FNAFFANGAME.py

Consist of:

- Import settings.py
- Class Enemy
- Game functions
- Main loop

Import settings.py

```
from settings import *  # Import all constants and libraries from settings.py
```

First we need to import all modules, initialization, and game variables from settings with this line of code.

Class Enemy

```
class Enemy:
    def __init__(self, screen_name, jumpscare_image=None):
        self.screen_name = screen_name  # Screen name where the enemy appears
        self.jumpscare_image = jumpscare_image  # Image for the jumpscare animation
        self.active = False  # Whether the enemy is active
        self.cooldown = random.randint(10000, 20000)  # Cooldown time before the enemy can spawn again
        self.last_spawn_time = 0  # Last time the enemy was spawned
        self.spawn_time = 0  # Time when the enemy was spawned
        self.spawn_duration = 10000  # Duration the enemy stays active
        self.jumpscare_frames = []  # List to store jumpscare frames
        self.jumpscare_played = False  # Track if jumpscare has been played
        self.load_jumpscare_frames()  # Load jumpscare frames
        self.jumpscare_sound = None  # Jumpscare sound effect
        self.load_jumpscare_sound()  # Load jumpscare sound
```

This is the first initialization of the enemy. This class only takes screen names and jumpscare images to create a new enemy.

Class Enemy initialization itself consist of:

1. screen_name= to initialize which screen does the animatronic appears
2. jumpscare_image= to initialize the animatronic jumpscare
3. active= a boolean to check whether the enemy is active or not
4. cooldown= a random cooldown after the animatronic despawn
5. last_spawn_time= a placeholder to store the ticks that had passed since the animatronic last spawn (not active)
6. spawn_time= a placeholder to store the ticks that had passed since the animatronic spawned (active)
7. spawn_duration= the duration of the enemy until it jumpscares the player if they didn't despawn the animatronic
8. jumpscare_frames= a list to store the jumpscare frames (animation)
9. jumpscare_played= to track/check whether the jumpscare has been played
10. load_jumpscare_frames= a function that is used to load the jumpscare from the sprite sheet
11. jumpscare_sound= to initialize which sound is used for each jumpscare

12. load_jumpscare_sound= a function used to play the sound according to each jumpscare

Class Enemy also consist of functions like:

1. ***spawn=***

```
def spawn(self):
    """Spawn the enemy at a random location."""
    if not self.active and pygame.time.get_ticks() - self.last_spawn_time > self.cooldown:
        self.active = True  # Activate the enemy
        self.spawn_time = pygame.time.get_ticks()  # Record the spawn time
```

What it does?

Spawning the enemy

How it works?

The function will check whether the animatronic is active or not AND whether the time that had passed minus the last time this animatronic spawn is bigger than the random cooldown. If the condition is true then the animatronic is switched to active and it'll record the ticks since it spawned

2. ***despawn=***

```
def despawn(self):
    """Despawn the enemy."""
    self.active = False  # Deactivate the enemy
    self.last_spawn_time = pygame.time.get_ticks()  # Record the last spawn time
```

What it does?

Despawning the enemy

How it works?

The function will change the state of the animatronic to be deactivated and record the ticks that the animatronic despawned

3. ***update=***

```
def update(self):
    """Automatically despawn the enemy after its duration."""
    if self.active and pygame.time.get_ticks() - self.spawn_time > self.spawn_duration:
        if not self.jumpscare_played:
            self.trigger_jumpscare()  # Trigger the jumpscare
            self.jumpscare_played = True  # Mark the jumpscare as played
```

What it does?

It updates the state of the animatronic, to check whether to jumpscare the player or not

How it works?

The function will check whether the animatronic is active AND the current ticks minus the ticks that have passed since the enemy spawned is bigger than the spawn duration (which is 10 seconds). It'll go to the next if statement, that is if the jumpscare has not been played yet, it'll trigger the jumpscare and change the state of the animatronic's jumpscare_played to be true (to indicate that the jumpscare had been played.

