

1. Personal information

Aleksi Alatalo 890155 Data Science 1st year 12.4.2024

2. General description

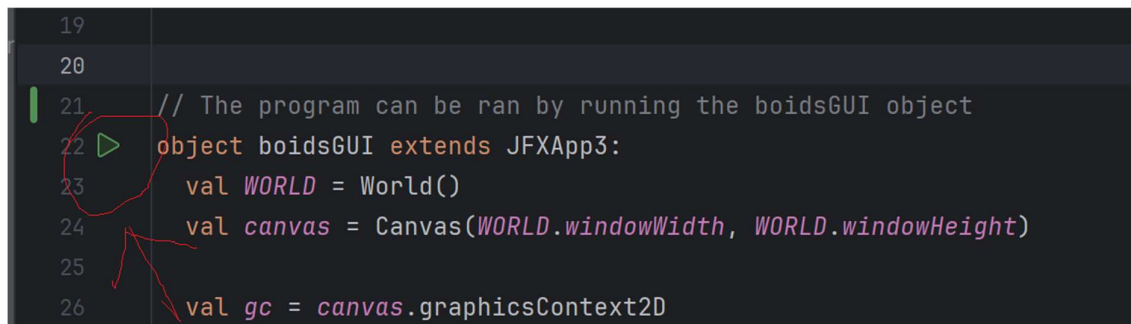
The general plan was to implement boids (bird-oid objects) in Scala. Boids move individually according to instructions, which create emergent behavior in them to cause them to move like flocks of birds or schools of fish.

The program also tries to simulate evolution. Boids try to collect food, which causes them to reproduce with a chance of mutations. To enact as a counterbalance to reproduction, I also implemented predators (shrimps), which seek boids and reproduce upon reaching one.

I believe the project was completed on the 'difficult'-level as I implemented all the requirements as well as live parameter control and predator-prey behavior.

