Introduction to Pygame

Introduction to Pygame

Pygame is a Python library designed for creating games and multimedia projects. It equips developers with tools for graphics, sound, and handling user input, making game development more approachable. With Pygame, anyone can start turning their game ideas into reality, regardless of their programming skill level.

- **Easy to Learn** - Ideal for beginners, offering simple syntax and intuitive game development processes.
- **Versatile** - Supports the creation of a broad range of games and applications.
- **Strong Community** - Access to a wealth of tutorials, forums, and resources.
- **Cross-Platform** - Enables games to run on various operating systems like Windows, Mac, and Linux.

you can download template here

Project Structure: Why It Matters

```
project/
│
├── assets/
│   ├── images/
│   └── sounds/
├── main.py
├── settings.py
├── models/
│   ├── player.py
│   ├── enemy.py
│   ├── food.py
│   ├── obstacle.py
│   └── ...
├── misc/
│   └── utils.py
├── README.md
├── requirements.txt
└── venv/
```

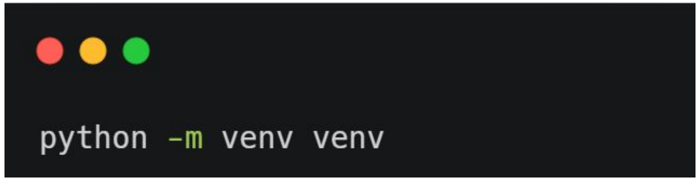
A well-planned project structure ensures code clarity and simplifies maintenance, setting a strong foundation for both immediate understanding and future growth. It facilitates the seamless integration of new features and updates, crucial for evolving projects.

Basic Project Structure Overview

- **assets** - contains game resources like images and sounds
- **main.py** - acts as the entry point for the application
- **settings.py** - contains all sorts of game settings
- **models** - contains definitions for various game entities
- **misc** - contains miscellaneous utilities and helper functions
- **README.md** - provides a project overview
- **requirements.txt** - lists necessary dependencies for the project

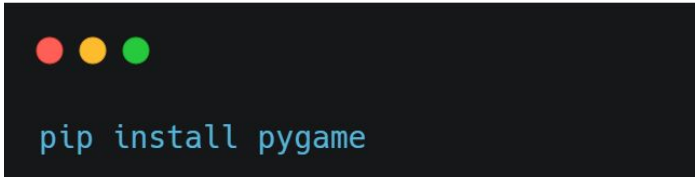
Getting Started with Pygame

- Set up the Python virtual environment in your project folder and activate it.



```
python -m venv venv
```

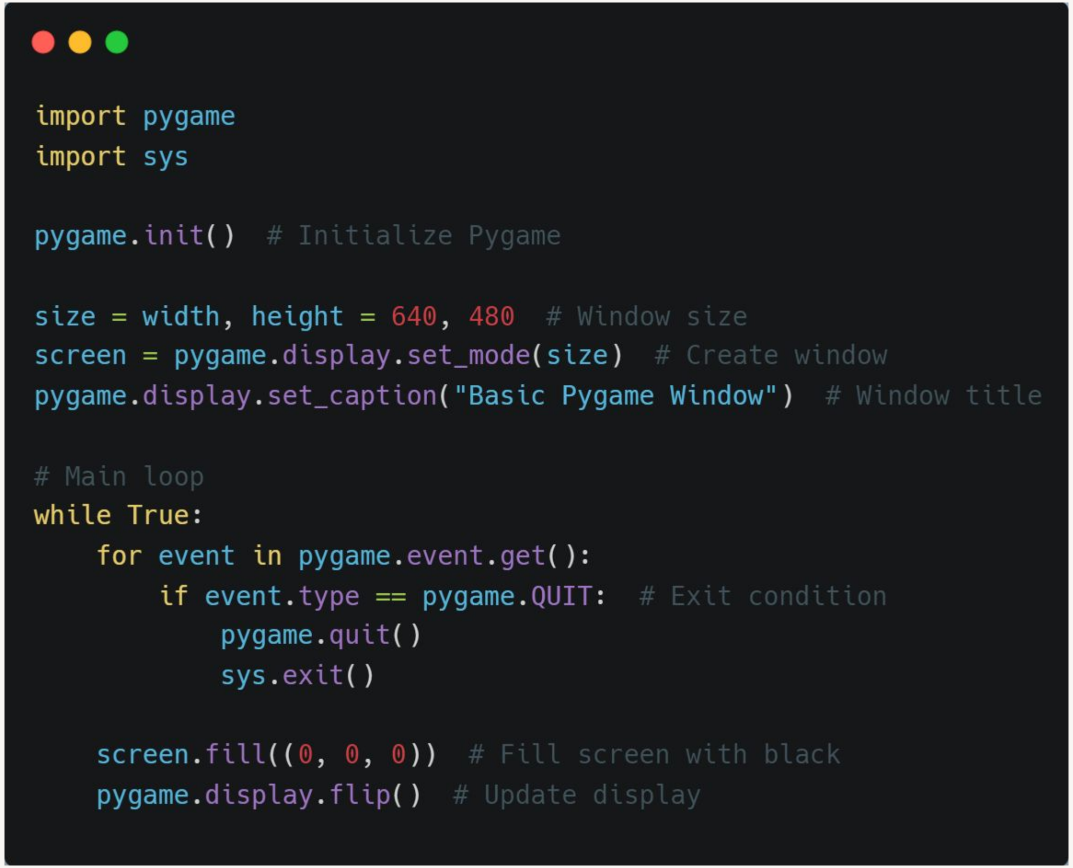
- Install Pygame using pip command in the terminal.



```
pip install pygame
```

