Assignment 2 Boids

Håkon Silseth

UiT - Norges Arktiske Universitet

March 6, 2023

Technical background

In this assignment we are to make a boids-simulator, but what are boids?
In 1986 Craig Reynolds created a model for simulating flock behaviour of animals, such as a flock of birds or a school of fish. The model uses three basic principles to model the complex behaviour of a leaderless flock of so-called boids. In this way the complex herding behaviour of such a flock can be modelled, without hard coding the motion of the boids, they adjust their velocity according to their given 'rules'. The rules that the boids must obey are the following:

Separation

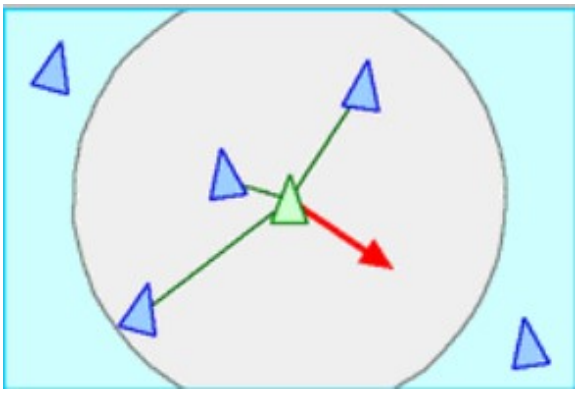

Figure 1: Separation (Reynolds)

Figure 1 shows the principle of separation. Boids will steer away from each other if they come to close. This is quite straight forward, the boids are given a threshold, and if another boid comes within that threshold the boids will steer away from each other.

Alignment

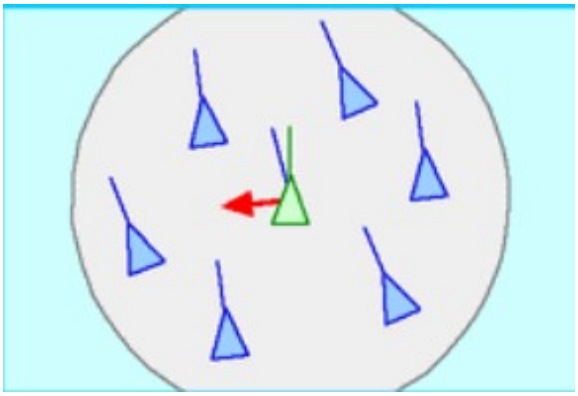

Figure 2: Alignment (Reynolds)

Figure 2 shows the principle of separation. Boids will steer towards the same direction as local boids. In other words all boids will steer towards the average heading of the local boids. Local here means the boids within a given limit.

Cohesion

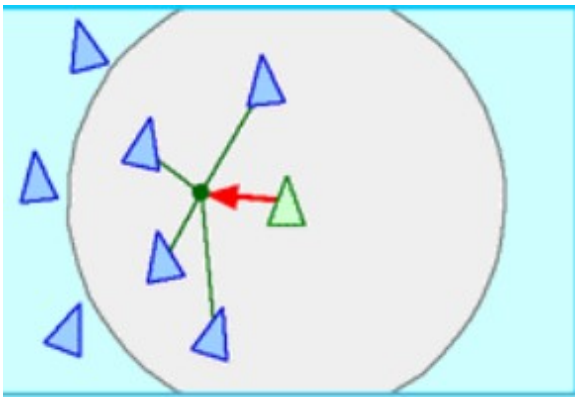

Figure 3: Cohesion (Reynolds)

Figure 3 shows the principle of cohesion. This principle is also very straight forward, each boid will steer towards the center of mass of local boids. In other words, each boid will try to come as close as possible to all other boids.

Taken separately these rules are not too complicated, and they don't result in such complex behaviour either. When we add all these together however we see complex flocking behaviour, the flock of boids move collectively. In addition to these rules one can also add more rules for the boids

to follow, as I have done in my code. These rules could be having hoiks that hunt the boids that the boids have to avoid. You can also add obstacles that the boids have to steer around, and bait that attracts them. And probably much more.

Inheritance

In this assignment I use inheritance in my code, but what exactly is inheritance. Inheritance in python is the process of creating a class that inherits properties and methods from a different class. Inheritance consists of a parent class, which is the class that is being inherited from, and a child class which inherits, or borrows, from the parent class. These can also be called base class and derived class respectively. Inheritance is used to avoid repetition in the code, if you have multiple classes that contain some of the same properties and/or methods it's a bit tedious to write these out for each class. Rather we can have one parent class which includes all the properties shared by the child classes. One example of how this might be done is if we have a system that keeps track of people associated with a university. All students need to have a name, age, which study program they are in, their courses, and maybe an ID-number. And the employees need a name, age and an ID-number, but not a study program or what courses they take, but they will need information about their employment status and salary. Here it makes sense to create a parent class called person with name, age and ID. We can then create two child classes named student and employee, where the unique properties are stored, study program and courses for student, and employee status and salary for employees. By structuring the code this way we avoid writing all the properties and methods that are shared between student and employee twice.

Sprites

In this assignment I utilize the sprite class that is built in to the pygame module (pygame.sprite.Sprite). This class is very helpful to use for creating visible objects on the screen. The class has built-in attributes for image and rect, which are particularly helpful, these basically makes it super easy to assign an image and a physical size to the sprite. It also makes it very easy to group objects of the same class together. In our case we can make a group of boids, a group of hoiks etc. By using sprites its also very easy to check if two sprites collide, which is helpful here when for example hoiks eat boids.