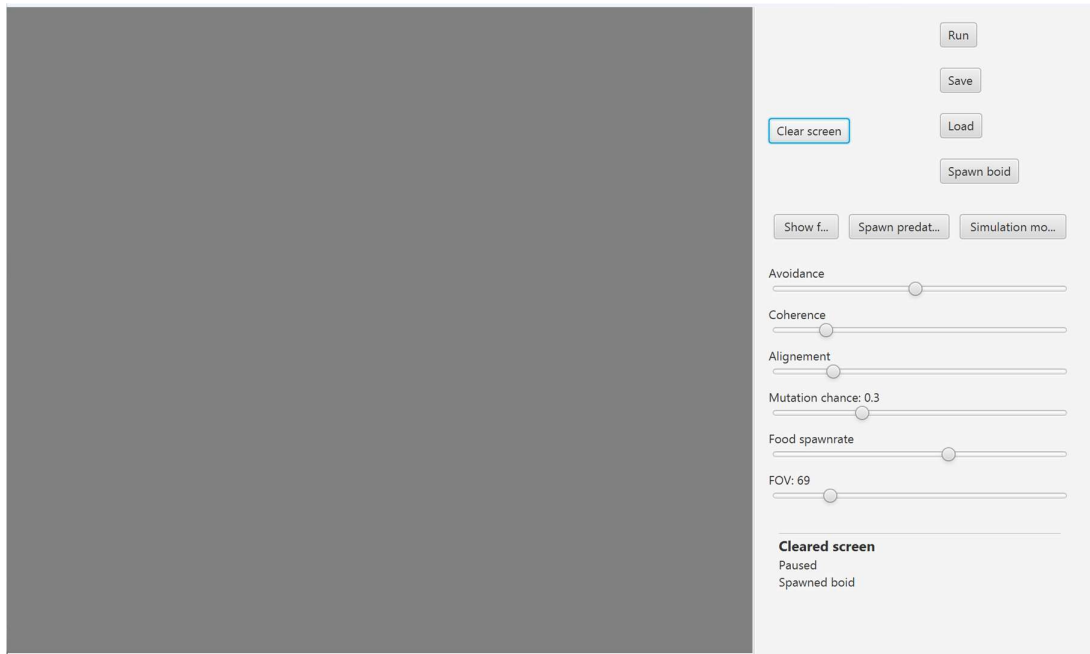
3. User interface

The program can be started by running the boidsGUI object from GUI.scala

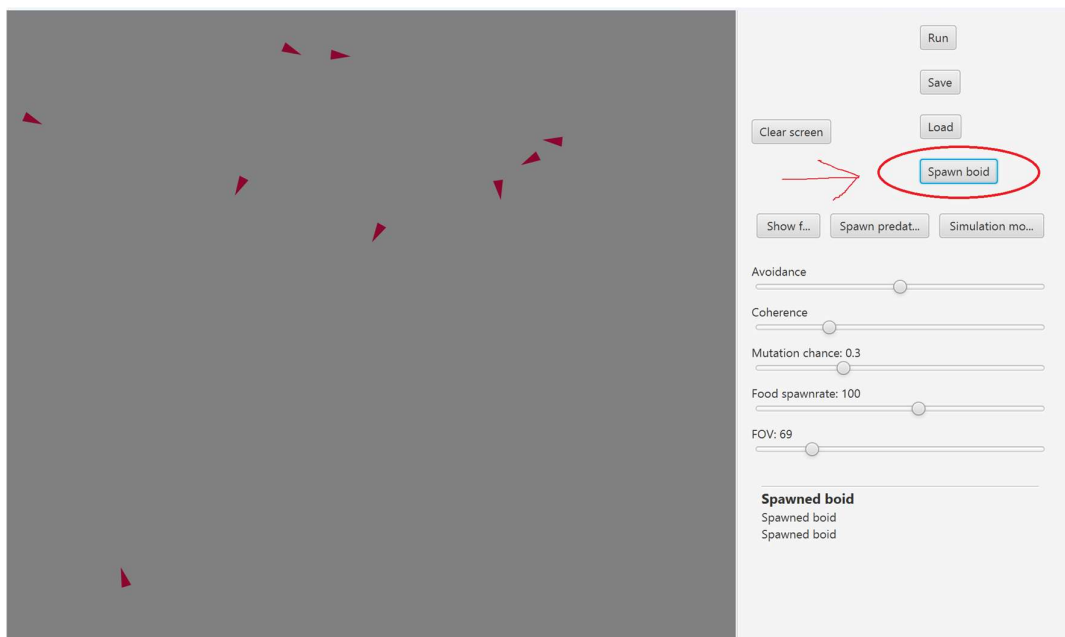


```
19
20
21 // The program can be ran by running the boidsGUI object
22 object boidsGUI extends JFXApp3:
23   val WORLD = World()
24   val canvas = Canvas(WORLD.windowWidth, WORLD.windowHeight)
25
26   val gc = canvas.graphicsContext2D
```

Upon launching the user is greeted with a screen like this. From here they have two options to start using the program: either loading a file or spawning boids manually.



Boids can be spawned from the “Spawn boid” button.



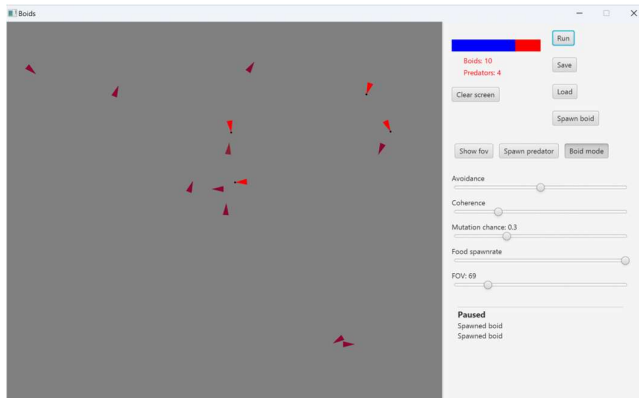
The program starts paused and can be ran from the “Run” button.

Parameters for the boids on screen can be changed from the Avoidance, Coherence and FOV sliders. These change the boids behavior as well as color. The avoidance sliders affect how much the boids avoid each other and the coherence slider affects how much an individual boid seeks the center of the boids visible to it. The FOV slider affects the degree of vision a boid has.

The program also starts in “Boid mode”, meaning that predator/prey functionalities are disabled. The spawn predator, mutation, and food spawn rate sliders do not do anything when Boid mode is enabled.

When “Simulation mode” is enabled, predators can be spawned and the food spawnrate slider affects the frequency of food spawning.

The world can be saved or loaded from the corresponding buttons and emptied from the clear screen.



When simulation mode is enabled, a graph showing the ratio of boids to predators is also shown.

