import pygame  
import random  
import pickle  
import os  
  
# Initialize Pygame  
pygame.init()  
  
# Screen dimensions  
WIDTH, HEIGHT = 800, 600  
PLAYABLE\_HEIGHT = HEIGHT - 60 # Leave space for inventory at the bottom  
screen = pygame.display.set\_mode((WIDTH, HEIGHT))  
pygame.display.set\_caption("Elysian Grove Adventure")  
  
# Load character images with error handling  
def load\_image(name):  
 try:  
 image = pygame.image.load(name)  
 return image.convert\_alpha() # Ensure transparency is handled correctly  
 except pygame.error as message:  
 print(f"Cannot load image: {name}")  
 raise SystemExit(message)  
  
# Load images  
luminara\_img = load\_image("luminara.png")  
luminara\_img = pygame.transform.scale(luminara\_img, (90, 90)) # Increased size by 50%  
luminara\_invuln\_img = pygame.transform.scale(load\_image("luminarainvuln.png"), (90, 90)) # Match player size  
enemy\_img = load\_image("enemy.png")  
enemy\_img = pygame.transform.scale(enemy\_img, (67, 67)) # Increased size by 50%  
bossenemy\_img = load\_image("bossenemy.png")  
bossenemy\_img = pygame.transform.scale(bossenemy\_img, (134, 134)) # Boss enemy is twice the size of regular enemy  
malakar\_img = pygame.transform.scale(load\_image("malakar.png"), (268, 268)) # Malakar is twice the size of boss enemy  
background\_img = load\_image("background.png")  
ripple\_img = load\_image("ripple.png")  
ripple\_img = pygame.transform.scale(ripple\_img, (45, 45))  
  
# Load fruit images  
gleamberry\_img = pygame.transform.scale(load\_image("gleamberry.png"), (45, 45)) # Increased by 50%  
shimmeringapple\_img = pygame.transform.scale(load\_image("shimmeringapple.png"), (45, 45)) # Increased by 50%  
etherealpear\_img = pygame.transform.scale(load\_image("etherealpear.png"), (45, 45)) # Increased by 50%  
flamefruit\_img = pygame.transform.scale(load\_image("flamefruit.png"), (45, 45)) # Increased by 50%  
moonbeammelon\_img = pygame.transform.scale(load\_image("moonbeammelon.png"), (45, 45)) # Increased by 50%  
  
# Colors  
WHITE = (255, 255, 255)  
GREEN = (0, 255, 0)  
RED = (255, 0, 0)  
BLUE = (0, 0, 255)  
BLACK = (0, 0, 0)  
RAINBOW\_COLORS = [(255, 0, 0), (255, 127, 0), (255, 255, 0), (0, 255, 0), (0, 0, 255), (75, 0, 130), (148, 0, 211)]  
  
# Player class  
class Player(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, image):  
 super().\_\_init\_\_()  
 self.default\_image = image  
 self.invuln\_image = luminara\_invuln\_img  
 self.image = self.default\_image  
 self.rect = self.image.get\_rect()  
 self.rect.center = (WIDTH // 2, PLAYABLE\_HEIGHT // 2)  
 self.speed = 5  
 self.experience = 0  
 self.level = 1  
 self.health = 100  
 self.max\_health = 100  
 self.inventory = {"Gleam Berry": 0, "Shimmering Apple": 0, "Ethereal Pear": 0, "Flamefruit": 0, "Moonbeam Melon": 0}  
 self.damage = 20 # Default damage  
 self.invulnerable = False  
 self.last\_hit = pygame.time.get\_ticks() # Track the last time the player was hit  
 self.invuln\_end\_time = 0 # Track when the invulnerability ends  
 self.melon\_end\_time = 0 # Track when the melon's effect ends  
 self.last\_regen\_time = pygame.time.get\_ticks() # Track the last time health was regenerated  
 self.special\_attack\_ready = True  
 self.special\_attack\_time = 0  
 self.flamefruit\_end\_time = 0  
 self.flamefruit\_active = False  
 self.flamefruit\_position = None  
 self.speed\_boost\_end\_time = 0  
 self.damage\_reduction = 0  
 self.damage\_reduction\_end\_times = []  
  
 def move(self, dx, dy):  
 self.rect.x += dx \* self.speed  
 self.rect.y += dy \* self.speed  
 self.rect.x = max(0, min(self.rect.x, WIDTH - self.rect.width))  
 self.rect.y = max(0, min(self.rect.y, PLAYABLE\_HEIGHT - self.rect.height))  
  
