnerability_duration:
                self.vulnerable = True
        if self.shield_cooldown:
            if current_time - self.shield_cooldown_start > 5000 :
                self.shield_available = True
                self.shield_down = False

    def update(self):

        self.animate()
        self.cooldown()
        self.input()
        self.get_status()
        self.move(self.speed)

import pygame

class SpriteSheet:
    def __init__(self, image):
        self.sheet = pygame.image.load(image).convert_alpha()

    def get_image(self, frame, width, height, scale, color):
        image = pygame.Surface((width, height)).convert_alpha()
        image.blit(self.sheet, (0, 0), ((frame * width), 0, width,
height))
        image = pygame.transform.scale(image, (width * scale, height *
scale))
        image.set_colorkey(color)

        return image

import pygame
```

```
from GameEngine.settings import *

class Tile(pygame.sprite.Sprite):
    def __init__(self, position, groups, indexes):
        super().__init__(groups)
        self.sprite_type = "Wall"
        self.image =
pygame.transform.scale(pygame.image.load(f"OtherAssets/Tiles/{indexes}.png
").convert_alpha(), (TILE_SIZE, TILE_SIZE))
        self.rect = self.image.get_rect(topleft = position)
        self.hitbox = self.rect.inflate(-10, -10)

# Library Import
import pygame

# Pygame Initialization
pygame.init()

# Button Class
class Button():

    # Initialization Function
    def __init__(self, x, y, image):
        self.x = x
        self.y = y

        self.image = pygame.image.load(image).convert_alpha()

        self.width = self.image.get_width()
        self.height = self.image.get_height()

        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)

        self.clicked = False

    # Draws the Button and Checks for the Button is Pressed or not
    def draw(self, window):
        pos = pygame.mouse.get_pos()
        action = False
```

```
        if self.rect.collidepoint(pos):
            if pygame.mouse.get_pressed()[0] and not self.clicked:
                self.clicked = True
                action = True

        if not pygame.mouse.get_pressed()[0]:
            self.clicked = False

        window.blit(self.image, (self.rect.x, self.rect.y))
        return action

# Library Import
import pygame

# Camera Class
class YSortCameraGroup(pygame.sprite.Group):

    # Initialization Function
    def __init__(self):
        # General Setup
        super().__init__()
        self.screen = pygame.display.get_surface()
        self.half_height = self.screen.get_size()[0] // 2
        self.half_width = self.screen.get_size()[1] // 2
        self.offset = pygame.math.Vector2()

        self.background =
pygame.transform.scale(pygame.image.load("Background/StoneBackground.png")
.convert(), (5000, 5000))
        self.background_rect = self.background.get_rect(topleft = (0,0))

    # Draw Function that Shows the Whole map moving so that the player is
always at the center of the map.
    def custom_draw(self, player):
        self.offset.x = player.rect.centerx - self.half_width
        self.offset.y = player.rect.centery - self.half_height

        floor_offset_pos = self.background_rect.topleft - self.offset
        self.screen.blit(self.background, floor_offset_pos)
```

```
        for sprite in sorted(self.sprites(), key = lambda sprite:
sprite.rect.centery):
            offset_position = sprite.rect.topleft - self.offset
            self.screen.blit(sprite.image, offset_position)

    # Enemy Update Function to Help Track the Player
    def enemy_update(self, player):
        enemy_sprites = [sprite for sprite in self.sprites() if
hasattr(sprite, "sprite_type") and sprite.sprite_type == "Mob"]
        for enemy in enemy_sprites:
            enemy.enemy_update(player)

# Pygame Info
import pygame
pygame.init()
font = pygame.font.Font(None,30)

# Shows the Information at the top of the Screen
def debug(info,y = 50, x = 10):
    display_surface = pygame.display.get_surface()
    debug_surf = font.render(str(info),True,'White')
    debug_rect = debug_surf.get_rect(topleft = (x,y))
    pygame.draw.rect(display_surface,'Black',debug_rect)
    display_surface.blit(debug_surf,debug_rect)

import pygame
from GameEngine.settings import *

# Healthbar Class
class UI:

    # Initialization Function
    def __init__(self):
        self.screen = pygame.display.get_surface()
        self.font = pygame.font.SysFont(UI_FONT, UI_FONT_SIZE)
        self.health_bar_rect = pygame.Rect(10, 10 , HEALTH_BAR_WIDTH,
BAR_HEIGHT)