4. ***draw_indicator=***

```
def draw_indicator(self):
    """Draw an image on the respective screen."""
    if self.active:
        if self.screen_name == "left":
            indicator_image = pygame.image.load("images/BB.png").convert_alpha()  # Load the indicator image
        elif self.screen_name == "right":
            indicator_image = pygame.image.load("images/toy_bonnie.png").convert_alpha()  # Load the indicator image
        indicator_image = pygame.transform.scale(indicator_image, (SCREEN_WIDTH, SCREEN_HEIGHT))  # Scale the image to fit the screen
        screen.blit(indicator_image, (0, 0))  # Draw the image on the screen
```

What it does?

It draws each animatronic on their respective screen

How it works?

If the animatronic is active, it'll go to the next if and elif statement where if the screen is showing the "left camera" it'll show Balloon Boy and if the screen is showing the "right camera", it'll show Toy Bonnie instead.

5. ***draw_indicator=***

```
def trigger_jumpscare(self):
    """Display an animated jumpscare and end the game."""
    if self.jumpscare_frames:
        if self.jumpscare_sound and not self.jumpscare_played:
            self.jumpscare_sound.play()  # Play the jumpscare sound effect if loaded and not already played
        for frame in self.jumpscare_frames:
            screen.blit(frame, (0, 0))  # Draw each frame on the screen
            pygame.display.flip()  # Update the display
            pygame.time.delay(20)  # Delay for smooth animation

        screen.blit(self.jumpscare_frames[-1], (0, 0))  # Draw the last frame
        pygame.display.flip()  # Update the display
        pygame.time.delay(3000)  # Delay for 3 seconds
    else:
        jumpscare_image = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT))  # Create a surface for the jumpscare
        jumpscare_image.fill(RED)  # Fill the surface with red color
        screen.blit(jumpscare_image, (0, 0))  # Draw the surface on the screen
        pygame.display.flip()  # Update the display
        pygame.time.delay(3000)  # Delay for 3 seconds
    self.game_over_screen()  # Display the game over screen
```

What it does?

It draws each animatronic on their respective screen

How it works?

If the animatronic is active, it'll go to the next if and elif statement where if the screen is showing the "left camera" it'll show Balloon Boy and if the screen is showing the "right camera", it'll show Toy Bonnie instead.

6. ***trigger_jumpscare=***

```
def trigger_jumpscare(self):
    """Display an animated jumpscare and end the game."""
    if self.jumpscare_frames:
        if self.jumpscare_sound and not self.jumpscare_played:
            self.jumpscare_sound.play()  # Play the jumpscare sound effect if loaded and not already played
        for frame in self.jumpscare_frames:
            screen.blit(frame, (0, 0))  # Draw each frame on the screen
            pygame.display.flip()  # Update the display
            pygame.time.delay(20)  # Delay for smooth animation

        screen.blit(self.jumpscare_frames[-1], (0, 0))  # Draw the last frame
        pygame.display.flip()  # Update the display
        pygame.time.delay(3000)  # Delay for 3 seconds
    else:
        jumpscare_image = pygame.Surface((SCREEN_WIDTH, SCREEN_HEIGHT))  # Create a surface for the jumpscare
        jumpscare_image.fill(RED)  # Fill the surface with red color
        screen.blit(jumpscare_image, (0, 0))  # Draw the surface on the screen
        pygame.display.flip()  # Update the display
        pygame.time.delay(3000)  # Delay for 3 seconds
    self.game_over_screen()  # Display the game over screen
```

What it does?

It displays the animated and the sound of the jumpscare, then end the game

How it works?

If the jumpscare_frames is filled with the frames then this function will check whether the jumpscare sound is not empty AND the jumpscare has not been played yet. and then it'll play the sound of the jumpscare. Then there's a loop function to display each frame from jumpscare_frames. The last frame will stay for 3 seconds before it shows the game over screen. If there's no jumpscare frames, then it'll show a red screen then game over screen.

