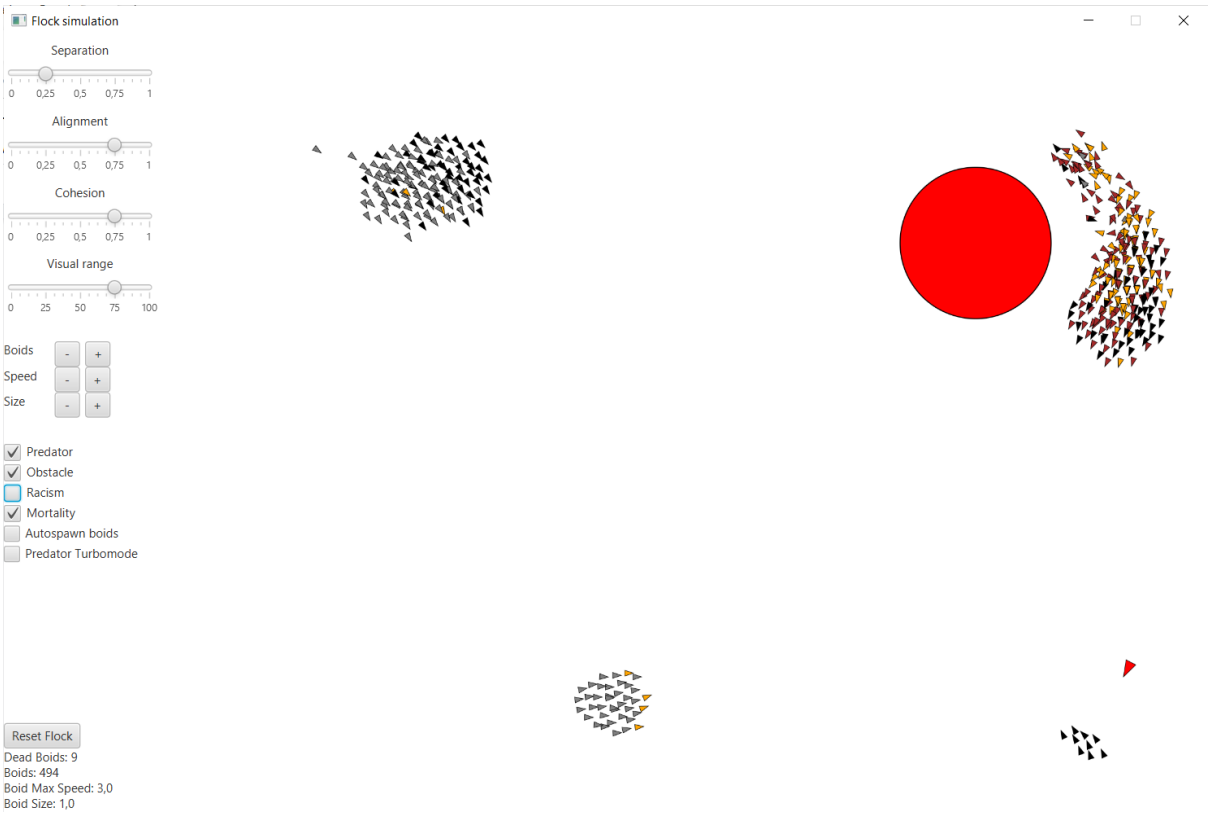
Flock simulation



Ville Prami
866011
SCI-2021
13.04.2022

Table of Contents

General description

The program is a flock simulation where a number of birds move in the flock. The program visualizes bird movement graphically.

The bird movements are described in a 2D world by depicting the movement of birds from above. Bird flocks are simulated by giving each bird three simple rules to choose its direction of travel. When there are several birds and they work according to the same internal rules, their cooperation resembles that of real birds.

The following three main rules are used to control the movement of the birds:

- Avoiding collisions to other birds (Separation).
- Flying as fast as the rest of the flock on average (Alignment). Using a vector with speed and direction.
- Aiming for the center of the flock (Cohesion).

To achieve natural behavior, the above rules are applied in the order of importance shown above.