 def collect\_fruit(self, fruit):  
 self.inventory[fruit.name] += 1  
 trigger\_rainbow\_splash()  
 if fruit.name == "Gleam Berry":  
 self.health = min(self.health + 10, self.max\_health)  
 spawn\_ripple(fruit.rect.center)  
 elif fruit.name == "Shimmering Apple":  
 self.speed\_boost\_end\_time = pygame.time.get\_ticks() + 12000 # 12 seconds  
 self.speed = 10 # Speed boost  
 elif fruit.name == "Ethereal Pear":  
 self.experience += 150  
 self.health = min(self.health + 20, self.max\_health + 20)  
 self.max\_health += 5  
 elif fruit.name == "Flamefruit":  
 self.experience += 100  
 self.flamefruit\_end\_time = pygame.time.get\_ticks() + 3000 # 3 seconds total  
 self.flamefruit\_active = True  
 self.flamefruit\_position = fruit.rect.center  
 if self.level > 50:  
 self.damage\_reduction += 10  
 self.damage\_reduction\_end\_times.append(pygame.time.get\_ticks() + 5000) # 5 seconds duration  
 elif fruit.name == "Moonbeam Melon":  
 self.experience += 200  
 self.damage = 100 # Increase damage by a factor of 5  
 self.melon\_end\_time = pygame.time.get\_ticks() + 9000 # 9 seconds total  
 self.invulnerable = True  
 self.image = self.invuln\_image  
 self.invuln\_end\_time = pygame.time.get\_ticks() + 2000 # 2 seconds invulnerability  
  
 self.experience += 100  
 if self.experience >= 1000:  
 self.level += 1  
 self.experience = 0  
 trigger\_level\_up\_splash()  
  
 return fruit.name  
  
 def attack(self, enemy):  
 damage\_dealt = max(self.damage - self.damage\_reduction, 0)  
 enemy.health -= damage\_dealt  
 trigger\_rainbow\_splash()  
 if enemy.health <= 0:  
 enemy.kill()  
 self.experience += 50 # Award experience for defeating enemies  
 if isinstance(enemy, BossEnemy):  
 self.experience += 500 # Award more experience for defeating boss enemies  
 if isinstance(enemy, Malakar):  
 self.experience += 1000 # Award even more experience for defeating Malakar  
 global malakar\_spawn\_allowed\_time  
 malakar\_spawn\_allowed\_time = pygame.time.get\_ticks() + 15000 # 15 seconds delay before Malakar can respawn  
  
 def take\_damage(self, amount):  
 if not self.invulnerable:  
 current\_time = pygame.time.get\_ticks()  
 if current\_time - self.last\_hit > 1000: # 1 second cooldown  
 self.health -= max(amount - self.damage\_reduction, 0)  
 self.speed = max(self.speed - 1, 1) # Permanent -1 speed penalty, but not less than 1  
 self.last\_hit = current\_time  
  
 def special\_attack(self):  
 if self.special\_attack\_ready:  
 for sprite in all\_sprites:  
 if isinstance(sprite, (NightCrawler, BossEnemy, Malakar)):  
 distance = ((self.rect.centerx - sprite.rect.centerx) \*\* 2 + (self.rect.centery - sprite.rect.centery) \*\* 2) \*\* 0.5  
 if distance <= WIDTH // 5:  
 sprite.health -= 500 # Increased special attack damage  
 sprite.speed = 0 # Prevent movement  
 sprite.freeze\_end\_time = pygame.time.get\_ticks() + 2000 # Freeze for 2 seconds  
 sprite.original\_color = sprite.image.copy()  
 sprite.image.fill(RED, special\_flags=pygame.BLEND\_MULT)  
 trigger\_rainbow\_splash()  
 if sprite.health <= 0:  
 sprite.kill()  
 self.experience += 50  
 if isinstance(sprite, BossEnemy):  
 self.experience += 500  
 if isinstance(sprite, Malakar):  
 self.experience += 1000  
 global malakar\_spawn\_allowed\_time  
 malakar\_spawn\_allowed\_time = pygame.time.get\_ticks() + 15000 # 15 seconds delay before Malakar can respawn  
 self.special\_attack\_ready = False  
 self.special\_attack\_time = pygame.time.get\_ticks() + 30000 # 30 seconds cooldown  
  
 def update(self):  
 current\_time = pygame.time.get\_ticks()  
  