    # Displays the Healthbar
```

```
    def display(self, player):
        pygame.draw.rect(self.screen, UI_BG_COLOR, self.health_bar_rect)
        left = int((player.hp / player.data["Health"]) * HEALTH_BAR_WIDTH)
        current_rect = self.health_bar_rect.copy()
        current_rect.width = left
        pygame.draw.rect(self.screen, HEALTH_COLOR, current_rect)
        pygame.draw.rect(self.screen, UI_BORDER_COLOR,
self.health_bar_rect, 3)

# Displays the Score
def score_display(score):
    screen = pygame.display.get_surface()
    font = pygame.font.SysFont(None, 45)
    text_rendered = font.render(f"Score: {score}", True,
color_constants["White"])
    score_rect = text_rendered.get_rect(topright = (SCREEN_WIDTH - 25,
25))
    pygame.draw.rect(text_rendered, color_constants["Black"], score_rect)
    screen.blit(text_rendered, score_rect)

# Pygame Info and Libary Import
import pygame
from GameEngine.settings import *

# Medkit Class
class Medkit(pygame.sprite.Sprite):
    # Initialization Function
    def __init__(self, x, y, groups):
        super().__init__(groups)
        self.sprite_type = "Healing"
        self.image = pygame.transform.scale(
            pygame.image.load("OtherAssets/Tiles/19.png").convert_alpha(),
(TILE_SIZE, TILE_SIZE))
        self.rect = self.image.get_rect(topleft = (x, y))

# Libary Import
import pygame
import math
from GameEngine.settings import *
from PlayerAndEnemies.spritesheet import SpriteSheet
```

```
# Arrow Class
class Arrows(pygame.sprite.Sprite):
    # Initialization Function
    def __init__(self, groups, pos, target, walls):

        # Sprite Setup

        super().__init__(groups)

        self.speed = 7
        self.walls = walls

        self.image = pygame.transform.scale(

pygame.image.load("OtherAssets/ArrowSprite.png").convert_alpha(), (34,
10))

        # Calcatulating Direction and Repositioning

        self.rect = self.image.get_rect(center = pos)

        self.arrow_vec = pygame.math.Vector2(self.rect.center)
        self.target_vec = pygame.math.Vector2(target.rect.center)

        diff = (self.target_vec - self.arrow_vec)
        if diff.magnitude() > 0:
            self.direction = diff.normalize()
        else:
            self.direction = pygame.math.Vector2()

        angle = -math.degrees(math.atan2(diff[1], diff[0]))
        self.image = pygame.transform.rotate(self.image, angle)
        self.rect = self.image.get_rect(center = self.rect.center)

    # Updates the Arrow's Position amd Wall Collision
    def update(self):

        for wall in self.walls:
            if self.rect.colliderect(wall.hitbox):
```

```
                self.kill()

        self.rect.x += self.direction.x * self.speed
        self.rect.y += self.direction.y * self.speed

# Fireball Class for the Wizard
class FireBall(pygame.sprite.Sprite):
    # Initialization Function
    def __init__(self, player, groups):
        super().__init__(groups)
        self.player = player
        self.frame_index = 0
        self.scale = 0.5
        self.spritesheet =
SpriteSheet("OtherAssets\\FireBallSpriteSheet.png")
        self.image = self.spritesheet.get_image(self.frame_index, 20,
                                                22, self.scale, (0, 0, 0))
        self.rect = self.image.get_rect(center = player.rect.center)

        self.time_started = pygame.time.get_ticks()
        self.animation_speed = 0.35

    # Update the Frame of the Fireball Ring
    def update_frame(self):
        self.frame_index += self.animation_speed
        self.scale += 0.4
        self.rect = self.image.get_rect(center = self.player.rect.center)
        self.image = self.spritesheet.get_image(int(self.frame_index), 20,
                                                22, self.scale, (0, 0, 0))