The project was possible to implement on an intermediate-difficult level which included the following requirements:

- Graphical user interface
- In the beginning every member moves in a random direction. Read the initial state from a file, define the format yourself
- The possibility to change the weights of the rules (by default use ones with the most natural results)
- Other functionality such as different starting conditions, live parameter control etc

The project was completed on the difficult level since all the above requirements have been met and in addition the following features were implemented:

- Predator that hunts birds.
- Obstacle that birds can collide with.
- Live parameter control for the weights of main rules and for the amount, speed, and size of the birds.
- Allowing boids to die from being eaten by the predator or colliding with the obstacle
- Turbo mode that makes the predator faster.
- Auto spawning of birds so that when one dies, one spawns
- Applying edges for birds to keep them bound on the screen and in case one escapes it will wrap around edges.
- Seek behavior for a predator to hunt birds.
- Flee behavior for boids to avoid the predator.
- Bird behavior for avoiding the obstacle.
- Racism behavior for birds (Prefering to flock amongst their own color).

User interface

The project was created and tested in IntelliJ IDEA. To run the Scala application in IntelliJ import the project normally, open it in the editor and execute the application. The project has been tested also on Visual Studio Code.

The user interface is a graphical user interface using ScalaFX. The GUI has a sidebar to handle all inputs from the user.

The sidebar has the following controls/features:

- Change the weighting of separation behavior (Slidebar)
- Change the weighting of alignment behavior (Slidebar)
- Change the weighting of cohesion behavior (Slidebar)
- Change the visual range of boids and the predator (Slidebar)
- Add or remove boids (Buttons)
- Decrease or increase the maximum speed of boids and the predator (Buttons)
- Decrease or increase the size of boids and the predator (Buttons)
- Toggle predator for simulation (Checkbox)
- Toggle obstacle for simulation (Checkbox)
- Toggle racism behavior (Checkbox)
- Toggle to allow boids to die (Checkbox)
- Toggle auto spawning of boids (Checkbox)
- Toggle turbo mode for predator (Checkbox)
- Reset the flock (Button)
- Information about current boids amount, maximum speed, size and amount of dead boids.

Next to the sidebar and the rest of the interface is the actual simulation that runs with the parameters based on the user inputs. On starting of the program the simulation starts immediately on the default settings.

Program structure

Note: A link to a larger version can be found in the references.

The UML diagram presents the final class structure used in the project. The green highlighted parts show all the classes, methods and parameters that were added to the UML diagram presented in the planning of the project.

The *Main* class mostly has GUI components like sliders, labels, buttons and checkboxes. The *Behavior* trait got some new methods and some changes to existing ones. Three new classes were added: *Ticker*, *Predator* and *Obstacle*. *Flock* class got more methods but mostly stayed the same. *Simulation* class went through a massive expansion and loads of new methods and parameters were added during the project implementation.

All of these changes were added on the fly as was needed and when new ideas came to mind. Although many new methods and parameters were added the structure still resembles the planning phase structure.

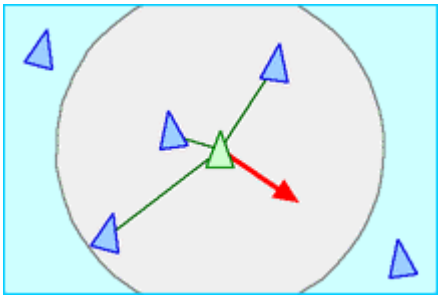
Behavior trait class is responsible for calculations of the velocity and direction of all moving objects. This class has all the rules that direct the boids and the predator. The *Main* class

has all the GUI parameters and methods so it alone describes the user interface. It also contains an Animate-method that gets passed to the *Ticker* to update the simulation. The class *Simulation* brings everything together and creates the simulation by updating boids, predator and obstacle and various other features based on the settings set by the user. *Boid*, *Object* and *Predator* represent a single item of themselves. *Flock* class represents a flock of boids and has methods to create and delete boids.

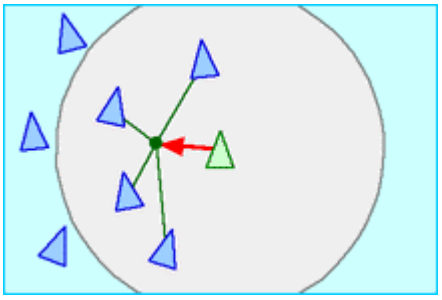
Overall the planned structure was used and expanded to the needs of the project. I got familiar with it and it became easier to add and change things to it. The final structure worked well.