- Write a simple Pygame script to create a window and display it.
- Explore basic Pygame functionalities like handling events and updating the screen.

Basic pygame code example

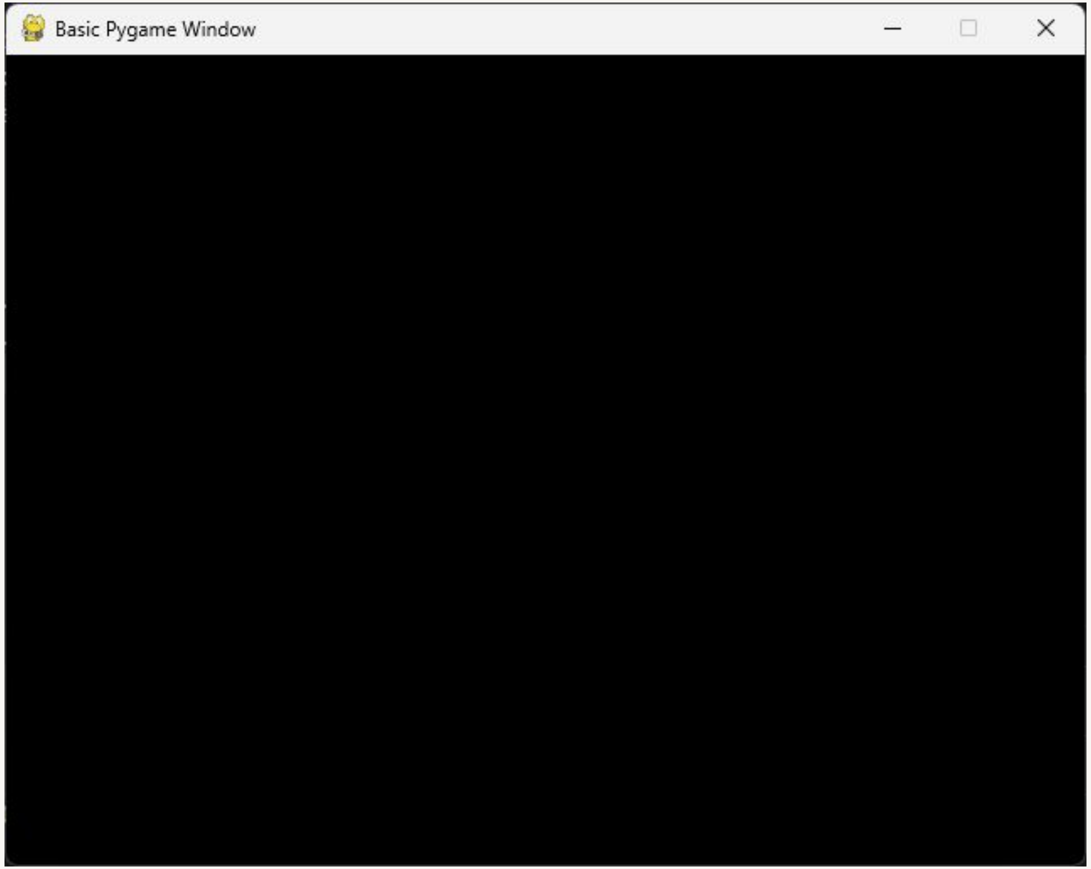


```python
import pygame
import sys

pygame.init()  # Initialize Pygame

size = width, height = 640, 480  # Window size
screen = pygame.display.set_mode(size)  # Create window
pygame.display.set_caption("Basic Pygame Window")  # Window title

# Main loop
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:  # Exit condition
            pygame.quit()
            sys.exit()

    screen.fill((0, 0, 0))  # Fill screen with black
    pygame.display.flip()  # Update display
```

This code initializes Pygame, creates a window with a title, and enters a loop to keep the window open and responsive. It fills the window with black and updates the display. The loop ends, and Pygame closes when the user exits the window, demonstrating a basic Pygame application setup.

