MARIO GAME

```
import pygame
import sys

# Initialize Pygame
pygame.init()

# Constants
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
FPS = 60

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
BLUE = (52, 152, 219)
GREEN = (46, 204, 113)
RED = (231, 76, 60)
YELLOW = (241, 196, 15)
BROWN = (139, 69, 19)

# Player class (Mario-style)
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.width = 40
        self.height = 50
        self.image = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
        self.draw_character()
        self.rect = self.image.get_rect()
        self.rect.x = 100
        self.rect.y = SCREEN_HEIGHT - 150
        self.velocity_y = 0
        self.velocity_x = 0
        self.on_ground = False
        self.speed = 5
        self.jump_power = 15
        self.gravity = 0.8
        self.facing_right = True

    def draw_character(self):
        """Draw a Mario-style character"""
        self.image.fill((0, 0, 0, 0))  # Clear with transparency

        # Hat (red)
        pygame.draw.rect(self.image, (220, 20, 20), (8, 5, 24, 8))
        # Brim of hat
        pygame.draw.rect(self.image, (139, 69, 19), (5, 12, 30, 4))

        # Face (peach/skin tone)
        pygame.draw.rect(self.image, (255, 220, 177), (10, 16, 20, 12))

        # Eyes (black dots)
        pygame.draw.circle(self.image, BLACK, (16, 22), 2)
        pygame.draw.circle(self.image, BLACK, (24, 22), 2)

        # Mustache (brown)
        pygame.draw.rect(self.image, (101, 67, 33), (12, 24, 16, 3))

        # Shirt (blue)
        pygame.draw.rect(self.image, (30, 144, 255), (8, 28, 24, 12))

        # Overalls (blue)
        pygame.draw.rect(self.image, (0, 100, 200), (10, 32, 20, 8))

        # Buttons (yellow)
        pygame.draw.circle(self.image, YELLOW, (14, 34), 2)
        pygame.draw.circle(self.image, YELLOW, (26, 34), 2)
        # Legs (blue overalls)
        pygame.draw.rect(self.image, (0, 100, 200), (12, 40, 6, 8))
        pygame.draw.rect(self.image, (0, 100, 200), (22, 40, 6, 8))

        # Shoes (brown)
        pygame.draw.rect(self.image, (139, 69, 19), (10, 48, 8, 4))
        pygame.draw.rect(self.image, (139, 69, 19), (22, 48, 8, 4))

    def update(self):
        # Apply gravity
        self.velocity_y += self.gravity

        # Horizontal movement
        keys = pygame.key.get_pressed()
        self.velocity_x = 0

        if keys[pygame.K_LEFT]:
            self.velocity_x = -self.speed
            if self.facing_right:
                self.facing_right = False
                self.image = pygame.transform.flip(self.image, True, False)
        if keys[pygame.K_RIGHT]:
            self.velocity_x = self.speed
            if not self.facing_right:
                self.facing_right = True
                self.image = pygame.transform.flip(self.image, True, False)

        # Update position
        self.rect.x += self.velocity_x
        self.rect.y += self.velocity_y

        # Keep player on screen (horizontal)
        if self.rect.left < 0:
            self.rect.left = 0
        if self.rect.right > SCREEN_WIDTH:
            self.rect.right = SCREEN_WIDTH

        # Bottom boundary
        if self.rect.bottom >= SCREEN_HEIGHT:
            self.rect.bottom = SCREEN_HEIGHT
            self.velocity_y = 0
            self.on_ground = True

    def jump(self):
        if self.on_ground:
            self.velocity_y = -self.jump_power
            self.on_ground = False

    def check_platform_collision(self, platforms):
        # Check collision with platforms
        for platform in platforms:
            if self.rect.colliderect(platform.rect):
                # Landing on top of platform
                if self.velocity_y > 0 and self.rect.bottom <= platform.rect.top + 20:
                    self.rect.bottom = platform.rect.top
                    self.velocity_y = 0
                    self.on_ground = True

# Platform class (Brick blocks)
class Platform(pygame.sprite.Sprite):
    def __init__(self, x, y, width, height):