Algorithms

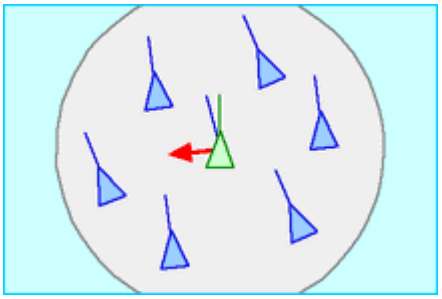
The simulation is based on three main rules that work together to mimic flock behavior in nature. These rules or steering behaviors are *separation, cohesion* and *alignment.* These behaviors determine how a character reacts to other characters in its local neighborhood. These rules are carried out as explained in an article [Steering Behaviours For Autonomous Characters] by Craig W. Reynolds.



Separation is the ability to maintain a certain separation distance from others. It is calculated by subtracting the positions of a character from a nearby character, normalizing and then applying a weighting. This is done for each nearby character and then summed together to produce the overall force.



Cohesion is the ability to approach and form a group with other nearby characters. It's calculated by computing the average position of all nearby characters and subtracting the characters position from it and applying a weighting to it.

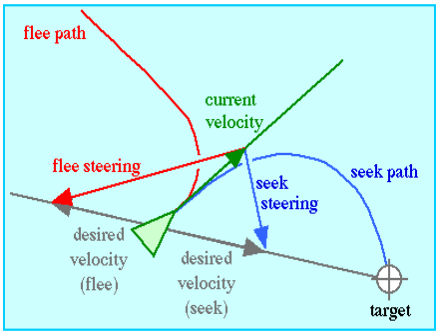


Alignment is the ability to head in the same direction as other nearby characters. This can be calculated by averaging together the velocity of nearby characters and subtracting the characters current velocity from it and then applying a weighting.

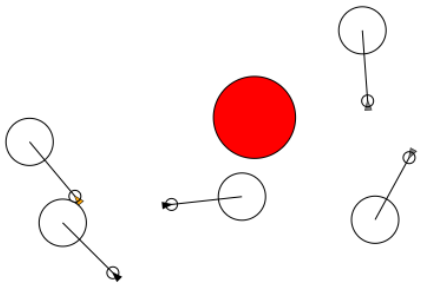
Flocking behavior is a combination of these three forces. For better control the three components will be normalized, scaled by weighing factors and then summed together.

In addition to the three main rules the following important algorithms were used:

To make the boids and the predator stay inside the window I applied margins from the top, bottom and sides of the window. If their position goes over these margins they are turned back by a certain factor. In testing some of the boids and the predator sometimes flew over the margins and the screen and never came back. I could not figure out why so I implemented a failsafe for them to wrap around the screen incase that happens.



Predator's seek behavior. Seek behavior is achieved by simply subtracting the predator's current position from the target's position and adding that to the predator's current velocity. Target here being the closest boid to the predator in a certain radius.
Avoiding predator or fleeing behavior. This behavior is almost the opposite of seek behavior. If the predator is in a certain radius from a boid, we steer the boid away from it. If the distance to the center of the predator is less than the sum of the boids collision sphere radius and predator's collision sphere radius there is a collision. If the mortality option is checked the boid dies. Radius of the collision spheres are the masses of the agents.



Avoiding obstacles. To avoid obstacles the boids utilize collision spheres. The boid has a collision sphere around itself and a future collision sphere which is bigger. The culture collision sphere is predicted by normalizing and multiplying the boids current velocity and that is added to the current position and then a circle is drawn around that center point. If the distance to the obstacle center is less than the sum of the obstacles radius and boids current locations spheres radius there is a collision. If the mortality option is checked the boid dies on collision. Otherwise the boid bounces from the obstacle by adding the halved subtraction between the boids current position and the obstacles center to the boids current velocity. The future collision happens if the distance from the obstacles center is less than the sum of the obstacles radius and the future collision spheres radius. In this case depending from which direction the boid is coming from we steer the boid away from the obstacle by a certain turning factor.

To update, the boids separation, alignment and cohesion rules are applied in that order. If the predator is present avoiding predator behavior is applied and if the obstacle is present also the obstacle avoidance rule is applied. Next speed limits and edges are applied. If the mortality option is checked then a check for boids alive status is applied and in case the boid is dead it is removed from the simulation and if the autospawn option is checked a new boid

with random position and velocity is spawned to replace the dead boid. Lastly all of these updates to the velocity of the boid are added to the boids current location.

