Onni Miettinen
793401
Aalto-yliopisto, Perustieteiden korkeakoulu
Tietotekniikan fuksi
28.4.2021

BIRDS AND FISH --- GROUP BEHAVIOUR

Back to school, flock of boids!

--------- Programme documentation ---------

INDEX

1. General description

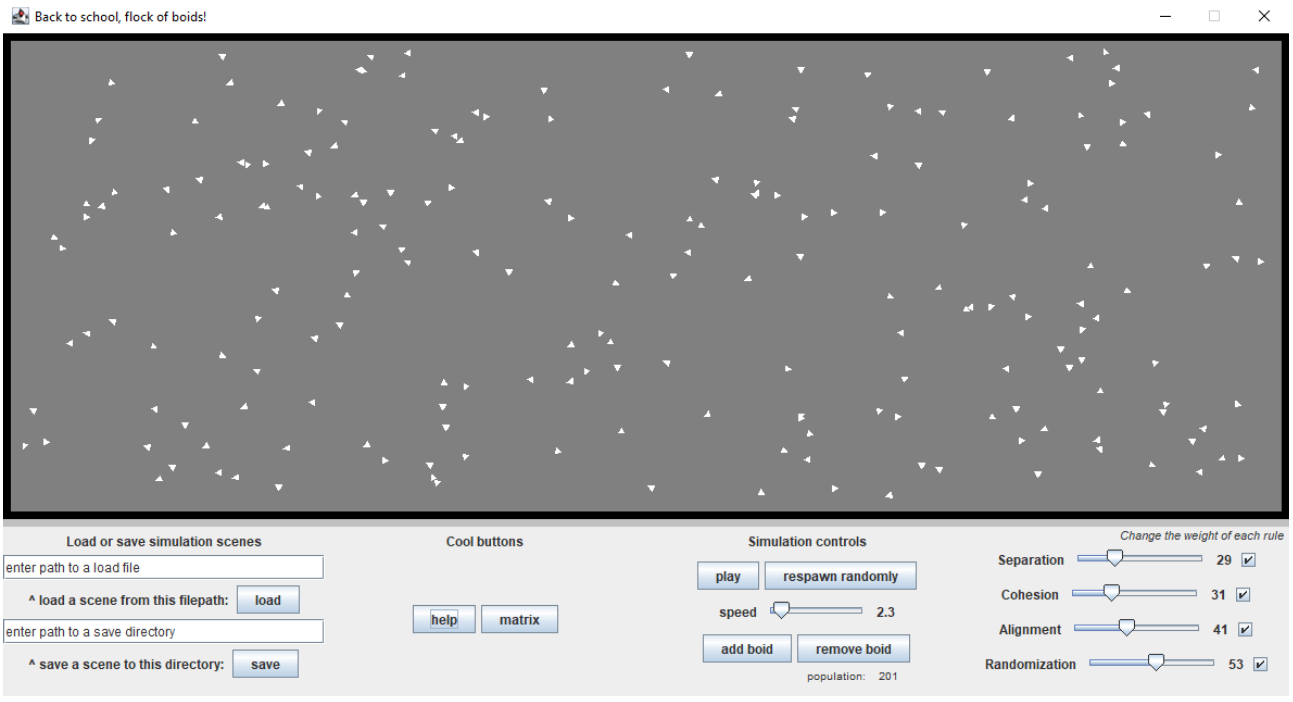
The purpose of this project is to create a beautiful simulation for the group behaviour of a flock of birds (or a school of fish, if you prefer that). The actual focus of the behaviour programming will be on a single entity, a boid (bird-oid entity). A group of these will result in the actual simulation.

A boid is programmed so that it will act based on its surroundings (this being other boids). This style of behaviour will result in a group-like behaviour when several boids are present. A single boid follows four separate rules that define its movement. These rules are: Separation, Cohesion, Alignment and Randomization. A further look into these rules is given at a later chapter, namely “4.1. Behaviour motives for boids”.

The simulation runs on a graphical user interface, that visualizes the simulation and allows the user to alter it. The GUI allows the user to change the weight factor of each rule, that is, how impactful they are compared to the other rules. This is the general gist and purpose of the programme: to create and visualize group-like behaviours of boids and see how each individually simple rule affects the whole.