4. Program structure

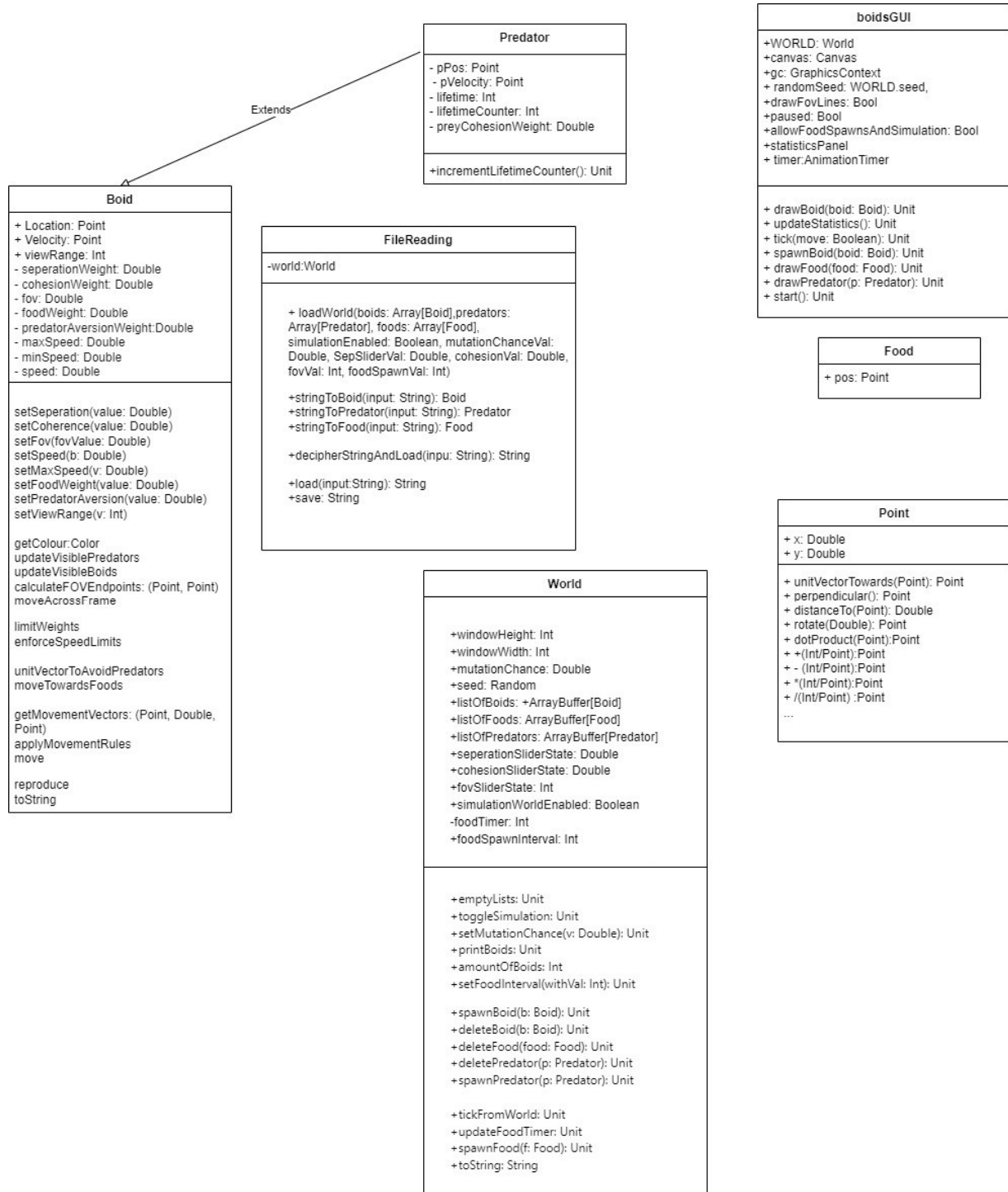
BoidsGUI is the main class that serves as the entry point for the application. It manages the graphical user interface and controls the animation loop, updating and redrawing the program state accordingly.

The World-class represents the simulation environment, meaning the objects inside it, variables for spawning objects as well as spawning and deleting them. The world class combined with boidsGUI and FileReading handle saving and loading files.

The food class contains just a Point where a food is to be drawn. The Point-class contains coordinates and operations that can be performed with them.

The Boid-class represents an individual boid. It defines it's movement rules and parameters and provides functions to move according to them. The class is extended by the Predator-class, which represents the predators inside the program. They work mostly like Boids but seek them instead of food. Boids, foods and predators make up the logic behind the program.

I would have liked to plan the class structure a bit more, since in the current state Predator extending Boid implemented quite poorly (but it works!). Since all of the three drawable objects contain positions, they could have been extensions of an object class containing a Point.



5. Algorithms

The flocking behaviour of boids consists of finding which boids are visible and figuring out how they affect the boid.