For updating the predator a seeking behavior is applied with limiting the speed to be slower than the boid. If the predator turbo mode is checked the predator's visual range and speed are increased drastically. Lastly edges are applied and the updated velocity is added to the predator's position. Predator does not collide with the obstacle.

Creating new boids happen recursively by choosing a random position in the screen, random velocity and a random color from a vector with some colors. If the boid's randomly generated position is inside or inside the collision range of an existing obstacle it tries to create a new boid with new values.

Drawing the boids and the predator happens by drawing a triangle using line drawing rotated in the direction of the velocity. Obstacle is drawn as an oval.

Data structures

I based my data structures on familiarity from what I have learned from previous programming courses. The program needs to handle many *Boids* at the same time so to store them I'm using *Arrays*. For some constants for example a list of *Colors* I used *Vectors*. For vectors in the simulation I will be using ScalaFX *Point2D*. It offers multiple useful tools to use in the algorithms. To store the drawn contents ScalaFX *graphicsContxt2D* were used.

Files and Internet access

Aside from scala classes the program doesn't have any files and does not need internet access.

Testing

System testing will happen through the user interface. Through the UI it is easy to see if the flock behavior looks natural. In testing I will look at each steering behavior individually to see if they are working correctly. I will also inspect them together with different weightings to see if there are any bugs. There shouldn't be any incorrect inputs from the user, since the goal is to use sliders and buttons, so there won't be any user input outside of the parameters given to the user.

Testing mainly happened through printing information while the simulation was running. For example, testing boid collision with an obstacle I printed "Collision" on expected

collision and when I saw some boids that flew through the obstacle without printing collision there was some fixing to do. Another problem I had with the obstacle was its location on the canvas. I drew some lines from the coordinates of the created oval and noticed that the graphicsContext strokeOval method created the oval from the left corner coordinates and not the center which I wasn't aware of. So all of the testing happened through trial and error by printing out information and checking visually the behavior from the GUI.

Known bugs and missing features

One known bug is the predator's behavior with a small amount of boids at high speed. Sometimes the predator fixates on one boid and rapidly spins around it. It happens very rarely and I could not figure out why it happens.

There are no missing features in the program.

3 best sides and 3 weaknesses

Best sides:

- Boids collide with the obstacle and bounce from it. The boids were originally meant to all avoid the obstacle, but when I saw that at high speeds some of the boids collided with the obstacle I decided to leave it that way since it was more realistic. It inspired me to add the ability for boids to die on collision and it became a feature in the simulation.
- The simulation has more features than just the 3 main rules for flocking. These additional rules may not be accurate in nature but they make the simulation more interesting.
- The ability to add new behaviors easily and apply them to the boids if wanted/needed.

Weaknesses:

- Resizing the window. For the program it is not possible to resize the window while it's running since it would change the fixed locations of nodes. There is an option at the start of the Main class to launch the program in the maximized screen by changing the value of “maximized” to true but this has not been fully tested and might not work on all devices.
- The simulation is not very efficient with large amounts of boids, the more boids there are the more it will slow down.
- The predator and boids have same behavioral traits like applying edges, limiting speed and the method to draw them, but they are not shared. Instead they have their own methods for them thus code recycling was not well implemented.

Deviations from the plan, realized process and schedule

Here is the planned schedule from the project planning phase:

Phase	To do	Time estimate (in hours)
07/02 - 20/02	PLAN	10
21/02 - 06/03	GUI	15
07/03 - 20/03	BEHAVIOR	20
21/03 - 03/04	SIMULATION	15
04/04 - 17/04	EXTRAS	15
18/04 - 27/04	POLISH, DEADLINE	15

And here is the actual schedule:

Phase	Completed task	Time spent (in hours)
16/03 - 30/03	Added classes and made the basics work: Main Ticker Boid Flock Simulation Behavior	10
31/03 - 13/04	Added classes: Obstacle Predator and made all the project minimum requirements work and added some extra features	30
14/04 - 27/04	Added extra features, cleaned up the code and finished it, wrote the documentation	20

The biggest difference was time management, at the start of the project I did not have as much time as I had planned for it so I had to adjust and do it more later. The GUI was pretty much created after getting most of the rules to work on a testing screen which was just a white screen. If I could do it again I would do it as in the planning phase and create the GUI before anything else.