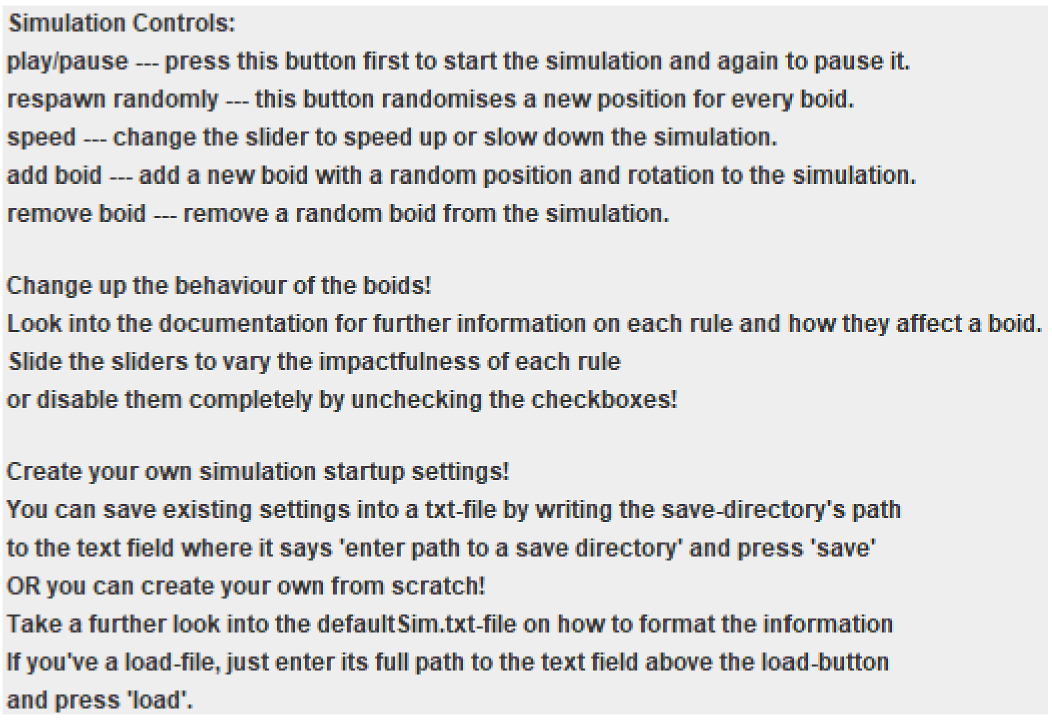
2. How to use the programme

Well, first things first, here is a screenshot of the programme, so you have a better idea of what is going on:


Back to school, flock of boids!
Load or save simulation scenes
enter path to a load file
^ load a scene from this filepath:
load
enter path to a save directory
^ save a scene to this directory:
save
Cool buttons
help
matrix
Simulation controls
play
respawn randomly
add boid
remove boid
Change the weight of each rule

speed 2.3
population: 201
Separation 29 ✓
Cohesion 31 ✓
Alignment 41 ✓
Randomization 53 ✓

After the programme is ran, a window should pop up (one like above). The window cannot be resized. There are several buttons and a relatively wide view of the visual simulation. There should a button that says ‘help’. Clicking this button should open a dialog that has information on how to use the UI. Here is the message from that dialog:



Simulation Controls:
play/pause --- press this button first to start the simulation and again to pause it.
respawn randomly --- this button randomises a new position for every boid.
speed --- change the slider to speed up or slow down the simulation.
add boid --- add a new boid with a random position and rotation to the simulation.
remove boid --- remove a random boid from the simulation.