    # Update the Fireball Position and the Radius of the Fireball
    def update(self):
        current_time = pygame.time.get_ticks()
        if current_time - self.time_started > 1000:
            self.player.attacking = False
            self.player.fireball = None
            self.kill()
        else:
            self.update_frame()
```

```
# Importing Libraries
import pygame

# Slash Class
class Slash(pygame.sprite.Sprite):
    # Initialization Function
    def __init__(self, player, groups):
        super().__init__(groups)
        self.sprite_type = "Melee"
        direction = player.state.split("_")[0]

        #graphic
        if direction == "Left" or direction == "Right":
            self.image = pygame.Surface((120,
player.image.get_height())).convert_alpha()
        elif direction == "Up" or direction == "Down":
            width = player.image.get_width()
            if width - 100 < 0:
                width = 64
            else:
                width -= 100
            self.image = pygame.Surface((player.image.get_width(),
60)).convert_alpha()

        self.image.set_colorkey((0, 0, 0))

        #Placement
        if direction == "Left":
            self.rect = self.image.get_rect(topright = player.rect.topleft
+ pygame.math.Vector2(100, 0))
        elif direction == "Right":
            self.rect = self.image.get_rect(topleft = player.rect.topright
+ pygame.math.Vector2(-100, 0))
        elif direction == "Up":
            self.rect = self.image.get_rect(bottomleft =
player.rect.topleft + pygame.math.Vector2(0, 50))
        else:
            self.rect = self.image.get_rect(topleft =
player.rect.bottomleft + pygame.math.Vector2(0, -50))
```

```
# Shield Class
class Shield(pygame.sprite.Sprite):
    # Initialization Function
    def __init__(self, player, groups):
        super().__init__(groups)
        self.player = player

        self.flip_x = True
        self.flip_y = False

        self.image =
pygame.transform.scale(pygame.image.load("OtherAssets\\AuraShield.png").co
nvert_alpha(), (player.image.get_width() + 10, player.image.get_height() +
10))
        self.rect = self.image.get_rect(center = (player.rect.center))
        self.time_started = pygame.time.get_ticks()

    # Updates the Animation and Checks if the Shield Should be destroyed
or not
    def update(self):

        self.flip_x = not self.flip_x
        self.flip_y = not self.flip_y

        self.image = pygame.transform.flip(self.image, self.flip_x,
self.flip_y)
        self.rect.center = self.player.rect.center
        current_time = pygame.time.get_ticks()
        if current_time - self.time_started > 5000 or "Attack" in
self.player.state:
            self.player.shield_cooldown_start = current_time
            self.player.shield_cooldown = True
            self.player.shield = None
            self.kill()

# Importing the Necessary Libraries

import pygame
import sys
from GameEngine.level import Level
```

```
from GameEngine.settings import *
from Utilities.button import Button
from Utilities.debug import debug
from Utilities.ui import score_display

# Game Class
class Game:
    """
    Main game class responsible for managing game states and logic.
    """

    def __init__(self):
        """
        Initialize the game.
        """
        pygame.init()

        # Set up the display
        pygame.display.set_caption("Forgotten Frontiers")
        self.screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))
        self.clock = pygame.time.Clock()

        # Initialize game state variables
        self.pause = False
        self.resume_button = Button(0, 0, "Buttons\\Resume.png")
        self.exit_button = Button(0, 0, "Buttons\\Exit.png")
        self.resume_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2 - self.resume_button.image.get_height() // 2 - 30)
        self.exit_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2 + self.exit_button.image.get_height() // 2 + 30)

    def update(self, player):
        """
        Update the game state.

        Args:
            player (str): Type of player character.
        """
        self.level.player.player_type = player
```

```
        self.level.player.import_player_assets()

    def change_level(self, levelnum, player_character):
        """
        Change the game level.

        Args:
            levelnum (int): Number of the level.
            player_character (str): Type of player character.
        """

        transition(f"Level {levelnum + 1}")
        self.level = Level(levelnum, player_character)
        self.levelnum = levelnum
        self.player_character = player_character

    def run(self):
        """
        Main game loop.

