PURDUE UNIVERSITY | Department of Computer Science

# Lesson 5: Advanced Project Development Concepts

## Objectives:

- Learn to create and manage game objects using classes.
- Implement basic interaction detection and scoring in a Pygame project.

## Lesson Plan

### 1. Introduction (10 minutes)

#### Engagement:

- Discuss how games and simulations use objects and interactions to create engaging experiences.
- Show a simple example of a game object in Pygame to demonstrate the concept.

### 2. Explanation (15 minutes)

**What is a class?**

- A class is a blueprint for creating objects (a particular data structure).
- Objects are instances of classes. They bundle data and functionality together.
- Each object created from the class can have different values for the attributes defined by the class.

Example Code:

```
class Planet:
    def __init__(self, name, color, radius, position):
        self.name = name  # Name of the planet
        self.color = color  # Color of the planet
        self.radius = radius  # Radius of the planet
        self.position = position  # Position of the planet

    def draw(self, screen):
        # Method to draw the planet on the screen
        pygame.draw.circle(screen, self.color, (int(self.position[0]),
int(self.position[1])), self.radius)

# Create instances of the Planet class
earth = Planet("Earth", (0, 0, 255), 20, [400, 300])
mars = Planet("Mars", (255, 0, 0), 15, [200, 300])
```

**Key Points to Explain:**

Class Definition: The class keyword is used to define a class.
Constructor Method: The __init__ method initializes the object's attributes.
Attributes: Attributes like name, color, radius, and position are characteristics of the object.
Methods: Methods like draw define behaviors of the object.

**What is Interaction Detection?**

- Interaction detection, such as collision detection, is crucial in game development to determine when objects interact with each other.
- A common approach to collision detection is to calculate the distance between the centers of two objects and check if it is less than the sum of their radii.

Example Code:

```
def detect_collision(planet1, planet2):
    # Calculate the difference in the x-coordinates
    dx = planet1.position[0] - planet2.position[0]
    # Calculate the difference in the y-coordinates
    dy = planet1.position[1] - planet2.position[1]
    # Calculate the distance between the two planets using the Euclidean distance
formula
    distance = math.sqrt(dx**2 + dy**2)
    # Return True if the distance is less than the sum of the radii (collision
detected)
    return distance < planet1.radius + planet2.radius
```

**Scoring**

- Scoring mechanisms are used to quantify and reward player actions.
- Scores can be updated based on certain interactions or events in the game.
- Global Variables: score is a global variable accessed and modified within the update_score function.
- Function: update_score updates the score based on the points passed to it.

Example Code:

```
score = 0  # Initialize score

def update_score(points):
    global score  # Use the global score variable
    score += points  # Update score by adding points
    print("Score:", score)  # Print the updated score
```

# 3. Hands-On Activity (20 minutes)

## Task:

Give students the code for them to be able to read through and understand the parts and pieces how they go together.