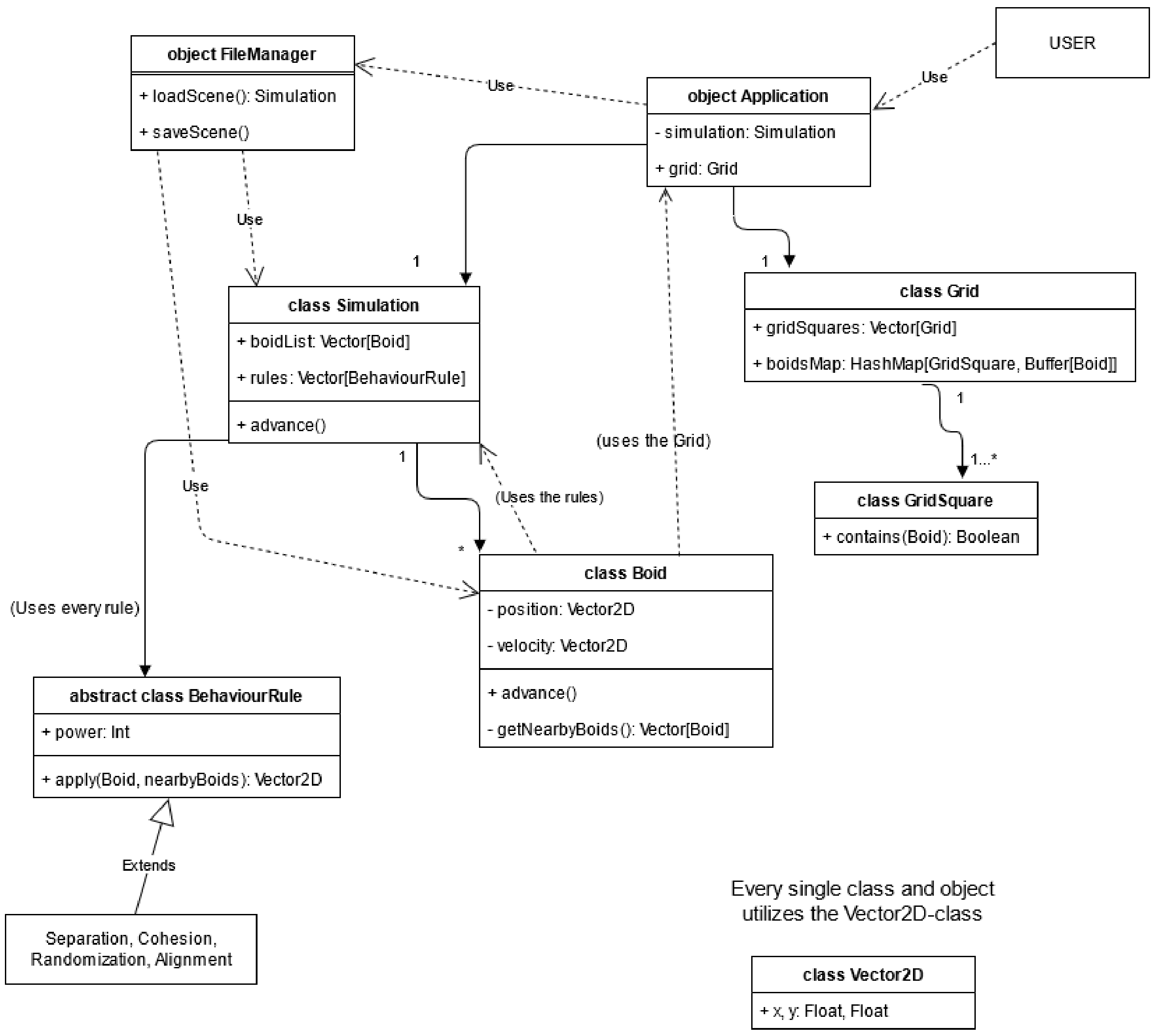
Change up the behaviour of the boids!
Look into the documentation for further information on each rule and how they affect a boid.
Slide the sliders to vary the impactfulness of each rule
or disable them completely by unchecking the checkboxes!

Create your own simulation startup settings!
You can save existing settings into a txt-file by writing the save-directory's path
to the text field where it says 'enter path to a save directory' and press 'save'
OR you can create your own from scratch!
Take a further look into the defaultSim.txt-file on how to format the information
If you've a load-file, just enter its full path to the text field above the load-button
and press 'load'.

Much like the dialog informs, there are several options for the user. The user can vary the weights of the rules, alter the population by removing or adding single boids, change the speed of the simulation and restart the simulation by randomly respawning each boid. The defaultSim.txt-file that is mentioned resides inside the project root, boids-os2. More information on file stuff is held captive at chapter 6. File foolery.