The screen size in the code is set to 640 pixels wide by 480 pixels high. This size determines the dimensions of the window that appears when the program runs, providing a fixed canvas for displaying graphics and game elements.

Basic pygame code example



Working with Images

Loading and Displaying Images

Learn how to load images into Pygame and display them on the screen to create visual elements for your game.

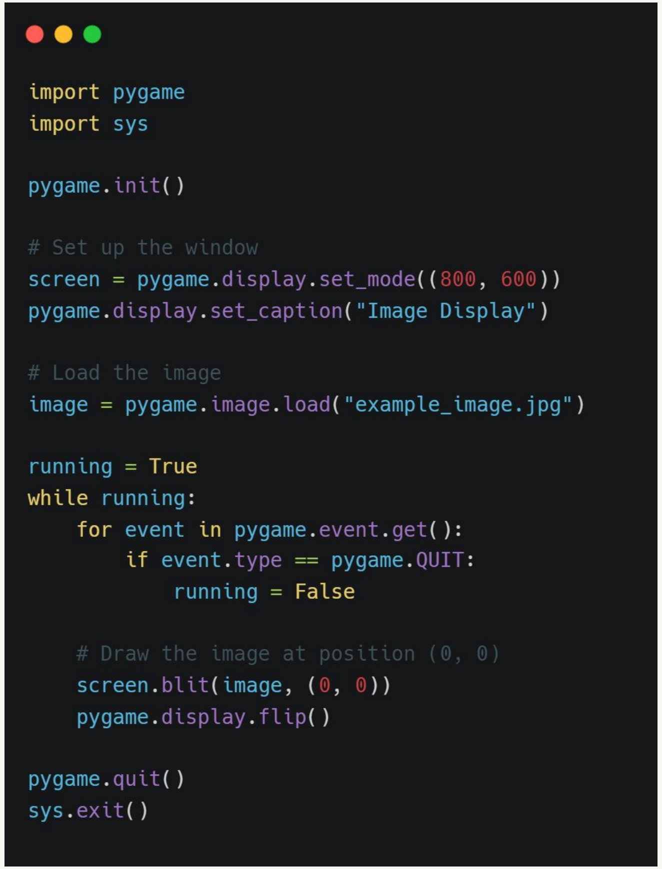
Manipulating Image Sizes

Adjust the sizes of images in Pygame to fit your game's design and layout, enhancing the visual appeal of your game.

Positioning Images on the Screen

Control the position of images within the game window to create dynamic visual effects and interactions for a more engaging gameplay experience.

