Table of Contents

1. Personal information

Name: Joonas Björk

Date: 25.4.2021

2. General description

My chosen project is the Simon says -simulation. In the simulation a leader circle is followed by a user defined amount of follower circles. The circles interact like the simple vehicle model described in Craig W. Raynolds' article "Steering Behaviors For Autonomous Characters" (Red3d.com, 1999). The followers follow the leader with one of two movement modes while trying to avoid hitting other individuals or getting in the way of the leader. The leader has three different movement modes and additionally tries to avoid the walls. The program has many adjustable parameters. Some adjust the simulations properties like the speed of the simulation or the number of followers, some change the visuals of the program and some change the properties of the leader and the followers. The program can read files that are in the correct format. The program has an additional editor mode where the user can set the settings, adjust the parameters of the leader and the followers, and save the settings into a file. I think the project was completed at the difficult level.

3. User interface

The program is started by running the Main object and is controlled entirely by the graphical user interface. The simulation resizes to the side of the window. The parameters of the simulation can be altered with the tab pane on the right side of the user interface. In the top bar the user can change the movement type of the leader and the followers, toggle the velocity and force arrows of the followers', and interact with files.

The program can open .csv files that have the correct form. These can be created in the editor. The editor can be accessed from the Simulation options tab by pressing the "Editor" button. In the editor, the user can add and remove followers and move the leader's starting position with a mouse by clicking the editor view as well as change the parameters of the simulation. These settings are saved from the file menu into a .csv file that can be opened in the simulation or the editor.

Small explanation for few of the options:

The leader's wall avoid range is the range where the leader starts getting pushed away from walls.

The arrival range is the range where an individual starts slowing down when approaching its destination. It only affects the leader when it is using the "Seek" movement type.

The followers' vision range is the range at which the followers start avoiding each other.

The User control movement type is used with the w, a, s, and d keys.

Setting the target fps changes how many times the program attempts to update in a second. The rate changes the size of a single update.

The sector force is disabled initially and can be turned on from the Simulation options.

Files can be opened and run again from the menu in the top left corner. In the editor mode they can also be saved.

The layout of the program can be seen in the attachments 1. and 2.

4. Program structure

The program's main two sub-parts are the user interface and the individuals. These communicate with each other through the Simulation object. The individuals are split into the Leader class and the Follower class. These classes use their respective Movement type classes. The structure can also be viewed as a UML diagram in attachment 3.

4.1. The entity package

The Leader and the Follower classes extend the trait Individual. Individuals have information like their location and velocity. They also have the move method. The Leader and Follower classes have methods ending in "Force". These calculate and return the different types of forces that affect the leader or follower. The move method is used to calculate the individual's forces and velocity together and apply them this individual. There are also some helper methods for these calculations.

The Leader and Follower have different options for what is their objective in moving around in the simulation. These are represented by the subclasses of the traits LeaderMovementType and FollowerMovementType. The subclasses have only one method: getForce. It returns a force according to what the objective of the movement type is. Its value is stored with the follower or the leader. This force is applied to the follower or the leader in their move methods.

The VectorVariable class contains x and y coordinates and methods for different calculations that can be used on and between VectorVariables. Instances of this class represent 2d vectors in the simulation space and are used as location coordinates, velocity vectors and force vectors of the Individuals.

4.2. Simulation and settings

The Simulation object contains the individuals, the parameters of the simulation and the editor, and methods to communicate with the individuals. The Settings class contains the simulation parameters that can be stored in a file. Simulation has two instances of Settings. One is for storing the settings of the simulation and the other one is for storing the settings of the editor.

The Settings object has one method which is called updateSettingFromFile. It uses it to set its own values from the FileReader object.

Simulation's method "update" calls the individuals' move methods, so it makes the followers move in the simulation. The updateGoal method moves the leader's goal around. The updateLeader method clears the leader's current information and gives it new information. The addFollower creates new followers into the followers variable in Simulation. Other methods are used in combination to the user interface. The clearEditorSettings is used to reset the settings of the editor and updateIndividualForces is used for updating the individual's forces once so that the simulation can be updated while paused.

