VRIJE UNIVERSITEIT AMSTERDAM

Code Tutorial 0

Project Collective intelligence

Simulator

All the simulations will be run in python by utilizing a python package known under the name Violet. For more information refer to Violet's Documentation In brief, Violet is a minimal simulation framework built on top of PyGame. In this course we are going to use it to simulate multiple agent interactions. The python version used in the course is python 3.9. To get a quick start with Violet, check the Violet Book

To install the simulator run:

```
pip install -U violet-simulator
```

Pygame Vector2

Vector2 is widely used in this first assignment. Please refer to the documentation of Vector2 for more information. Some useful methods are: **Vector2.length()** and **Vector2.normalize()**

1 Tips for the Assignment

Image rotation

One crucial part of the flocking is the direction in which the agents are moving, and, to properly tune the algorithm, you will need to visualize the direction of every agent. The Violet simulator has that option deactivated by default, but it is straightforward

to activate the image rotation. You only need to set `image_rotation = True` in the simulation config when launching it.

```
(
    Simulation(Config(image_rotation=True, movement_speed=1, radius=
                                        50, seed=1))
    .batch_spawn_agents(50, Bird, images=["images/bird.png"])
    .run()
)
```

It would be a good idea to use triangular images or and assets that gives you some sense of direction.

Normalization

During this assignment, you will be working with vectors of different magnitudes. To facilitate coordination between agents, removing the scaling factor and operating all the agents in the same magnitude is better. That is when vector normalization comes into action. Since `pygame.math.Vector2` includes a method `vector2.normalize()` the implementation is done for you.

Speed Limit

Flocking is a collective behaviour, and to ensure its proper functionality, all the collections' agents need to operate in similar magnitudes. Including a speed limit in your implementation will help you get an easier to tune flocking behaviour.

Utilizing Violet .config

Part of the assignment involves tuning the algorithm to generate a fluid flocking behaviour. You can make your life easier by utilizing the live parameter update with the .config submodule. In the file provided, you will find `class FlockingConfig(Config)`, a class to store the parameters for your flocking.

```
from vi.config import Config, dataclass, deserialize
@deserialize
@dataclass
class FlockingConfig(Config):
    alignment_weight: float = 0.5
    cohesion_weight: float = 0.5
    separation_weight: float = 0.5
    delta_time: float = 3
    mass: int = 20

    def weights(self) -> tuple[float, float, float]:
        return (self.alignment_weight, self.cohesion_weight, self.
                                            separation_weight)
```

The easiest way to include the config class in your agent is to reference it inside the class to access it later to extract the parameters needed.

```
class Bird(Agent):
    config: FlockingConfig

    def get_alignment_weigth(self)-> float:
        return self.config.alignment_weight


my_bird = Bird()
my_bird.get_alignment_weigth()
```

```
>>> 0.5
```

By utilizing the `class FlockingConfig`, the parameter live update will be enable. To update the parameter select the desire parameter with the number keys [1,2,3] and press *UP_KEY* to increase the parameter or *DOWN_KEY* to decrease it.