Project Design

of

# Bio-Inspired Swarm Robotics Simulation Using Python

**AINT 44052**

**Intelligent Autonomous Robotics**

**H.M. Sudith Amarasinghe**

**CS/2018/003**

# 1. System Architecture

## 1.1 Overview

Using bio-inspired algorithms, a number of robots, or booids, move and interact in a 2D simulation environment. The simulation environment, the boid model, the control algorithms, and the user interface are the primary parts.

## 1.2 Components

- **Robot Model (Boid):** Each boid has a position, velocity, and acceleration. The boids use simple movement and sensor capabilities to interact with their environment.
- **Environment:** A 2D plane with static obstacles. The environment is represented using a grid where each cell can be either empty or occupied by an obstacle.
- **Control Algorithm:** Implements the bio-inspired flocking behavior using three main rules: alignment, cohesion, and separation.

# 2. Flowcharts and Diagrams

## 2.1 Flowchart for Boid Behavior:

**01. Initialize Boid:**

- Set initial position, velocity, and acceleration.

**02. Edges Check:**

- Wrap around edges if the boid goes out of bounds.

**03. Flocking Behavior:**

- Calculate alignment, cohesion, and separation forces.

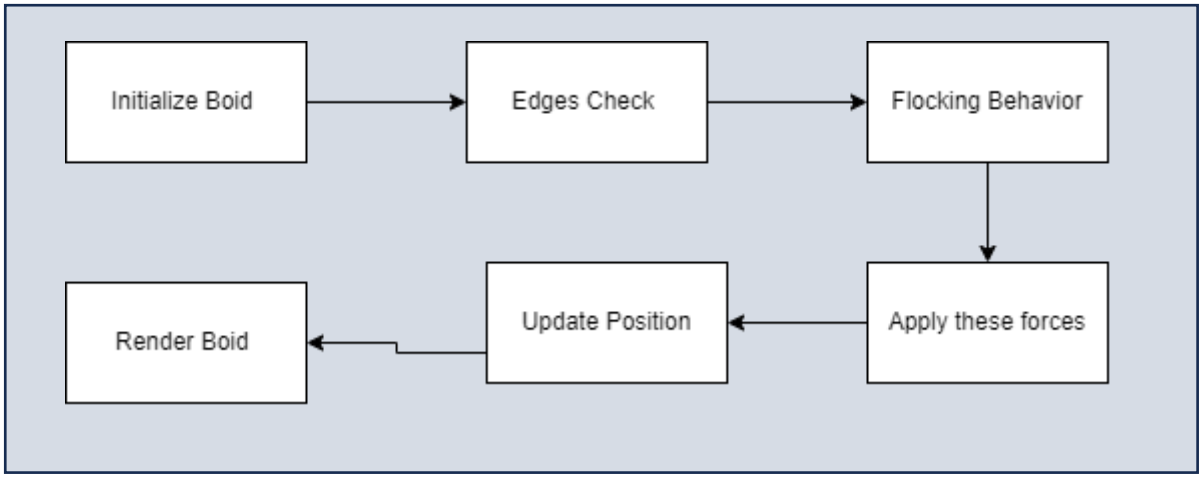
- Apply these forces to update the boid’s acceleration.

**04. Update Position:**

- Update the boid’s velocity and position based on acceleration.

**05. Render Boid:**

- Draw the boid on the screen.




-Simple flowchart for the system-

# 3. Implementation Details

## 3.1 Classes and Functions:

**Boid Class:**

- __init__: Initializes the boid with random position and velocity.
- update: Updates the boid's position and velocity.
- apply_force: Applies a force to the boid's acceleration.
- align, cohesion, separation: Calculate alignment, cohesion, and separation forces.

**Simulation Class:**

- __init__: Initializes the simulation environment.
- run: Main loop to update and render boids.

## 3.2 Pseudocode:

```
class Boid:
    init(position, velocity):
        self.position = position
        self.velocity = velocity
        self.acceleration = Vector(0, 0)

    apply_force(force):
        self.acceleration += force

    align(boids):
        steering = Vector(0, 0)
        total = 0
        for boid in boids:
            if distance(self, boid) < perception_radius:
                steering += boid.velocity
                total += 1
        if total > 0:
            steering /= total
            steering = normalize(steering) * max_speed
            steering -= self.velocity
            limit(steering, max_force)
        return steering

    cohesion(boids):
        steering = Vector(0, 0)
        total = 0
        for boid in boids:
            if distance(self, boid) < perception_radius:
                steering += boid.position
                total += 1
        if total > 0:
            steering /= total
            steering -= self.position
            steering = normalize(steering) * max_speed
            steering -= self.velocity
            limit(steering, max_force)
        return steering
```

```
    separation(boids):
        steering = Vector(0, 0)
        total = 0
        for boid in boids:
            if distance(self, boid) < perception_radius:
                diff = self.position - boid.position
                diff /= distance(self, boid)
                steering += diff
                total += 1
        if total > 0:
            steering /= total
            steering = normalize(steering) * max_speed
            steering -= self.velocity
            limit(steering, max_force)
        return steering

    update():
        self.velocity += self.acceleration
        limit(self.velocity, max_speed)
        self.position += self.velocity
        self.acceleration *= 0

    edges():
        if self.position.x > WIDTH:
            self.position.x = 0
        elif self.position.x < 0:
            self.position.x = WIDTH
        if self.position.y > HEIGHT:
            self.position.y = 0
        elif self.position.y < 0:
            self.position.y = HEIGHT

class Simulation:
    init():
        self.boids = [Boid() for _ in range(num_boids)]

    run():
        while running:
            for boid in self.boids:
                boid.edges()
                boid.flock(self.boids)
                boid.update()
                boid.show()
```