FAF.FIA16.1 -- Artificial Intelligence Fundamentals

> **Lab 2:** Flocking Behavior
> **Performed by:** Furdui Alexandru, group FAF-192
> **Verified by:** Mihail Gavrilita, asist. univ.

Imports and Utils

```
from boid import Boid
from p5 import stroke, circle
from utils import Vector
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
```

Task 1 -- Implement the Vector class in Python that works on simple Python lists.

```
class Vector:
    def __init__(self, *args):
        self.values = np.array(args)

    def norm(self):
        return np.linalg.norm(self.values)

    def __add__(self, other):
        if self.values.shape != other.values.shape:
            raise ValueError("Both vectors must have the same number of elements
        return Vector(*(self.values + other.values))

    def __sub__(self, other):
        if self.values.shape != other.values.shape:
            raise ValueError("Both vectors must have the same number of elements
        return Vector(*(self.values - other.values))

    def __mul__(self, scalar):
        return Vector(*(self.values * scalar))

    def __truediv__(self, scalar):
        return Vector(*(self.values / scalar))

    def dot(self, other):
        if self.values.shape != other.values.shape:
            raise ValueError("Both vectors must have the same number of elements
        return np.dot(self.values, other.values)

    def cross(self, other):
        if self.values.shape != (3,) or other.values.shape != (3,):
            raise ValueError("Both vectors must have 3 elements")
        return Vector(*np.cross(self.values, other.values))
```

Output

```
norm:  3.7416573867739413
addition:  [3 5 7]
substraction:  [-1 -1 -1]
multiplication:  [2 4 6]
div:  [0.5 1.  1.5]
dot:  7.0
cross:  [0. 0. 0.]
v1: Vector(1, 2, 3)
v2: Vector(2, 3, 4)
```

This is a simple Vector class that uses NumPy lib to implement vector operations

Task 2 -- Implement the Boid class with the steering behaviors

1. **Separation**

```
def separation(self, boids):
        steering = Vector(*np.zeros(2))
        total = 0
        avg_vector = Vector(*np.zeros(2))
        for boid in boids:
            distance = np.linalg.norm(boid.position - self.position)
            if self.position != boid.position and distance < self.perception:
                if distance < 100:  # Minimum distance threshold
                    diff = self.position - boid.position
                    diff /= distance**2
                    avg_vector += diff
                    total += 1
        if total > 0:
            avg_vector /= total
            avg_vector = Vector(*avg_vector)
            if np.linalg.norm(avg_vector) > 0:
                avg_vector = (avg_vector / np.linalg.norm(avg_vector)) * self.ma
            steering = avg_vector - self.velocity
            if np.linalg.norm(steering) > self.max_force:
                steering = (steering / np.linalg.norm(steering)) * self.max_forc

        return steering
```

2. **Cohesion**

```
def cohesion(self, boids):
    steering = Vector(*np.zeros(2))
    total = 0
    center_of_mass = Vector(*np.zeros(2))
    for boid in boids:
        if np.linalg.norm(boid.position - self.position) < self.perception:
            center_of_mass += boid.position
            total += 1
    if total > 0:
        center_of_mass /= total
        center_of_mass = Vector(*center_of_mass)
        vec_to_com = center_of_mass - self.position
        if np.linalg.norm(vec_to_com) > 0:
            vec_to_com = (vec_to_com / np.linalg.norm(vec_to_com)) * self.ma
        steering = vec_to_com - self.velocity
        if np.linalg.norm(steering)> self.max_force:
            steering = (steering /np.linalg.norm(steering)) * self.max_force

    return steering
```

3. **Alignment**

```
def align(self, boids):
    steering = Vector(*np.zeros(2))
    total = 0
    avg_vector = Vector(*np.zeros(2))
    for boid in boids:
        if np.linalg.norm(boid.position - self.position) < self.perception:
            avg_vector += boid.velocity
            total += 1
    if total > 0:
        avg_vector /= total
        avg_vector = Vector(*avg_vector)
        avg_vector = (avg_vector / np.linalg.norm(avg_vector)) * self.max_sp
        steering = avg_vector - self.velocity

    return steering
```

In order for all the above to work expected, the class need some additional functions for boid init, draw, initial value.

