Assignment 2 report - INF-1400

Sera Madeleine Elstad

March 10, 2022

1 Introduction

For the second assignment in INF-1400, the task is to create a clone of the classic flock simulator boids. The implementation will be written in python with a goal of using the principles of object-oriented programming like classes, methods, and inheritance. The simulation should simulate a moving flock consisting of boids operating in a lifelike manner and following a set of rules. The simulation shall also contain hoiks and obstacles. The rules the boids should follow are:

1. Boids steer towards the average position of local flockmates.
2. Boids attempt to avoid crashing into other boids.
3. Boids steer towards the average heading of local flockmates.

Some additional features must also be added, these are obstacles that the boids need to avoid and predators that will try to eat the boids.

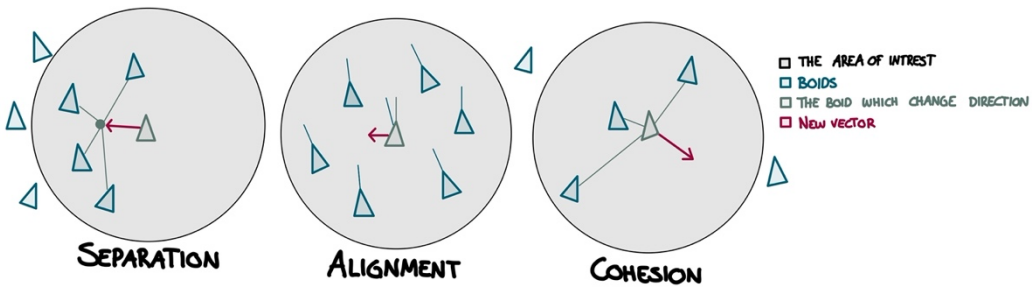
2 Technical Background

2.1 Boids algorithm

Craig Reynolds created Boids, an artificial life program, in 1986. The simulation's goal was to mimic the behavior of a flock of birds. The boids simulation specifies the behavior of each individual boid rather than controlling the interactions of an entire flock. This is accomplished through the application of a set of rules:

1. Alignment: steer in the direction of the average heading of local
2. Separation: steer away from local flockmates to avoid crashing.
3. Cohesion: steer to the average position of local flockmates.

The simulation can be enhanced with additional rules[1].



2.2 Inheritance

A programmer can use inheritance to create a class that inherits all of the methods and properties of another class. This is advantageous when it accurately represents a real-world relationship, allows for code reusability, and is transitive. The parent class is the class that the child class inherits, and the child class is the class that the parent class inherits. [2]

2.3 Sprites

Sprite is a module that contains some higher-level classes for dealing with moving objects on the screen. When playing a 2D or 3D game, sprites are the objects that appear on the screen. Sprites can be animated, controlled by the player, interacted with, and added to Sprite groups.

A Sprite group is a container class that holds and manages multiple Sprite objects at once. By using a Sprite group, the programmer can easily add, remove, draw, and update one or more sprites with a single command. This is advantageous because it shortens the code and makes it easier to read. [3]

3 Design and implementation

3.1 Moving objects

The parent class for this implementation is Moving Objects. This class contains some properties that are used in the children's class. Figure 1 depicts a UML diagram of inheritance between the parent class, moving objects, and the children classes, boids and hoiks.

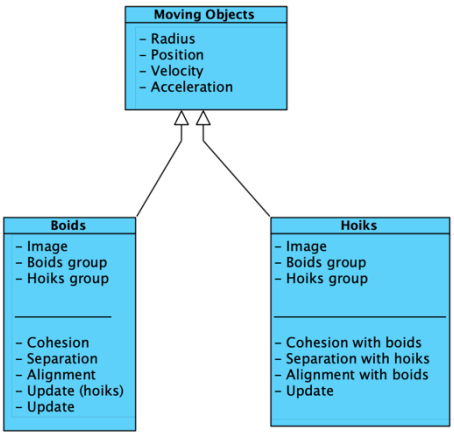


Figure 1, UML-diagram

3.2 Boids

The boids class consists of an image, the rules for the boids, an update for the hoiks, and an update for the boids themselves. All of the boids are added to a group, which is then added to another group containing all of the objects. Within the game loop, this group is drawn and updated.