        Returns:
            bool: True if the game is won, False if the game is over.
        """
        while True:

            # Event Handling
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

                # Pause Function
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_ESCAPE:
                        if self.pause:
                            self.pause = False
                        else:
                            self.pause = True

            # Screen Display
```

```
            self.screen.fill(color_constants["Black"])
            if not self.pause:
                self.level.run()
                score_display(self.level.player.score * 5)
            else:
                if self.exit_button.draw(self.screen):
                    pygame.quit()
                    sys.exit()
                if self.resume_button.draw(self.screen):
                    self.pause = False

            # Screen Updating
            pygame.display.update()
            self.clock.tick(FPS)

            # Level Transition
            if self.level.player.hp <= 0:
                return False
            if len(self.level.attackable_sprites) == 0:
                self.levelnum += 1
                if self.levelnum == 5:
                    return True
                elif self.levelnum == 4:
                    current_score = self.level.player.score

                    transition("Boss Level")
                    self.level = Level(self.levelnum,
self.player_character)

                    self.level.player.score = current_score
                else:
                    current_score = self.level.player.score

                    transition(f"Level {self.levelnum + 1}")
                    self.level = Level(self.levelnum,
self.player_character)

                    self.level.player.score = current_score

# Transition Screen In between the Level
```

```
def transition(text):

    # Graphics Window
    screen = pygame.display.get_surface()

    # Text Display Setup
    font = pygame.font.SysFont("Arial", 100)
    text = font.render(text, True, (255, 255, 255))
    text_rect = text.get_rect(center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2))

    # Time
    start_time = pygame.time.get_ticks()
    current_time = pygame.time.get_ticks()
    while current_time - start_time < 50:

        # Event Handler
        current_time = pygame.time.get_ticks()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # Screen Fill
        screen.fill((0, 0, 0))
        screen.blit(text, text_rect)

        # Screen Update
        game.clock.tick(FPS)
        pygame.display.update()

# Menu1 Function
def menu1():
    """
    First menu screen.

    Returns:
        None: Returns to the Main Loop.
    """
```

```
    # Get the Screen

    screen = pygame.display.get_surface()

    # Create the Buttons

    start_button = Button(0, 0, "Buttons/Start.png")
    credit_button = Button(0, 0, "Buttons/Credit.png")
    exit_button = Button(0, 0, "Buttons/Exit.png")
    info_button = Button(0, 0, "Buttons/Info.png")

    # Variable is Shown

    menu_shown = False
    info_shown = False

    # Centers the Button on the Screen

    start_button.rect.center = (SCREEN_WIDTH // 2, 125)
    credit_button.rect.center = (SCREEN_WIDTH // 2, 310)
    info_button.rect.center = (SCREEN_WIDTH // 2, 490)
    exit_button.rect.center = (SCREEN_WIDTH // 2, 675)

    # Menu Main Loop

    while True:

        # Event Getter

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            # Buttons Pressed Event
            if event.type == pygame.MOUSEBUTTONDOWN:
                if start_button.rect.collidepoint(event.pos):
                    return
                elif credit_button.rect.collidepoint(event.pos):
                    menu_shown = True
```

```
                    display_start = pygame.time.get_ticks()

                elif info_button.rect.collidepoint(event.pos):
                    info_shown = True
                    display_start = pygame.time.get_ticks()

                elif exit_button.rect.collidepoint(event.pos):
                    pygame.quit()
                    sys.exit()

        # Draws the Screen

        screen.fill((0, 0, 0))
        start_button.draw(screen)
        credit_button.draw(screen)
        exit_button.draw(screen)
        info_button.draw(screen)

        # Credit Menu Opened

        if menu_shown:
            current_time = pygame.time.get_ticks()
            if current_time - display_start > 5000:
                menu_shown = False
            pygame.draw.rect(screen, (85, 85, 85), (50, 50, 850, 700))
            blit_text(screen, 800, "Game Created by Undefined Studios.
Images are Credited in the Game Files.", (100, 75), credit_font,
color_constants["Black"])

        # Info Menu Opened

        if info_shown:
            current_time = pygame.time.get_ticks()
            if current_time - display_start > 7500:
                info_shown = False
            pygame.draw.rect(screen, (85, 85, 85), (50, 50, 850, 700))
            blit_text(screen, 800, "This is a Dungeon Crawler Game where
you get to choose the level you will play and the character you play as.
Player Movements are WASD, Attack button is Right Alt, and Shield is Right
```