Basic pygame image insert example



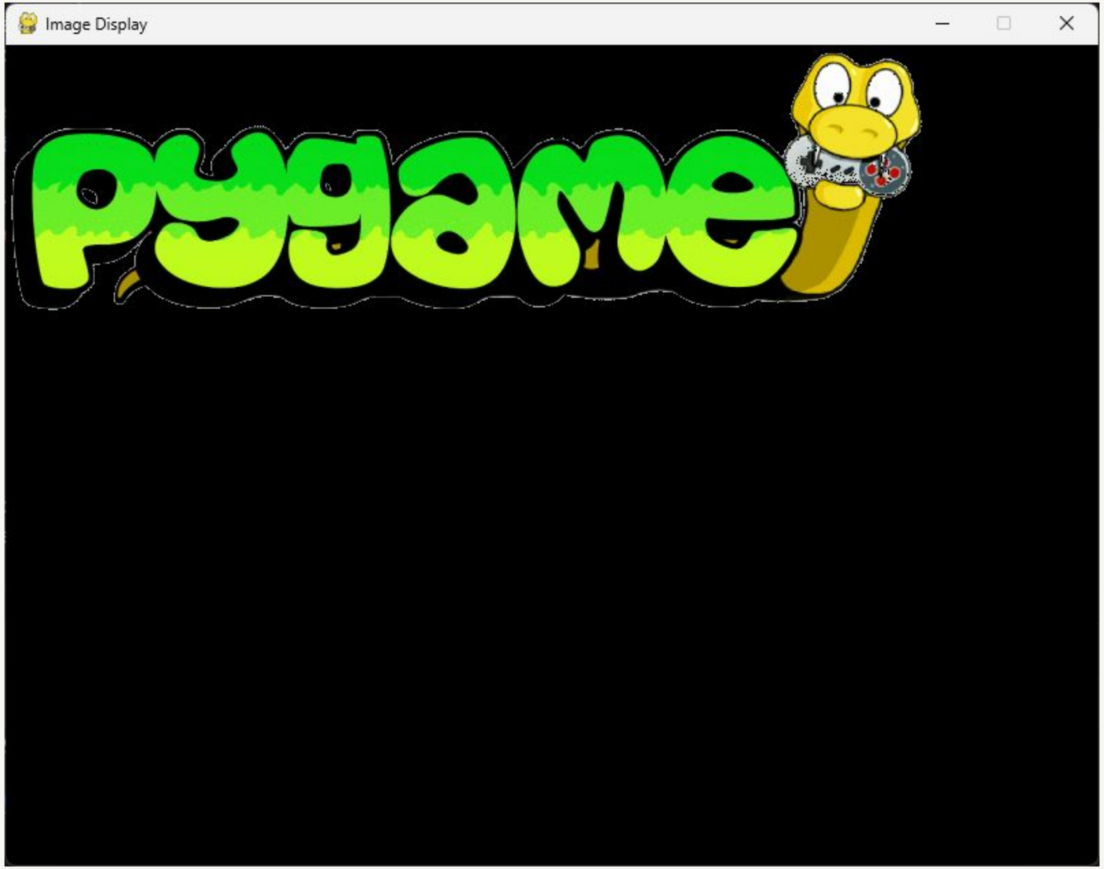
```python
import pygame
import sys

pygame.init()

# Set up the window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Image Display")

# Load the image
image = pygame.image.load("example_image.jpg")

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Draw the image at position (0, 0)
    screen.blit(image, (0, 0))
    pygame.display.flip()

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

In Pygame, images can be utilized for various purposes such as creating sprites, setting the background of the game, implementing animation, and more. Here's how images can be used:

- **Creating Sprites**: Images can be loaded and used to create sprites - moving objects in the game, such as characters, enemies, or items.
- **Backgrounds**: Images serve as backgrounds for game levels, menus, and interfaces.
- **Animation**: Sequences of images can be loaded and displayed sequentially, creating animation effects for objects or characters in the game.
- **Interactive Elements**: Images can be loaded and used to create interactive buttons, menus, and other user interface elements.
- **State Indication**: Different images can be used to represent various states of objects in the game, such as "alive" and "dead" for characters or "open" and "closed" for doors or chests.

Basic pygame image insert example

Source code



Music and Sound Effects

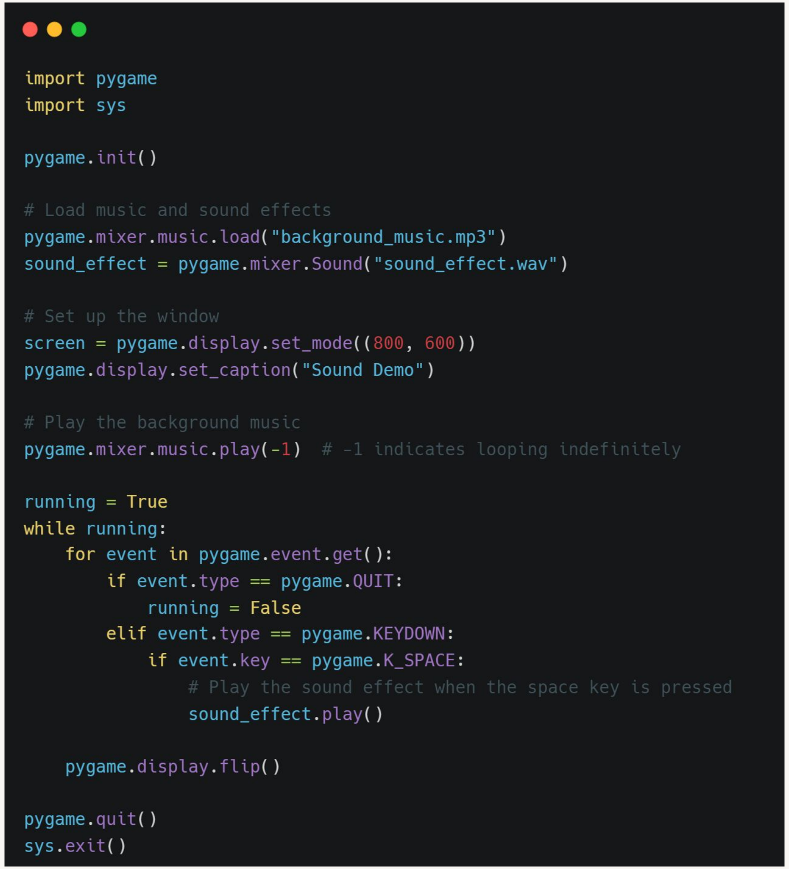
Incorporating Audio

- Add background music and sound effects to enhance game experience.
- Use Pygame functions to load and play audio files.
- Control volume levels and manage playback within Pygame.

Volume Control

- Adjust volume settings dynamically during gameplay.
- Implement sound effects at specific game events for feedback.
- Ensure a balanced audio mix for a more immersive gaming experience.

Basic pygame Music and Sound Effects example



