Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Docente: Fred Torres Cruz
Autor : Edilberto Wilson Mamani Emanuel

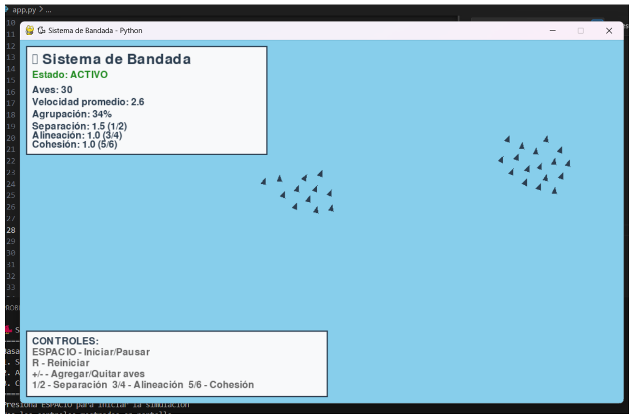
Trabajo Encargado - Realizar la codificaci´ón del ejemplo realizado en clase:

Mi tareaa

```
"""
Sistema Bioinspirado: Bandada de Aves
Implementación en Python usando pygame




import pygame
import math
import random
import sys
from typing import List, Tuple

# Inicializar pygame
pygame.init()

# Constantes
SCREEN_WIDTH = 1000
SCREEN_HEIGHT = 600
FPS = 60
BACKGROUND_COLOR = (135, 206, 235)  # Sky blue
BIRD_COLOR = (44, 62, 80)  # Dark blue-gray
UI_COLOR = (255, 255, 255)  # White
PANEL_COLOR = (248, 249, 250)  # Light gray

class Vector2D:
    """Clase simple para manejar vectores 2D"""
    def __init__(self, x: float = 0, y: float = 0):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)
    def __mul__(self, scalar: float):
        return Vector2D(self.x * scalar, self.y * scalar)

    def __truediv__(self, scalar: float):
        if scalar != 0:
            return Vector2D(self.x / scalar, self.y / scalar)
        return Vector2D(0, 0)

    def magnitude(self) -> float:
        return math.sqrt(self.x**2 + self.y**2)

    def normalize(self):
        mag = self.magnitude()
        if mag > 0:
            return self / mag
        return Vector2D(0, 0)

    def limit(self, max_val: float):
        if self.magnitude() > max_val:
            return self.normalize() * max_val
        return self

    def distance_to(self, other) -> float:
        return (self - other).magnitude()

class Bird:
    """Clase que representa un ave individual en la bandada"""

    def __init__(self, x: float, y: float):
        self.position = Vector2D(x, y)
        self.velocity = Vector2D(
            random.uniform(-2, 2),
            random.uniform(-2, 2)
        )
        self.acceleration = Vector2D(0, 0)

        # Parámetros de comportamiento
        self.max_speed = 3.0
        self.max_force = 0.1
        self.size = 6

        # Radios de percepción
        self.separation_radius = 25
        self.alignment_radius = 50
        self.cohesion_radius = 50

    def separate(self, birds: List['Bird']) -> Vector2D:
        """Regla 1: Separación - evitar vecinos muy cercanos"""
        steer = Vector2D(0, 0)
        count = 0

        for bird in birds:
            distance = self.position.distance_to(bird.position)
            if 0 < distance < self.separation_radius:
                # Vector que apunta lejos del vecino
                diff = self.position - bird.position
                diff = diff.normalize()
                diff = diff / distance  # Peso inversamente proporcional a la distancia
                steer = steer + diff
                count += 1

        if count > 0:
            steer = steer / count
            steer = steer.normalize() * self.max_speed
            steer = steer - self.velocity
            steer = steer.limit(self.max_force)

        return steer

    def align(self, birds: List['Bird']) -> Vector2D:
        """Regla 2: Alineación - moverse en la misma dirección"""
        sum_velocity = Vector2D(0, 0)
        count = 0

        for bird in birds:
            distance = self.position.distance_to(bird.position)
            if 0 < distance < self.alignment_radius:
                sum_velocity = sum_velocity + bird.velocity
                count += 1

        if count > 0:
            sum_velocity = sum_velocity / count
            sum_velocity = sum_velocity.normalize() * self.max_speed
            steer = sum_velocity - self.velocity
            steer = steer.limit(self.max_force)
            return steer

        return Vector2D(0, 0)
    def cohesion(self, birds: List['Bird']) -> Vector2D:
        """Regla 3: Cohesión - moverse hacia el centro del grupo"""
        sum_position = Vector2D(0, 0)
        count = 0

        for bird in birds:
            distance = self.position.distance_to(bird.position)
            if 0 < distance < self.cohesion_radius:
                sum_position = sum_position + bird.position
                count += 1

        if count > 0:
            sum_position = sum_position / count
            return self.seek(sum_position)

        return Vector2D(0, 0)

    def seek(self, target: Vector2D) -> Vector2D:
        """Buscar un punto específico"""
        desired = target - self.position
        desired = desired.normalize() * self.max_speed
        steer = desired - self.velocity
        steer = steer.limit(self.max_force)
        return steer

    def flock(self, birds: List['Bird'], weights: Tuple[float, float, float]):
