Pygame Development

Developing game with Object-Oriented Design
Object-Oriented Programming 2/2567

@wichit2s
https://wichit2s.github.io/courses/oop/

Hourly Agenda

Hour 1: Introduction to Pygame

Hour 2: Drawing and Interactivity

Hour 3: Adding Animation and Sounds

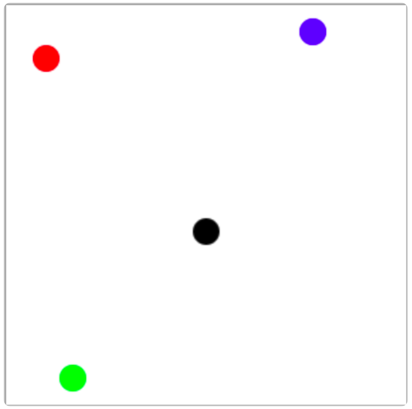
Hour 4: Building a Complete Game

Developing game with Object-Oriented Design
Object-Oriented Programming 2/2567

@wichit2s
https://wichit2s.github.io/courses/oop/

Hour 1

Introduction to Pygame



Developing game with Object-Oriented Design
Object-Oriented Programming 2/2567

@wichit2s
https://wichit2s.github.io/courses/oop/

Pygame

1st Hour Objectives:

- **Understand Pygame Fundamentals**:
  - Learn the core concepts of 2D game development.
  - Explore how to set up and use Pygame effectively.
- **Learn Design Patterns in Game Development**:
  - Discover essential patterns like the Game Loop and State Management.
- **Build Interactive Games**:
  - Gain hands-on experience creating simple games.
  - Animate objects, handle user input, and add sound.
- **Develop Problem-Solving Skills**:
  - Apply coding concepts to solve real-world game design challenges.

What is Pygame?

Definition: A Python library for 2D game development.

Features:

- Easy to use.
- Provides tools for graphics, sound, and input.

Common Use Cases:

- Prototyping games.
- Learning game development.

Why Learn Pygame?

- **Accessible:** Suitable for beginners with basic Python knowledge.
- **Cross-Platform:** Runs on Windows, Mac, and Linux.
- **Skill-Building:** Teaches core game development concepts.
- **Fun Factor:** Turn your ideas into playable games!

Installation and Setup

Create	Create environment • python3 -m venv venv
Activate	Activate environment • ./venv/scripts/activate
Install	Install Pygame: • pip install pygame
Check	Check Installation: • python -m pygame --version
Set Up	Set Up Your IDE: Use VS Code, PyCharm, or any Python editor.

Game Development Basics

Game Loop Overview:

- **Initialize:** Set up game variables and assets.
- **Update:** Process input, update objects.
- **Draw:** Render everything to the screen.

Core Pygame Modules:

- pygame.display for the window.
- pygame.event for input handling.

Design Pattern: The Game Loop

Definition: A loop that controls the flow of the game.

Steps:

- Process events (e.g., user input).
- Update game objects.
- Render updates to the screen.

```
running = True
# loop
while running:
  # 1. check events
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      running = False
  # 2. update game objects state/data
  # 3. draw on screen
  screen.fill((0, 0, 0))
  # 4. show screen on display
  pygame.display.flip()
```

Hands-On: Create Your First Window

- **Goal:** Create a Pygame window with a background color.
- With fix FPS

```
import pygame

pygame.init()
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Pygame Mygame")
clock = pygame.time.Clock()
running = True
while running:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      running = False
  screen.fill((0, 128, 255))
  pygame.display.flip()
  clock.tick(30)
pygame.quit()
```

Key Takeaways

- **Game Loop is Fundamental:**

  Initialize, Update, Draw.

- **Pygame is Beginner-Friendly:**

  Quick to learn and experiment.

- **Hands-On Practice:** Start

  building simple games now.

Hour 2

Drawing and Interactivity in Pygame

Composite Design Pattern

- Drawing shapes and images
- Handling user input (keyboard and mouse)
- Creating a simple interactive scene

@wichit2s
https://wichit2s.github.io/courses/oop/

Composite Design Pattern

Separates game objects into reusable components:

- **Visual Component** (e.g., sprite, shape)
- **Behavior Component** (e.g., movement, collision logic)

Advantages:

- Easy to maintain and expand.
- Encourages modular development.

Example:

- A "Player" object may have components like Image, Position, and Movement.