7. ***load_jumpscare_frame=***

```
def load_jumpscare_frames(self):
    """Load jumpscare frames from sprite sheet."""
    if self.jumpscare_image:
        try:
            if self.jumpscare_image == "images/BBjumpscare.png":
                sheet = pygame.image.load(self.jumpscare_image).convert_alpha()  # Load the sprite sheet
                sheet_width, sheet_height = sheet.get_size()  # Get the size of the sprite sheet
                cols, rows = 5, 10  # Number of columns and rows in the sprite sheet
                total_frames = 51  # Total number of frames
            elif self.jumpscare_image == "images/toy_bonniejumpscare.png":
                sheet = pygame.image.load(self.jumpscare_image).convert_alpha()  # Load the sprite sheet
                sheet_width, sheet_height = sheet.get_size()  # Get the size of the sprite sheet
                cols, rows = 5, 9  # Number of columns and rows in the sprite sheet
                total_frames = 41  # Total number of frames
            else:
                print(f"Unknown jumpscare image: {self.jumpscare_image}")
                return
            frame_width = sheet_width // cols  # Width of each frame
            frame_height = sheet_height // rows  # Height of each frame

            for row in range(rows):
                for col in range(cols):
                    if len(self.jumpscare_frames) < total_frames:
                        frame_rect = pygame.Rect(col * frame_width, row * frame_height, frame_width, frame_height)  # Rectangle representing the frame
                        frame = sheet.subsurface(frame_rect)  # Extract the frame from the sprite sheet
                        scaled_frame = pygame.transform.scale(frame, (SCREEN_WIDTH, SCREEN_HEIGHT))  # Scale the frame to fit the screen
                        self.jumpscare_frames.append(scaled_frame)  # Add the frame to the list
        except pygame.error as e:
            print(f"Error loading jumpscare frames: {e}")
```

What it does?

To load the jumpscare frames to the jumpscare_frames

How it works?

It takes the cols, rows, and total_frames to split each sprite sheet according to their size. Then append them to the jumpscare_frames

8. ***load_jumpscare_sound=***

```
def load_jumpscare_sound(self):
    """Load the jumpscare sound effect."""
    if self.jumpscare_image == "images/BBjumpscare.png":
        self.jumpscare_sound = pygame.mixer.Sound("sound/BBjumpscaresound.wav")  # Load the jumpscare sound effect
    elif self.jumpscare_image == "images/toy_bonniejumpscare.png":
        self.jumpscare_sound = pygame.mixer.Sound("sound/bonniejumpscaresound.wav")  # Load the jumpscare sound effect
    else:
        print(f"Unknown jumpscare image: {self.jumpscare_image}")
        self.jumpscare_sound = None  # Set to None if unknown image
```

What it does?

To load the jumpscare frames to the jumpscare_frames

How it works?

It takes the cols, rows, and total_frames to split each sprite sheet according to their size, then append them to the jumpscare_frames.

9. ***game_over_screen=***

```
def game_over_screen(self):
    """Display a game over screen with retry and placeholder for main menu."""
    font_large = pygame.font.Font('font/minecraft.ttf', 100)  # Load a large font
    font_small = pygame.font.Font('font/minecraft.ttf', 50)  # Load a small font

    game_over = True
    while game_over:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()  # Quit the game
                exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_r:
                    game_over = False  # Exit the game over screen
                    restart_dinosaur_game()  # Restart the game

        screen.fill(BLACK)  # Fill the screen with black color
        game_over_text = font_large.render("GAME OVER", True, RED)  # Render the game over text
        retry_text = font_small.render("Press R to Retry", True, WHITE)  # Render the retry text
        screen.blit(game_over_text, game_over_text.get_rect(center=(gameover_x,gameover_y)))  # Draw the game over text
        screen.blit(retry_text, retry_text.get_rect(center=(retry_x,retry_y)))  # Draw the retry text
        pygame.display.flip()  # Update the display
```

What it does?

To display the game over screen

How it works?

Displays the game over screen with an option to retry

With these functions, I made 2 enemies:

```
# Create two enemies
red_enemy = Enemy("left", "images/BBjumpscare.png")  # Create a red enemy
blue_enemy = Enemy("right", "images/toy_bonniejumpscare.png")  # Create a blue enemy
```

Game functions

I also created some game functions that helps a lot in gameplay sense:

1. *load_dino_sprite_sheet:*

```python
def load_dino_sprite_sheet():
    """Load the dinosaur sprite sheet and extract frames for running and jumping."""
    sheet = pygame.image.load(sprite_sheet_path).convert_alpha()  # Load the sprite sheet
    sprite_width = sheet.get_width() // 5  # 5 columns in the sprite sheet
    sprite_height = sheet.get_height() // 7  # 7 rows in the sprite sheet
    running_frames = []

    # Extract frames for running (top row)
    for i in range(5):
        frame = sheet.subsurface((i * sprite_width, 0, sprite_width, sprite_height))  # Extract frame
        running_frames.append(pygame.transform.scale(frame, (DINOSAUR_WIDTH, DINOSAUR_HEIGHT)))  # Scale and add to list

    # Extract frame for jumping (bottom-left frame)
    jump_frame = sheet.subsurface((0, 6 * sprite_height, sprite_width, sprite_height))  # Extract frame
    jump_frame = pygame.transform.scale(jump_frame, (DINOSAUR_WIDTH, DINOSAUR_HEIGHT))  # Scale the frame

    return running_frames, jump_frame  # Return the frames
```

What it does?

Loads the dino animation based on it's sprite sheet

How it works?

It takes the sprite width and height based on how many frames there are in the sprite sheet. For the running frame it takes the top row and for the jump frame, it takes the bottom left frame because it fits best for the jumping animation, then returns it as running frame and jump frame.

```python
running_frames, jump_frame = load_dino_sprite_sheet()  # Load the frames
```

Is used to load the variables outside the function

2. *update_dino_animation:*

```python
def update_dino_animation():
    """Update Dino animation based on running or jumping state."""
    global current_frame_index, animation_timer
    if jumping:
        # If the dinosaur is jumping, return the jump frame
        return jump_frame
    else:
        # If the dinosaur is running, cycle through running frames
        animation_timer += ANIMATION_SPEED  # Increment the animation timer by the animation speed
        if animation_timer >= 1:
            # If the timer exceeds or equals 1, update the frame index
            current_frame_index = (current_frame_index + 1) % len(running_frames)  # Move to the next frame, looping back to the start if necessary
            animation_timer = 0  # Reset the animation timer
        return running_frames[current_frame_index]  # Return the current running frame based on the frame index
```

What it does?

Updates the dino animation when it's walking and jumping

How it works?

When it's jumping it uses the jump_frame variable and for the running it cycles through the running_frames by using all the code inside the else statement.

3. *wingamecondition:*

```python
def wingamecondition():
    if score >= 143:  # Check if the score reaches 143
        return True  # End the game if the score reaches 143
    return False  # Continue the game
```

What it does?

Define the first win condition, that is if you have a score equal or exceed 143

How it works?

If the score equal to 143 it'll return True, other than that it's False

4. ***restart_dinosaur_game:***

```
def restart_dinosaur_game():
    """Reset the game variables."""
    global dino_game_active, score, obstacles, jumping, velocity_y, dino_paused, current_pov
    global hours, game_start_time, current_time_label, dinosaur_y
    # Reset variables
    dino_game_active = True
    score = 0
    hours = 12
    obstacles = []
    jumping = False
    velocity_y = 0
    dino_paused = False
    current_pov = "center"
    dino.y = dinosaur_y
    red_enemy.despawn()
    blue_enemy.despawn()
    pygame.event.clear(SPAWN_OBSTACLE_EVENT)
    red_enemy.jumpscare_played = False
    blue_enemy.jumpscare_played = False
    game_start_time = pygame.time.get_ticks()
```

What it does?

It actually restarts the whole game, not just the dinosaur game. It restarts it by making sure each variable that's used returns to its original state.

How it works?

Rewrite each variable to each original state.

5. ***draw_pause_menu:***

```
def draw_pause_menu():
    """Draw the pause menu for the Dinosaur Game."""
    pause_text = font.render("Dinosaur Game Paused", True, BLACK)  # Render the pause text
    resume_text = font.render("Press ESC to Resume", True, BLACK)  # Render the resume text

    # Draw the pause text in the center of the screen
    screen.blit(pause_text, (pause_x, pause_y))
    # Draw the resume text below the pause text
    screen.blit(resume_text, (resume_x,resume_y))

    pygame.display.update()  # Update the display to show the pause menu
```

What it does?

Draws the pause menu for the dinosaur game

How it works?

