Flock Simulation Final Document

Roope Kettunen
Student#: 100583964
Bachelor in Data Science 2022
19.4.2023

1. **General Description**

I created a flock simulation, where a user-adjustable number of individuals' movements are simulated. The program visualizes bird movement graphically in a 2D world with a perspective of the birds from above. Taking inspiration from Craig Reynolds work on steering behavior and so-called “boids”, the bird flock will be 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 and their movement patterns resemble that of real birds.

The three bird movement rules are as follows: (i) avoid collisions to other birds (Separation), (ii) fly as fast as the rest of the flock on average (Alignment), and (iii) aim for the center of the flock (Cohesion). The applied strength effect of each of these rules will be adjustable in the GUI. The implementation of threads will also be necessary to run a real time simulation.

I have attempted to meet all the requirements for a Demanding/Difficult level project that are outlined above.

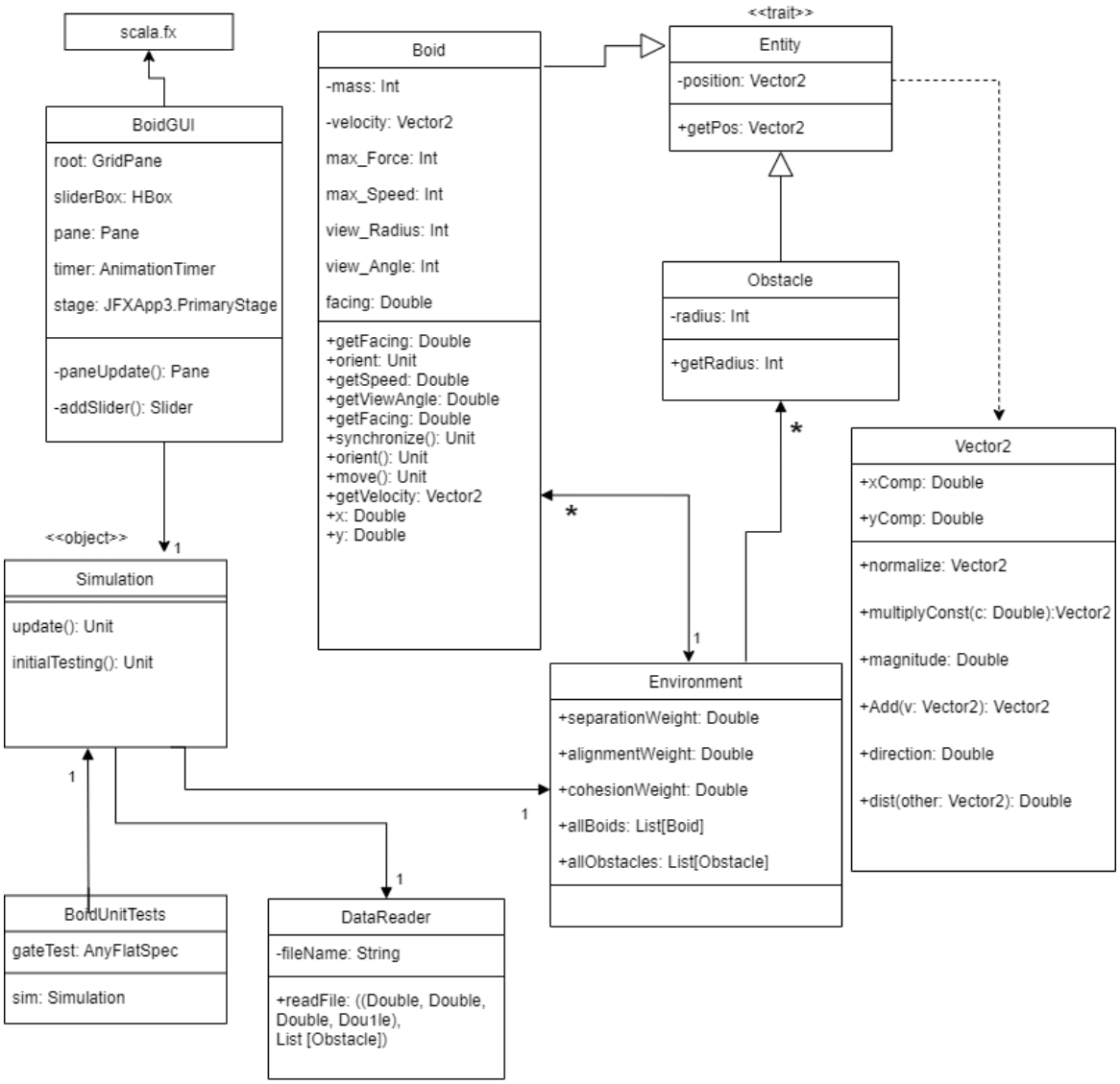
2. **User Interface**

Before running the program, the user can also specify some initial starting conditions according to the rules explained briefly in the file named “BoidData.txt”. Once these are set (or left at the default values), the user can run the BoidGUI object, which will create a scalafx window displaying the simulation, and have a number of interactive features. These features include six sliders that can adjust each of the three movement weights, the view radius of the boids, the speed of the boids, and the total number of boids. Additionally, there are five buttons that can toggle the view circles of the boids,

add/remove single boids, pause/resume the simulation, and finally clear all obstacles on the simulation plane. Furthermore, the user can manually add obstacles at any location within the simulation by simply clicking on that spot with any mouse button. Finally, the user can sit back, and enjoy the soothing and wonderful emergent behavior of the boids. An example screenshot in the appendix. (section 13)

3. Program Structure

The uml diagram concisely sums up the basic program structure, which is not overly complex.



4. Algorithms

Here, I will briefly go over the largest and most complex methods, as the rest of the code has quite trivial logic.

Each Boids orient method

1. First a search of all nearby Boids within a Boids defined view_Radius and view_Angle that is saved into a list and will be used in the next three major phases
2. During this search, the wrapping of the edges in the simulation (how the boids are teleported when they hit the edges) must be taken into account
3. Separation - For each nearby Boid, a repulsive force is computed by subtracting the positions of the current Boid from that of nearby Boids. Each of these forces is then normalized, and applylied a 1/r weighting based on the distance to the boid. Then the forces are added together to create the combined Separation steering force
4. Alignment - The average velocity of all nearby Boids is calculated, and this is the steering force