        super().__init__()
        self.image = pygame.Surface((width, height))
        self.draw_brick_pattern(width, height)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    def draw_brick_pattern(self, width, height):
        """Draw brick block pattern"""
        # Base color
        self.image.fill((180, 100, 50))

        # Draw brick pattern
        brick_width = 30
        brick_height = 15

        for row in range(0, int(height), brick_height):
            offset = (row // brick_height) % 2 * (brick_width // 2)
            for col in range(-offset, int(width), brick_width):
                if col >= 0 and col < width:
                    # Brick outline
                    pygame.draw.rect(self.image, (150, 80, 40),
                                   (col, row, min(brick_width, width - col),
                                    min(brick_height, height - row)), 2)
                    # Highlight
                    pygame.draw.line(self.image, (220, 140, 80),
                                   (col + 2, row + 2),
                                   (min(col + brick_width - 2, width - 2), row + 2), 2)

# Grass decoration
class GrassPlatform(pygame.sprite.Sprite):
    def __init__(self, x, y, width):
        super().__init__()
        self.image = pygame.Surface((width, 10), pygame.SRCALPHA)
        self.draw_grass(width)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    def draw_grass(self, width):
        """Draw grass decoration"""
        for i in range(0, width, 8):
            # Grass blades
            pygame.draw.polygon(self.image, (34, 139, 34),
                    [(i, 10), (i + 2, 0), (i + 4, 10)])
            pygame.draw.polygon(self.image, (50, 205, 50),
                    [(i + 3, 10), (i + 5, 2), (i + 7, 10)])

# Mushroom power-up class
class Mushroom(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.Surface((24, 24), pygame.SRCALPHA)
        self.draw_mushroom()
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.velocity_x = 1.5

    def draw_mushroom(self):
        """Draw a Mario-style super mushroom"""
        self.image.fill((0, 0, 0, 0))

        # Mushroom stem (cream)
        pygame.draw.rect(self.image, (250, 240, 230), (8, 14, 8, 10))

        # Mushroom cap (red)
        pygame.draw.ellipse(self.image, (220, 20, 60), (2, 4, 20, 14))

        # White spots
        pygame.draw.circle(self.image, WHITE, (7, 9), 3)
        pygame.draw.circle(self.image, WHITE, (17, 9), 3)
        pygame.draw.circle(self.image, WHITE, (12, 13), 2)

        # Eyes
        pygame.draw.circle(self.image, BLACK, (9, 17), 1)
        pygame.draw.circle(self.image, BLACK, (15, 17), 1)

    def update(self):
        """Move the mushroom"""
        self.rect.x += self.velocity_x
        if self.rect.left <= 0 or self.rect.right >= SCREEN_WIDTH:
            self.velocity_x *= -1

# Coin class (Mario-style spinning coin)
class Coin(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.Surface((24, 24), pygame.SRCALPHA)
        self.draw_coin()
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.frame = 0

    def draw_coin(self):
        """Draw a Mario-style coin"""
        self.image.fill((0, 0, 0, 0))

        # Outer circle (gold)
        pygame.draw.circle(self.image, (255, 215, 0), (12, 12), 10)
        # Inner circle (darker gold)
        pygame.draw.circle(self.image, (218, 165, 32), (12, 12), 8)
        # Shine effect
        pygame.draw.circle(self.image, (255, 255, 150), (9, 9), 3)

    def update(self):
        """Animate the coin"""
        self.frame += 1
        if self.frame % 10 == 0:  # Slow rotation
            # Simple width oscillation for spin effect
            pass  # Keep it simple for now

# Goal/Flag class for level completion
```

```
class Goal(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.Surface((40, 60), pygame.SRCALPHA)
        self.draw_flag()
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    def draw_flag(self):