Put the text of pause menu when the player hit "esc"

6. ***update_timer:***

```
def update_timer():
    """Update the timer and display the current time."""
    global current_time_label, minutesinmil, hours
    elapsed_time = pygame.time.get_ticks() - game_start_time  # Calculate elapsed time since game start
    total_minutes = elapsed_time // minutesinmil  # Convert elapsed time to minutes
    if total_minutes >= 12:
        return True  # End the game if 12 minutes have passed

    hours = 12 + total_minutes // 2  # Calculate the current hour
    minutes = (total_minutes % 2) * 30  # Calculate the current minutes (0 or 30)
    if hours > 12:
        hours -= 12  # Adjust hours to 12-hour format

    current_time_label = f"{hours}:{minutes:02d} AM"  # Format the current time label
    return False  # Continue the game
```

What it does?

Define the second winning condition, that's to reach 6 AM (12 minutes since 12 AM (game time))

How it works?

Update the time based on the elapsed time and set the boolean if statement for the winning state. Other than that is just calculation to show each hour and minute the right way (updates each 30 minutes in game time)

7. ***draw_timer:***

```
def draw_timer():
    """Draw the timer on the screen."""
    timer_text = font.render(current_time_label, True, BLACK)
    screen.blit(timer_text, (SCREEN_WIDTH - timer_text.get_width() - 20, SCREEN_HEIGHT - timer_text.get_height() - 20))
```

What it does?

Draw the timer on the bottom right of the screen

How it works?

It draws it based on the calculation of the screen and its own width and height. I put minus 20 to act as a "padding" so it doesn't stick to the inner box

Main loop

For the main loop, it consist of:

1. ***Importing global & define running***

```
def main():
    global jumping, velocity_y, score, dino_game_active, dino_paused, current_pov, last_pov_change, flashlight, flashlightkeyduration
    global gameover_x, gameover_y, retry_x, retry_y, pause_x, pause_y, resume_x, resume_y
    running = True
```

Importing global variables so I can use it in the main function. Also defining running as true, this is the variable I use for the loop sequence.

2. ***Main loop & win condition***

```
while running:
    if update_timer() or wingamecondition():
        winsound.play()  # Play the win sound
        scaled_win_image = pygame.transform.scale(win_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
        screen.blit(scaled_win_image, (0, 0))
        pygame.display.flip()
        pygame.time.delay(5000)  # Display the win screen for 5 seconds
        running = False  # Game won by surviving until 6 AM or reaching 143 points
```

This is where I define my main loop which is “while running”. Then I created an if statement for the win condition (if the function update_timer() or wingamecondition() is true). If the win condition is met, the win screen and sound are played, then the game is closed.

3. ***Cameras***

```
# Apply camera function based on POV (covers the entire screen)
if current_pov == "left":
    screen.fill(BLACK)
    red_enemy.draw_indicator()
elif current_pov == "right":
    screen.fill(BLACK)
    blue_enemy.draw_indicator()
else:
    screen.blit(background_image, (0, 0))
```

This is where I created a placeholder for each camera

4. *Dinosaur jumping & obstacles*

```
# Dinosaur game (inside the white box)
# Draw the game box border
pygame.draw.rect(screen, BLACK, (gameborder_x, gameborder_y, gamerborder_height, gameborder_width), 5)

if dino_game_active:
    if not dino_paused:
        # Update dinosaur position if jumping
        if jumping:
            dino.y += velocity_y  # Update vertical position
            velocity_y += gravity  # Apply gravity
            if dino.y >= dinosaur_y:
                dino.y = dinosaur_y  # Reset position if on the ground
                jumping = False  # Stop jumping

        # Update obstacles
        obstacle_speed = base_obstacle_speed + score // 1000  # Increase obstacle speed based on score
        for obstacle in list(obstacles):
            obstacle['rect'].x -= obstacle_speed  # Move obstacle to the left
            if obstacle['rect'].right < BOX_X + 30:  # Check if obstacle is near the left edge of the box
                obstacle['alpha'] -= 68  # Decrease obstacle transparency
                if obstacle['alpha'] <= 0:  # Remove obstacle if fully transparent
                    obstacles.remove(obstacle)
                    score += 1  # Increase score for successfully avoiding the obstacle
            else:
                if obstacle['alpha'] < 255:  # Increase obstacle transparency if not fully opaque
                    obstacle['alpha'] += 10  # Increase obstacle transparency

            if dino.colliderect(obstacle['rect']):  # Check collision with obstacles
                score -= 10  # Decrease score by 10 for collision
                if score < 0:
                    score = 0  # Ensure score doesn't go below 0
                obstacles.remove(obstacle)  # Remove the obstacle
                dino.x = dinosaur_x
                dino.y = dinosaur_y  # Reset dinosaur's vertical position
                break  # Exit the loop after handling collision
```

