Coding Example: Implement the behavior of Boids (Python approach)

In this lesson we will try to implement the Boids class using the traditional Pythonic approach and analyze it in terms of efficiency.

Boid Class Implementation (Python)

Since each boid is an autonomous entity with several properties such as position and velocity, it seems natural to start by writing a `Boid` class:

```python
import math
import random
from vec2 import vec2

class Boid:
    def __init__(self, x=0, y=0):
        self.position = vec2(x, y)
        angle = random.uniform(0, 2*math.pi)
        self.velocity = vec2(math.cos(angle), math.sin(angle))
        self.acceleration = vec2(0, 0)
```

The `vec2` object is a very simple class that handles all common vector operations with 2 components. It will save us some writing in the main `Boid` class. Note that there are some vector packages in the Python Package Index, but that would be an overkill for such a simple example.

Boid is a difficult case for regular Python because a boid has interaction with local neighbors. Since the boids are moving so, at every step, we need to calculate the distance of one boid with every other boid that comes within the interaction radius and then sort those distances. The prototypical way of writing the three rules is thus something like:

```python
def separation(self, boids):
  count = 0
  for other in boids:
    d = (self.position - other.position).length()
    if 0 < d < desired_separation:
      count += 1
      # ...
  if count > 0:
    # ...

def alignment(self, boids): # ...
def cohesion(self, boids): # ...
```

To complete the picture, we can also create a `Flock` object:

```python
class Flock:
  def __init__(self, count=150):
    self.boids = []
    for i in range(count):
      boid = Boid()
      self.boids.append(boid)

  def run(self):
    for boid in self.boids:
      boid.run(self.boids)
```

Complete Solution

Merging all the logic from above, given below is the complete implementation of Boids.

main.py
vec2.py

```python
# -----------------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
# -----------------------------------------------------------------------------
import math
import random
from vec2 import vec2


class Boid:
    def __init__(self, x, y):
        self.acceleration = vec2(0, 0)
        angle = random.uniform(0, 2*math.pi)
        self.velocity = vec2(math.cos(angle), math.sin(angle))
        self.position = vec2(x, y)
        self.r = 2.0
        self.max_velocity = 2
        self.max_acceleration = 0.03

    def seek(self, target):
        desired = target - self.position
        desired = desired.normalized()
        desired *= self.max_velocity
        steer = desired - self.velocity
        steer = steer.limited(self.max_acceleration)
        return steer

```

RUN SAVE RESET

output.mp4

Drawbacks

Using this approach, we can have up to 50 boids until the computation time becomes too slow for a smooth animation. As you may have guessed, we can do much better using NumPy, but let me first point out the main problem with this Python implementation.

- If you look at the code, you will certainly notice there is a lot of redundancy. More precisely, we do not exploit the fact that the *Euclidean* distance is reflexive, that is, $|x - y| = |y - x|$.
- In this naive Python implementation, each rule (function) computes n^2 distances while $\frac{n^2}{2}$ would be sufficient if properly cached.
- Furthermore, each rule re-computes every distance without caching the result for the other functions. In the end, we are computing $3n^2$ distances instead of $\frac{n^2}{2}$.

Now let's see what NumPy has to offer to give a better and more efficient solution. See you in the next lesson!