        """Draw a goal flag"""
        self.image.fill((0, 0, 0, 0))

        # Flag pole (black)
        pygame.draw.rect(self.image, BLACK, (18, 0, 4, 60))

        # Flag (checkered pattern)
        flag_colors = [(0, 200, 0), (255, 255, 255)]
        for row in range(2):
            for col in range(2):
                color = flag_colors[(row + col) % 2]
                pygame.draw.rect(self.image, color,
                               (0 + col * 10, 5 + row * 10, 10, 10))

        # Flag outline
        pygame.draw.rect(self.image, BLACK, (0, 5, 20, 20), 2)

        # Pole top (golden ball)
        pygame.draw.circle(self.image, YELLOW, (20, 3), 4)

# Enemy class (Goomba-style)
class Enemy(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.width = 30
        self.height = 30
        self.image = pygame.Surface((self.width, self.height), pygame.SRCALPHA)
        self.draw_goomba()
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.velocity_x = 2

    def draw_goomba(self):
        """Draw a Goomba-style enemy"""
        self.image.fill((0, 0, 0, 0))

        # Body (brown mushroom shape)
        pygame.draw.ellipse(self.image, (139, 90, 43), (3, 8, 24, 20))

        # Angry eyebrows
        pygame.draw.line(self.image, BLACK, (8, 12), (12, 14), 3)
        pygame.draw.line(self.image, BLACK, (22, 12), (18, 14), 3)

        # Eyes (white with black pupils)
        pygame.draw.circle(self.image, WHITE, (10, 16), 4)
        pygame.draw.circle(self.image, WHITE, (20, 16), 4)
        pygame.draw.circle(self.image, BLACK, (10, 16), 2)
        pygame.draw.circle(self.image, BLACK, (20, 16), 2)

        # Fangs/teeth
        pygame.draw.polygon(self.image, WHITE, [(12, 22), (15, 22), (13, 25)])
        pygame.draw.polygon(self.image, WHITE, [(15, 22), (18, 22), (16, 25)])

        # Feet
        pygame.draw.ellipse(self.image, (101, 67, 33), (2, 26, 10, 6))
        pygame.draw.ellipse(self.image, (101, 67, 33), (18, 26, 10, 6))

    def update(self):
        self.rect.x += self.velocity_x
        # Bounce off screen edges
        if self.rect.left <= 0 or self.rect.right >= SCREEN_WIDTH:
            self.velocity_x *= -1

# Game class
class Game:
    def __init__(self):
        self.screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))
        pygame.display.set_caption("Mario-Style Platformer")
        self.clock = pygame.time.Clock()
        self.running = True
        self.score = 0
        self.lives = 3
        self.powered_up = False
        self.current_level = 1
        self.max_level = 3

        # Sprite groups
        self.all_sprites = pygame.sprite.Group()
        self.platforms = pygame.sprite.Group()
        self.coins = pygame.sprite.Group()
        self.enemies = pygame.sprite.Group()
        self.mushrooms = pygame.sprite.Group()

        # Create player
        self.player = Player()
        self.all_sprites.add(self.player)

        # Create level
        self.create_level()

        # Add decorative clouds
        self.clouds = []
        for i in range(5):
            self.clouds.append({
                'x': i * 200,
                'y': 50 + (i % 3) * 40,
                'speed': 0.3 + (i % 2) * 0.2
            })

        # Font
        self.font = pygame.font.Font(None, 36)

    def create_level(self):
        """Create level based on current_level"""
        # Clear existing level objects
        for sprite in self.platforms:
            sprite.kill()
        for sprite in self.coins:
            sprite.kill()
        for sprite in self.enemies:
            sprite.kill()
        for sprite in self.mushrooms:
            sprite.kill()

        if self.current_level == 1:
            self.create_level_1()
        elif self.current_level == 2:
            self.create_level_2()
        elif self.current_level == 3:
            self.create_level_3()

    def create_level_1(self):
        """Level 1 - Easy introduction"""
        # Ground platforms
        platform_data = [
            (0, 550, 300, 50),