3. The internal structure

The programme's structure consists of objects, that are responsible for certain areas of the programme, and several classes that represent parts of the simulation. Here is a simple UML-diagram of the objects and classes and their relations. Some methods and variables are named just to better showcase the holder-objects functionality.



USER
Use
object FileManager
+ loadScene(): Simulation
+ saveScene()
Use
object Application
- simulation: Simulation
+ grid: Grid
Use
1
class Simulation
+ boidList: Vector[Boid]
+ rules: Vector[BehaviourRule]
+ advance()
1
class Grid
+ gridSquares: Vector[Grid]
+ boidsMap: HashMap[GridSquare, Buffer[Boid]]
1
1...*
class GridSquare
+ contains(Boid): Boolean
(uses the Grid)
(Uses the rules)
1
Use
*
class Boid
- position: Vector2D
- velocity: Vector2D
+ advance()
- getNearbyBoids(): Vector[Boid]
(Uses every rule)
abstract class BehaviourRule
+ power: Int
+ apply(Boid, nearbyBoids): Vector2D
Extends
Separation, Cohesion, Randomization, Alignment
Every single class and object utilizes the Vector2D-class
class Vector2D
+ x, y: Float, Float

Let’s start with the most important ones for the simulation.

3.1. Simulation and Boids

The simulation consists of boids. We represent these with the class Boid. The simulation should contain these boids that are vivid, moving, lifelike and most importantly – group-minded.

For the movement behaviours (the four rules that were mentioned in the opening chapter, 1. General description), we have separate objects that extend the abstract class BehaviourRule. These rules exist to provide answers to the boids looking for the purpose in life. So, each boid (instances of Boid-class), move so that every timeframe they ask the rules for direction and move accordingly. This is done inside the advance()-method of Boid.

All these boids that are present in a simulation are contained so that they can each be assessed. This container is an instance of the class Simulation. A Simulation contains every boid and every behaviour rule. The instance of Simulation contains many methods for controlling boids and the simulation, and the user interacts with the simulation (we’re ignoring the actual UI here), and not with individual boids. When a simulation advances(), it calls the advance()-method of every boid it contains. The boids carry a variable for their host-simulation, so that they can use the rules that the simulation holds.

3.2. The Grid

I tried to picture clusters of boids as they moved through the computer. What do they look like? Birds? Fish? Were the circuits like rivers, or as spacious as the vast open air?

At first the simulation ran with every boid checking every other boid and the distance between them. As you’d imagine, this method wasn’t really the greatest in terms of performance. So, I added a grid. An instance of the class Grid contains several GridSquares, that are as lengthy as a single boid’s vision. This idea and its pros are further explained at a later chapter inside the Algorithms-chapter. Structurally, A grid contains small squares, that each contain boids. The instance of Grid contains the map of boids (HashMap of [GridSquare → Buffer[Boid]]) and all the GridSquares. A GridSquare’s only function is to provide the method contains(boid), that checks whether it holds a certain boid.

3.3. GUI

The object Application in the UML is where the programme is ran. The Application uses Swing-library and extends SimpleSwingApplication. The GUI thus consists of containers (panels) that contain other UI-elements (more containers, buttons, text fields etc.). The GUI-layout (look at the picture at chapter 2. How to use the programme) has two big components: the simulation window, where the simulation is visualized, and a controls-panel.

The controls panel itself consists of four panels, from left to right: the file management -panel, cool buttons -panel, simulation controls -panel, and the rule controllers -panel. The file management -panel has two text fields and two buttons for loading and saving (path and the corresponding action button). The simulation controls -panel has buttons that manipulate the current Simulation-instance, and the rule controllers -panel has controls that manipulate each BehaviourRule-object.

The Application holds the one ongoing simulation and the grid. The Application manipulates the BehaviourRules by using the simulation’s rules-list. The Application also uses the FileManager in order to create or read from files.

3.4. FileManager

FileManager has two methods: loadScene and saveScene. These methods are called by the Application when certain buttons are pressed. When loading, FileManager reads a txt-file from the given path and creates instances of Boids and an instance of Simulation to contain these.