This is part of the dinosaur game code. In this line of codes, I put the jumping mechanics by using the variables that could affect dino's y position. Then I define the obstacle that would get faster each time the player gets a higher score. For the obstacle to smoothly enter the inner box, I used alpha as a way to increase or decrease the opacity of each obstacle. In the loop I created a colliderect if statement to check whether the player hit the obstacle or not. If they are then the score is deduced and the whole loop is broken (but for error control, I put another if statement so the score won't get any lower than 0).

5. *Obstacles spacing correction*

```
        # Ensure obstacles don't spawn too close to each other
        if obstacles and obstacles[-1]['rect'].x < OBSCTALE_SPAWN_X:
            pygame.time.set_timer(SPAWN_OBSTACLE_EVENT, random.randint(1500, 2500))
```

To make sure the obstacle won't spawn right next to each other and make the player can't jump over it, I created a function to make sure the "OBSTACLE_SPAWN_X" acts as a barrier. If the obstacle went past the barrier, then it can append another obstacle.

6. *Updating sky, ground, and dino*

```
#Draw sky and ground
screen.blit(sky_image, (BOX_X, BOX_Y))  # Draw the sky image
screen.blit(ground_image, (ground_x,ground_y))  # Draw the ground image

# Draw Dino with animation
dino_frame = update_dino_animation()  # Get the current frame for the dinosaur animation
screen.blit(dino_frame, dino)  # Draw the dinosaur frame
```

Pretty self-explanatory, it displays the surface of sky, ground, and dino animation in order to make sure it won't overlap.

7. *Draw obstacle*

```
# Draw obstacles
for obstacle in obstacles:
    obstacle_image = pygame.image.load("images/obstacle.png").convert_alpha()  # Load the obstacle image
    obstacle_image = pygame.transform.scale(obstacle_image, (obstacle['rect'].width, obstacle['rect'].height))  # Scale the obstacle image
    obstacle_image.set_alpha(obstacle['alpha'])  # Set the transparency of the obstacle image
    screen.blit(obstacle_image, obstacle['rect'].topleft)  # Draw the obstacle image
```

It applies "obstacle.png" into the rectangle.

8. *Draw score*

```
# Draw score
score_text = font.render(f"Score: {score}", True, BLACK)  # Render the score text
screen.blit(score_text, (BOX_X + BOX_WIDTH - score_text.get_width() - 20, BOX_Y + BOX_HEIGHT - score_text.get_height() - 20))  # Draw the score text
```

Draw the score on the bottom right of the inner box

9. *Dino pause, spawning, and updating enemies*

```
        if dino_paused:
            draw_pause_menu()  # Draw the pause menu if the game is paused

        # Spawn and handle enemies
        red_enemy.spawn()  # Spawn the red enemy
        blue_enemy.spawn()  # Spawn the blue enemy

        red_enemy.update()  # Update the red enemy
        blue_enemy.update()  # Update the blue enemy
```

Drawing the pause menu using the "draw_pause_menu()" function. Updating the enemy by spawning it or updating it based on the class.

10. *Event handling*

```
        # Event handling
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == SPAWN_OBSTACLE_EVENT and dino_game_active and not dino_paused:
                obstacle_height = random.randint(20, 60)

                obstacle = {
                    'rect': pygame.Rect(580, 350- obstacle_height, OBSTACLE_WIDTH, obstacle_height),
                    'alpha': 0
                }
                obstacles.append(obstacle)
                pygame.time.set_timer(SPAWN_OBSTACLE_EVENT, random.randint(1500, 2500))
```

Event handling loop, if the player want to quit the game and for spawning the obstacle