 # End invulnerability  
 if self.invulnerable and current\_time > self.invuln\_end\_time:  
 self.invulnerable = False  
 self.image = self.default\_image  
  
 # End melon's damage increase  
 if current\_time > self.melon\_end\_time:  
 self.damage = 20  
  
 # End damage reduction  
 self.damage\_reduction = sum(10 for end\_time in self.damage\_reduction\_end\_times if current\_time <= end\_time)  
 self.damage\_reduction\_end\_times = [end\_time for end\_time in self.damage\_reduction\_end\_times if current\_time <= end\_time]  
  
 # Reset speed after speed boost ends  
 if current\_time > self.speed\_boost\_end\_time:  
 self.speed = 5  
  
 # Health regeneration over time  
 if current\_time - self.last\_regen\_time > 5000 and self.health < self.max\_health: # Regenerate every 5 seconds  
 self.health += 1  
 self.last\_regen\_time = current\_time  
  
 # Special attack cooldown  
 if not self.special\_attack\_ready and current\_time > self.special\_attack\_time:  
 self.special\_attack\_ready = True  
  
 # End flamefruit effect  
 if self.flamefruit\_active and current\_time > self.flamefruit\_end\_time:  
 self.flamefruit\_active = False  
  
# Fruit class  
class Fruit(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y, name, image):  
 super().\_\_init\_\_()  
 self.image = image  
 self.rect = self.image.get\_rect()  
 self.rect.topleft = (x, y)  
 self.name = name  
  
# Ripple class  
class Ripple(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y):  
 super().\_\_init\_\_()  
 self.image = ripple\_img  
 self.rect = self.image.get\_rect()  
 self.rect.center = (x, y)  
 self.speed = 1 # Reduced speed  
  
 def update(self):  
 nearest\_enemy = None  
 nearest\_distance = float('inf')  
 for enemy in enemies:  
 distance = ((self.rect.centerx - enemy.rect.centerx) \*\* 2 + (self.rect.centery - enemy.rect.centery) \*\* 2) \*\* 0.5  
 if distance < nearest\_distance:  
 nearest\_distance = distance  
 nearest\_enemy = enemy  
 for bossenemy in bossenemies:  
 distance = ((self.rect.centerx - bossenemy.rect.centerx) \*\* 2 + (self.rect.centery - bossenemy.rect.centery) \*\* 2) \*\* 0.5  
 if distance < nearest\_distance:  
 nearest\_distance = distance  
 nearest\_enemy = bossenemy  
 for malakar in malakar\_group:  
 distance = ((self.rect.centerx - malakar.rect.centerx) \*\* 2 + (self.rect.centery - malakar.rect.centery) \*\* 2) \*\* 0.5  
 if distance < nearest\_distance:  
 nearest\_distance = distance  
 nearest\_enemy = malakar  
  
 if nearest\_enemy:  
 if self.rect.centerx < nearest\_enemy.rect.centerx:  
 self.rect.x += self.speed  
 elif self.rect.centerx > nearest\_enemy.rect.centerx:  
 self.rect.x -= self.speed  
 if self.rect.centery < nearest\_enemy.rect.centery:  
 self.rect.y += self.speed  
 elif self.rect.centery > nearest\_enemy.rect.centery:  
 self.rect.y -= self.speed  
  
 if pygame.sprite.collide\_rect(self, nearest\_enemy):  
 nearest\_enemy.health -= 10000 # Ripple deals massive damage  
 if nearest\_enemy.health <= 0:  
 nearest\_enemy.kill()  
 self.kill()  
  
# Night Crawler class with improved movement  
class NightCrawler(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y):  
 super().\_\_init\_\_()  
 self.image = enemy\_img  
 self.rect = self.image.get\_rect()  
 self.rect.topleft = (x, y)  
 self.health = 100  
 self.speed = random.uniform(0.5, 1.5) # Reduced by half  
 self.aggro\_radius = WIDTH // 5  
 self.freeze\_end\_time = 0 # Track freeze time  
  
 def update(self):  
 current\_time = pygame.time.get\_ticks()  
 if current\_time < self.freeze\_end\_time:  
 self.speed = 0  
 self.image.fill(RED, special\_flags=pygame.BLEND\_MULT)  
 else:  
 self.speed = random.uniform(0.5, 1.5) # Restore speed  
 if hasattr(self, 'original\_color'):  
 self.image = self.original\_color  
  