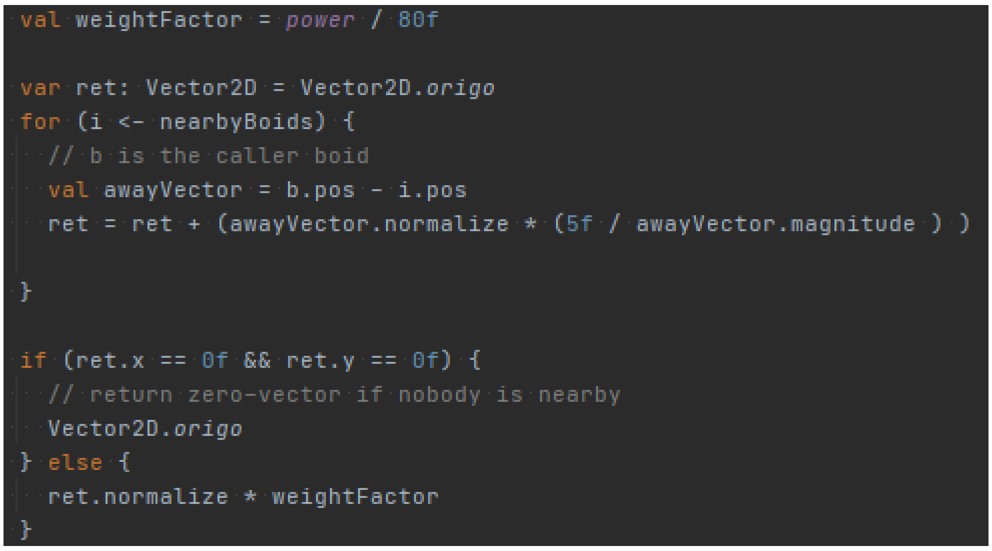
4. Algorithms

In this section, we dive a bit deeper inside the motives of the simulation’s boids, namely the four BehaviourRules that control the boids’ movement. We also take a look at the reason I implemented a grid onto the simulation.

4.1. Behaviour motives for boids

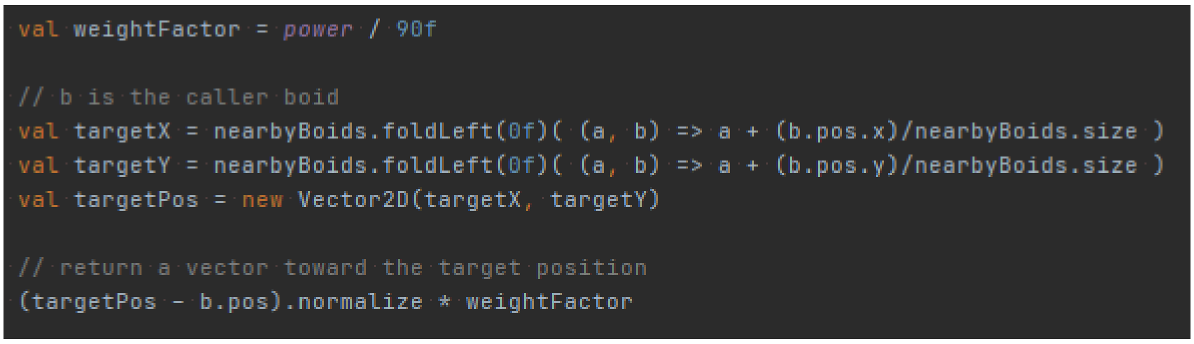
Let us look at each rule. Again, the rules are: Separation, Cohesion, Alignment and Randomization. I will give a quick explanation on each rule and then show the code for it. Each rule, except for Randomization, receives a list of the nearby boids as a parameter for its apply()-function. This list of nearby boids is later referenced just as ‘vicinity’ in text and ‘nearbyBoids’ in code. In the code it should be noted that a BehaviourRule has a variable for ‘power’.

Separation: the boid should look around and move away from the average position of the boids in its vicinity.