Final evaluation

I enjoyed making and seeing the program come together. I learned a lot about scalaFX and interaction between the user and the program. I managed to implement the required functionality of the program and some extra features too. I'm happy with the quality of it. It is not perfect and a lot could be improved but for the purpose of the course it is good.

The more significant shortcomings are the 3 bad sides of the program explained earlier; resizing the window, efficiency and repeated code. The program could be improved by fixing these. I also had some ideas for extra features that I decided was not worth the time that could be added in the future:

- Weather like wind, rain, lighting etc.
- Wonder behavior
- Multiple obstacles and predators
- Perching
- Path following
- Insects for boids to eat
- User controlled boid that other boids would avoid or follow
- User controlled character that could shoot boids

If I would do the program again from the beginning I would follow the plan more carefully, like starting from the GUI and I would recycle some methods in the classes better.

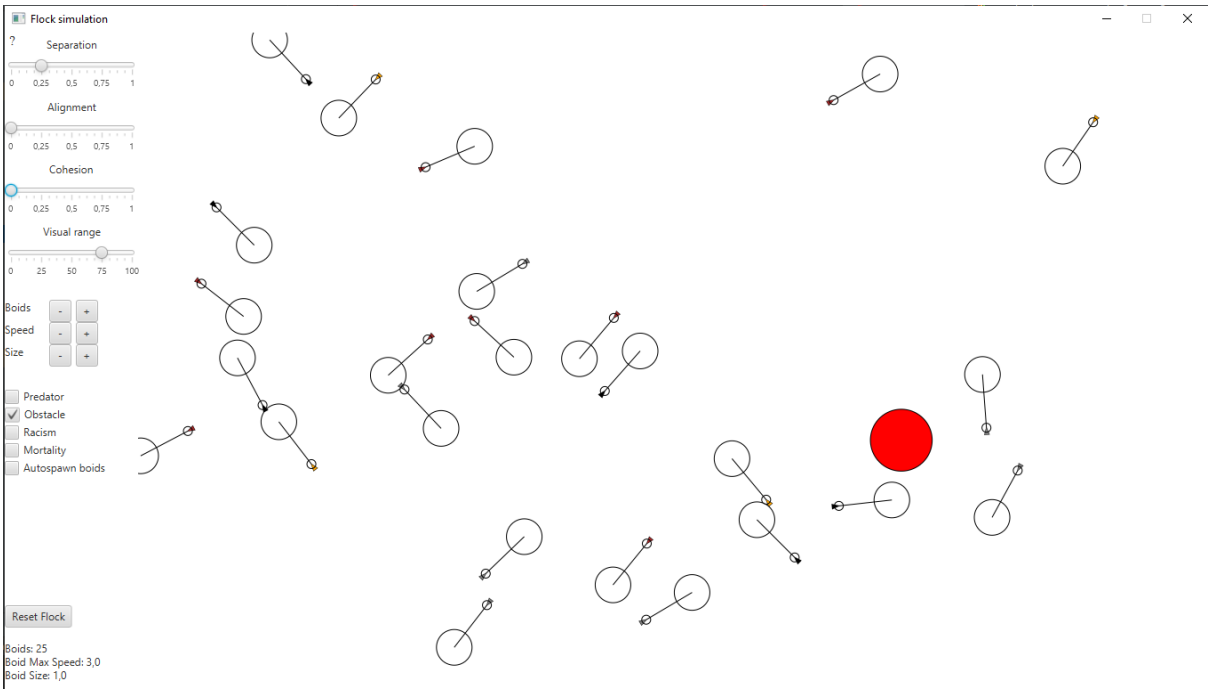
References

Here I have gathered some links for material that I have used to aid my project. My favourite of them all is the YouTube video “Coding Adventure: Boids” by Sebastian Lague. I have seen this video prior to the course and I picked my project subject based on it. “The Nature of Code” is an online book with a lot of information on the physics of the boids.

