FAF.FIA16.1 -- Artificial Intelligence Fundamentals

> **Lab 1: FIA \** Performed by:** Gavirlita Ion, group FAF-191 \ **Verified by:** Mihail Gavrilita, asist. univ.

Imports and Utils

In [16]:

```
import math
```

Task 1 -- Implement the Vector class in Python that works on simple Python lists. The Vector class should implement the vector operations:

In [17]:

```
class Vector:
    def __init__(self, arr=None):
        if arr is None:
            arr = []
        self.vec = arr

    def norm(self):
        return math.sqrt(sum(x ** 2 for x in self.vec))

    def __add__(self, other):
        return Vector([x + y for x, y in zip(self.vec, other.vec)])

    def __sub__(self, other):
        return Vector([x - y for x, y in zip(self.vec, other.vec)])

    def __mul__(self, other):
        if isinstance(other, Vector):
            return sum([x * y for x, y in zip(self.vec, other.vec)])

        if isinstance(other, int) or isinstance(other, float):
            return Vector([x * other for x in self.vec])


    def __truediv__(self, other):
        return Vector([x / other for x in self.vec])

    def __str__(self):
        return str(self.vec)

    def __repr__(self):
        return str(self.vec)
    def __abs__(self):
        return Vector([abs(self.vec[0]), abs(self.vec[1])])

    def cross_2d(self, other):
        return self.vec[0] * other.vec[1] - self.vec[1] * other.vec[0]


vector1 = Vector([1, 2, 3])
vector2 = Vector([4, 5 ,6])

print(vector1.norm())
print(vector1 + vector2)
print(vector1 - vector2)
print(vector1 * vector2)
print(vector1.cross_2d(vector2))
```

```
3.7416573867739413
[5, 7, 9]
[-3, -3, -3]
32
-3
```

Task 2 -- Using the Vector class and the provided paper, implement the Boid class with the steering behaviors

In [18]:

```
class Boid:
    def __init__(self, position_vector, velocity_vector=Vector([1,1])):
        self.vectors = [position_vector, velocity_vector]
        self.position = position_vector
        self.velocity = velocity_vector
        self.radius = 100
        self.alignment_factor = 3.0
        self.separation_factor = 1.0
        self.cohesion_factor = 2.0



    def get_proximity(self, other_boid):
        return abs(self.position - other_boid.position)

    def flocking(self, vector_group):
        self.cohesion(vector_group)
        self.separation(vector_group)
        self.alignment(vector_group)

    # steers
    def alignment(self, vector_group):
        align_vec = Vector([0, 0])
        proximity_len = 0
        for vector in vector_group:
            if self.vectors == vector:
                continue
            other_boid = Boid(vector)
            proximity = self.get_proximity(other_boid)
            if proximity.vec[0] <= self.radius and proximity.vec[1] <= self.
                proximity_len += 1
                # sum of velocities
                align_vec += other_boid.velocity

        if proximity_len > 0:
            # average velocity
            align_vec /= proximity_len
            align_vec /= align_vec.norm()
            self.velocity += align_vec * self.alignment_factor
            self.velocity /= self.velocity.norm()

    def separation(self, vector_group):
        separation_vector = Vector([0, 0])
        proximity_len = 0
        for coordinates in vector_group:
            if coordinates == self.vectors:
                continue

            other_boid = Boid(coordinates)
            proximity = self.get_proximity(other_boid)

            if proximity.vec[0] <= self.radius and proximity.vec[1] <= self.
                # calculate the separation vector
                separation_vector += (self.position - other_boid.position)
                proximity_len += 1

        if proximity_len > 0:
            # average the separation vector and apply it to the velocity
            separation_vector /= proximity_len
            self.velocity += separation_vector * self.separation_factor
            self.velocity /= self.velocity.norm()

    def cohesion(self, vector_group):
        cohesion_vec = Vector([0, 0])
        proximity_len = 0
        for coordinates in vector_group:

            if self.vectors == coordinates:
                continue

            other_boid = Boid(coordinates)
            proximity = self.get_proximity(other_boid)

            if proximity.vec[0] <= self.radius and proximity.vec[1] <= self.
                proximity_len += 1
                # sum of positions
                cohesion_vec += other_boid.position

        if proximity_len > 0:
            # center of mass
            cohesion_vec /= proximity_len
            # cohesion vector
            cohesion_vec -= self.position
```

```
                self.velocity += cohesion_vec * self.cohesion_factor
                self.velocity /= self.velocity.norm()
```

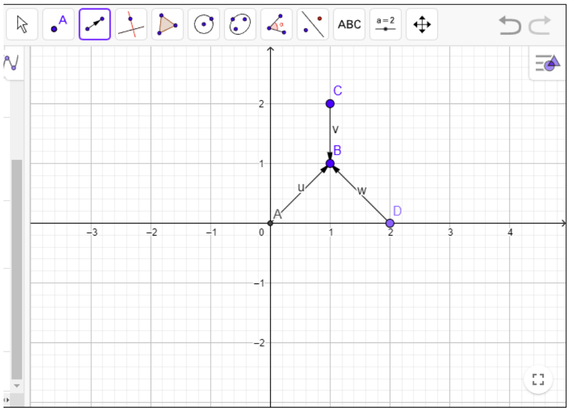
Task 3 -- Add the calm flocking behaviour to the Boid class according to the provided paper, using the 3 steering behaviours implemented in the Task 2.

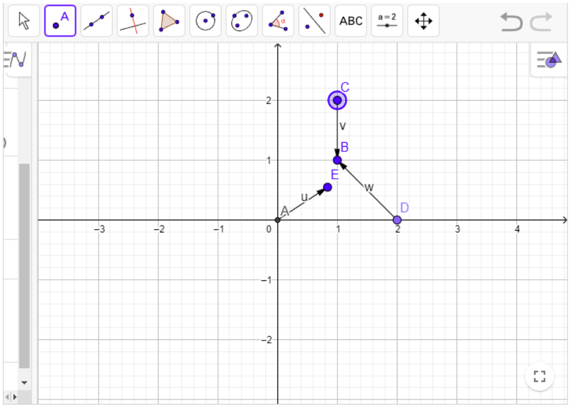
In [21]:
```
boid = Boid(Vector([0, 0]))
boidz = [Vector([1, 2]), Vector([2, 1])]

print(boid.position, boid.velocity) # before flocking
boid.flocking(boidz)
print(boid.position, boid.velocity) # after calm flocking, the velocity vect
```

```
[0, 0] [1, 1]
[0, 0] [0.8366224044184944, 0.5477800219294394]
```

Change of velocity plotted bellow:





Conclusions:

Your conclusions go here

Bibliography:

The sources you've used go here