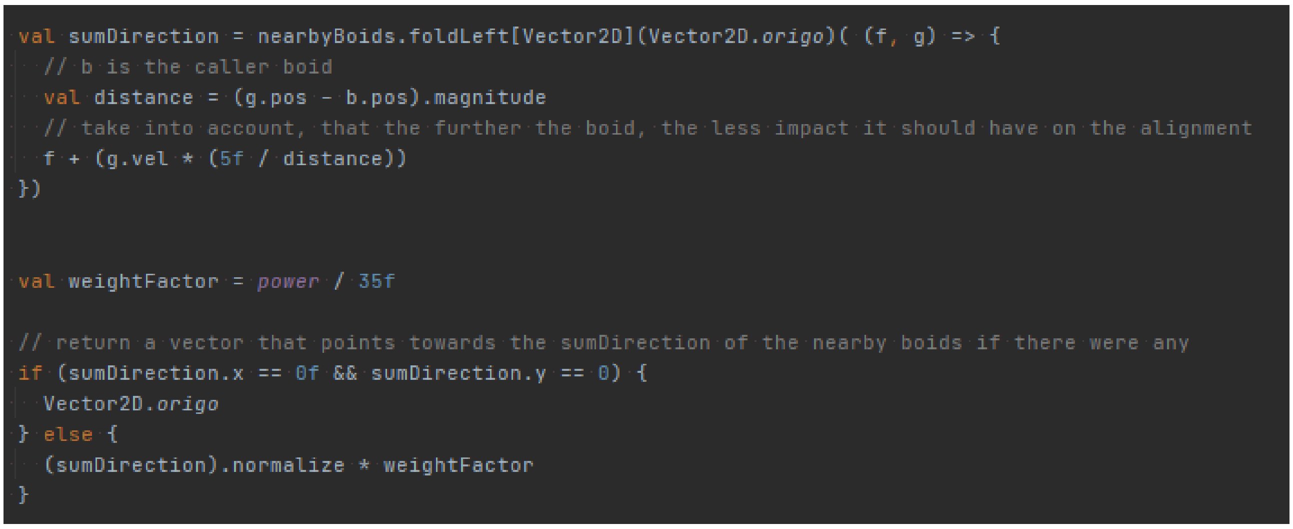
```
val weightFactor = power / 80f

var ret: Vector2D = Vector2D.origo
for (i <- nearbyBoids) {
  // b is the caller boid
  val awayVector = b.pos - i.pos
  ret = ret + (awayVector.normalize * (5f / awayVector.magnitude ) )

}

if (ret.x == 0f && ret.y == 0f) {
  // return zero-vector if nobody is nearby
  Vector2D.origo
} else {
  ret.normalize * weightFactor
}
```

Cohesion: the boid should look around and move towards the average position of the boids in its vicinity.



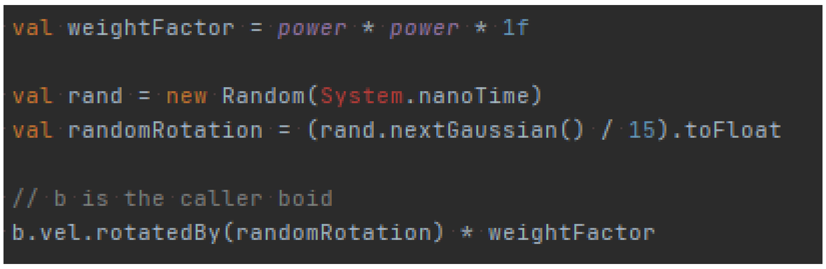
```
val weightFactor = power / 90f

// b is the caller boid
val targetX = nearbyBoids.foldLeft(0f)( (a, b) => a + (b.pos.x)/nearbyBoids.size )
val targetY = nearbyBoids.foldLeft(0f)( (a, b) => a + (b.pos.y)/nearbyBoids.size )
val targetPos = new Vector2D(targetX, targetY)

// return a vector toward the target position
(targetPos - b.pos).normalize * weightFactor
```

Alignment: the boid should look around and adjust its direction so that it has the same direction as the average boid in its vicinity. The average direction is just calculated as a sum vector of directions.



```
val sumDirection = nearbyBoids.foldLeft[Vector2D](Vector2D.origo)( (f, g) => {
  // b is the caller boid
  val distance = (g.pos - b.pos).magnitude
  // take into account, that the further the boid, the less impact it should have on the alignment
  f + (g.vel * (5f / distance))
})


val weightFactor = power / 35f

// return a vector that points towards the sumDirection of the nearby boids if there were any
if (sumDirection.x == 0f && sumDirection.y == 0) {
  Vector2D.origo
} else {
  (sumDirection).normalize * weightFactor
}
```

Randomization: the boid should change its direction a little bit, randomly. This is calculated using the Vector2D's method rotatedByAngle(angle) and randomizing a random input angle using the normal distribution.