5. Cohesion - The average position vector of all nearby Boids is calculated and then a steering force in directed from the current Boids position to that average position
6. Avoidance - similar to Separation, a simplified version of obstacle avoidance will be applied, treating the obstacles as a boid that would be more repulsive
7. Each of the above mentioned steering forces will be multiplied by the weights given via the Environment class, and will also be modified by the weight and max_force
8. This force is then added to current velocity, and is capped at maximum speed. The velocity is then later added to position, in the move() method

In BoidGUI, the one longer method was the updatePane() method:

1. First, we create a new scalafx.scene.layout.Pane, and set the size and paddings
2. Next, a blank rectangle is drawn as the background, and we begin drawing Boids
3. As we loop through the boids list, we first check if the boid is within the boids of the environment, and then using the boid's direction and position information, draw a kite-like Boid visual representation, drawing a corresponding view circle if toggled
4. Finally, we draw in the obstacles from the obstacle list, and return the pane

File Reading Method in file reader:

1. Read each line using methods from first few chapters of Programming Studio A

5. Data Structures

- Mutable List of Boids in Environment Class
- Mutable List of Obstacle Options in Environment Class
- Vars/Vals and Pairs for Scalars and Vectors in each Boid and certain Environment classes
- Vector2 Class that contains two Doubles, for x and y components of a 2D vector

6. Files and Internet Access

A text file with all basic initialization information using the following example human readable format (bolded parts can change):
—----------
NumberOfBoids: **50**
SeparationWeight: **1.20**
AlignmentWeight: **0.80**
CohesionWeight: **0.60**
Obstacle: (**100**, **100**)
Obstacle: (**300**, **300**)
#this to comment lines out

7. Testing

Using the GUI as a visual cue, a lot of basic tests were easily done just by eye, and once there were no visual discrepancies with multiple combinations of settings, many functionalities were easily tested. However, to ensure that the calculations for the orient method of each boid was correctly modifying the velocity of the boids, a simple two Boid unit test file was created named BoidUnitTests. The unit test runs a simulation, starting with two Boids in relatively close proximity. The test then calls the update() method in the simulation once, and then compares the velocities of the two Boids to an expected, hand-calculated result. These simple unit tests were passed, and this testing was done pretty much as planned.

8. Known bugs and missing features

From a decent amount of testing, there are no major known bugs that would impede the results of the simulation. There are certain situations and settings that may cause the Boids to act slightly unnaturally, making very quick turns, or shaking slightly (making many small repeated needless adjustments). However, these movements take little away from the simulation.