            (400, 550, 400, 50),
    (200, 450, 150, 20),
    (450, 400, 150, 20),
    (100, 350, 100, 20),
    (600, 350, 150, 20),
    (300, 250, 200, 20),
]

for x, y, w, h in platform_data:
    platform = Platform(x, y, w, h)
    self.platforms.add(platform)
    self.all_sprites.add(platform)

    # Add grass on top of ground platforms
    if y >= 500:
        grass = GrassPlatform(x, y - 10, w)
        self.all_sprites.add(grass)

# Create coins
coin_positions = [
    (230, 420), (480, 370), (130, 320),
    (630, 320), (400, 220), (500, 500)
]

for x, y in coin_positions:
    coin = Coin(x, y)
    self.coins.add(coin)
    self.all_sprites.add(coin)

# Create mushrooms
mushroom_positions = [(350, 220), (150, 500)]

for x, y in mushroom_positions:
    mushroom = Mushroom(x, y)
    self.mushrooms.add(mushroom)
    self.all_sprites.add(mushroom)
    # Create enemies
    enemy_positions = [(500, 520), (250, 420)]

    for x, y in enemy_positions:
        enemy = Enemy(x, y)
        self.enemies.add(enemy)
        self.all_sprites.add(enemy)

    # Add level goal flag
    self.goal = Goal(720, 480)
    self.all_sprites.add(self.goal)

def create_level_2(self):
    """Level 2 - Medium difficulty with more jumps"""
    # Platforms - more challenging layout
    platform_data = [
        (0, 550, 200, 50),
        (600, 550, 200, 50),
        (100, 480, 100, 20),
        (250, 420, 100, 20),
        (400, 360, 100, 20),
        (550, 300, 100, 20),
        (700, 360, 80, 20),
        (200, 300, 120, 20),
        (450, 240, 150, 20),
        (100, 180, 100, 20),
        (650, 200, 100, 20),
    ]

    for x, y, w, h in platform_data:
        platform = Platform(x, y, w, h)
        self.platforms.add(platform)
        self.all_sprites.add(platform)
    if y >= 500:
        grass = GrassPlatform(x, y - 10, w)
        self.all_sprites.add(grass)

# More coins
coin_positions = [
    (130, 450), (280, 390), (430, 330), (580, 270),
    (730, 330), (230, 270), (480, 210), (130, 150),
    (680, 170), (50, 520)
]

for x, y in coin_positions:
    coin = Coin(x, y)
    self.coins.add(coin)
    self.all_sprites.add(coin)

# Mushrooms
mushroom_positions = [(500, 210), (700, 520)]

for x, y in mushroom_positions:
    mushroom = Mushroom(x, y)
    self.mushrooms.add(mushroom)
    self.all_sprites.add(mushroom)

# More enemies
enemy_positions = [(150, 450), (300, 390), (600, 270), (100, 520)]

for x, y in enemy_positions:
    enemy = Enemy(x, y)
    self.enemies.add(enemy)
    self.all_sprites.add(enemy)

# Goal
self.goal = Goal(720, 330)
self.all_sprites.add(self.goal)
```

```
def create_level_3(self):
    """Level 3 - Hard difficulty with precise jumps"""
    # Challenging platform layout
    platform_data = [
        (0, 550, 150, 50),
        (650, 550, 150, 50),
        (80, 490, 80, 20),
        (200, 440, 70, 20),
        (320, 390, 70, 20),
        (440, 340, 70, 20),
        (560, 290, 70, 20),
        (680, 340, 80, 20),
        (150, 340, 100, 20),
        (300, 240, 80, 20),
        (450, 190, 100, 20),
        (600, 240, 90, 20),
        (200, 150, 120, 20),
        (500, 120, 100, 20),
    ]

    for x, y, w, h in platform_data:
        platform = Platform(x, y, w, h)
        self.platforms.add(platform)
        self.all_sprites.add(platform)

        if y >= 500:
            grass = GrassPlatform(x, y - 10, w)
            self.all_sprites.add(grass)

    # Many coins
    coin_positions = [
        (110, 460), (230, 410), (350, 360), (470, 310), (590, 260),
        (710, 310), (180, 310), (330, 210), (480, 160), (630, 210),
        (230, 120), (530, 90), (70, 520), (680, 520)
        ]

        for x, y in coin_positions:
            coin = Coin(x, y)
            self.coins.add(coin)
            self.all_sprites.add(coin)

        # Mushrooms
        mushroom_positions = [(330, 210), (260, 120)]

        for x, y in mushroom_positions:
            mushroom = Mushroom(x, y)
            self.mushrooms.add(mushroom)