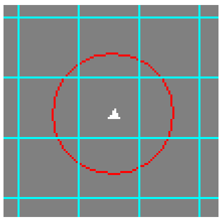


```
val weightFactor = power * power * 1f

val rand = new Random(System.nanoTime)
val randomRotation = (rand.nextGaussian() / 15).toFloat

// b is the caller boid
b.vel.rotatedBy(randomRotation) * weightFactor
```

Each rule returns a Vector2D from its apply()-method. The sum of these vectors is the direction and velocity of the boid after it has been influenced by these rules.

4.2. A grid of squares as a map

As I mentioned before, before the grid-system, each boid had to calculate every boid inside the simulation in order to get the vicinity. The grid system introduces GridSquares that have the side length of the boid’s vision. Thus, if the boid can see no farther than X pixels, it is enough to just calculate the vicinity from the neighboring GridSquares.


The red circle represents the boid’s vision.

Although the simulation needs to update the grid’s boidsMap every frame, this grid-system fastens the calculations tremendously, when there are lots of boids present.

5. Data structures

For the maps of the Grid, I used HashMap, which should be very fast. For lists, I used Vectors when mutability was not required, because they are generally fast in lots of aspects. For mutable lists, I used Buffers.

I also created a new class for 2-dimensional mathematical vectors, Vector2D. The class has many useful vector methods, and the values are all Floats. I used Float instead of Double because the precision of Double is not necessary in this programme. According to several sources (random forgotten google searches), a Double is more ‘expensive’ than a Float and should be used instead when accuracy is of great importance.

6. File foolery

So, the programme allows the user to load simulation scenes from txt-files and to save simulations as txt-files. The file format is explained inside the defaultSim.txt-file, that works as the default simulation for the programme when it is started. Here is the file format:

```
POP 201
RUL Separation; 29; 1
RUL Cohesion; 31; 1
RUL Alignment; 41; 1
RUL Randomization; 53; 1
SPE 2.3
BOI 928.85126,78.23431;-3.0450428,0.5811317
```

There are four information types, each information needs to be on its own line. For the programme to be able to read each information line, they must start with one of the following: POP, RUL, SPE or BOI. This also means that the user can write whatever on any line, if that line doesn’t start with any of those 3-letter prefixes.

Each of these information types that are POP, RUL, SPE or BOI contain information of the population, a behaviour rule, simulation speed or a single boid, respectively.

Any line that does not start with these prefixes are ignored by the programme. Whitespace is allowed everywhere, but each information type needs to contain all of its information on its one line. Here is further explanation of the information lines, inputs are given where the parenthesis are:

POP (population amount as an integer)

RUL (name of the rule: Separation, Cohesion, Alignment or Randomization) ; (weight as an integer) ; (enabled as 1 or 0)

SPE (simulation speed as a number with one decimal within the range of 0.1 and 20.0)

The user doesn’t have to give any boids their starting position, as they are randomized by default. If the user does give some boids initial values, the rest of the population is randomized. a BOI-piece contains boid’s initial xy-position and its direction as a two-dimensional vector.

BOI (x-position as a float), (y-position as a float) ; (x-vector as a float), (y-vector as a float)

7. Testing, testing…

Initially after I implemented the class of Vector2D, I created a bunch of unit tests for the methods to see if they return wrong values. For the behaviour rules though, I tested them visually directly from the GUI. As the vector-mathematics was not too complicated, I could easily see from the effect on the visual simulation whether the rules worked correctly.