Drawing Shapes

- Use Pygame's built-in methods:

- **Rectangles:**

```
pygame.draw.rect(screen, color, rect)
pygame.draw.rect(screen, (255, 0, 0), (50, 50, 100, 50))
```

- **Circles:**

```
pygame.draw.circle(screen, color, center, radius)
pygame.draw.circle(screen, (0, 255, 0), (200, 150), 40)
```

- **Lines:**

```
pygame.draw.line(screen, color, start_pos, end_pos, width)
pygame.draw.line(screen, (0, 0, 255), (300, 200), (400, 300), 5)
```

Displaying Images

- **Steps to display images**:
  - Load an image:

```
image = pygame.image.load("path/to/image.png")
```

  - Draw the image on the screen:

```
screen.blit(image, (x, y))
```

  - Create new image:

```
rect = pygame.Rect((40,60))
image = pygame.Surface(rect.size).convert()
```

- **Important Notes:**
  - Images must be in the same directory or provide the correct path.
  - Use .convert() or .convert_alpha() for performance.

```
image = pygame.image.load("player.png")
screen.blit(image, (100, 100))
```

Handling User Input

Use Pygame's event system to handle input:

Keyboard Events:

Detect key presses:

pygame.KEYDOWN and pygame.KEYUP

Example: Move left/right with arrow keys.

Mouse Events:

Detect clicks:

pygame.MOUSEBUTTONDOWN

pygame.MOUSEBUTTONUP

```
from pygame.locals import (
   K_UP, K_DOWN, K_LEFT, K_RIGHT, QUIT, KEYDOWN
)

for event in pygame.event.get():
  if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_LEFT:
      print("Left arrow pressed!")
    elif event.type == pygame.MOUSEBUTTONDOWN:
      print("Mouse clicked at", event.pos)
```

Custom User Event

- Define event

```
SPAWN_EVENT = pygame.USEREVENT+1
```

- Set event timer

```
pygame.time.set_timer(SPAWN_EVENT, 1000)
```

- Detect event

```
if event.type == SPAWN_EVENT
```

Creating Interactivity

- **Goal:** Move a shape using arrow keys.
- **Steps:**
  - Draw the shape.
  - Update its position based on input.
  - Refresh the screen.

```
x, y = 100, 100
running = True
while running:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      running = False
  keys = pygame.key.get_pressed()
  if keys[pygame.K_LEFT]:
    x -= 5
  if keys[pygame.K_RIGHT]:
    x += 5
  screen.fill((0, 0, 0))
  pygame.draw.rect(screen, (255, 0, 0), (x, y, 50, 50))
  pygame.display.flip()
```

Hands-on Activity

- Create a Simple Interactive Scene
- **Task:**
  - Draw a circle that moves with arrow keys.
  - Change the circle's color when the spacebar is pressed.
- **Guidance:**
  - Use pygame.KEYDOWN to detect spacebar press.
  - Update the screen each frame to reflect changes.

```
color = (randint(0, 255), randint(0, 255), randint(0, 255))
```

Key Takeaways

- **Design Pattern:** Composite Design Pattern
  Component-Based Design promotes modularity.
- **Core Concepts:**
  - Drawing shapes and images.
  - Handling user input.
  - Combining these concepts for interactivity.
- **Next Hour:** Animation and Sound.

@wichit2s
https://wichit2s.github.io/courses/oop/

Hour 3

Animation and Sound

State Management Design Pattern

Topics:

- •Animating objects
- •Adding sound effects and music
- •Building a mini-game

@wichit2s
https://wichit2s.github.io/courses/oop/

State Management

Managing different states of a game (e.g., menus, gameplay, pause, game over).

- **Why it's useful:**
  - Keeps game logic organized.
  - Allows transitions between states.
- **Example:**
  - A "Game Over" screen that appears when the player loses, while gameplay is paused.

```
state = "menu"
while running:
 if state == "menu":
  show_menu()
 elif state == "gameplay":
  play_game()
 elif state == "game_over":
  show_game_over()
```