11. Player's input

```
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_r:
                Enemy.jumpscare_played = False
                if not dino_game_active:
                    restart_dinosaur_game()
                elif dino_paused:
                    dino_paused = False
            if event.key == pygame.K_ESCAPE and current_pov == "center":
                dino_paused = not dino_paused
            if event.key == pygame.K_f:
                key_held_start = pygame.time.get_ticks()  # Start tracking key press time
                flashlight = True

        # Detect key release
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_f:
                key_held_start = 0  # Reset when key is released
                flashlight = False
```

Handling player's input by using "pygame.KEYDOWN" and "pygame.KEYUP". Here's the list of what they can do:

a. "R" for restarting the game
b. "esc" to pause dinosaur game
c. "F" to use the flashlight and to record how long the flashlight has been on

12. Flashlight images

```
# Show flashlight image if flashlight is true
if flashlight:
    if red_enemy.active and current_pov == "left":
        indicator_image = pygame.image.load("images/BBflashlight.png").convert_alpha()
        indicator_image = pygame.transform.scale(indicator_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
        screen.blit(indicator_image, (0, 0))
    elif blue_enemy.active and current_pov == "right":
        indicator_image = pygame.image.load("images/toy_bonnieflashlight.png").convert_alpha()
        indicator_image = pygame.transform.scale(indicator_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
        screen.blit(indicator_image, (0, 0))
    elif current_pov == "left" or current_pov == "right":
        indicator_image = pygame.image.load("images/noneflashlight.png").convert_alpha()
        indicator_image = pygame.transform.scale(indicator_image, (SCREEN_WIDTH, SCREEN_HEIGHT))
        screen.blit(indicator_image, (0, 0))
    elif current_pov == "center":
        flashlight = False  # Disable flashlight in center POV
```

Implementing each flashlight image to each condition.

13. Another user input

```
# User input
keys = pygame.key.get_pressed()
if keys[pygame.K_SPACE] and not jumping and dino_game_active and not dino_paused:
    jumping = True
    velocity_y = jump_velocity
if keys[pygame.K_a] and pygame.time.get_ticks() - last_pov_change > 500:
    current_pov = "left" if current_pov == "center" else "center"
    last_pov_change = pygame.time.get_ticks()
if keys[pygame.K_d] and pygame.time.get_ticks() - last_pov_change > 500:
    current_pov = "right" if current_pov == "center" else "center"
    last_pov_change = pygame.time.get_ticks()
if keys [pygame.K_f]:
    if key_held_start and pygame.time.get_ticks() - key_held_start > flashlightkeyduration:
        if red_enemy.active and current_pov == "left":
            red_enemy.despawn()
        elif blue_enemy.active and current_pov == "right":
            blue_enemy.despawn()
        key_held_start = pygame.time.get_ticks()  # Reset the timer to allow continuous despawning
```

This one is using the “pygame.key.get.pressed()” to check player input continuously. Here’s the list of what the player can do:

a. “spacebar” for jumping
b. “a” to check left camera with a cooldown of 0.5 seconds
c. “d” to check right camera with a cooldown of 0.5 seconds
d. “f” to record how long the flashlight has been activated and despawn the enemy

14. Timer, updating the screen, setting the FPS, closing the game, and running the main loop

```
        # Draw timer
        draw_timer()

        pygame.display.flip()
        clock.tick(FPS)

    pygame.quit()

if __name__ == "__main__":
    main()
```

Drawing the timer using “draw_timer()”, updating the whole screen with the updates, setting the FPS, calling “pygame.quit()” to make sure there’s no resources left, and using the if statement to call main() so it can’t be imported to other files.

settings.py

Consist of:

- Import and initialization
- Game variables

Import library

```
import pygame  # Import the pygame library
import random  # Import the random library
```

Importing “pygame” as our base library to make the game and “random” for the event handling.

Initialization

```
# Initialize pygame and mixer
pygame.init()
pygame.mixer.init()
```

To initialize pygame modules and the mixer modules to make sure it’s properly set up.