```python
import pygame
import sys

pygame.init()

# Load music and sound effects
pygame.mixer.music.load("background_music.mp3")
sound_effect = pygame.mixer.Sound("sound_effect.wav")

# Set up the window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Sound Demo")

# Play the background music
pygame.mixer.music.play(-1)  # -1 indicates looping indefinitely

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                # Play the sound effect when the space key is pressed
                sound_effect.play()

    pygame.display.flip()

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

In Pygame, music and sound effects serve several important functions:

- Atmospheric Enhancement: Music sets the mood and ambiance of the game, while sound effects add realism and interactivity to the gameplay.
- Event Signaling: They can signal important in-game events, such as the start of a boss battle or the discovery of a hidden item, providing cues to the player.
- Feedback Mechanism: Music and sound effects provide auditory feedback, helping players understand the consequences of their actions and making the game more responsive.

Source code

Geometric Drawing

Creating visual elements such as game characters, obstacles, and backgrounds using geometric shapes in Pygame.

Using rectangles, circles, and lines to design interactive game environments and enhance player experiences.

Customizing shapes with colors, sizes, and positions to add visual appeal and functionality to games.

Basic pygame Geometric Drawing example

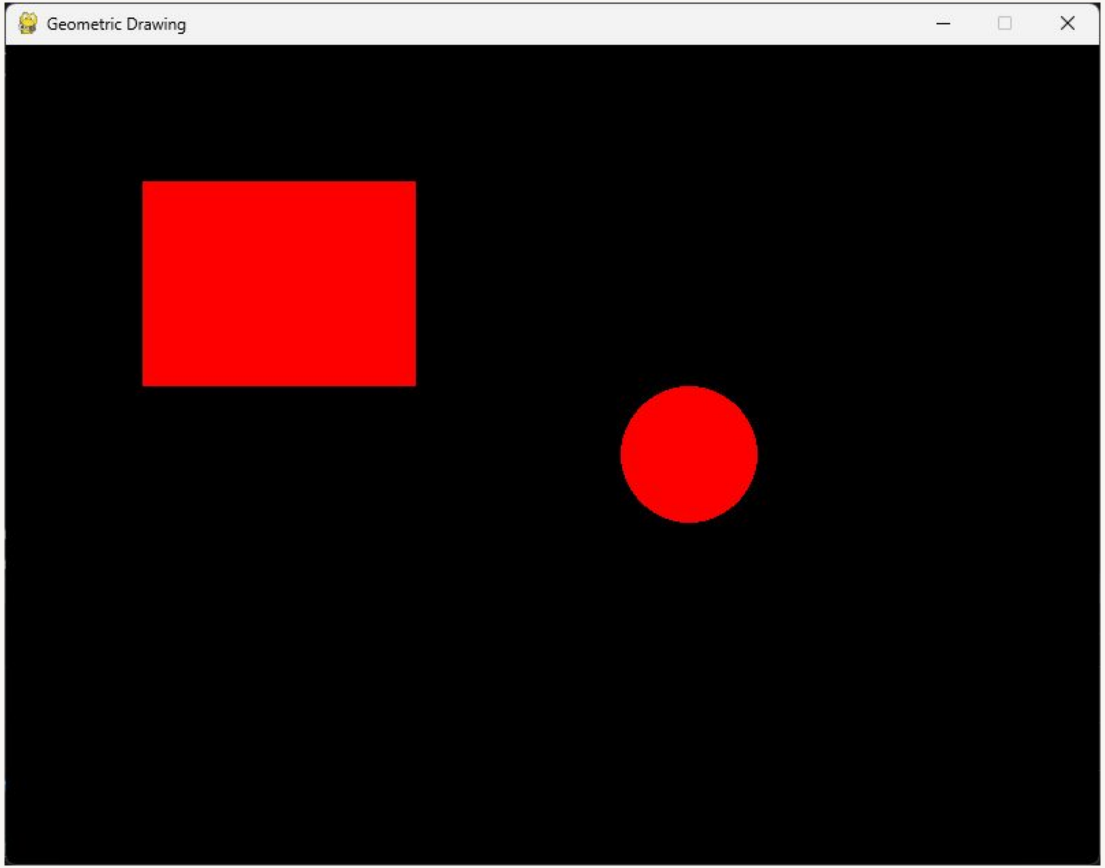
```python
import pygame
import sys

pygame.init()

size = width, height = 800, 600  # Set window size
screen = pygame.display.set_mode(size)  # Create window
pygame.display.set_caption("Geometric Drawing")  # Set window title

black = 0, 0, 0  # Define black color
red = 255, 0, 0  # Define red color

running = True
while running:  # Main loop
    for event in pygame.event.get():  # Event handling
        if event.type == pygame.QUIT:  # Check for quit event
            running = False  # Exit loop if quit event is detected

    screen.fill(black)  # Fill window with black background

    # Draw a red rectangle at position (100, 100) with width 200 and
height 150
    pygame.draw.rect(screen, red, pygame.Rect(100, 100, 200, 150))

    # Draw a red circle with center at (500, 300) and radius 50
    pygame.draw.circle(screen, red, (500, 300), 50)

    pygame.display.flip()  # Update display

pygame.quit()  # Quit Pygame
sys.exit()  # Exit script
```