```
Ctrl. There will be medical packs that when ran into will heal the
player.", (100, 75), credit_font, color_constants["Black"])

        # Refresh the Frame

        game.clock.tick(FPS)
        pygame.display.update()

# Menu2 Function
def menu2():
    """
    Third menu screen for selecting game level.

    Returns:
        int: Number of the selected game level.
    """
    screen = pygame.display.get_surface()
    level1 = Button(0, 0, "Buttons\\Level1.png")
    level2 = Button(0, 0, "Buttons\\Level2.png")
    level3 = Button(0, 0, "Buttons\\Level3.png")

    level1.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 - 180)
    level2.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    level3.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 180)

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if level1.rect.collidepoint(event.pos):
                    levelnum = 1
                    return levelnum
                elif level2.rect.collidepoint(event.pos):
                    levelnum = 2
                    return levelnum
                elif level3.rect.collidepoint(event.pos):
                    levelnum = 3
                    return levelnum
```

```
        screen.fill((0, 0, 0))  # Clear the screen

        level1.draw(screen)
        level2.draw(screen)
        level3.draw(screen)

        game.clock.tick(FPS)
        pygame.display.update()

# Menu3 Function
def menu3():
    """
    Second menu screen for selecting player character.

    Args:
        boolean (bool): Boolean controlling menu loop.

    Returns:
        str: Type of selected player character.
    """
    screen = pygame.display.get_surface()

    knight_button = Button(0, 0, "Buttons\\Knight.png")
    ranger_button = Button(0, 0, "Buttons\\Ranger.png")
    wizard_button = Button(0, 0, "Buttons\\Wizard.png")

    knight_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 -
200)
    ranger_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    wizard_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 +
150)

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if knight_button.rect.collidepoint(event.pos):
```

```
                    player_type = "Knight"
                    return player_type
                elif ranger_button.rect.collidepoint(event.pos):
                    player_type = "Ranger"
                    return player_type
                elif wizard_button.rect.collidepoint(event.pos):
                    player_type = "Wizard"
                    return player_type

        screen.fill((0, 0, 0))  # Clear the screen
        knight_button.draw(screen)
        ranger_button.draw(screen)
        wizard_button.draw(screen)

        game.clock.tick()
        pygame.display.update()

# Results Function
def results(text):
    """
    Display the game result.

    Args:
        text (str): Text to be displayed.
    """

    # Fonts
    font = pygame.font.SysFont("Arial", 100)
    score_font = pygame.font.SysFont(None, 50)

    # Render Fonts
    text = font.render(text, True, (255, 255, 255))
    score_text = score_font.render(f"Score: {game.level.player.score *
5}", True, color_constants["Black"])

    # Scores Printed
    print("Congratulations!")
    print(f"You got: {game.level.player.score * 5}")

    # Text Rects
```

```
    text_rect = text.get_rect(center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2))
    score_rect = score_text.get_rect(midtop = text_rect.midbottom)

    #Time
    start_time = pygame.time.get_ticks()

    while True:
        current_time = pygame.time.get_ticks()

        # Event Handler
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # Screen Display
        game.screen.fill(color_constants["Black"])
        game.screen.blit(text, text_rect)
        game.screen.blit(score_text, score_rect)

        # Display Update
        game.clock.tick(FPS)
        pygame.display.update()

        # Timer
        if current_time - start_time > 10000:
            pygame.quit()
            sys.exit()

# Main Game if This is the main function
if __name__ == "__main__":
    # Initialize the game
    game = Game()

    # Display menu screens and get user inputs

    menu1()
    levelnum = menu2()
    player_character = menu3()
```

```
    # Start the selected game level
    game.change_level(levelnum, player_character)
    result = game.run()

    # Display game result
    if result:
        results("You Win!")
    else:
        results("Game Over.")

# Importing the Necessary Libraries

import pygame
import sys
from GameEngine.level import Level
from GameEngine.settings import *
from Utilities.button import Button
from Utilities.debug import debug
from Utilities.ui import score_display

# Game Class
class Game:
    """
    Main game class responsible for managing game states and logic.
    """

    def __init__(self):
        """
        Initialize the game.
        """
        pygame.init()