Some initially planned features that did not get fully or properly implemented include proper obstacle avoidance (a different complex behavior to separation described by Craig Reynolds) and the use of additional threads to speed up computation with very large amounts of Boids. However, to compensate, a plethora of non-planned features have been added to increase the interactivity of the simulation.

9. 3 best sides and 3 weaknesses

I believe that the three best sides of my program are (i) the smoothness of Boid motion, (ii) the wide variety of live modifiers, yet clarity in the GUI, (iii) the ability to add obstacles with the click of a mouse. The third point allows for complete creativity to create even an obstacle course for the Boids and is a lot of fun to try.

The three clearest weaknesses of my program include (i) a sub-par implementation of obstacle avoidance, sometimes looking odd, (ii) the boids having rare crashes due to non-perfect separation behavior, (iii) the use of only a scalafx animation timer to run the simulation as opposed to multi-threading.

10. Deviations from the plan, realized process and schedule

Brief summaries of progress reports:

Calendar Weeks 7 and 8: Initial class structure and very basic GUI frame

Calendar Weeks 9 and 10: Boid rendering was completed, and the orient method of the Boids was drafted, and created some behavior

Calendar Weeks 11 and 12: Complete overhaul of the orient method, and major GUI fixes and adjustments

Calendar Weeks 13 and 14: Unit tests for boid behavior were done, obstacles in the simulation were implemented, the text reading functionality was added

Calendar Week 15: Edge wrapping and teleporting was finally fixed and numerous aforementioned live adjustment features were added, and comments and code clean up

Overall, the initial schedule was followed surprisingly well. The biggest deviations were definitely during weeks 12-14, where little ended up being done, and there was some lack of motivation. However, during the last week, I was able to completely transform the program by fixing the build up of bugs and messy code. Additionally, I was able to add the live adjustment features that made the boid simulation much more interactive and overall powerful. Creating the GUI first, to be able to at least somewhat visually test new features was essential to getting progress when creating any sort of changes. Additionally, even the small unit tests that I created were able to find numerous bugs that would otherwise have been unnoticed. This was mainly because the creation of the unit tests required a precise understanding of what the true functionality and algorithms of the program were.

11. Final evaluation

I am definitely satisfied with the final results of my project. The visuals are simple, yet able to effectively show the power of the Boids, and the three simple rules of motion. By far, being able to mess with so many different parameters live is one of the strongest aspects of the project. When assessing code structure, the general class structure seems quite solid to me, and this is supported by how it did not change much from the initial plan to realization.

Despite this, within the classes, many of the methods and constants could likely be organized more effectively, as there were certain phases during programming where certain actions seemed repetitive. However, I did attempt to fix many of these errors during the final week, and try to organize the code into readable blocks paired with simple comments. I think the best ways to improve the program would be to properly clean up the simulation process with a solid implementation of multi-threading to completely

stabilize the simulation, even with thousands of Boids. Additionally, performance can be improved significantly in areas such as the edge wrapping detection, that does not perform accurately when the Boids are in the corners. I think one of the worst written files is the DataReader, which contains a lot of directly mirrored code from the Programming Studio A exercises for a human readable format. I think this file reading aspect could have been expanded on more, or at least made much more logical through improvements in file format coupled with file reading techniques.

I believe the overall structure of the program is quite suitable for changes overall, as the Boid class is still easy to navigate and thus many changes can be made to refine the overall steering behavior. Additionally, from the Entity polymorphism, there are countless ways to easily implement other drawable objects with a position into the simulation, that can have interesting effects on Boid behavior. The GUI also has a lot of space and somewhat organized blocking, which makes adding buttons and sliders quite manageable. If I were to start the project from the beginning, I would create many more unit tests along the way. By doing this, I would be able to advance and debug faster, knowing that my previous work is solid. With this confidence, I would have been able to look into multi-threading, and be able to test more complicated boid behavior calculation methods. This would have all led to an even more satisfying and feature packed simulation, with larger simulation capabilities.