Geometric drawing in Pygame offers a versatile way to create visual elements for games and applications:

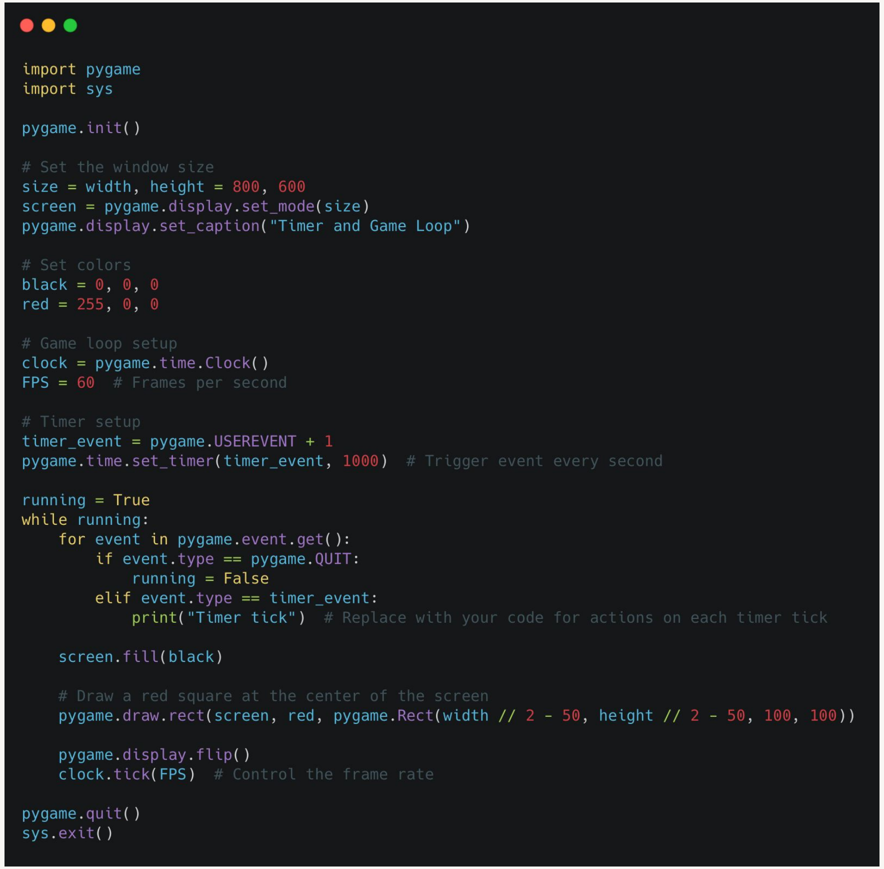
- **Basic Shapes**: Rectangles, circles, lines, and polygons serve as building blocks for visual design.
- **Game Elements**: Shapes represent game elements like platforms, obstacles, or boundaries, defining the game environment.
- **Visual Feedback**: They provide clear visual cues for in-game events and interactions.
- **Customization**: Shapes can be customized in size, color, and transparency for creative design choices.
- **Dynamic Rendering**: Adaptation to game logic or user input ensures responsive and engaging visuals.