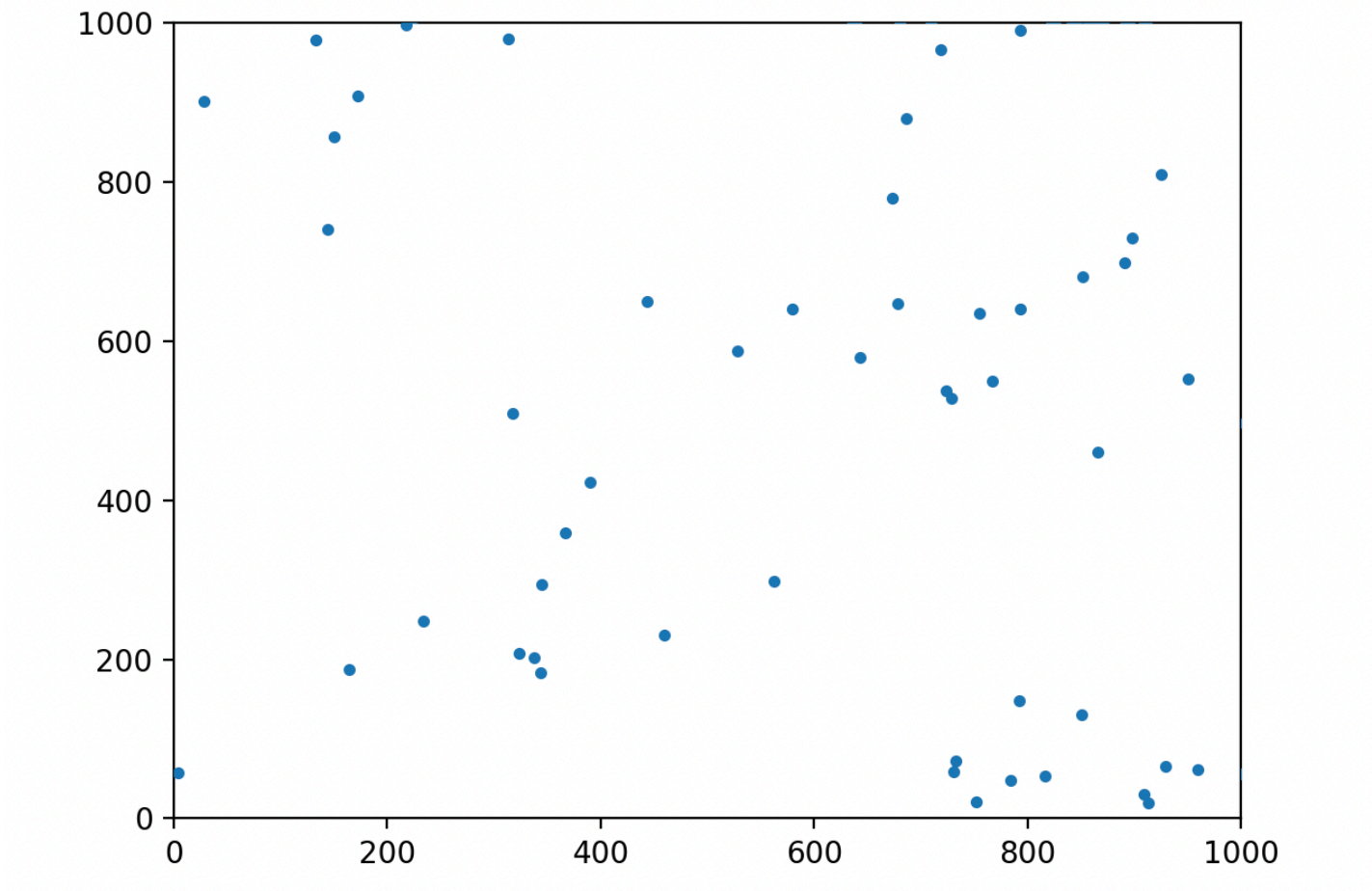
```
class Boid():
    def __init__(self, x, y, width, height):
        self.position = Vector(x, y)
        vec = (np.random.rand(2) - 0.5)*10
        self.velocity = Vector(*vec)

        vec = (np.random.rand(2) - 0.5)/2
        self.acceleration = Vector(*vec)
        self.max_force = 0.3
        self.max_speed = 5
        self.perception = 150

        self.width = width
        self.height = height

    def update(self):
        self.position += self.velocity
        self.velocity += self.acceleration
        #limit
        if np.linalg.norm(self.velocity) > self.max_speed:
            self.velocity = self.velocity / np.linalg.norm(self.velocity) * self

        self.acceleration = Vector(*np.zeros(2))

    def show(self):
        stroke(255)

        circle((self.position.x, self.position.y), radius=10)

    def edges(self):
        if self.position.x > self.width:
            self.position.x = 0
        elif self.position.x < 0:
            self.position.x = self.width

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

    # Continue with steering behaviours
```