 # Only move towards the player if within aggro radius  
 if pygame.sprite.collide\_circle\_ratio(self.aggro\_radius / self.rect.width)(self, player):  
 # Smooth, random movement  
 if self.rect.x < player.rect.x:  
 self.rect.x += self.speed  
 elif self.rect.x > player.rect.x:  
 self.rect.x -= self.speed  
 if self.rect.y < player.rect.y:  
 self.rect.y += self.speed  
 elif self.rect.y > player.rect.y:  
 self.rect.y -= self.speed  
 if pygame.sprite.collide\_rect(self, player):  
 player.take\_damage(1 + player.level // 4) # Increase damage by 1/4 player level  
 elif player.flamefruit\_active and player.flamefruit\_position:  
 # Move towards the location where flamefruit was collected  
 if self.rect.x < player.flamefruit\_position[0]:  
 self.rect.x += self.speed  
 elif self.rect.x > player.flamefruit\_position[0]:  
 self.rect.x -= self.speed  
 if self.rect.y < player.flamefruit\_position[1]:  
 self.rect.y += self.speed  
 elif self.rect.y > player.flamefruit\_position[1]:  
 self.rect.y -= self.speed  
  
# Boss Enemy class  
class BossEnemy(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y):  
 super().\_\_init\_\_()  
 self.image = bossenemy\_img  
 self.rect = self.image.get\_rect()  
 self.rect.topleft = (x, y)  
 self.health = 500  
 self.speed = random.uniform(0.5, 1.5) # Reduced by half  
 self.aggro\_radius = WIDTH // 5  
 self.freeze\_end\_time = 0 # Track freeze time  
  
 def update(self):  
 current\_time = pygame.time.get\_ticks()  
 if current\_time < self.freeze\_end\_time:  
 self.speed = 0  
 self.image.fill(RED, special\_flags=pygame.BLEND\_MULT)  
 else:  
 self.speed = random.uniform(0.5, 1.5) # Restore speed  
 if hasattr(self, 'original\_color'):  
 self.image = self.original\_color  
  
 # Only move towards the player if within aggro radius  
 if pygame.sprite.collide\_circle\_ratio(self.aggro\_radius / self.rect.width)(self, player):  
 if self.rect.x < player.rect.x:  
 self.rect.x += self.speed  
 elif self.rect.x > player.rect.x:  
 self.rect.x -= self.speed  
 if self.rect.y < player.rect.y:  
 self.rect.y += self.speed  
 elif self.rect.y > player.rect.y:  
 self.rect.y -= self.speed  
 if pygame.sprite.collide\_rect(self, player):  
 player.take\_damage(10 + player.level // 2) # Increase damage by 1/2 player level  
  
# Malakar class  
class Malakar(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y):  
 super().\_\_init\_\_()  
 self.image = malakar\_img  
 self.rect = self.image.get\_rect()  
 self.rect.topleft = (x, y)  
 self.health = 5000  
 self.speed = random.uniform(0.5, 1.5) # Reduced by half  
 self.aggro\_radius = WIDTH // 2 # Aggro radius increased by a factor of 2  
 self.freeze\_end\_time = 0 # Track freeze time  
  
 def update(self):  
 current\_time = pygame.time.get\_ticks()  
 if current\_time < self.freeze\_end\_time:  
 self.speed = 0  
 self.image.fill(RED, special\_flags=pygame.BLEND\_MULT)  
 else:  
 self.speed = random.uniform(0.5, 1.5) # Restore speed  
 if hasattr(self, 'original\_color'):  
 self.image = self.original\_color  
  
 # Only move towards the player if within aggro radius  
 if pygame.sprite.collide\_circle\_ratio(self.aggro\_radius / self.rect.width)(self, player):  
 if self.rect.x < player.rect.x:  
 self.rect.x += self.speed  
 elif self.rect.x > player.rect.x:  
 self.rect.x -= self.speed  
 if self.rect.y < player.rect.y:  
 self.rect.y += self.speed  
 elif self.rect.y > player.rect.y:  
 self.rect.y -= self.speed  
 if pygame.sprite.collide\_rect(self, player):  
 player.take\_damage(10 + player.level) # Increase damage by player level  
  