Basic pygame Geometric Drawing example

Source code



PyGame Timer and Game Loop example



```python
import pygame
import sys

pygame.init()

# Set the window size
size = width, height = 800, 600
screen = pygame.display.set_mode(size)
pygame.display.set_caption("Timer and Game Loop")

# Set colors
black = 0, 0, 0
red = 255, 0, 0

# Game loop setup
clock = pygame.time.Clock()
FPS = 60  # Frames per second

# Timer setup
timer_event = pygame.USEREVENT + 1
pygame.time.set_timer(timer_event, 1000)  # Trigger event every second

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == timer_event:
            print("Timer tick")  # Replace with your code for actions on each timer tick

    screen.fill(black)

    # Draw a red square at the center of the screen
    pygame.draw.rect(screen, red, pygame.Rect(width // 2 - 50, height // 2 - 50, 100, 100))

    pygame.display.flip()
    clock.tick(FPS)  # Control the frame rate

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

Timer and game loop management in Pygame are essential for controlling game behavior and timing:

- **Timing Control**: The game loop regulates frame rates for smooth gameplay and performance.
- **Timer Events**: Timers schedule actions like updates, animations, and game events.
- **Game State**: The loop updates player input, object interactions, and game progression.
- **Animation**: Timer drive visual effects, enhancing the player experience.
- **Frame Rate**: Control ensures consistent performance across platforms.

Source code

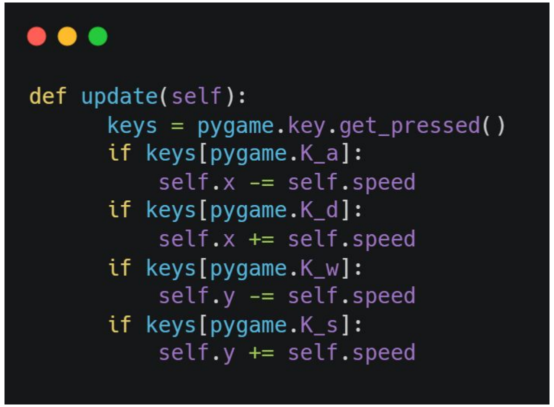
Event Handling

User Input Management

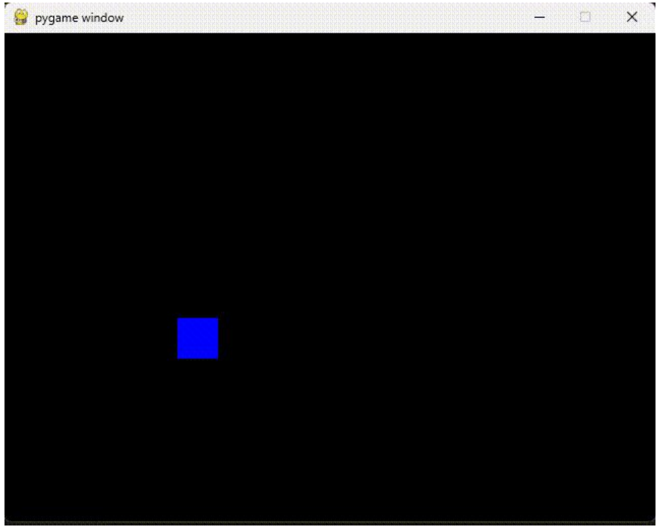
Handling keyboard and mouse input to control game actions and interactions.

Interactive Game Experiences

Creating engaging gameplay by integrating user input with event handling in Pygame.