The stored values in Simulation and Settings are used by the individuals to know the parameters of the simulation. The user interface updates the values of Settings and Simulation according to user inputs. Simulation also contains the limits to adjustable values.

4.3. Graphical user interface

The Main object is used to run the program and contains the stage and the scene of the program which work as the base for all GUI elements. Keyboard inputs of the simulation are handled by the Keypress

object, which detects the keypresses on the scene. The SimulationGui contains the timeline variable that calls functions inside of it constantly when the simulation is running. Additionally the SimulationGui object contains the simulations rootpane where all of the base user interface elements are. These are the ViewContainer, SettingsPane and TopMenu.

4.3.1. Main simulation view

The ViewContainer object can contain different views which are an instance of either the SimulationPane class or the EditorPane class. These are the main two views of the program. It has the createNewEditorPane method to refresh the editor when it is started.

The SimulationPane class gets its parameters from the Simulation object and draws the simulation based on the inputs of the user. Its variables contain the information of the circles and lines drawn in the simulation that correspond to the leader and the followers. Its methods change these variables. The circles and lines are stored in Groups or as single variables so that they can be observed. The drawFollowerNodes method redraws the circles usually after they are updated by the user. The updatePositions is called constantly to move the circles and lines in the simulationPane. updateFollowerCount changes the number of circles and lines and redraws them. The resetPaneChildren is a helper method to reset the circles and the lines on the SimulationPane. The updateIndividualsFromFile is used when redrawing the simulation based on a file and calls the previously mentioned methods to do so.

The EditorPane class is used in the editor mode as an interactable part of the view. Its variables contain information of the circles and lines shown in the editor that correspond to the leader and the followers. It also contains information about the state of the editor. Its methods are used to add, remove, or change the information of the circles and the lines. These include addFollowerCircle, addLeader, moveLeader, removeFollower and addVelocityArrow. Additionally, the circles and lines can be updated from a file with the updateIndividualsFromFile method. The saveToFile method is used to save the information of the circles and arrows into a file.

4.3.2. Settings side bar

The SettingsPane object is the base for the side bar. It contains at one time either the SimulationTabPane object or the EditorTabPane object. The SimulationTabPane is used while in the simulation mode as a container for tabs. It contains three tabs that are used to change the parameters of the simulation. These tabs are instances of the SimulationTab, LeaderTab and FollowerTab. Its method updateAllSimulationTabs is used to update the information of these three tabs and is used when a file is run.

The EditorTabPane is like the SimulationTabPane as it also contains three tabs, but it is used while in the editor mode. The three tabs are instances of the EditorTab, LeaderTab and FollowerTab. While in the editor, these are used to change the saved parameters that are then written to a file. The updateAllEditorTabs method is used to change the values of the tabs when opening a file in the editor. The createNewChildren method is used to refresh the editor after starting it.

4.3.3. The tabs

The program has four different tab classes in total. The tabs contain GUI elements such as buttons, textfields, labels and sliders which are used to change the parameters of the simulation. They communicate with the Simulation class. The Simulation class contains the values and the instances of the Settings class which the GUI elements change. Also, the limits of the GUI elements are in the Simulation class.

The sliders and buttons in the class SimulationTab set the parameters of the program that are not stored in a file. It contains elements such as a button for changing the simulation's framerate, a pause button, a button to change the number of followers, sliders to change the arrow lengths and a button to toggle the

force that makes the followers avoid getting in front of the leader. The methods in the class carry out these actions and communicate with the ViewContainer, SimulationGui and SettingsPane.

The class also contains a button to enter the editor mode. When this is pressed, the EditorTab communicates with the ViewContainer to change its view to the EditorPane and with the SettingsPane to change its view to the EditorTabPane. Both the SettingsPane and the EditorTabPane are set to their default settings.