# Additional functions  
def spawn\_ripple(position):  
 ripple = Ripple(\*position)  
 all\_sprites.add(ripple)  
  
def trigger\_rainbow\_splash():  
 global rainbow\_splash, rainbow\_splash\_time  
 rainbow\_splash = True  
 rainbow\_splash\_time = pygame.time.get\_ticks()  
  
def draw\_rainbow\_splash(surface):  
 global rainbow\_splash  
 if rainbow\_splash:  
 color = random.choice(RAINBOW\_COLORS)  
 surface.fill(color, special\_flags=pygame.BLEND\_MULT)  
 rainbow\_splash = False # Single frame duration  
  
def trigger\_level\_up\_splash():  
 global level\_up\_splash, level\_up\_splash\_time  
 level\_up\_splash = True  
 level\_up\_splash\_time = pygame.time.get\_ticks()  
  
def draw\_level\_up\_splash(surface):  
 global level\_up\_splash  
 if level\_up\_splash:  
 color = (255, 255, 0) # Yellow for level-up splash  
 surface.fill(color, special\_flags=pygame.BLEND\_MULT)  
 level\_up\_splash = False # Single frame duration  
  
def draw\_inventory(surface, player):  
 pygame.draw.rect(surface, BLUE, (0, PLAYABLE\_HEIGHT, WIDTH, HEIGHT - PLAYABLE\_HEIGHT))  
  
 x\_offset = 10  
 y\_offset = PLAYABLE\_HEIGHT + 10  
  
 draw\_text(surface, f'Collected Fruits:', 18, WIDTH // 2, PLAYABLE\_HEIGHT + 10, WHITE)  
  
 for fruit, image in fruit\_names\_images:  
 surface.blit(image, (x\_offset, y\_offset))  
 draw\_text(surface, str(player.inventory[fruit]), 18, x\_offset + 30, y\_offset, WHITE)  
 x\_offset += 50  
  
 # Display player status effects  
 if player.invulnerable:  
 draw\_text(surface, 'Status: Invulnerable', 18, x\_offset + 30, PLAYABLE\_HEIGHT + 10, GREEN)  
 if player.damage == 100:  
 draw\_text(surface, 'Status: Increased Damage', 18, x\_offset + 30, PLAYABLE\_HEIGHT + 30, GREEN)  
  
def draw\_legend(surface):  
 legend\_text = ["Arrow Keys: Move", "Spacebar: Attack", "N: Special Attack", "P: Pause"]  
 x = WIDTH - 150  
 y = HEIGHT - 50  
 for line in legend\_text:  
 draw\_text(surface, line, 18, x, y, WHITE)  
 y += 20  
  
def draw\_health\_bar(surface, x, y, health, max\_health, width, height, border=2):  
 if health < 0:  
 health = 0  
 fill = (health / max\_health) \* width  
 outline\_rect = pygame.Rect(x, y, width, height)  
 fill\_rect = pygame.Rect(x, y, fill, height)  
 pygame.draw.rect(surface, RED if health > max\_health \* 0.1 else WHITE, fill\_rect)  
 pygame.draw.rect(surface, BLUE, outline\_rect, border)  
  
def draw\_text(surface, text, size, x, y, color):  
 font = pygame.font.SysFont(None, size)  
 text\_surface = font.render(text, True, color)  
 text\_rect = text\_surface.get\_rect()  
 text\_rect.midtop = (x, y)  
 surface.blit(text\_surface, text\_rect)  
  
def save\_game():  
 with open("savegame.dat", "wb") as f:  
 pickle.dump({  
 'experience': player.experience,  
 'level': player.level,  
 'health': player.health,  
 'max\_health': player.max\_health,  
 'inventory': player.inventory,  
 'damage': player.damage,  
 'invulnerable': player.invulnerable,  
 'last\_hit': player.last\_hit,  
 'invuln\_end\_time': player.invuln\_end\_time,  
 'melon\_end\_time': player.melon\_end\_time,  
 'last\_regen\_time': player.last\_regen\_time,  
 'special\_attack\_ready': player.special\_attack\_ready,  
 'special\_attack\_time': player.special\_attack\_time,  
 'flamefruit\_end\_time': player.flamefruit\_end\_time,  
 'flamefruit\_active': player.flamefruit\_active,  
 'flamefruit\_position': player.flamefruit\_position,  
 'speed\_boost\_end\_time': player.speed\_boost\_end\_time,  
 'damage\_reduction': player.damage\_reduction,  
 'damage\_reduction\_end\_times': player.damage\_reduction\_end\_times  
 }, f)  
  