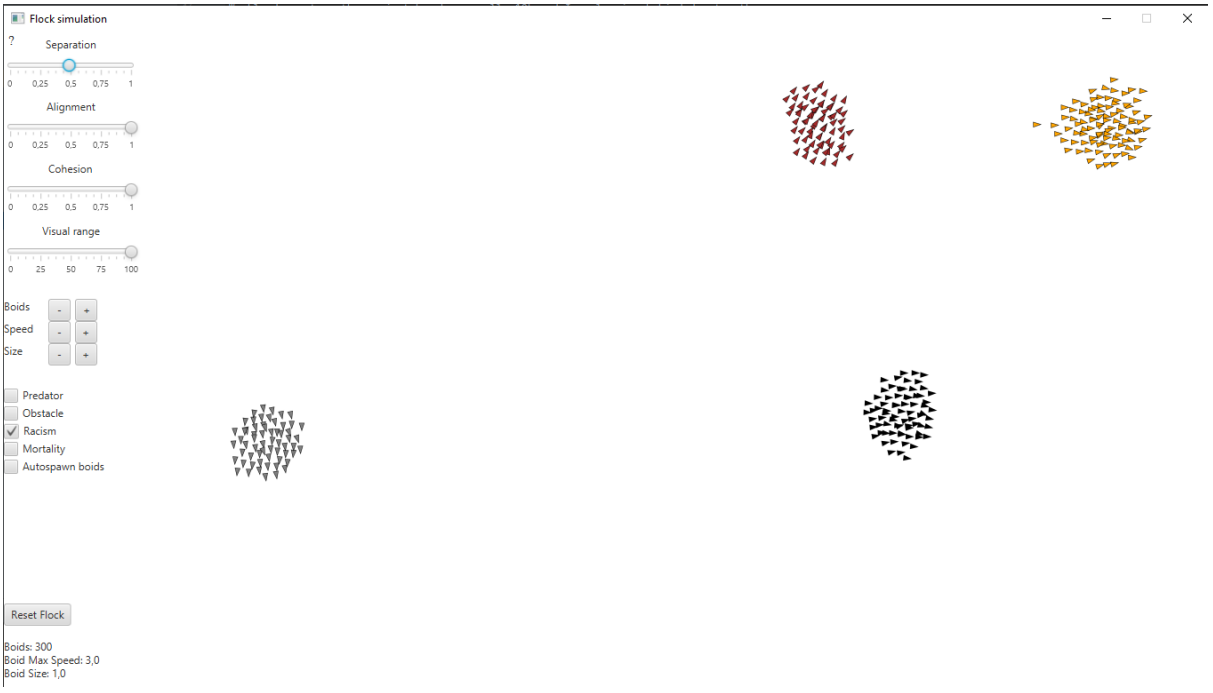
Uml-link

Coding Adventure: Boids - YouTube
Coding Challenge #124: Flocking Simulation - YouTube
The Nature of Code
Properties (scalafx.org)
scalafx/scalafx/src/main/scala/scalafx/scene at master · scalafx/scalafx · GitHub
Steering Behaviors For Autonomous Characters (red3d.com)
Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model) (red3d.com)
Performance Characteristics | Collections (Scala 2.8 - 2.12) | Scala Documentation (scala-lang.org)
Larger application: Asteroids - Java Programming (mooc.fi)