After creating the implementation for the Grid, I added a feature that allowed me to visually check whether it worked. Also, I could compare known simulation scenes with and without the grid-system. The feature, that allowed me to visually see whether the grids aligned nicely or at all, is the matrix-button on the controls-panel.

Well, pretty much everything was tested through the GUI after implementation of the Vector2D-class. Stress testing for the capacity was also through the GUI, though those tests were only ‘feeling-tests’. Sometimes I did test, whether some code-blocks were assessed a correct number of times or at all, with just println()’s.

8. Bugs? Bugs!!

Even though I tried to up the performance by implementing the grid-system and calculating the simulation’s advance() in another thread separate from the Swing’s Event Dispatch Thread, the stress tests showed that, at least on my computer, after a certain number of boids, the simulation can suddenly stop for a great number of the boids. Even though on the simulation window there are plenty of boids still, some continue moving.

Also, sometimes I saw that a Boid’s velocity would get the values of NaN. I suspected this was due to the fact that the velocity is normalized, which probably returns a NaN if the velocity to be normalized is 0 (dividing by zero when trying to normalize). I suspect that sometimes the rules could sum together a zero-vector. This bug was ‘fixed’ by adding if-statements to check whether the value is NaN (or zero) before updating the boid’s rotation (this was where the issue of Nan was a problem). Now, or even before if it was not for the requirements of some Vector2D-methods, it does not affect the simulation at all.

9. Top 3 and the other top 3

Well, all things considered, I am very proud of my work and it is not easy to find a 'bottom 3', nor a 'top 3', at that. Anyway, let us start with the other top 3, or in other words: those that I do not find the best things ever in the world.

The other top 3:

- Performance. I think the performance could be much better, though I have not too many ideas to improve it anymore. One thing I did learn a couple days ago from the course Programming 2 is that for-loops and foreach-methods can be replaced with while-loops for better performance. I can say for sure that there are a lot of for-loops in the programme.

- The Grid's and Simulation's relationship. As mentioned before, the Grid was a late-add. I think it works wonderfully but it is very separate from the Simulation in terms of the class structures. I think it would be better if they would be more connected. Now, the Application holds an instance of Grid that it just adds as an input to every simulation-method. Well, this is not too bad either, I think.

- The visual side of the UI. This is not that big of a deal for me, but I wish I would have had more time to better improve the programme's visual appeal. I do like the colour-theme of the simulation window, but the default buttons and everything in Swing are not too pleasing for me.

The top 3 (not in any particular order). These I am a bit proud of:

- Performance and The Grid. Wow, what a surprise, right? Well personally, I didn't have high hopes for the performance of the programme, but I was pleasantly surprised by how much the grid system improved it. Even though it can be improved, sure, I do think it is quite good now.

- The Simulation and the BehaviourRules. I think the simulation is very good. The rules are perfect for what they stand, and they produce magnificent-looking results for the flocks. I very much enjoy looking at my work: my favourite thing is to amp up the Cohesion and then toggle it. Just watching the boids explode out of formation is quite cool. Also, I tried to create the project so that it would be easy to implement new rules. I think this was successful as one only needs to extend the abstract class of BehaviourRule and add it to the list of rules inside the class Simulation.

- File management. This is not that big of a deal for me, but I do think the file system is pretty good: it's easy and simple for the user (okay, the path-inputting might be troublesome, but I like it) and the file format is easy to understand and use. I think it's a bit funny for the user to be able to create a simulation txt-file and add all these individual boids on their own lines. Well, at least the saved scenes look pretty nice on the txt-files.

10. Final words

The project and implementing a simulation for group-like behaviour of individual boids was very successful, I would say. I am very proud of my work. There was never really a solid schedule for anything project related, I just worked on it whenever I had time and energy. There were plenty of weeks where I did not advance the project much, but on the weeks that I did, the progress flew. There was no one plan or schedule I followed; it was more like I created lists of things I wanted to finish on certain time frames. This was more fluid in terms of progress and a bit more fun when compared to a solid long-term schedule. Though, I cannot really compare which would be better since I didn't really give the other one a try.

11. References

- Craig Reynolds, article: "Steering Behaviours For Autonomous Characters", last updated on 3.5.1999, http://www.red3d.com/cwr/steer/gdc99/