def load\_game():  
 global player, all\_sprites, fruits, enemies, bossenemies, malakar\_group  
 if os.path.exists("savegame.dat"):  
 with open("savegame.dat", "rb") as f:  
 data = pickle.load(f)  
 player.experience = data['experience']  
 player.level = data['level']  
 player.health = data['health']  
 player.max\_health = data['max\_health']  
 player.inventory = data['inventory']  
 player.damage = data['damage']  
 player.invulnerable = data['invulnerable']  
 player.last\_hit = data['last\_hit']  
 player.invuln\_end\_time = data['invuln\_end\_time']  
 player.melon\_end\_time = data['melon\_end\_time']  
 player.last\_regen\_time = data['last\_regen\_time']  
 player.special\_attack\_ready = data['special\_attack\_ready']  
 player.special\_attack\_time = data['special\_attack\_time']  
 player.flamefruit\_end\_time = data['flamefruit\_end\_time']  
 player.flamefruit\_active = data['flamefruit\_active']  
 player.flamefruit\_position = data['flamefruit\_position']  
 player.speed\_boost\_end\_time = data['speed\_boost\_end\_time']  
 player.damage\_reduction = data['damage\_reduction']  
 player.damage\_reduction\_end\_times = data['damage\_reduction\_end\_times']  
 all\_sprites.empty()  
 fruits.empty()  
 enemies.empty()  
 bossenemies.empty()  
 malakar\_group.empty()  
 all\_sprites.add(player)  
  
# Create player and groups  
player = Player(luminara\_img)  
all\_sprites = pygame.sprite.Group()  
fruits = pygame.sprite.Group()  
enemies = pygame.sprite.Group()  
bossenemies = pygame.sprite.Group()  
malakar\_group = pygame.sprite.Group()  
  
all\_sprites.add(player)  
  
fruit\_names\_images = [("Gleam Berry", gleamberry\_img), ("Shimmering Apple", shimmeringapple\_img), ("Ethereal Pear", etherealpear\_img), ("Flamefruit", flamefruit\_img), ("Moonbeam Melon", moonbeammelon\_img)]  
  
# Create initial fruits and enemies  
for \_ in range(10): # Increased number of fruits  
 name, image = random.choice(fruit\_names\_images)  
 fruit = Fruit(random.randint(0, WIDTH-30), random.randint(0, PLAYABLE\_HEIGHT-30), name, image)  
 fruits.add(fruit)  
 all\_sprites.add(fruit)  
  
for \_ in range(10): # Increased number of enemies  
 enemy = NightCrawler(random.randint(0, WIDTH-45), random.randint(0, PLAYABLE\_HEIGHT-45))  
 enemies.add(enemy)  
 all\_sprites.add(enemy)  
  
# Main game loop  
running = True  
paused = False  
game\_won = False  
clock = pygame.time.Clock()  
  
# Timers for spawning  
fruit\_spawn\_time = pygame.time.get\_ticks()  
enemy\_spawn\_time = pygame.time.get\_ticks()  
bossenemy\_spawn\_time = pygame.time.get\_ticks()  
malakar\_spawn\_allowed\_time = pygame.time.get\_ticks() + 30000 # Malakar spawns after 30 seconds  
ripple\_spawn\_allowed\_time = 0  
  
# Variables for displaying fruit name  
show\_fruit\_name = False  
fruit\_name = ""  
fruit\_name\_time = 0  
  
# Variables for rainbow splash  
rainbow\_splash = False  
rainbow\_splash\_time = 0  
  
# Variables for level-up splash  
level\_up\_splash = False  
level\_up\_splash\_time = 0  
  
# Main menu  
def show\_menu():  
 screen.fill(BLACK)  
 draw\_text(screen, "Elysian Grove Adventure", 50, WIDTH // 2, HEIGHT // 4, WHITE)  
 draw\_text(screen, "Press S to Start", 30, WIDTH // 2, HEIGHT // 2, WHITE)  
 draw\_text(screen, "Press L to Load Game", 30, WIDTH // 2, HEIGHT // 2 + 40, WHITE)  
 pygame.display.flip()  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_s:  
 waiting = False  
 if event.key == pygame.K\_l:  
 load\_game()  
 waiting = False  
  