Code structure

The code is set up in the following way:

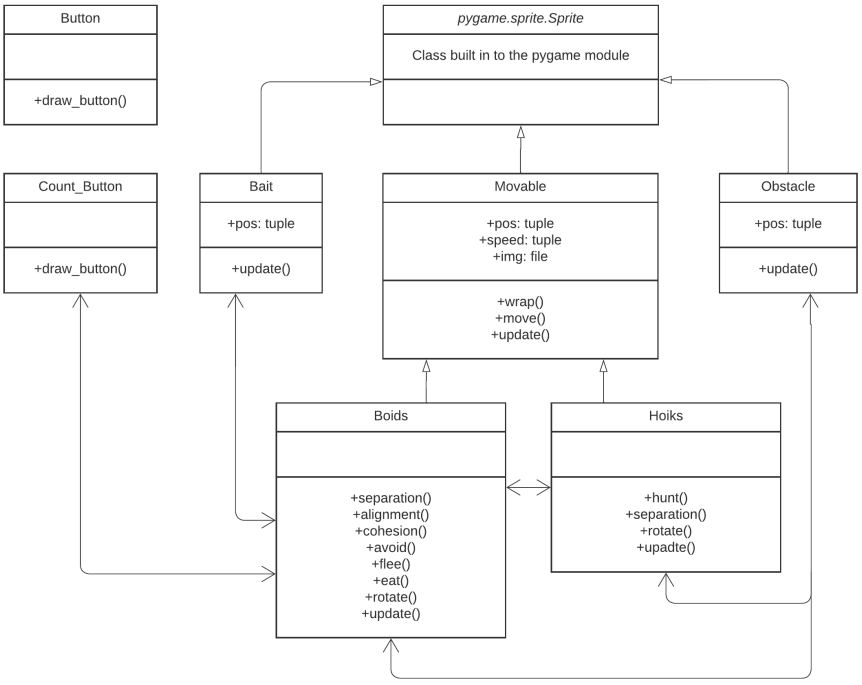

Figure 4: UML-diagram of the boid simulation code

I have used the 'sprite' class which is built in to the pygame module to simplify my code. The Movable, Bait and Obstacle classes all inherit from the sprite class. This is because the sprite class is very helpful for all objects that are drawn on screen and have a physical size in the simulation. So I have used the sprite class in the same way one might use a 'drawable' class.
The bait and obstacle classes don't really do much other than have a physical size and an image, it is the other classes that alter their behaviour when in close proximity to these.
The Movable class however has more to it. This is the class that lays the groundwork for the elements that move on screen, the boids and the hoiks. This class takes in the position and speed of the object as well as its image (might be more accurate to say sprite instead of image here, but I will use image to avoid confusion with the sprite class). Movable has the methods move, wrap and update, wrap just takes care of the situation where an object flies off screen. Instead of disappearing wrap makes it so that they appear on the other side of the screen. Move just updates the position based on the objects speed, and update calls on the move method.
The class Boids defines the behaviour of the boid, and I have done it in the following way:

- Separation: This method checks if another boid is within a given minimum distance to it. If there is a boid that is to close this method calculates the distance between itself and the boid that is to close and returns a velocity vector that is the opposite of the distance, i.e. the distance vector times -1.

- Alignment: This method checks if there are other boids in the neighbourhood of the boid. If

there are boids in the neighbourhood it calculates the sum of velocity vectors of all boids in the neighbourhood, excluding itself, and divides it by the number of boids in the neighbourhood. One can think of this as the center of mass, but for velocity instead of position.

- Cohesion: This method checks for boids in the neighbourhood, if there are boids in the neighbourhood the method calculates the center of mass of the neighbouring boids. It then calculates a velocity vector in the direction of this center of mass.
- Avoid: This method utilizes the exact same technique as the cohesion method, only it checks for obstacles in the neighbourhood instead of boids, and returns a velocity vector in the opposite direction of the obstacle.
- Flee: This method is identical to the avoid method, only it searches for hoiks in the neighbourhood.
- Eat: This method searches for bait in the entire screen, if there is bait on the screen then this method calculates a velocity vector towards the bait.
- Rotate: This method does not affect the behaviour of the boid, it only keeps it 'nose' pointed in the direction of motion.
- Update: This method adds all the velocity vectors up and then scales this sum to the boids velocity. This method also weighs the different velocity vectors differently according to the variables given in the 'variables.py' file. Also in this method is updating the position according to the velocity, and it removes bait from the screen if a boid 'eats' it.

The Hoik class has the methods Hunt, separation, rotate and update. Hunt works in the exact same way as the eat-method for boids, separation is also the same as for the boids. And so is update and rotate.
The Button and Count button classes wont be explained in detail here as I have not written them myself. Their functionality is to adjust the behaviour of the boids during the simulation. They allow the user to change the weighting of cohesion, separation and alignment. They also display helpful information, like the keys you can press to add objects and the number of boids. I would like to emphasise that I did NOT write the code for these buttons myself, I used code from `https://github.com/russs123/pygame_button/blob/main/button.py` written by Russs123, and modified it slightly. I also want to note that I used the psuedocode written by Conrad Parker at: `http://www.kfish.org/boids/pseudocode.html` as a starting point to base my code on.

Areas for improvement

My code is not perfect and I have myself found a few areas where it can be improved. The rotate method can be found in both the hoik class and the boid class, even though this is a property all moving objects should have. I however met many errors when trying to put this method in Movable, therefore I have them separately in the boid and hoik classes. Ideally rotate should be in Movable. There are probably a number of different changes I should make, but not that I have found myself, I am sure the code can be written more concisely and be less intensive on the computer.

References

Craig Reynolds. Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model), July 2007. [Online; accessed 16. Feb. 2023].

Conrad Parker. Boids Pseudocode, February 2023. [Online; accessed 16. Feb. 2023].

usss123. pygame button, February 2023. [Online; accessed 16. Feb. 2023].