A boid has a position (a point) and a velocity vector (also a point). It moves by taking a unit vector towards the velocity and making its new position its current position plus the unit vector multiplied by its speed.

To figure out the boids visible to a boid, each boid iterates over the list of all the boids in the world to check if they are within sight:

```

visibleBoids=List
for each boid:
if boidIsWithinRange:
    if angleToBoid > viewRange:
        add each to visibleBoids

```

Based on the list of visible boids, each boid calculates a new velocity vector. The boid has two main forces that drive it: separation and cohesion. (Other boid algorithms have a third rule which sets the heading of a boid equal to the average of the group but this is included in cohesion.)

Cohesion is calculated by iterating over the list of visible boids and calculating their average position. Then a unit vector from the current boids position to the average is returned. Same is done for alignment but taking the average of all boids direction vectors.

Separation on the other hand is calculated by iterating again but subtracting the position of each visible boid from the current boids position and returning a unit vector towards the result.

A boid also sets its speed to the average speed of visible boids.

```

def getMovementVectors: (Point, Double, Point, Point) =
    var pointForCohesion = pos
    var pointForSeparation: Point = pos
    var pointForAlignment: Point = dest

    for each <- visibleBoids do
        pointForCohesion = pointForCohesion + (pos.unitVectorTowards(each.pos))
        pointForAlignment = pointForAlignment + each.dest
        speedSum += each.speed
        if pos.distanceTo(velocity) > separationActivationDistance then

pointForSeparation = pointForSeparation
    (pos.unitVectorTowards(each)) .* (pos.distanceTo(each))
    val separation = pos.unitVectorTowards(pointForSeparation)
    val newSpeed = speedSum / (amountOfBoids + 1)
    val cohesion = pos.unitVectorTowards(pointForCohesion)
    val alignment = pos.unitVectorTowards(pointForAlignment / amountOfVisibleBoids + 1)

    (separation, newSpeed, cohesion, alignment)

```

The returned unit vectors are scaled by their corresponding weights which are simply a value in [0,2]. The results are added to the current velocity vector to get the new velocity vector. The boid then moves based on this new velocity vector.

Boids also have additional vectors calculated to avoid predators and seek food. They are acquired from iterating the list of visible foods or predators and either subtracting or adding a unit vector towards them to the velocity vector to avoid and seek respectively. These also have their own weights.

Predators function much like boids, replacing the flocking behaviour only with avoiding other predators and food seeking with seeking boids.

The world has an evolution chance in the range [0,1] which the user can adjust. Whenever a reproduction event occurs, the chance for mutations is the one set by the user.

If a mutation is deemed to occur, there is then again, a 0.5 chance for each parameter of a boid or predator to change.

6. Data structures

The heavy lifting data wise in the program is done by ArrayBuffers. They contain all the objects to be drawn in the program and are iterated continuously when the program runs.

I created a class called Point, which consists of x and y-coordinates as well as common operations for vectors such as adding, multiplication, taking a unit vector and so on.

The locations of boids, predators and foods are stored in the objects as points. The same is true for the velocity vectors of boids and predators. Arrays were my first consideration for storing elements and as it worked, I stuck with it.

The weights of boids and predators are stored in the objects as variables.

The project leans more towards object oriented programming, thus most of the data structures are mutable, but it has some features resembling other paradigms as well. The arrays containing a world's objects are for example appended by functions inside the World-class.

7. Files and Internet access

I chose to go with a custom file format for saving and loading a world as it seemed the easiest thing to implement. The format is similar to csv but uses "END" strings to differentiate blocks of data.

The worlds are saved as single strings that are generated by the program. The strings are in the following format:

(boids separated by ;) END (predators separated by ;) END (foods separated by ;) END (simulation enabled) END (mutation chance) END (separation slider value) END (cohesion slider value) END (fov slider value) END (foodspawninterval)

A small file could be for example:

```
531.363257086175,538.9949883591282,526.1909387788277,542.3931057055026,1.55244755244755
25,1.25,326.9527972027972,2.0,2.0;202.49309509950973,573.5521035588039,199.103699934894
08,615.4682589760811,1.5524475524475525,1.25,326.9527972027972,2.0,2.0;446.865754432442
15,580.8723829987339,453.45912020747596,577.6039662903368,1.5524475524475525,1.25,326.9
527972027972,2.0,2.0;ENDENDENDfalseEND0.3END1.0END1.25END69END150
```

Since the strings get quite long, larger test files are provided in the code.