def show\_upgrade\_menu():  
 screen.fill(BLACK)  
 draw\_text(screen, "Upgrade Menu", 50, WIDTH // 2, HEIGHT // 4, WHITE)  
 draw\_text(screen, "Press 1 to Increase Max Health with Gleam Berry", 30, WIDTH // 2, HEIGHT // 2 - 60, WHITE)  
 draw\_text(screen, "Press 2 to Increase Speed with Shimmering Apple", 30, WIDTH // 2, HEIGHT // 2 - 30, WHITE)  
 draw\_text(screen, "Press 3 to Increase Level with Ethereal Pear", 30, WIDTH // 2, HEIGHT // 2, WHITE)  
 draw\_text(screen, "Press 4 to Increase Damage Reduction with Flamefruit", 30, WIDTH // 2, HEIGHT // 2 + 30, WHITE)  
 draw\_text(screen, "Press 5 to Increase Damage with Moonbeam Melon", 30, WIDTH // 2, HEIGHT // 2 + 60, WHITE)  
 draw\_text(screen, "Press P to Resume Game", 30, WIDTH // 2, HEIGHT // 2 + 90, WHITE)  
 pygame.display.flip()  
  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_1 and player.inventory["Gleam Berry"] > 0:  
 player.max\_health += 10  
 player.inventory["Gleam Berry"] -= 1  
 if event.key == pygame.K\_2 and player.inventory["Shimmering Apple"] > 0:  
 player.speed += 1  
 player.inventory["Shimmering Apple"] -= 1  
 if event.key == pygame.K\_3 and player.inventory["Ethereal Pear"] > 0:  
 player.level += 1  
 player.inventory["Ethereal Pear"] -= 1  
 if event.key == pygame.K\_4 and player.inventory["Flamefruit"] > 0:  
 player.damage\_reduction += 5  
 player.inventory["Flamefruit"] -= 1  
 if event.key == pygame.K\_5 and player.inventory["Moonbeam Melon"] > 0:  
 player.damage += 5  
 player.inventory["Moonbeam Melon"] -= 1  
 if event.key == pygame.K\_p:  
 return  
  
show\_menu()  
  
while running:  
 current\_time = pygame.time.get\_ticks()  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_p:  
 paused = not paused  
 if paused:  
 show\_upgrade\_menu()  
 if event.key == pygame.K\_n and not paused:  
 player.special\_attack()  
 if event.key == pygame.K\_s:  
 save\_game()  
  
 if not paused and not game\_won and player.health > 0:  
 # Handle player movement  
 keys = pygame.key.get\_pressed()  
 dx, dy = 0, 0  
 if keys[pygame.K\_LEFT]:  
 dx = -1  
 if keys[pygame.K\_RIGHT]:  
 dx = 1  
 if keys[pygame.K\_UP]:  
 dy = -1  
 if keys[pygame.K\_DOWN]:  
 dy = 1  
 player.move(dx, dy)  
  
 # Check for collisions with fruits  
 collected\_fruits = pygame.sprite.spritecollide(player, fruits, True)  
 for fruit in collected\_fruits:  
 fruit\_name = player.collect\_fruit(fruit)  
 show\_fruit\_name = True  
 fruit\_name\_time = current\_time  
  
 # Check for collisions with enemies  
 enemies\_hit = pygame.sprite.spritecollide(player, enemies, False)  
 bossenemies\_hit = pygame.sprite.spritecollide(player, bossenemies, False)  
 malakar\_hit = pygame.sprite.spritecollide(player, malakar\_group, False)  
 if keys[pygame.K\_SPACE]:  
 for enemy in enemies\_hit:  
 player.attack(enemy)  
 for bossenemy in bossenemies\_hit:  
 player.attack(bossenemy)  
 for malakar in malakar\_hit:  
 player.attack(malakar)  
  
 # Spawn new fruit every 2 seconds  
 if current\_time - fruit\_spawn\_time >= 2000:  
 name, image = random.choice(fruit\_names\_images)  
 fruit = Fruit(random.randint(0, WIDTH-30), random.randint(0, PLAYABLE\_HEIGHT-30), name, image)  
 fruits.add(fruit)  
 all\_sprites.add(fruit)  
 fruit\_spawn\_time = current\_time  
  