        """Aplicar las tres reglas de la bandada"""
        sep_weight, ali_weight, coh_weight = weights

        separation = self.separate(birds) * sep_weight
        alignment = self.align(birds) * ali_weight
        cohesion = self.cohesion(birds) * coh_weight

        self.acceleration = separation + alignment + cohesion

    def update(self, screen_width: int, screen_height: int):
        """Actualizar posición y velocidad"""
        # Aplicar aceleración
        self.velocity = self.velocity + self.acceleration
        self.velocity = self.velocity.limit(self.max_speed)
        self.position = self.position + self.velocity

        # Resetear aceleración
        self.acceleration = Vector2D(0, 0)
        # Wrap around en los bordes
        if self.position.x < 0:
            self.position.x = screen_width
        elif self.position.x > screen_width:
            self.position.x = 0

        if self.position.y < 0:
            self.position.y = screen_height
        elif self.position.y > screen_height:
            self.position.y = 0

    def draw(self, screen):
        """Dibujar el ave como un triángulo apuntando en la dirección del movimiento"""
        if self.velocity.magnitude() > 0:
            angle = math.atan2(self.velocity.y, self.velocity.x)

            # Puntos del triángulo
            point1 = (
                self.position.x + self.size * math.cos(angle),
                self.position.y + self.size * math.sin(angle)
            )
            point2 = (
                self.position.x + self.size * math.cos(angle + 2.4),
                self.position.y + self.size * math.sin(angle + 2.4)
            )
            point3 = (
                self.position.x + self.size * math.cos(angle - 2.4),
                self.position.y + self.size * math.sin(angle - 2.4)
            )

            pygame.draw.polygon(screen, BIRD_COLOR, [point1, point2, point3])

class FlockSystem:
    """Sistema principal que maneja la simulación de la bandada"""

    def __init__(self):
        self.screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
        pygame.display.set_caption(" Sistema de Bandada - Python")
        self.clock = pygame.time.Clock()
        self.font = pygame.font.Font(None, 36)
        self.small_font = pygame.font.Font(None, 24)

        # Parámetros de la simulación
        self.bird_count = 100
        self.birds = []
        self.running = True
        self.simulation_active = False

        # Pesos para las reglas de flocking
        self.separation_weight = 1.5
        self.alignment_weight = 1.0
        self.cohesion_weight = 1.0

        self.create_birds()

    def create_birds(self):
        """Crear aves en posiciones aleatorias"""
        self.birds = []
        for _ in range(self.bird_count):
            x = random.uniform(50, SCREEN_WIDTH - 50)
            y = random.uniform(50, SCREEN_HEIGHT - 50)
            self.birds.append(Bird(x, y))

    def handle_events(self):
        """Manejar eventos de pygame"""
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    self.simulation_active = not self.simulation_active
                elif event.key == pygame.K_r:
                    self.create_birds()
                elif event.key == pygame.K_PLUS or event.key == pygame.K_EQUALS:
                    if self.bird_count < 100:
                        self.bird_count += 5
                        self.create_birds()
                elif event.key == pygame.K_MINUS:
                    if self.bird_count > 5:
                        self.bird_count -= 5
                        self.create_birds()
                elif event.key == pygame.K_1:
                    self.separation_weight = max(0, self.separation_weight - 0.1)
                elif event.key == pygame.K_2:
                    self.separation_weight += 0.1
                elif event.key == pygame.K_3:
                    self.alignment_weight = max(0, self.alignment_weight - 0.1)
                elif event.key == pygame.K_4:
                    self.alignment_weight += 0.1
                elif event.key == pygame.K_5:
```

```
                self.cohesion_weight = max(0, self.cohesion_weight - 0.1)
            elif event.key == pygame.K_6:
                self.cohesion_weight += 0.1

    def update(self):
        """Actualizar la simulación"""
        if self.simulation_active:
            weights = (self.separation_weight, self.alignment_weight, self.cohesion_weig

            for bird in self.birds:
                bird.flock(self.birds, weights)
                bird.update(SCREEN_WIDTH, SCREEN_HEIGHT)