Animating Objects

- **Concept:**
  - Change an object's position, size, or appearance frame by frame.
- **Steps:**
  - Define object properties (e.g., position, velocity).
  - Update properties in each frame.
  - Redraw the object with new properties.

```
x, y = 100, 100
speed_x, speed_y = 3, 2
while running:
  x += speed_x
  y += speed_y
  if x < 0 or x > 800:
    speed_x = -speed_x
  if y < 0 or y > 600:
    speed_y = -speed_y
  screen.fill((0, 0, 0))
  pygame.draw.circle(screen, (255, 0, 0), (x, y), 20)
  pygame.display.flip()
```

Sprites

- **Definition:**
  - Sprites are 2D images or animations integrated into a game.
  - Used to represent characters, objects, or effects.
- **How to Load a Sprite:**
  - Create a Sprite class that inherits from pygame.sprite.Sprite.
  - Load the image in the constructor.

```
class Player(pygame.sprite.Sprite):
 def __init__(self):
  super().__init__()
  self.image = pygame.image.load("player.png").convert_alpha()
  self.rect = self.image.get_rect()

 def update(self):
  self.rect.move_ip(-3, 0)
```

Using Sprite Groups

- **Purpose:**
  - Organize and manage multiple sprites efficiently.
- **Steps:**
  - Create a pygame.sprite.Group to hold all sprites.
  - Use group.draw(screen) to render all sprites.
  - Use group.update() to update their logic.

```
player = Player()
sprites = pygame.sprite.Group() sprites.add(player)
while running:
  sprites.update()
  screen.fill((0, 0, 0))
  sprites.draw(screen)
  pygame.display.flip()
```

Adding Sound Effects and Music

- **Loading Sounds:**
  - Load a sound:

    ```
    sound = pygame.mixer.Sound("path/to/sound.wav")
    ```
  - Play a sound:

    ```
    sound.play()
    ```
- **Playing Music:**
  - Load music:

    ```
    pygame.mixer.music.load("path/to/music.mp3")
    ```
  - Start playback:

    ```
    pygame.mixer.music.play(-1)
    ```

```
pygame.mixer.init()
sound = pygame.mixer.Sound("click.wav")
music = "background.mp3" pygame.mixer.music.load(music)
pygame.mixer.music.play(-1)
```

Combining Animation and Sound

- **Goal:** Add sound effects and animations to a bouncing ball.
- **Steps:**
  - Add bouncing animation (from previous demo).
  - Play a sound when the ball hits the screen edge.

```
bounce_sound = pygame.mixer.Sound("bounce.wav")
if x < 0 or x > 800:
  speed_x = -speed_x
  bounce_sound.play()
if y < 0 or y > 600:
  speed_y = -speed_y
  bounce_sound.play()
```

Hands-On Activity

Build a Mini Game

Task: Create a mini-game where a ball bounces around the screen:

- Add background music.
- Play a "bounce" sound effect when the ball hits the edges.
- Change the ball’s color when it bounces.

Hints:

- Use pygame.mixer for sounds.
- Randomize color using:

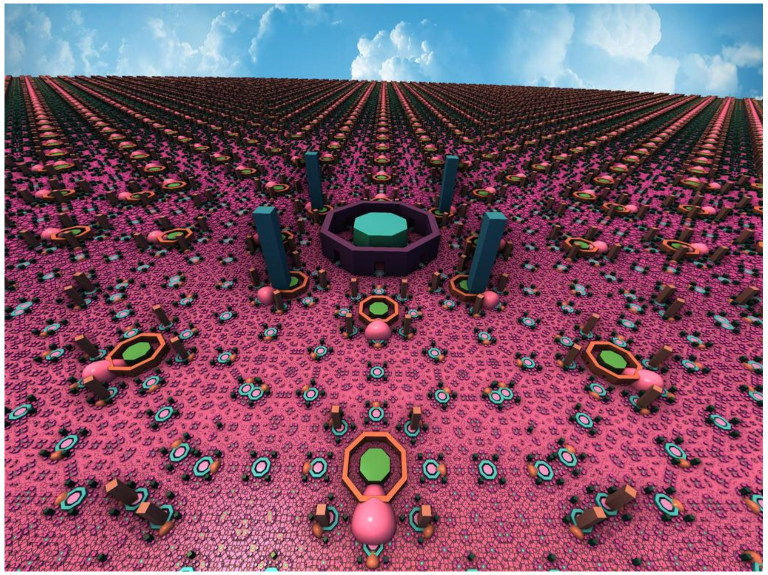
Key Takeaways

- **Design Pattern:** State Management keeps game logic organized.
- **Core Concepts:**
    - Animating objects.
    - Adding sound effects and music.
    - Combining these to build a more immersive experience.
- **Next Hour:** Building a complete game.