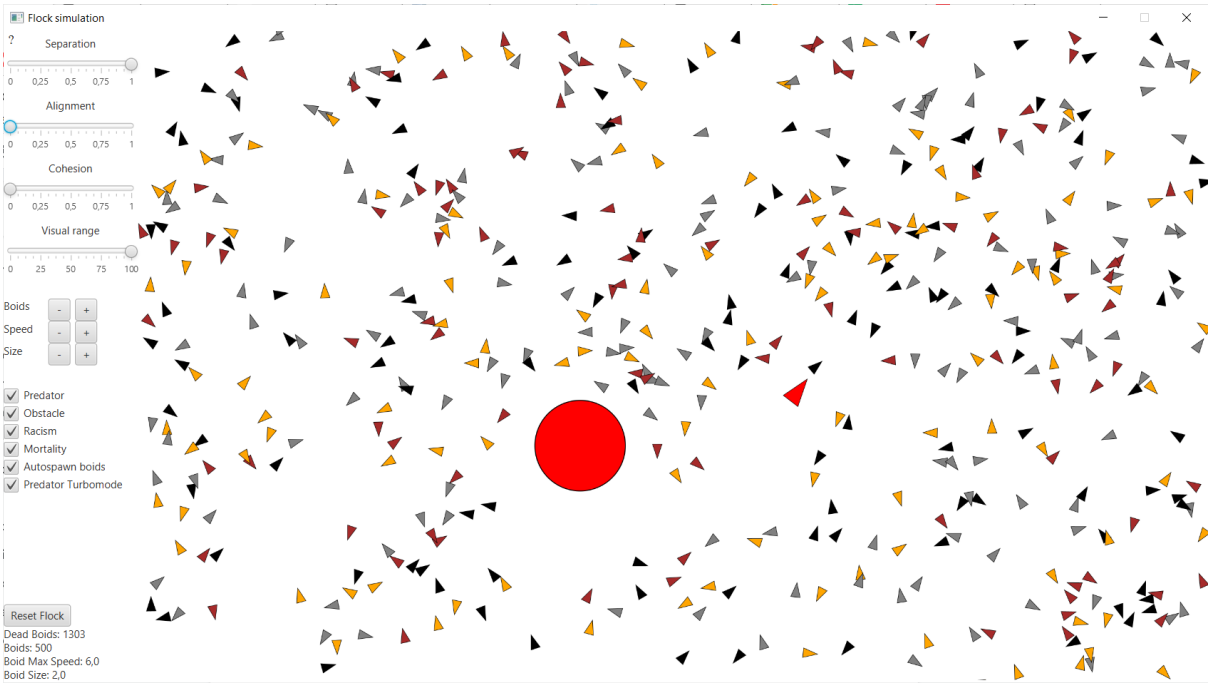
Appendixes



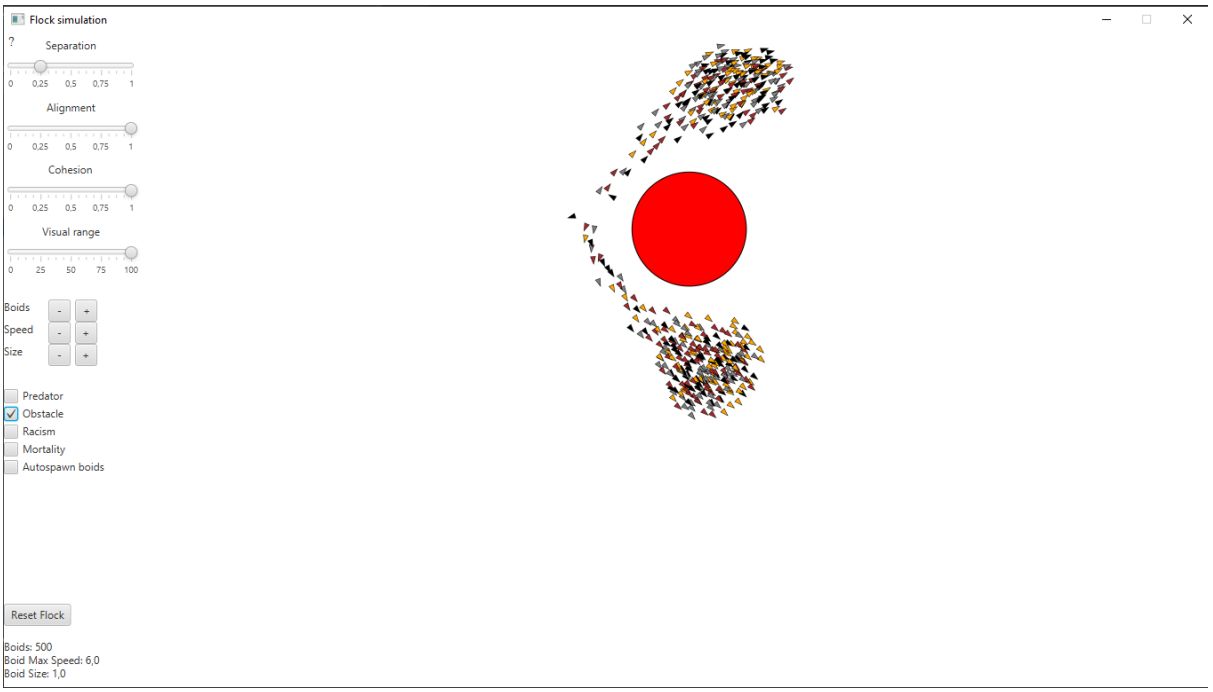
Boids with their collision spheres visible.



Boids separated to their own colors.



Giant mess and rapid boid death rate



Obstacle splitting a flock