 # Spawn regular enemy every 2 seconds  
 if current\_time - enemy\_spawn\_time >= 2000:  
 enemy = NightCrawler(random.randint(0, WIDTH-45), random.randint(0, PLAYABLE\_HEIGHT-45))  
 enemies.add(enemy)  
 all\_sprites.add(enemy)  
 enemy\_spawn\_time = current\_time  
  
 # Spawn boss enemy every 5 seconds if fewer than 3 are present  
 if len(bossenemies) < 3 and current\_time - bossenemy\_spawn\_time >= 5000:  
 bossenemy = BossEnemy(random.randint(0, WIDTH-90), random.randint(0, PLAYABLE\_HEIGHT-90))  
 bossenemies.add(bossenemy)  
 all\_sprites.add(bossenemy)  
 bossenemy\_spawn\_time = current\_time  
  
 # Spawn Malakar if no boss enemies are present and allowed  
 if len(bossenemies) == 0 and len(malakar\_group) == 0 and current\_time > malakar\_spawn\_allowed\_time:  
 malakar = Malakar(random.randint(0, WIDTH-90), random.randint(0, PLAYABLE\_HEIGHT-90))  
 malakar\_group.add(malakar)  
 all\_sprites.add(malakar)  
  
 # Update sprites  
 all\_sprites.update()  
  
 # Draw everything  
 screen.blit(background\_img, (0, 0)) # Draw the background  
 all\_sprites.draw(screen)  
 draw\_rainbow\_splash(screen)  
 draw\_level\_up\_splash(screen)  
 draw\_inventory(screen, player)  
 draw\_legend(screen)  
  
 # Display enemy health as bars above enemy sprites  
 for enemy in enemies:  
 draw\_health\_bar(screen, enemy.rect.x, enemy.rect.y - 10, enemy.health, 100, 40, 5)  
 for bossenemy in bossenemies:  
 draw\_health\_bar(screen, bossenemy.rect.x, bossenemy.rect.y - 10, bossenemy.health, 500, 40, 5)  
 for malakar in malakar\_group:  
 draw\_health\_bar(screen, malakar.rect.x, malakar.rect.y - 10, malakar.health, 5000, 40, 5)  
  
 # Display player health as a larger, more prominent bar  
 draw\_health\_bar(screen, player.rect.x - 10, player.rect.y - 15, player.health, player.max\_health, 60, 10, 2)  
  
 # Draw player stats  
 draw\_text(screen, f'Speed: {player.speed}', 18, WIDTH - 100, 10, WHITE)  
 draw\_text(screen, f'Damage: {player.damage}', 18, WIDTH - 100, 30, WHITE)  
 draw\_text(screen, f'Damage Reduction: {player.damage\_reduction}', 18, WIDTH - 100, 50, WHITE)  
  
 draw\_text(screen, f'Level: {player.level}', 18, 50, 10, WHITE)  
 draw\_text(screen, f'Experience: {player.experience}', 18, 150, 10, WHITE)  
 draw\_text(screen, f'Health: {int(player.health)}', 18, 250, 10, WHITE) # Display health as an integer  
  
 if player.health <= 0:  
 draw\_text(screen, 'GAME OVER', 50, WIDTH // 2, HEIGHT // 2, RED)  
 pygame.display.flip()  
 pygame.time.wait(3000)  
 running = False  
  
 if player.level >= 100:  
 draw\_text(screen, 'YOU WIN', 50, WIDTH // 2, HEIGHT // 2, GREEN)  
 pygame.display.flip()  
 pygame.time.wait(3000)  
 running = False  
 game\_won = True  
  
 if show\_fruit\_name and current\_time - fruit\_name\_time < 1000: # Display fruit name for 1 second  
 draw\_text(screen, f'Collected: {fruit\_name}', 30, WIDTH // 2, PLAYABLE\_HEIGHT + 10, GREEN)  
 else:  
 show\_fruit\_name = False  
  
 if player.invulnerable:  
 draw\_text(screen, 'Status: Invulnerable', 18, WIDTH // 2, PLAYABLE\_HEIGHT + 30, GREEN)  
 if player.damage == 100:  
 draw\_text(screen, 'Status: Increased Damage', 18, WIDTH // 2, PLAYABLE\_HEIGHT + 50, GREEN)  
  
 if paused:  
 draw\_text(screen, 'PAUSED', 50, WIDTH // 2, HEIGHT // 2, BLUE)  
  
 pygame.display.flip()  
 clock.tick(60)  
  
pygame.quit()