INF-1400: Objektorientert programmering

Assignment 2

Magnus Kanck

March 14th, 2019

1 Introduction

In this assignment a clone of the flock simulator "boids" will be implemented in accordance with object-oriented design, i.e. with the use of classes, methods and inheritance. At the end of this assignment we expect to have created a system that exhibits emergent behaviour.

1.1 Requirements

The requirements are as follows:

- Boids steer towards the average position of other boids in its group.
- Boids avoid colliding with obstacles and other boids.
- Boids move towards the average direction of other boids in its group.
- Boids should flee from hoiks (predators) and the hoiks should hunt the boids.

2 Technical Background

This assignment was implemented in Python 3.6.5 along with the modules *pygame* (1.9.4), *numpy* (1.14.3), and the *random* module.

- Classes: In Python, a class can be defined as blueprint for an Object, containing methods, which are functions inside a class, and attributes which can be utilized to modify the state of an instance of a class.[1]
- Objects: an instance of a class is called an Object and is a combination attributes and methods depending on its class, and parent classes that the object inherits from.

[1] https://docs.python.org/3/tutorial/classes.html

- Inheritance: Object-oriented languages like Python supports class inheritance as opposed to object-based languages which do not.[2] Inheritance means that attributes and methods defined inside one class can be inherited by another class, allowing for more specialized or abstract instances of the same class.
- Simulations: In the context of this assignment, a simulation is a model of real-phenomena using rules and mathematics.

3 Design and Implementation

Three classes were implemented to serve as visible objects in the simulation while a fourth class was used to handle the simulation loop and the creation and destruction of objects. The Simulation class instantiates the visible objects and stores them in "groups" or container classes, as seen in Fig.1. This allows for easier handling of the objects as their states are updated. Global attributes were implemented to allow for easier modification of object attributes and methods during testing.

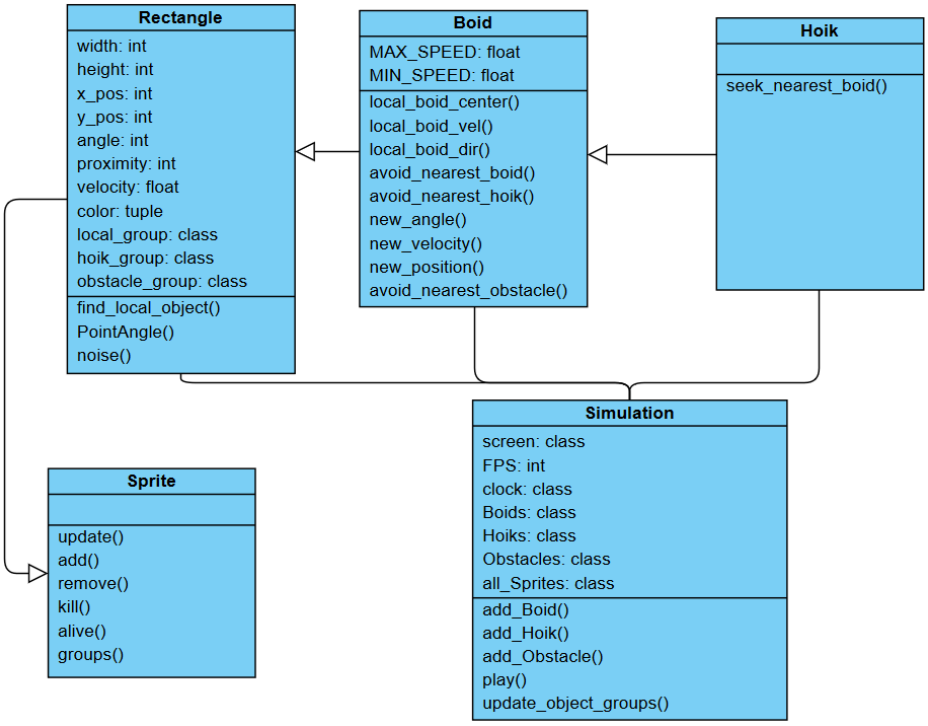

Figure 1: Class Diagram for an implementation of the "Boids" flock simulator. The Simulation class has an association with the visible game objects in that it instantiates them and updates their behaviour.

[2] https://en.wikipedia.org/wiki/Object_(computer_science)

The Sprite class from pygame is used as the base-class allowing for the use of sprite groups, which simplify the process of updating the state of several objects. The methods defined in the Rectangle class handle situations that do not pertain directly to updating object position, they serve mainly as utility methods called on by the methods defined in the child classes Boid and Hoik. The Hoik class is a child of the Boid class simply so that it can inherit the modified *update()* method which in turn calls on the necessary methods to calculate and update positions.