Example Code:

```
import pygame
from pygame.locals import *
import math
import os
import random

# Set SDL audio driver to avoid driver error in console
os.environ['SDL_AUDIODRIVER'] = 'dsp'

# Initialize Pygame
pygame.init()

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

# Define the Planet class
class Planet:
    def __init__(self, name, color, radius, position):
        self.name = name  # Name of the planet
        self.color = color  # Color of the planet
        self.radius = radius  # Radius of the planet
        self.position = position  # Position of the planet
        self.velocity = [random.uniform(-2, 2), random.uniform(-2, 2)]  # Random
initial velocity

    def draw(self, screen):
        # Method to draw the planet on the screen
        pygame.draw.circle(screen, self.color, (int(self.position[0]),
int(self.position[1])), self.radius)

    def move(self):
        # Update the position of the planet
        self.position[0] += self.velocity[0]
        self.position[1] += self.velocity[1]

        # Bounce off the edges of the screen
        if self.position[0] <= self.radius or self.position[0] >= 800 -
self.radius:
            self.velocity[0] = -self.velocity[0]
        if self.position[1] <= self.radius or self.position[1] >= 600 -
self.radius:
            self.velocity[1] = -self.velocity[1]

# Define the function to detect collisions between planets
def detect_collision(planet1, planet2):
    # Calculate the difference in the x-coordinates
    dx = planet1.position[0] - planet2.position[0]
    # Calculate the difference in the y-coordinates
    dy = planet1.position[1] - planet2.position[1]
    # Calculate the distance between the two planets using the Euclidean distance
formula
    distance = math.sqrt(dx**2 + dy**2)
```

```
    # Return True if the distance is less than the sum of the radii (collision
detected)
    return distance < planet1.radius + planet2.radius

# Initialize the score
score = 0

# Define the function to update the score
def update_score(points):
    global score  # Use the global score variable
    score += points  # Update score by adding points
    print("Score:", score)  # Print the updated score

# Create instances of the Planet class
earth = Planet("Earth", (0, 0, 255), 20, [400, 300])
mars = Planet("Mars", (255, 0, 0), 15, [200, 300])

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == KEYDOWN and event.key == K_ESCAPE:
            running = False

    # Fill the screen with black
    screen.fill((0, 0, 0))

    # Move and draw the planets
    earth.move()
    mars.move()
    earth.draw(screen)
    mars.draw(screen)

    # Check for collisions between the planets
    if detect_collision(earth, mars):
        print("Collision detected!")
        update_score(10)
        # Reposition Mars to simulate a reset after collision
        mars.position = [random.randint(100, 700), random.randint(100, 500)]

    # Update the display
    pygame.display.flip()

# Quit Pygame
pygame.quit()
```

# 4. Review (10 minutes)

## Q&A:

- Address any questions students might have about variables, data types, or the example code.

# Exit Ticket: Advanced Project Development Concepts

## 1. What is the primary purpose of using a class in Python?

A) To execute a block of code repeatedly.

B) To store a collection of items.

C) To create a blueprint for objects that bundle data and functionality together.

D) To handle exceptions in a program.

## 2. What method is used to initialize the attributes of a class?

A) __start__

B) __init__

C) __main__

D) __setup__

## 3. In the context of the Planet class, what does the `draw` method do?

A) It calculates the velocity of the planet.

B) It updates the position of the planet.

C) It detects collisions between planets.

D) It draws the planet on the screen.

## 4. How is the distance between two points calculated using the Euclidean distance formula?

A) distance = (x2 - x1) + (y2 - y1)

B) distance = (x2 + x1) * (y2 + y1)

C) distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)

D) distance = (x2 - x1) / (y2 - y1)

## 5. What is the purpose of the `detect_collision` function in the lesson?

A) To move the planets across the screen.

B) To draw the planets on the screen.

C) To detect if two planets have collided based on their positions and radii.

D) To update the score of the game.

## 6. What global variable is used to keep track of the player's score?

A) points
B) counter
C) score
D) value

## 7. How does the `update_score` function modify the score?

A) It resets the score to zero.
B) It subtracts points from the score.
C) It multiplies the score by a factor.
D) It adds points to the score and prints the updated score.

## 8. Which Pygame function is used to fill the screen with a specific color?

A) `pygame.draw.circle`
B) `pygame.display.flip`
C) `pygame.fill`
D) `screen.fill`

## 9. What is the effect of the following code snippet?

```
if detect_collision(earth, mars):
    update_score(10)
```

A) It stops the game if a collision is detected.
B) It moves the planets to a new position.
C) It draws a new planet on the screen.
D) It updates the score by 10 points if a collision is detected.

## 10. Why is the `pygame.display.flip` function used in the main loop?

A) To initialize the Pygame library.
B) To handle user input events.
C) To update the display with the latest changes.
D) To close the Pygame window.

# Answer Key:

1. C) To create a blueprint for objects that bundle data and functionality together.
2. B) __init__
3. D) It draws the planet on the screen.
4. C) distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)
5. C) To detect if two planets have collided based on their positions and radii.
6. C) score
7. D) It adds points to the score and prints the updated score.
8. D) screen.fill
9. D) It updates the score by 10 points if a collision is detected.
10. C) To update the display with the latest changes.