Assignment 2 report - INF-1400

Sigurd Uhre

March 11, 2021

1 Introduction

In this assignment, we make a simple simulator "or clone" of the classic flock simulator boids. The objective is to simulate a moving flock consisting of boids that operates by a set of rules. The main goal in this assignment is to make the boids behave in a lifelike manner with implementing the boids movement based on these three behavior rules:

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

Requirements
1. Implement the simulator in accordance with object-oriented design, using object and classes.
2. Inheritance must be used to implement at lest one class.
3. The simulator must follow the rules described above.
4. Hoiks and obstacles must be implemented.
5. The report must give a description of inheritance in object-oriented programming, and how you have chosen to use this feature.
6. The hand-in must include a class diagram (as shown in lectures) which describes relations between the different classes.

2 Background

For this assignment, Python 2.7 was used, which is an interpreted programming language, which supports the OOP principles. In addition, pygame library is used along with Visual studio code editor.

3 Design

Vector
To move the boids we have used the 2DVector from the pygame library. The direction and speed of the circles movement can be stored in a vector. This makes it easy for the direction and speed of a moving object to be changed. One way to think of a vector is the difference between two points. For every frame of animation, you instruct each object on the screen to move a certain number of pixles horizontally and a certain number of pixels vertically. We can say that for every frame: The new location is equal to velocity applied to current location. Nevertheless, another way to describe a location is the path taken from the origin to reach that location. Hence, a location can be the vector representing the difference between location and origin. [1]

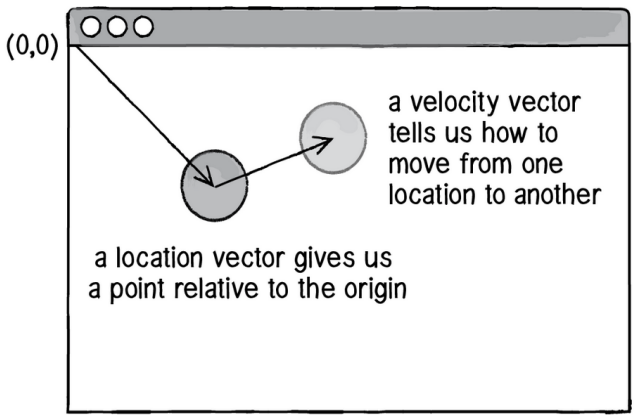

Figure 1: illustrates the two types of usage for vectors on screen for this assignment. [1]

Inheritance
We apply inheritance structure to the code in this assignment. Inheritance

is a well-known relationship in object-oriented programming. This relationship is sort of like a family tree where one class can inherit attributes and methods from another class. But different from a family tree they might also have their own attributes which are not inherited from any "parent" class. The attributes that we want to be specific for this class, is kept in this "child"-class only and no other class inherit these attributes. The attributes that we want to be general for different classes are kept in the parent class.

Inheritance is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation. [2] This way of organising the classes is beneficial for several reasons: Instead of writing the same attributes inside each class that needs the desired behavior, we save us from writing a lot of code by creating one parent class with these attributes, inheritance makes more sense the larger the code gets. Its easier to write and make changes along the way, as well as keeping the code organized. The code also take up less space and will most likely run faster. An illustration of the inheritance design is illustrated in figure 2.

Main
The different functions and iterations of lists are called upon in the main while loop. This design makes it easy to alternate different changes an adjustments in the loop, and keeps it structured and organized. If we want to debug the simulation, we can comment out different function to see what causes the bug in the first place.

4 Implementation

Vectors in code
The first an foremost implementation of the vectors are set in the parent class: bird. Originally we wanted both boids and hoiks to inherit these vector attributes, since we also wanted hoiks to have more speed than boids we used the super.() function to alter the velocity, making the hoiks faster than boids. We have used vector to give each boid and hoik a random starting position, speed and orientation on the screen within given parameters. Unlike the vectors mentioned above, the vectors in the alignment, separation and cohesion functions are reset.

Inheritance in code

The bird class is the parent class in in this code structure. This class holds several behavior functions that are desired for both boids and hoiks. By making the parent -class small with only specific necessary features it gets a larger area of use for different sub classes "child-classes". The "border" function is a god example for wide usage. This function tells the object to do "swing around" on the screen, which means that if the object leaves the screen on one side it will come around with the same direction and speed on the other side. When boids and hoiks inherit this function we don't need to specify it any further unless we want to.