The method *PointAngle()* finds and returns an angle in degrees that points towards a target coordinate relative to the object that called on the method. An example of this method being used is when a boid wants to find the direction towards the average centre of the boids in its vicinity. An illustration of how this method finds the return angle can be seen in Fig.2.

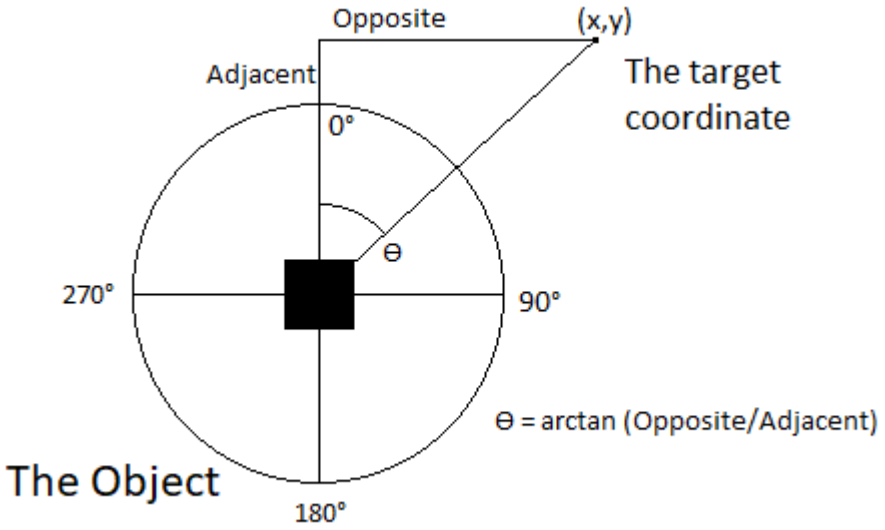

Figure 2: An example of how the *PointAngle()* method finds the angle between two points.

The return angle needs to be calculated differently based on where the target coordinate is relative to the object. If, for example, the target coordinate was situated in the lower left quadrant of the unit circle, the method would have to increment the angle by +180°, the returned angle would then be:

$$\theta_{returned} = 180° + \theta$$

In all there are eight cases for calculating the return angle; four for each quadrant and four for reach cardinal direction.

The rules governing the behaviour of the boids were ordered as follows, from highest to lowest priority:

1. Avoiding obstacles
2. Fleeing from hoiks
3. Avoiding boids that are too close
4. Steering towards the average heading and steering towards the average position of nearby boids.

The ordering is achieved by implementing if-else statements.

Flock center steering and flock directinon steering have equal priority but steering towards the center of the flock is weighted by an integer to achieve a more natural flock behaviour. The weighting is implemented by taking the weighted average of two coordinates:

$$M = \left(1 - \frac{1-t}{t}\right) * A + \frac{1-t}{t} * B$$

Where M is the new coordinate, A is the coordinate given by the average heading of the flock and B is the coordinate given by the average position of the flock. The integer t is given by the user.

4 Discussion

The problem of calculating the average direction of several vectors when they are defined as an angle and a magnitude became apparent more than once during this assignment. One cannot simply add together the angles and divide by the number of them. For example, by taking the average of three angles 359°, 0° and 1° we get:

$$\frac{359° + 0° + 1°}{3} = 120°$$

But the answer should be 0° or 360° because of the cyclic nature of degrees or radians. To solve this problem, all potential future positions had to be calculated, then averaged, and then one could find the angle towards that point. The reason this final angle has to be calculated is due to the way the simulation calculates the next position of the objects in the *new_position()* function.

Implementing the Hoik class required very little additional coding compared to the Boid class, this was in part due to how the Boid class was implemented and the simplicity of the rules governing the hoik. The *update()* method was generalized in such a way that only one additional method had to be defined to the Hoiks class, in addition to minor modifications to others, in order for the predator to be implemented.

The simulation handles checking for nearby objects by iterating over all visible objects on the screen, twice for the boids, in the *update_object_groups()* method in the Simulation class. For each iteration the alogorithm checks if there are obstacles, then hoiks, then boids near another boid, doing the same thing for hoiks in the process. This is a very time-consuming algorithm

and undoubtably has very poor worst-case complexity, which is a measure of an algorithms running time when iterating over very large numbers.

5 Conclusion

In this assignment a clone of the flock simulator boids was implemented in accordance with object-oriented design. Class inheritance, method and attribute overriding, and mathematical equations were utilized to achieve the resulting simulation.

6 Sources

https://en.wikipedia.org/wiki/Class_diagram

https://en.wikipedia.org/wiki/Boids

https://www.pygame.org/docs/

https://www.themathdoctors.org/averaging-angles/