Game Variables

```
# Screen dimensions
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)

# Game settings
# Ground height for the dinosaur game
GROUND_HEIGHT = 500

# Dimensions and position of the game box
BOX_WIDTH = 400
BOX_HEIGHT = 300
BOX_X = BOX_WIDTH // 2
BOX_Y = BOX_HEIGHT // 2
```

Initializing screen (height and width), colors, ground height, and the components for inner box

```
# Initialize the game screen
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("FNAF FAN GAME")

# Load background image for "center" POV
background_image = pygame.image.load("images/Background.jpg").convert()
background_image = pygame.transform.scale(background_image, (SCREEN_WIDTH, SCREEN_HEIGHT))

# Clock for controlling the frame rate
clock = pygame.time.Clock()
FPS = 60

# Game variables
DINOSAUR_WIDTH, DINOSAUR_HEIGHT = 40, 60  # Dimensions of the dinosaur
dinosaur_x, dinosaur_y = 300, 290 # Initial position of the dinosaur
dino = pygame.Rect(dinosaur_x, dinosaur_y, DINOSAUR_WIDTH, DINOSAUR_HEIGHT)

# Obstacle settings
OBSTACLE_WIDTH = 20
OBSCTALE_SPAWN_X= 500 # Initial x position of the obstacle to make sure it's far enough before adding another one
obstacles = []  # List to store obstacles
SPAWN_OBSTACLE_EVENT = pygame.USEREVENT + 1  # Custom event for spawning obstacles
pygame.time.set_timer(SPAWN_OBSTACLE_EVENT, random.randint(1500, 2500))  # Set timer for obstacle spawning
```

Initializing the screen, caption, main background image, clock, FPS, dinosaur game variables, and obstacle variables

```
# Jumping mechanics
jumping = False
jump_velocity = -15  # Initial jump velocity
gravity = 1  # Gravity affecting the dinosaur
velocity_y = 0  # Vertical velocity of the dinosaur

# Game state variables
score = 0  # Player's score
flashlight = False  # Flashlight state
flashlightkeyduration = 2000  # Duration for holding the flashlight key
dino_game_active = True  # State of the dinosaur game
dino_paused = False  # Pause state of the game
current_pov = "center"  # Current point of view (camera)
last_pov_change = pygame.time.get_ticks()  # Time of the last POV change
```

Initializing jumping mechanic variables, also initial game state variables

```
# Draw sky
sky_image = pygame.image.load("images/sky.png").convert_alpha()  # Load the sky image
sky_x, sky_y = 400, 240  # Position of the sky image
sky_image = pygame.transform.scale(sky_image, (sky_x, sky_y))  # Adjust height to fit between ground and top of box

# Draw ground
ground_image = pygame.image.load("images/ground.png").convert_alpha()  # Load the ground image
groundimg_height, groundimg_width = 400, 60  # Position of the ground image
ground_x,ground_y= 200, 330  # Position of the ground image
ground_image = pygame.transform.scale(ground_image, (groundimg_height,groundimg_width))  # Scale the ground image to fit the box width and dinosaur height

#Text settings
gameover_x, gameover_y = 400, 200  # Position of the game over image
retry_x, retry_y = 400, 300  # Position of the retry image
pause_x, pause_y = 260, 250 # Position of the pause image
resume_x, resume_y= 260, 300 # Position of the resume image
gameborder_x, gameborder_y, gamerborder_height, gameborder_width= 195, 145, 410, 310  # Position and dimensions of the game border
```

Initializing sky, ground, and text placement variables

```
# Timer variables
game_start_time = pygame.time.get_ticks()  # Start time of the game
minutesinmil= 60000 # 1 Minute in milliseconds
current_time_label = "12:00 AM"  # Initial time label

# Font settings
font = pygame.font.Font("font/Minecraft.ttf", 24)  # Font for rendering text

# Initialize key_held_start
key_held_start = 0 # Start time for tracking key press duration

# Base speed for obstacles
base_obstacle_speed = 5

# Load win screen image
win_image = pygame.image.load("images/winscreen.jpg").convert()
winsound= pygame.mixer.Sound("sound/winsound.wav")
```

Initializing timer variables, font, flashlight, obstacle speed, and win screen variables

```
# Dinosaur Animation State
current_frame_index = 0  # Current frame index for running animation
ANIMATION_SPEED = 0.15  # Speed of the animation (bigger is faster)
animation_timer = 0  # Timer for animation

# Load Dino Sprite Sheet
sprite_sheet_path = "images/purpguy.png"  # Path to the sprite sheet
```

Initializing dino animation variables and loading dino sprite sheet