            self.all_sprites.add(mushroom)

        # Many enemies
        enemy_positions = [
            (100, 460), (240, 410), (350, 360), (590, 260),
            (190, 310), (610, 210), (70, 520), (680, 520)
        ]

        for x, y in enemy_positions:
            enemy = Enemy(x, y)
            self.enemies.add(enemy)
            self.all_sprites.add(enemy)

        # Goal
        self.goal = Goal(730, 510)
        self.all_sprites.add(self.goal)

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            if event.type == pygame.KEYDOWN:
```

```
            if event.key == pygame.K_SPACE or event.key == pygame.K_UP:
                self.player.jump()

    def update(self):
        # Update all sprites
        self.all_sprites.update()

        # Check platform collisions
        self.player.check_platform_collision(self.platforms)

        # Check coin collection
        coins_collected = pygame.sprite.spritecollide(self.player, self.coins, True)
        self.score += len(coins_collected) * 10

        # Check mushroom collection
        mushrooms_collected = pygame.sprite.spritecollide(self.player,
self.mushrooms, True)
        if mushrooms_collected:
            self.powered_up = True
            self.score += 50
            # You could make the player bigger here if desired

        # Check enemy collision
        enemies_hit = pygame.sprite.spritecollide(self.player, self.enemies, False)
        if enemies_hit:
            self.lives -= 1
            self.player.rect.x = 100
            self.player.rect.y = SCREEN_HEIGHT - 150
            self.player.velocity_y = 0

            if self.lives <= 0:
                self.game_over()

        # Check goal/flag collision
        if hasattr(self, 'goal') and self.player.rect.colliderect(self.goal.rect):
            self.level_complete()

    def draw(self):
        # Sky blue background like Mario
        self.screen.fill((92, 148, 252))

        # Draw clouds
        for cloud in self.clouds:
            self.draw_cloud(cloud['x'], cloud['y'])
            cloud['x'] += cloud['speed']
            if cloud['x'] > SCREEN_WIDTH:
                cloud['x'] = -100

        # Draw all sprites
        self.all_sprites.draw(self.screen)

        # Draw HUD
        score_text = self.font.render(f"Score: {self.score}", True, WHITE)
        lives_text = self.font.render(f"Lives: {self.lives}", True, WHITE)
        level_text = self.font.render(f"Level: {self.current_level}", True, WHITE)

        # Add shadow to text for better visibility
        shadow_offset = 2
        score_shadow = self.font.render(f"Score: {self.score}", True, BLACK)
        lives_shadow = self.font.render(f"Lives: {self.lives}", True, BLACK)
        level_shadow = self.font.render(f"Level: {self.current_level}", True,
BLACK)

        self.screen.blit(score_shadow, (12, 12))
        self.screen.blit(score_text, (10, 10))
        self.screen.blit(lives_shadow, (12, 52))
        self.screen.blit(lives_text, (10, 50))
        self.screen.blit(level_shadow, (SCREEN_WIDTH - 142, 12))
        self.screen.blit(level_text, (SCREEN_WIDTH - 140, 10))
        # Show power-up status
        if self.powered_up:
            power_text = self.font.render("POWERED UP!", True, YELLOW)
            power_shadow = self.font.render("POWERED UP!", True, RED)
            self.screen.blit(power_shadow, (12, 92))
            self.screen.blit(power_text, (10, 90))

        pygame.display.flip()

    def draw_cloud(self, x, y):
        """Draw a Mario-style cloud"""
        cloud_color = (255, 255, 255)
        # Main cloud body
        pygame.draw.ellipse(self.screen, cloud_color, (x, y, 60, 30))
        pygame.draw.ellipse(self.screen, cloud_color, (x + 20, y - 10, 40, 30))
        pygame.draw.ellipse(self.screen, cloud_color, (x + 40, y, 50, 25))

    def level_complete(self):
        """Handle level completion"""
        self.screen.fill((92, 148, 252))

        if self.current_level >= self.max_level:
            # Game completed!
            title_text = self.font.render("CONGRATULATIONS!", True, YELLOW)
            message_text = self.font.render("You Beat All Levels!", True, WHITE)
        else:
            title_text = self.font.render(f"LEVEL {self.current_level} COMPLETE!",
True, YELLOW)
            message_text = self.font.render(f"Next Level: {self.current_level + 1}",
True, WHITE)

        score_text = self.font.render(f"Score: {self.score}", True, WHITE)
        continue_text = self.font.render("Press ENTER to Continue", True, WHITE)

        # Shadows
        title_shadow = self.font.render("CONGRATULATIONS!" if
self.current_level >= self.max_level else f"LEVEL {self.current_level}
COMPLETE!", True, BLACK)

        self.screen.blit(title_shadow, (SCREEN_WIDTH // 2 - 182,
SCREEN_HEIGHT // 2 - 102))
        self.screen.blit(title_text, (SCREEN_WIDTH // 2 - 180, SCREEN_HEIGHT
// 2 - 100))
        self.screen.blit(message_text, (SCREEN_WIDTH // 2 - 120,
SCREEN_HEIGHT // 2 - 40))
        self.screen.blit(score_text, (SCREEN_WIDTH // 2 - 80, SCREEN_HEIGHT
// 2 + 20))
        self.screen.blit(continue_text, (SCREEN_WIDTH // 2 - 150,
SCREEN_HEIGHT // 2 + 80))

        pygame.display.flip()

        waiting = True
        while waiting:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                    waiting = False
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN:
                        if self.current_level >= self.max_level:
                            # Reset to level 1
                            self.current_level = 1
                            self.lives = 3
                            self.score = 0
                        else:
                            # Next level
                            self.current_level += 1

                        # Reset player position
                        self.player.rect.x = 100
                        self.player.rect.y = SCREEN_HEIGHT - 150
                        self.player.velocity_y = 0
                        self.powered_up = False

                        # Load new level
                        self.create_level()
                        waiting = False
                    if event.key == pygame.K_q:
                        self.running = False
                        waiting = False

        def game_over(self):
        self.screen.fill(BLACK)
        game_over_text = self.font.render("GAME OVER!", True, RED)
        score_text = self.font.render(f"Final Score: {self.score}", True, WHITE)
        level_text = self.font.render(f"Reached Level: {self.current_level}", True,
WHITE)
        restart_text = self.font.render("Press R to Restart or Q to Quit", True,
WHITE)

        self.screen.blit(game_over_text, (SCREEN_WIDTH // 2 - 100,
SCREEN_HEIGHT // 2 - 80))
        self.screen.blit(score_text, (SCREEN_WIDTH // 2 - 120, SCREEN_HEIGHT
// 2 - 20))
        self.screen.blit(level_text, (SCREEN_WIDTH // 2 - 120, SCREEN_HEIGHT
// 2 + 20))
        self.screen.blit(restart_text, (SCREEN_WIDTH // 2 - 200,
SCREEN_HEIGHT // 2 + 80))

        pygame.display.flip()

        waiting = True
        while waiting:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.running = False
                    waiting = False
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_r:
                        # Restart from level 1
                        self.current_level = 1
                        self.score = 0
                        self.lives = 3
                        self.powered_up = False
                        self.player.rect.x = 100
                        self.player.rect.y = SCREEN_HEIGHT - 150
                        self.player.velocity_y = 0
                        self.create_level()
                        waiting = False
                    if event.key == pygame.K_q:
                        self.running = False
                        waiting = False

    def run(self):
        while self.running:
            self.handle_events()
            self.update()
            self.draw()
            self.clock.tick(FPS)

        pygame.quit()
        sys.exit()

# Run the game
if __name__ == "__main__":
    game = Game()
    game.run()
```