        # Set up the display
        pygame.display.set_caption("Forgotten Frontiers")
        self.screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))
        self.clock = pygame.time.Clock()

        # Initialize game state variables
```

```
        self.pause = False
        self.resume_button = Button(0, 0, "Buttons\\Resume.png")
        self.exit_button = Button(0, 0, "Buttons\\Exit.png")
        self.resume_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2 - self.resume_button.image.get_height() // 2 - 30)
        self.exit_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2 + self.exit_button.image.get_height() // 2 + 30)

    def update(self, player):
        """
        Update the game state.

        Args:
            player (str): Type of player character.
        """
        self.level.player.player_type = player
        self.level.player.import_player_assets()

    def change_level(self, levelnum, player_character):
        """
        Change the game level.

        Args:
            levelnum (int): Number of the level.
            player_character (str): Type of player character.
        """

        transition(f"Level {levelnum + 1}")
        self.level = Level(levelnum, player_character)
        self.levelnum = levelnum
        self.player_character = player_character

    def run(self):
        """
        Main game loop.

        Returns:
            bool: True if the game is won, False if the game is over.
        """
        while True:
```

```
        # Event Handling
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            # Pause Function
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    if self.pause:
                        self.pause = False
                    else:
                        self.pause = True

        # Screen Display
        self.screen.fill(color_constants["Black"])
        if not self.pause:
            self.level.run()
            score_display(self.level.player.score * 5)
        else:
            if self.exit_button.draw(self.screen):
                pygame.quit()
                sys.exit()
            if self.resume_button.draw(self.screen):
                self.pause = False

        # Screen Updating
        pygame.display.update()
        self.clock.tick(FPS)

        # Level Transition
        if self.level.player.hp <= 0:
            return False
        if len(self.level.attackable_sprites) == 0:
            self.levelnum += 1
            if self.levelnum == 5:
                return True
            elif self.levelnum == 4:
                current_score = self.level.player.score
```

```
                    transition("Boss Level")
                    self.level = Level(self.levelnum,
self.player_character)

                    self.level.player.score = current_score
                else:
                    current_score = self.level.player.score

                    transition(f"Level {self.levelnum + 1}")
                    self.level = Level(self.levelnum,
self.player_character)

                    self.level.player.score = current_score

# Transition Screen In between the Level
def transition(text):

    # Graphics Window
    screen = pygame.display.get_surface()

    # Text Display Setup
    font = pygame.font.SysFont("Arial", 100)
    text = font.render(text, True, (255, 255, 255))
    text_rect = text.get_rect(center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2))

    # Time
    start_time = pygame.time.get_ticks()
    current_time = pygame.time.get_ticks()
    while current_time - start_time < 50:

        # Event Handler
        current_time = pygame.time.get_ticks()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # Screen Fill
```

```
        screen.fill((0, 0, 0))
        screen.blit(text, text_rect)

        # Screen Update
        game.clock.tick(FPS)
        pygame.display.update()

# Menu1 Function
def menu1():
    """
    First menu screen.

    Returns:
        None: Returns to the Main Loop.
    """

    # Get the Screen

    screen = pygame.display.get_surface()

    # Create the Buttons

    start_button = Button(0, 0, "Buttons/Start.png")
    credit_button = Button(0, 0, "Buttons/Credit.png")
    exit_button = Button(0, 0, "Buttons/Exit.png")
    info_button = Button(0, 0, "Buttons/Info.png")

    # Variable is Shown

    menu_shown = False
    info_shown = False

    # Centers the Button on the Screen

    start_button.rect.center = (SCREEN_WIDTH // 2, 125)
    credit_button.rect.center = (SCREEN_WIDTH // 2, 310)
    info_button.rect.center = (SCREEN_WIDTH // 2, 490)
    exit_button.rect.center = (SCREEN_WIDTH // 2, 675)

    # Menu Main Loop
```

```
while True:

    # Event Getter

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        # Buttons Pressed Event
        if event.type == pygame.MOUSEBUTTONDOWN:
            if start_button.rect.collidepoint(event.pos):
                return
            elif credit_button.rect.collidepoint(event.pos):
                menu_shown = True
                display_start = pygame.time.get_ticks()

            elif info_button.rect.collidepoint(event.pos):
                info_shown = True
                display_start = pygame.time.get_ticks()

            elif exit_button.rect.collidepoint(event.pos):
                pygame.quit()
                sys.exit()

    # Draws the Screen

    screen.fill((0, 0, 0))
    start_button.draw(screen)
    credit_button.draw(screen)
    exit_button.draw(screen)
    info_button.draw(screen)

    # Credit Menu Opened

    if menu_shown:
        current_time = pygame.time.get_ticks()
        if current_time - display_start > 5000:
            menu_shown = False
```

```
            pygame.draw.rect(screen, (85, 85, 85), (50, 50, 850, 700))
            blit_text(screen, 800, "Game Created by Undefined Studios.
Images are Credited in the Game Files.", (100, 75), credit_font,
color_constants["Black"])

        # Info Menu Opened

        if info_shown:
            current_time = pygame.time.get_ticks()
            if current_time - display_start > 7500:
                info_shown = False
            pygame.draw.rect(screen, (85, 85, 85), (50, 50, 850, 700))
            blit_text(screen, 800, "This is a Dungeon Crawler Game where
you get to choose the level you will play and the character you play as.
Player Movements are WASD, Attack button is Right Alt, and Shield is Right
Ctrl. There will be medical packs that when ran into will heal the
player.", (100, 75), credit_font, color_constants["Black"])

        # Refresh the Frame

        game.clock.tick(FPS)
        pygame.display.update()

# Menu2 Function
def menu2():
    """
    Second menu screen for selecting player character.

    Args:
        boolean (bool): Boolean controlling menu loop.

    Returns:
        str: Type of selected player character.
    """
    screen = pygame.display.get_surface()

    knight_button = Button(0, 0, "Buttons\\Knight.png")
    ranger_button = Button(0, 0, "Buttons\\Ranger.png")
    wizard_button = Button(0, 0, "Buttons\\Wizard.png")
```

```
    knight_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 -
200)
    ranger_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
    wizard_button.rect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 +
150)

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if knight_button.rect.collidepoint(event.pos):
                    player_type = "Knight"
                    return player_type
                elif ranger_button.rect.collidepoint(event.pos):
                    player_type = "Ranger"
                    return player_type
                elif wizard_button.rect.collidepoint(event.pos):
                    player_type = "Wizard"
                    return player_type

        screen.fill((0, 0, 0))  # Clear the screen
        knight_button.draw(screen)
        ranger_button.draw(screen)
        wizard_button.draw(screen)

        game.clock.tick()
        pygame.display.update()

# Results Function
def results(text):
    """
    Display the game result.

    Args:
        text (str): Text to be displayed.
    """

    # Fonts
```

```
    font = pygame.font.SysFont("Arial", 100)
    score_font = pygame.font.SysFont(None, 50)

    # Render Fonts
    text = font.render(text, True, (255, 255, 255))
    score_text = score_font.render(f"Score: {game.level.player.score *
5}", True, color_constants["Black"])

    # Text Rects
    text_rect = text.get_rect(center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT
// 2))
    score_rect = score_text.get_rect(midtop = text_rect.midbottom)

    # Scores Printed
    print("Congratulations!")
    print(f"You got: {game.level.player.score * 5}")

    #Time
    start_time = pygame.time.get_ticks()

    while True:
        current_time = pygame.time.get_ticks()

        # Event Handler
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # Screen Display
        game.screen.fill(color_constants["Black"])
        game.screen.blit(text, text_rect)
        game.screen.blit(score_text, score_rect)

        # Display Update
        game.clock.tick(FPS)
        pygame.display.update()

        # Timer
        if current_time - start_time > 10000:
```

```
            pygame.quit()
            sys.exit()

# Main Game if This is the main function
if __name__ == "__main__":

    # Initialize the game
    game = Game()

    # Display menu screens and get user inputs
    menu1()
    player_character = menu2()

    # Start the selected game level
    game.change_level(levelnum, player_character)
    result = game.run()

    # Display game result
    if result:
        results("You Win!")
    else:
        results("Game Over.")
```