```python
def update(self):
        keys = pygame.key.get_pressed()
        if keys[pygame.K_a]:
            self.x -= self.speed
        if keys[pygame.K_d]:
            self.x += self.speed
        if keys[pygame.K_w]:
            self.y -= self.speed
        if keys[pygame.K_s]:
            self.y += self.speed
```

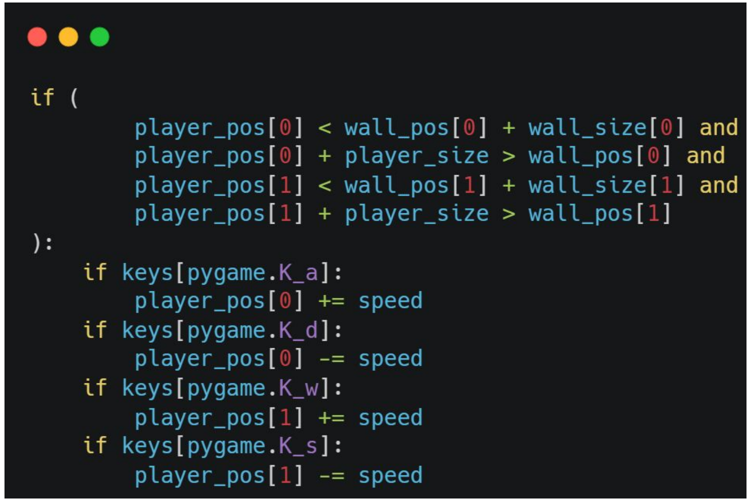


Collision Detection

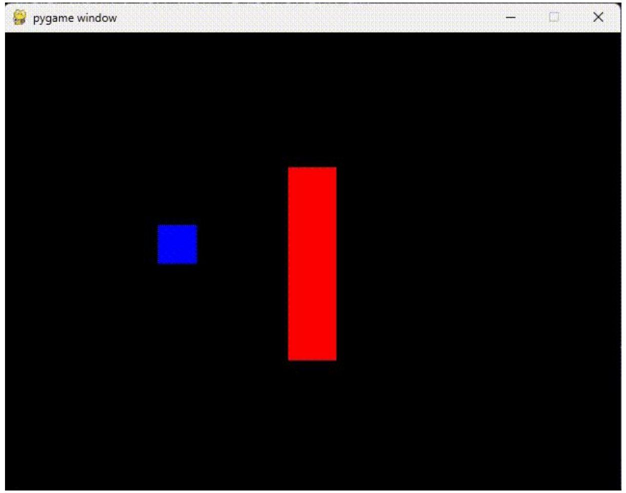
Collision Detection Methods

Implementing bounding boxes to check for collisions between sprites.

Source code



```
if (
        player_pos[0] < wall_pos[0] + wall_size[0] and
        player_pos[0] + player_size > wall_pos[0] and
        player_pos[1] < wall_pos[1] + wall_size[1] and
        player_pos[1] + player_size > wall_pos[1]
):
    if keys[pygame.K_a]:
        player_pos[0] += speed
    if keys[pygame.K_d]:
        player_pos[0] -= speed
    if keys[pygame.K_w]:
        player_pos[1] += speed
    if keys[pygame.K_s]:
        player_pos[1] -= speed
```



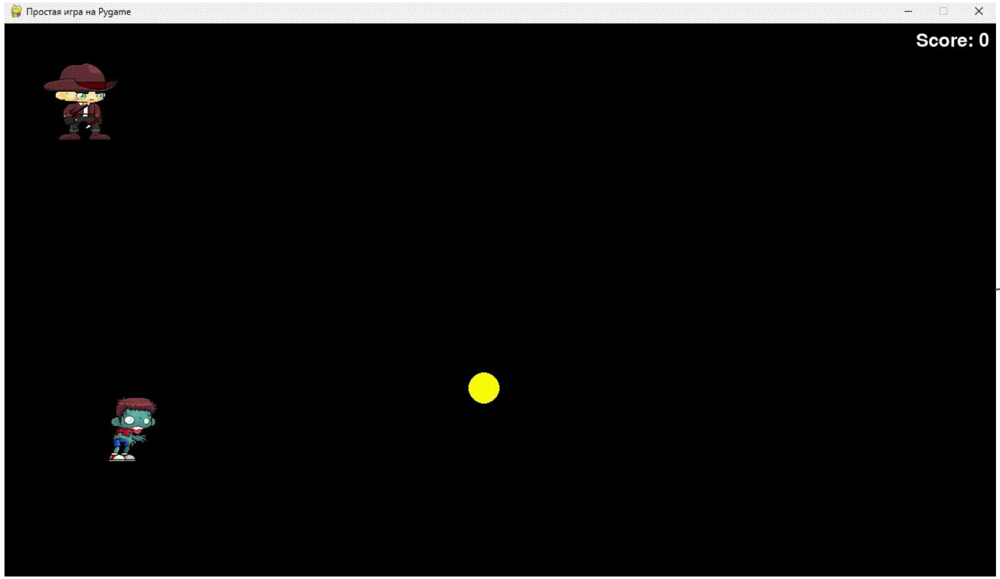
Resources

- Russian-language guide to the basics of pygame
- official pygame documentation
- guide link from labs

Source code

Bonus example

here you can see a basic example of a game using architecture, classes, and using images


Простая игра на Pygame
Score: 0