8. Testing

Due to the graphical nature of the project, testing was mostly done manually. Unit testing was tried and quickly dropped in favor of just testing everything manually. Some edge cases could have been tested through unit testing but as it had already formed a habit to test everything manually, that's what I stuck with.

9. Known bugs and missing features

An error thrown all the time while the program runs is ConcurrentModificationException, which rises from appending lists while the program

is iterating them. In the code it is ignored as it doesn't affect vital functions of the program. It does cause slight stuttering when there are mainly predators on screen, as they disappear. This could possibly be solved by adding all boids to be added and removed to a list and removing or adding them only at the end of a frame.

I'd planned to implement being able to click on boids and highlight and modify their parameters individually but deemed it unnecessary for the workload and final project. It could have been implemented by taking the location of the cursor when clicked on the canvas and getting a boid near that location from World,

10. 3 best sides and 3 weaknesses

Strengths:

Colors

I really like the way boids change colors as you change their parameter. Also seeing the way colors gradually change in the evolution mode is fun and rewarding as a programmer.

Drawing boids

I am also happy of the way the boids look. It took some time to figure out the math behind drawing a triangle as funny as it sounds and it was not a given from the start.

Weaknesses:

Predator extending Boid

The implementation is really clunky, and it could have been implemented much better. Correcting this would probably require refactoring Predator in its entirety.

Performance

The program is quite taxing on a computer, as each boid iterates over each other boid, then predator, then food. The program could be made lighter to run by using something like a quadtree and just refactoring things to be neater.

11. Deviations from the plan, realized process and schedule

Much care was not taken for the initial schedule. During the first few weeks I got to work more than expected, finishing the UI and writing a part of the Boid class. After that, most of the time was spent on getting boids to work properly.

Implementing food and evolution took slightly more time than expected as well. Saving and loading was also implemented as one of the last steps instead of before evolution. This was done to have a clear picture of what should be loaded and saved.

If I did this project again, I would start logging the amount of time spent on it, maybe spend slightly more time planning or at least update the plan as in agile development.

12. Final evaluation

In the end I am pleased with the program I've built. I chose this project for studio A as it seemed the most interesting to me, combining graphics and a certain degree of math. I'm also a big fan of emergent gameplay in games so the emergent nature of the flocking caught my eye.

Throughout writing this document I've often wandered back to fix a bug or develop something a bit further but as deadlines approach, I just leave them be.

The actual boids are something I'm quite happy with. They flock and respond to parameter changes nicely and the color changing part is fun. And though the evolution part could take a bit more refining the system is able to maintain around 20 to 30 boids and predators with their amounts varying as in natural population graphs. I did some digging into UI design and incorporated the golden ratio into the main layout which felt rewarding.

I feel like planning was one of the bigger shortcomings in the project. Especially the Predator class could've been much more thought out. Though I think my initial plans were good, it would've been good to step back at some point and think of a plan for the next few steps instead of starting to implement the next thing that seemed the most interesting or bothered me the most.

In the future I plan on refactoring the code to be easier to read and manage, as well as reworking the Predator class. This would include writing proper overwrite functions instead of basically having predators have two sets of locations. I will also most likely continue to work on the evolution part to make it more realistic.

If I started the project again, I would spend a bit more time planning out the basic class structure. This time around it was still slightly unclear what the end result would look like and when I noticed that it would be good to switch the structure around I felt it was too late. I would also use git more efficiently by for example by using branches to test around with new things instead of pushing partial progress of something that might not work in the end to main. I would also spend some time in writing unit tests after some main parts of the program are established.

13. References and code written by someone else

The scalaFX series by Mark Lewis has been really handy:

https://www.youtube.com/watch?v=sPE8STqy_ns&ab_channel=MarkLewis

https://www.youtube.com/watch?v=bqtqltqcQhw&ab_channel=SebastianLague

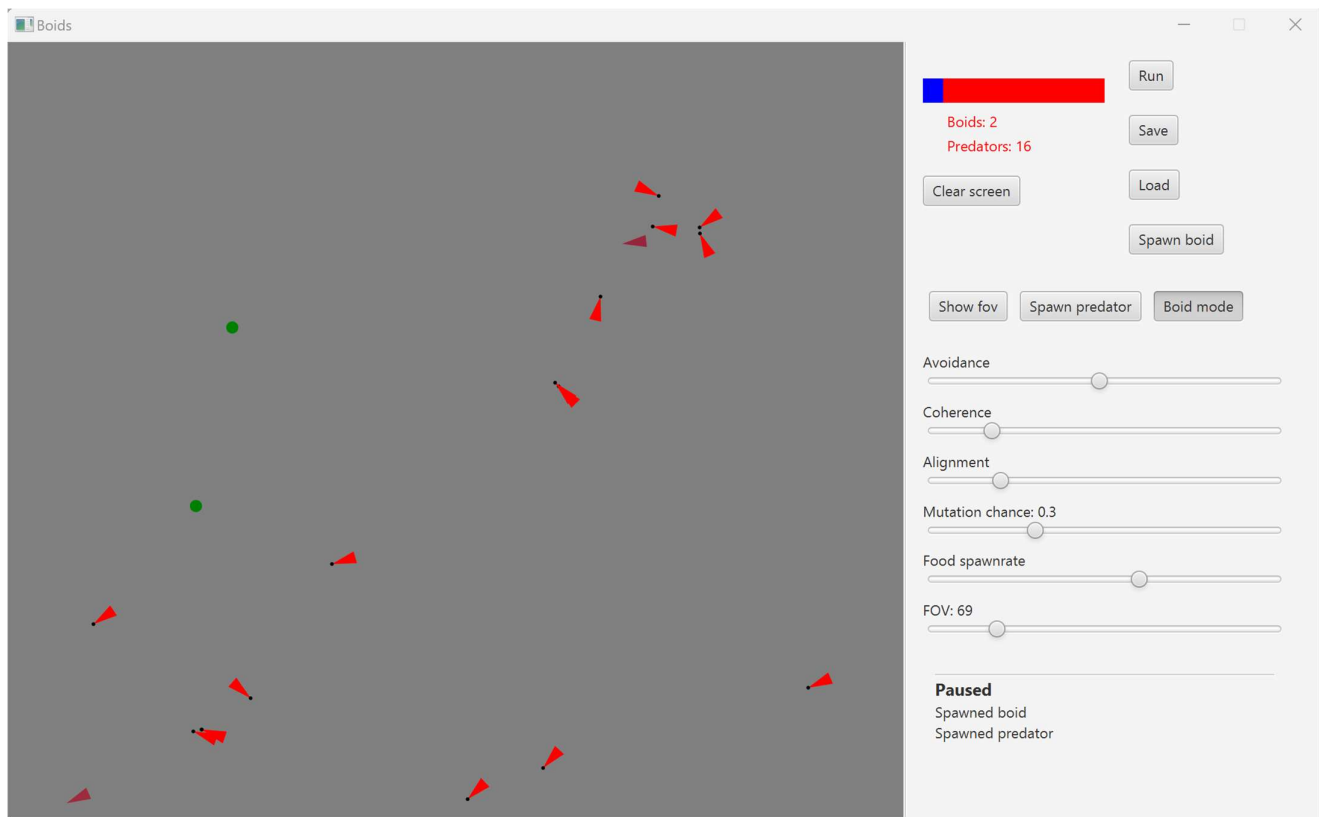
Some others projects I looked at:

<https://github.com/exigow/flocking-boids/tree/master/src>

<https://github.com/jimm/scala/tree/master/boids/src/boids>

<https://boids.cubedhuang.com/>

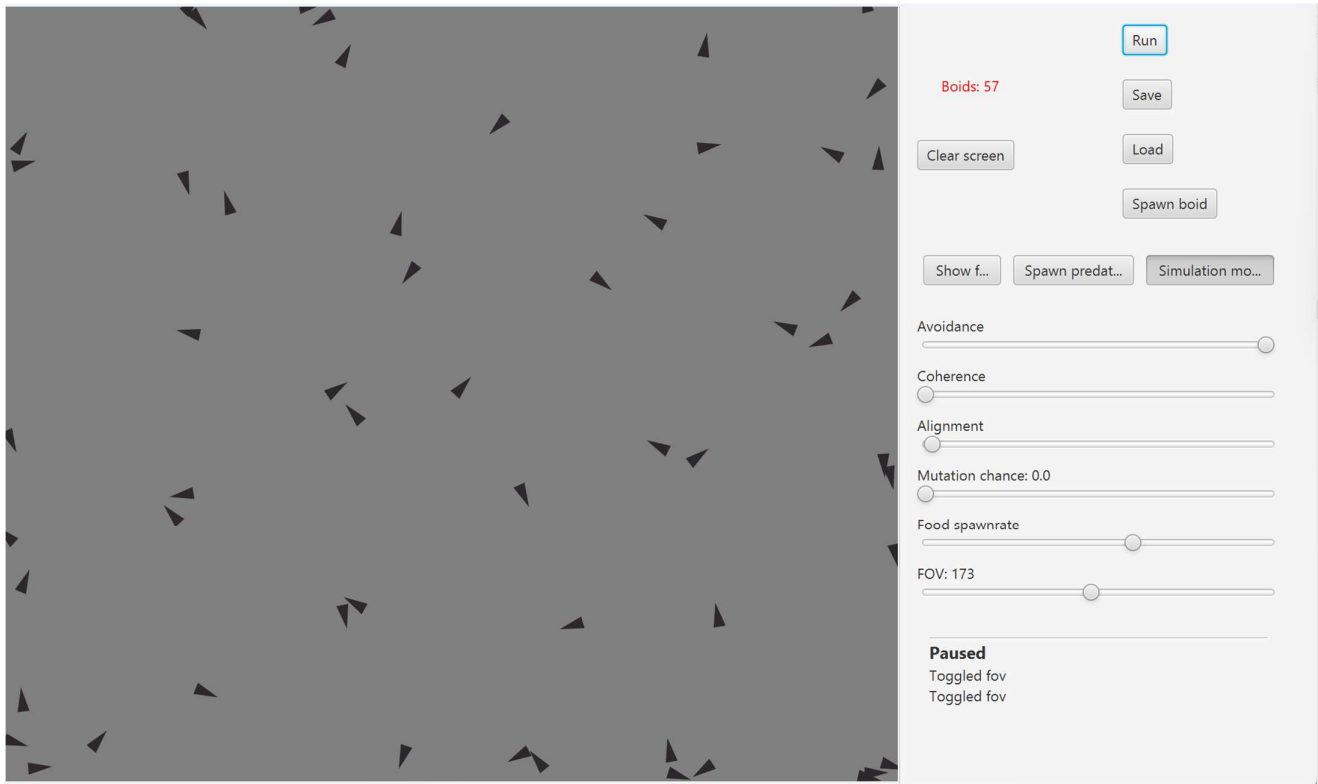
Appendices



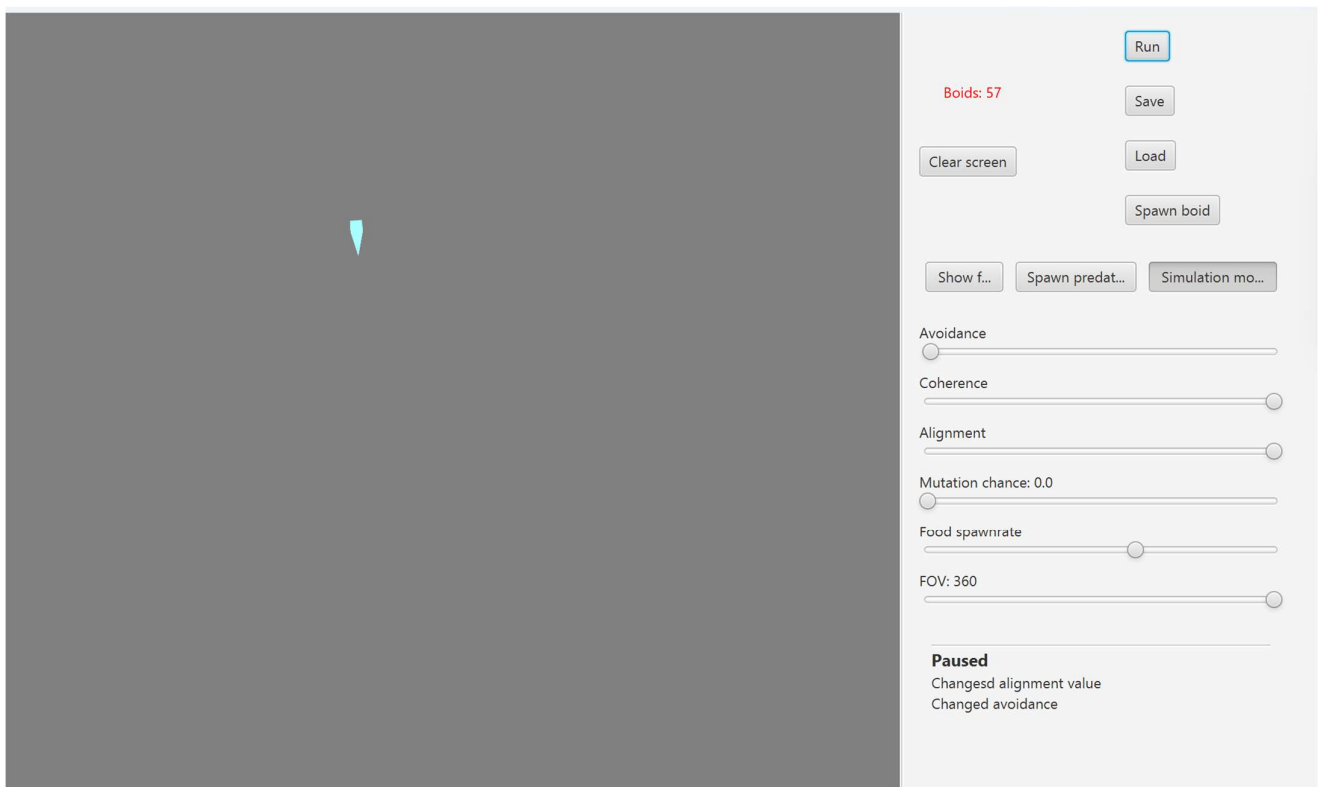