@wichit2s
https://wichit2s.github.io/courses/oop/

Hour 4

Building a Complete Game



Game Loop
Composite
State Management

Topics:
•Overview of the game loop
•Structuring a game project
•Building a simple game step-by-step

Game Loop Architecture

- The core structure of any game.**Steps:**
  - Process input
  - Update the game state
  - Render the game world
- **Benefits:**Ensures the game runs smoothly at a consistent frame rate.
- Decouples game logic from rendering.

Structuring Your Game Project

- Suggested folder structure:assets/ (Images, sounds, etc.)
- src/ (Game logic and classes)
- main.py (Entry point)
- Use modular code:Separate files for different parts of the game (e.g., player, enemies).

1. Game Concept and Setup

Example Game: Simple Dodge the Falling Objects

Setup: Screen size: 800x600

- Player (rectangle) at the bottom of the screen
- Falling objects (circles) that the player must avoid

Code: Initialize Pygame and set up the screen.

```
import pygame
pygame.init()
screen = pygame.display.set_mode((800, 600)) clock
= pygame.time.Clock()
running = True
```

2. Adding the Player

Design: Represent the player with a rectangle.

- Allow the player to move left and right.

```
player_x, player_y = 375, 550
player_speed = 5
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
  player_x -= player_speed
if keys[pygame.K_RIGHT]:
  player_x += player_speed

pygame.draw.rect(screen, (0, 255, 0), (player_x, player_y, 50, 50))
```

3. Adding Falling Objects

Design:

- Represent objects as circles that spawn randomly at the top.
- Move down the screen at a constant speed.

```
from random import randint

player_x, player_y = 375, 550
player_speed = 5
objects = []
for _ in range(5):
  objects.append({"x": randint(0, 750), "y": randint(-200, -50)})

for obj in objects:
  obj["y"] += 5

pygame.draw.circle(screen, (255, 0, 0), (obj["x"], obj["y"]), 20)
```

4. Detecting Collisions

Goal: End the game if the player collides with an object.
Logic: Use Pygame's colliderect() function.

```
player_rect = pygame.Rect(player_x, player_y, 50, 50)
for obj in objects:
  object_rect = pygame.Rect(obj["x"], obj["y"], 20, 20)
  if player_rect.colliderect(object_rect):
    print("Game Over!")
    running = False
```

5. Adding a Scoring System

Design:

- Increase the score for each frame the player survives.
- Display the score on the screen.

```
score = 0
score += 1
font = pygame.font.Font(None, 36)
score_text = font.render(f"Score: {score}", True, (255, 255, 255))
screen.blit(score_text, (10, 10))
```

6. Polishing the Game

Ideas:

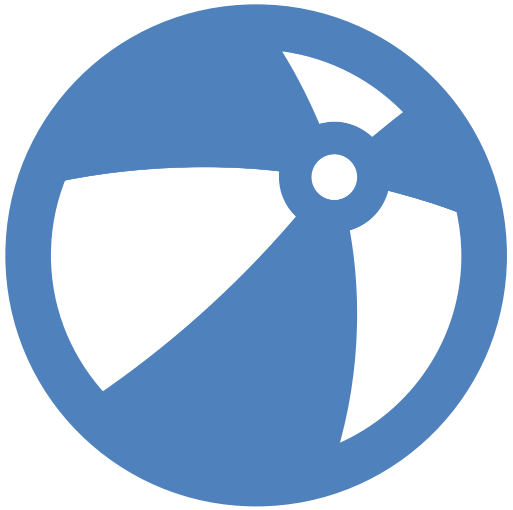
- Add sound effects (e.g., when objects fall).
- Introduce difficulty levels (e.g., increase speed over time).
- Add a restart or game-over screen.

Hands-On Activity

Task: Finish the "Dodge the Falling Objects" game.

- **Add** at least one new feature:
  - Power-ups (e.g., shields, extra lives).
  - Different types of falling objects (e.g., larger ones that move faster).

Key Takeaways



Core Concepts:

- The game loop is central to game design.
- Organize your game project into modular components.
- Combine all Pygame features to build a complete game.

E is for EXPLORE!™

@wichit2s
https://wichit2s.github.io/courses/oop/