12. References

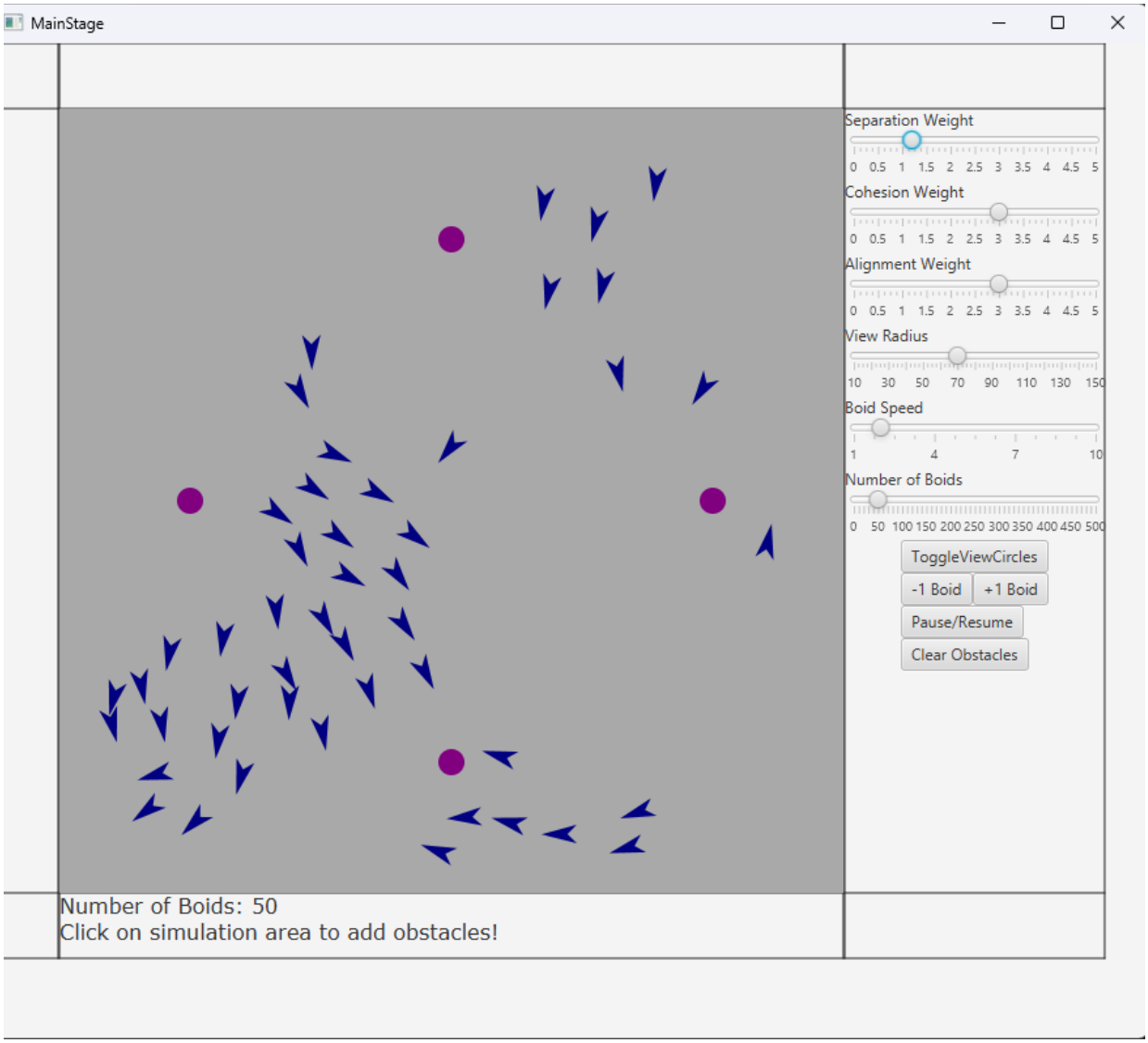
Steering Behaviors For Autonomous Characters (red3d.com)

1. Craig Reynolds: Flocks, Herds, and Schools: A Distributed Behavioral Model (toronto.edu)
2. Coding Adventure: Boids - YouTube
3. ProScalaFX/JavaFXSceneInSwingExample.scala at master · scalafx/ProScalaFX (github.com)

13. Appendixes

Full source code attached via intellij project

Example running image of simulation below


MainStage
Separation Weight
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
Cohesion Weight
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
Alignment Weight
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
View Radius
10 30 50 70 90 110 130 150
Boid Speed
1 4 7 10
Number of Boids
0 50 100 150 200 250 300 350 400 450 500
ToggleViewCircles
-1 Boid
+1 Boid
Pause/Resume
Clear Obstacles
Number of Boids: 50
Click on simulation area to add obstacles!