    def calculate_stats(self) -> Tuple[float, float]:
        """Calcular estadísticas de la bandada"""
        if not self.birds:
            return 0.0, 0.0

        # Velocidad promedio
        total_speed = sum(bird.velocity.magnitude() for bird in self.birds)
        avg_speed = total_speed / len(self.birds)

        # Nivel de agrupación (basado en distancia al centro)
        center_x = sum(bird.position.x for bird in self.birds) / len(self.birds)
        center_y = sum(bird.position.y for bird in self.birds) / len(self.birds)
        center = Vector2D(center_x, center_y)

        total_distance = sum(bird.position.distance_to(center) for bird in self.birds)
        avg_distance = total_distance / len(self.birds)
        grouping = max(0, 100 - (avg_distance / 3))

        return avg_speed, grouping

    def draw_ui(self):
        """Dibujar la interfaz de usuario"""
        # Panel de información
        panel_rect = pygame.Rect(10, 10, 400, 180)
        pygame.draw.rect(self.screen, PANEL_COLOR, panel_rect)
        pygame.draw.rect(self.screen, BIRD_COLOR, panel_rect, 2)

        # Título
        title = self.font.render(" Sistema de Bandada", True, BIRD_COLOR)
        self.screen.blit(title, (20, 20))

        # Estado
        status = "ACTIVO" if self.simulation_active else "PAUSADO"
        status_color = (34, 139, 34) if self.simulation_active else (220, 20, 60)
        status_text = self.small_font.render(f"Estado: {status}", True, status_color)
        self.screen.blit(status_text, (20, 50))

        # Estadísticas
        avg_speed, grouping = self.calculate_stats()
        stats = [
            f"Aves: {len(self.birds)}",
            f"Velocidad promedio: {avg_speed:.1f}",
            f"Agrupación: {grouping:.0f}%"
        ]

        for i, stat in enumerate(stats):
            text = self.small_font.render(stat, True, BIRD_COLOR)
            self.screen.blit(text, (20, 75 + i * 20))

        # Parámetros
        params = [
            f"Separación: {self.separation_weight:.1f} (1/2)",
            f"Alineación: {self.alignment_weight:.1f} (3/4)",
            f"Cohesión: {self.cohesion_weight:.1f} (5/6)"
        ]

        for i, param in enumerate(params):
            text = self.small_font.render(param, True, BIRD_COLOR)
            self.screen.blit(text, (20, 135 + i * 15))

        # Controles
        controls_rect = pygame.Rect(10, SCREEN_HEIGHT - 120, 500, 110)
        pygame.draw.rect(self.screen, PANEL_COLOR, controls_rect)
        pygame.draw.rect(self.screen, BIRD_COLOR, controls_rect, 2)

        controls = [
            "CONTROLES:",
            "ESPACIO - Iniciar/Pausar",
            "R - Reiniciar",
            "+/- - Agregar/Quitar aves",
            "1/2 - Separación  3/4 - Alineación  5/6 - Cohesión"
        ]

        for i, control in enumerate(controls):
            color = BIRD_COLOR if i == 0 else (100, 100, 100)
            text = self.small_font.render(control, True, color)
            self.screen.blit(text, (20, SCREEN_HEIGHT - 110 + i * 18))
```

```
    def draw(self):
        """Dibujar todo en pantalla"""
        self.screen.fill(BACKGROUND_COLOR)

        # Dibujar aves
        for bird in self.birds:
            bird.draw(self.screen)

        # Dibujar UI
        self.draw_ui()

        pygame.display.flip()

    def run(self):
        """Bucle principal del juego"""
        print(" Sistema de Bandada de Aves - Python")
        print("=" * 40)
        print("Basado en las reglas de Craig Reynolds (1987)")
        print("1. Separación: Evitar vecinos cercanos")
        print("2. Alineación: Moverse en la misma dirección")
        print("3. Cohesión: Mantenerse cerca del grupo")
        print("=" * 40)
        print("Presiona ESPACIO para iniciar la simulación")
        print("Usa los controles mostrados en pantalla")
        print("=" * 40)

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

        pygame.quit()
        sys.exit()

if __name__ == "__main__":
    # Verificar que pygame esté disponible
    try:
        import pygame
        system = FlockSystem()
        system.run()
    except ImportError:
        print("Error: pygame no está instalado.")
        print("Instálalo con: pip install pygame")
```



Sistema de Bandada - Python

Sistema de Bandada

Estado: ACTIVO

Aves: 30
Velocidad promedio: 2.6
Agrupación: 34%
Separación: 1.5 (1/2)
Alineación: 1.0 (3/4)
Cohesión: 1.0 (5/6)

CONTROLES:
ESPACIO - Iniciar/Pausar
R - Reiniciar
+/- - Agregar/Quitar aves
1/2 - Separación 3/4 - Alineación 5/6 - Cohesión

Figura 1: Enter Caption

```
sys.exit(1)
```