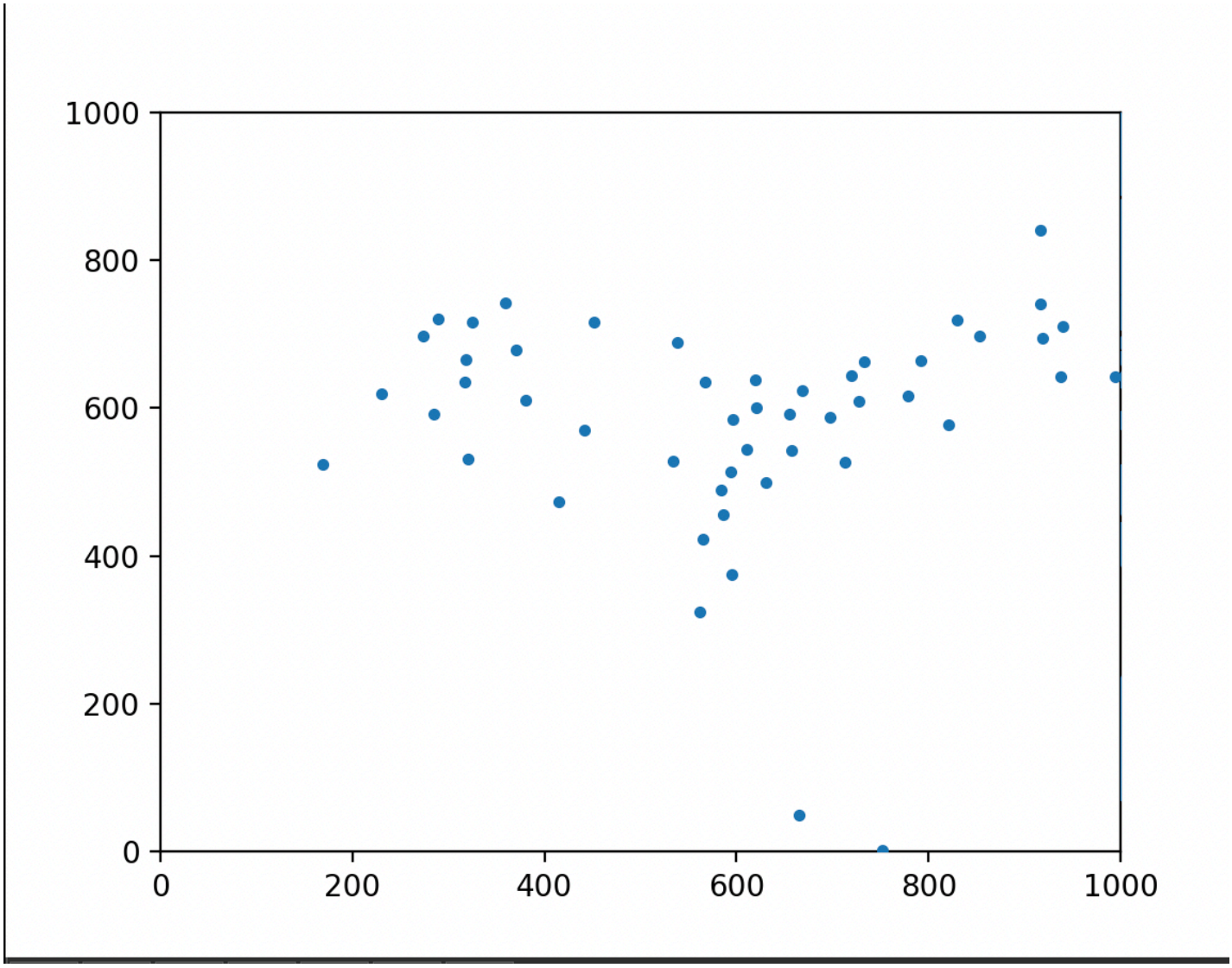
Task 3 -- Add the calm flocking behaviour to the Boid class

```
class Boid():

    # ...

    def apply_behaviour(self, boids):
        alignment = self.align(boids)
        cohesion = self.cohesion(boids)
        separation = self.separation(boids)

        self.acceleration += alignment
        self.acceleration += cohesion
        self.acceleration += separation
    # ...
```

To make it work all together, an update function is needed inside the Boid class to apply all the steering behaviours.

Combinin all together inside a main function, here is how the result will look like:





Conclusions:

Concluding, after implementing this labo
operations and implemented in a simple V
operations, i tried to implement the sir
bibliography was very helpful

Concluding, after implementing this laboratory work, i rewound the vector class operations and implemented in a simple Vector class. And using those operations, i tried to implement the simple flocking behaviour, the post from bibliography was very helpful

Bibliography:

https://betterprogramming.pub/boids-simulating-birds-flock-behavior-in-python-9fff99375118 - Simulating Bird Flock Behavior in Python Using Boids