3.2.1 Cohesion

According to the first rule, boids in a specific area will try to move towards the center of mass, which is the center of all neighboring boids. This is referred to as cohesion. Cohesion calculates the average position of all boids and refers to this as the perceived center. To find this center within a given radius, a loop is created that traverses all the boids in the boids group. The program then checks to see if the distance between the two boids, which are not the same boid, is within the radius. If this is the case, the vector for the perceived center is updated with the position of the given boid, as is the number of boids within the radius.

Because dividing by zero is "illegal," the program must ensure that there are some boids within the given radius. If there are any bois, the perceived center vector is divided by the number of boids contained within the given radius. The given boid's position is then subtracted from the perceived center vector. The length of the perceived center vector is then checked and scaled in order to manage velocity. Finally, the velocity of all boids in the given area is calculated by subtracting the velocity of the given boids from the perceived center vector.

3.2.2 Seperation

The second rule is separation, which states that the boids should keep a small distance between themselves and other objects. For the separation of boids, three loops are created: one to keep the boids apart, one to keep the boids apart from the hoiks, and one to keep boids apart from the obstacles. The main difference between these loops is that the first loop iterates through all boids and the second loop iterates through all hoiks. Iteratively, it calculates the distance between two boids or hoiks and determines whether the distance is within the given radius. The difference between the positions of the two objects is then calculated by subtracting the position of the first from the position of the second. The difference is then divided by the distance, and the vector is updated as a result.

The program then checks to see if there are any objects within the given radius, and if so, divides the vector by the number of objects and scales it. The velocity of the given object must be subtracted from the vector before it is returned.

3.2.3 Alignment

The third rule is alignment, which is supposed to make boids match the velocity of the boids around them. This is accomplished by iterating through all of the boids in the group to see if any are within the given radius; if so, the given vector is set to be the velocity of the given boid, and the number of boids is changed. If any boids are found within the given radius, the vector is divided by the number of boids and scaled. The vector is then subtracted from the velocity of the given boid.

3.3 Hoiks

The hoiks follow the same rules as the boids, but rather than iterating over the boids group, it iterates over the hoiks group. The number of hoiks that the programmer desires in the game is added in a loop, which is then added to another group that contains all of the game's objects. Within the game-loop, this group is both drawn and updated.

3.4 Obstacles

Obstacles includes a function for drawing obstacles. A loop is also created to create the desired number of obstacles and add them to the group of all obstacles. This group is added to another group containing all of the game objects, and this group is both drawn and updated within the game-loop.

3.5 Screen bounding

To prevent objects from disappearing from the screen a screen bounding is created and put in the update function for both the boids and hoiks. If the object goes outside the screen the coordinates of the object is moved to the opposite side of the screen. It is done like this to get a smoother movement.

4 Evaluation and discussion

A screen bounding is created and placed in the update function for both the boids and hoiks to prevent objects from disappearing from the screen. If an object moves outside the screen, its coordinates are moved to the opposite side of the screen. This is done to achieve a smoother movement.

5 Conclusion

The task for this assignment was to create a Python clone of the classic flock simulator boids using object-oriented programming principles. The simulation should depict a moving flock of boids that operate in a lifelike manner and adhere to a set of rules. Obstacles and hoiks will also be included in the simulation. The program contains all of the required elements, but there are some bugs and it does not accurately represent a flock of boids.

6 References

[1] Wikipedia, "Boids," 21 Febrary 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Emergence&action=history. [Accessed March 2022].

[2] ShivangiSrivastava1, «Inheritance in Python,» 22 February 2022. [Internett]. Available: https://www.geeksforgeeks.org/inheritance-in-python/. [Funnet March 2022].

[3] pygame, "pygame documentation," [Online]. Available: https://www.pygame.org/docs/ref/math.html. [Accessed 2022].