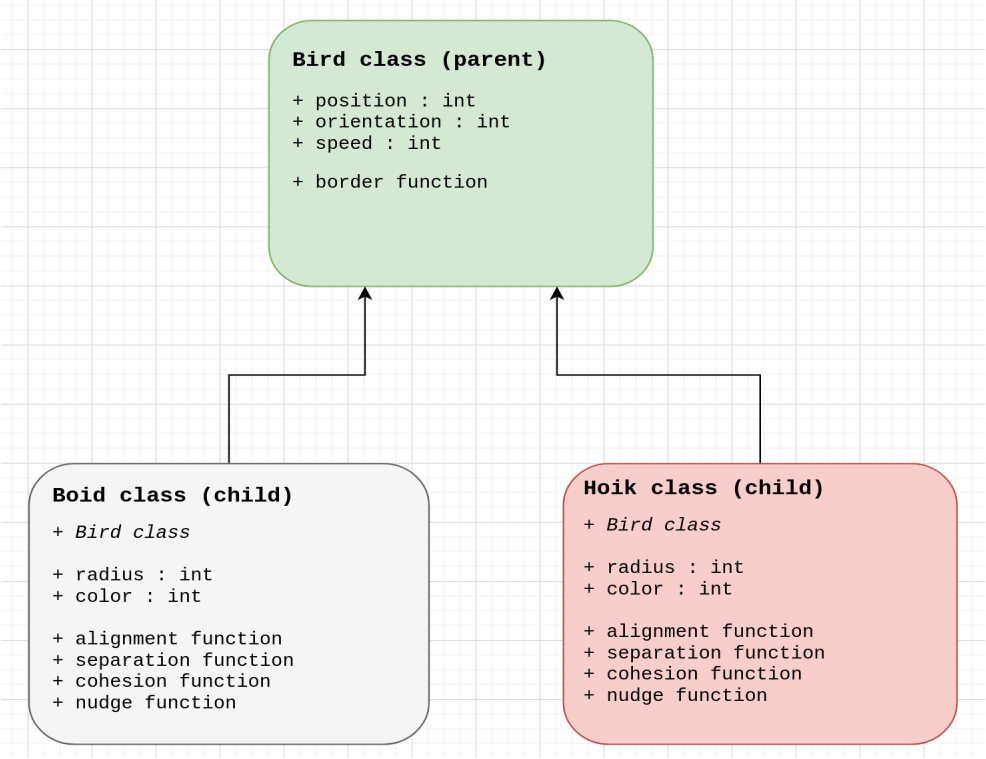

Figure 2: describes relation between the different classes.

Implementing the rules

Alignment, separation and cohesion are the rules that makes the boids behave in a flock like manner. We implemented a function for each rule and based them on the positional orientations and velocity given from the parent class. These three rules are dependent on one-another in the way that if one is to strong, the effects of the two others will be overrun. To achieve a nature-like flock movement we had to adjust the rules so that they where

balanced.

The nudge is an additional feature added to give the simulation more uninterpreted randomness in their movement. This function gives boids flock a new directional velocity, creating sudden separation and cohesion for different lists of boids. We use *lists* and iteration to gather the nearby boids in to a flock. The iteration goes trough all the boids to check if any other boids is within distanced to create a flock-list. In the same way, another iteration checks if the distance between the boids becomes to large and split up the list if the requirement is met.

5 Evaluation

In this solution, all requirements except number 4 are fulfilled. Hoiks have been implemented with unique behavior, but obstacles are not successfully implemented. They appear on the display as random positioned yellow circles with no movement. The function "avvoid obs" was unsuccessfully in making the circles an obstacle which the moving objects reflected upon. The function is written in the bird parent-class, it was meant to be a function that both boid- and hoik-class would inherit.

The requirement did not specify the behavior for hoiks. We have tried to implement some feature where the hoiks behavior is diffident compered to the boids. We had to make the alignment, separation and cohesion specific for hoik in the hoik class. This solution is sub optimal since we want to write one on each function so that it can be used for several classes, but we didn't succeed implementing it. Like the display show the hoiks also operate in a flock-like manner when close to each other, but they have higher speed and stronger nudges.

The other requirements in the assignment are met. We have implemented object-oriented design, using objects and classes. Inheritance have been used from the bird parent class to both boid- and hoik-class and the rules of alignment, separation and cohesion have been added into functions.

6 Discussion

Obviously, the implementation of the of the obstacles could be done better. This requirement was expected to be easier compared to some of the oth-

ers. Instead, it proved to one of the most challenging features to add to the simulation. The other disappointment in this assignment is that we didn't succeed in implementing the alignment-, separation- and cohesion -function in a way so that both hoik and boid could use it from the parent class. Since failing at this task, it meant we had to implement each function for each subclass making the code much larger.

We wanted to balance the different rules so that no rule had to strong of a "force" compered to the others. Several problems occurred when trying to solve this, but we experienced that "running one function at a time" is a good way to debug the code step by step. On the positive side, we successfully made the boids behave in a flock-like simulation with the use of the recommended tool: vectors.

7 Conclusion

I have implemented a solution that fulfills the requirements, except for parts of requirement 4. The implementation runs as expected and there is no problems with crashes or unexpected behavior. If given more time the focus would be on completing functional obstacles so that requirement number 4 would be fulfilled. The next focus would be on mplementing the alignment-, separation- and cohesion -function in a way so that both hoik and boid could use it from the parent class.

References:

1. The Nature of Code [Internet]. Natureofcode.com. 2021 [cited 10 March 2021]. Available from: https://natureofcode.com/book/chapter-1-vectors/

2. Inheritance (object-oriented programming) [Internet]. En.wikipedia.org. 2021 [cited 11 March 2021]. Available from: https://en.wikipedia.org/wiki/Inheritance$_(object-oriented_programming)$