Boids and predators coexisting



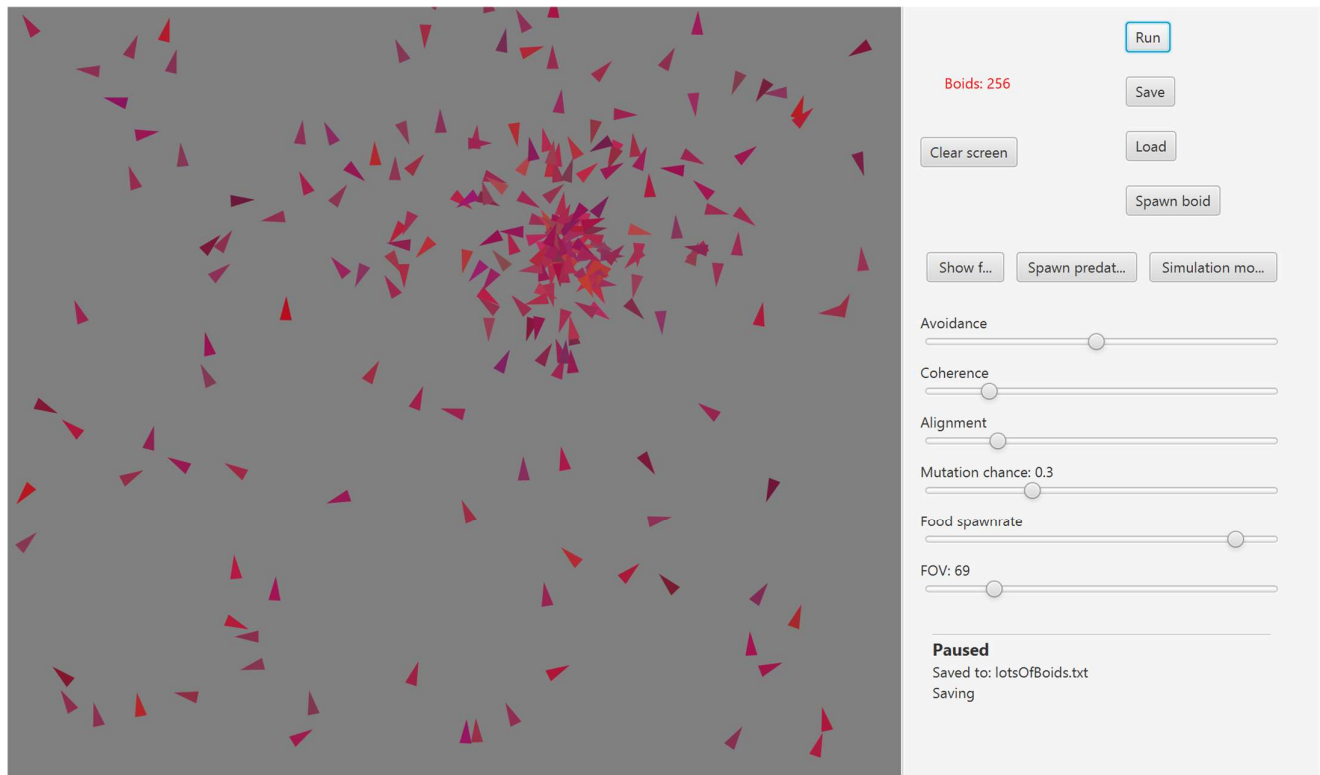
Boids with FOVs drawn.



Parameters to get black boids.



57 boids stacked on top of each other through parameter control



Lots of boids