The EditorTab contains a button to swap between moving the leader and the followers. It also has a button to return to the simulation. This works similarly to the button in SimulationTab. It calls the returnToSimulator method which communicates with the ViewContainer and the SettingsPane to return their children to SimulationPane and SimulationTabPane respectively.

The FollowerTab class and the LeaderTab class are very similar. They get an instance of the Settings class as their parameter and alter the variables of the Settings with sliders.

4.4. The file package

The file package consists of the object FileReader and the object FileWriter. The FileReader is used to read files. Its method readFile sets the objects variables based on the file found at the filepath given as its parameter or throws an error if the file is somehow incorrect. Its variables contain the information of the read file such as the individual information and the parameters of the simulation. It also stores the possible error messages and information about if the read was successful or not. It also has a helper method parametersAreCorrect to check if the values of the file are in the allowed ranges specified in the Simulation object.

The FileWriter object is used for writing into files. It only has a variable succeeded which contains information about if the write was successful. Its only method is writeToFile which takes in a file to write to, data of the individuals and an instance of Settings. It writes the information of the individuals and the settings into the file specified.

4.5. The top menu bar

The TopMenu object contains menus that can be used to change the movement types of the followers and the leader, toggle the followers' force and velocity arrows and interact with files. For toggling the followers' arrows, it communicates with the Simulation object to change a variable that is used by SimulationPane to check if it needs to draw the arrows. For changing the movement types of followers and the leader, it communicates with the followers and the leader in Simulation and sets their movement type. For interacting with the files. For opening files TopMenu communicates with Simulation to update its settings and then with simulationPane or editorPane in ViewContainer to update their information about the individuals as well as the SimulationTabPane or EditorTabPane to update their tabs. Which one depends on if the simulation is in the simulation or the editor mode. Rerunning the file does the same thing to the most recent file. For saving a file TopMenu communicates with FileWriter.

4.6. The structure

The main program and the GUI are split apart and only communicate through the Simulation object which makes sense. This came at the cost of the Simulation class containing a lot of variables. Another solution would have been to make an additional object that would store all the values so that every class wouldn't have to interact with Simulation. I feel like this would have made the structure a lot more unclear as some classes would still have to access the Simulation object and there would be two objects that many of the classes use.

The actions of GUI elements such as openItem in TopMenu should maybe be in their own methods. I chose not to make them functions because it sounded like a bad idea to have functions that have multiple functions with side effects inside of them.

There are some places where putting the values as parameters of methods or classes might have made the code cleaner and also the program more adaptable to new changes, for example in the updateSettingsFromFile method in the Settings class. I didn't have time to make these adjustments.

5. Algorithms

5.1. Force direction

One of the main algorithms in the project for modeling the movement of autonomous characters mentioned in Craig W. Reynolds' article is *steering_force = desired_velocity – current_velocity* (Red3d.com, 2021). It is used multiple times in many of the different force functions to calculate the direction of the steering force to reach a certain point or get away from a point.

5.2. Space in front of the leader

One hard problem during the project was finding if a point is in an area in front of the leader. This was needed for the followers to avoid getting in the way of the leader. In the end I chose to go with a triangle shaped area because it looked the best and it had good enough performance. For finding if a point is inside a triangle, I used an algorithm that checks if the point is on the same side of all the triangle's sides. The algorithm is based on checking the sign of the determinant of vectors AB and AM where A and B are two corners of the triangle and M is the followers location. If the signs for all the sides are same, we then know that the point is inside the triangle. I chose this method because it was more readable and efficient than my original implementation.

In my original implementation I searched for all followers in a nearby area and calculated their angle related to the angle that the leader is facing. This method was inefficient because of all the calculations of angles and many edge cases. The edge cases were caused by the values that the math.atan2 function returned in specific directions since I needed to check all of the directions around the leader.

Another attempt to find if a follower is inside the triangle was to check if the total surface area of the triangle ABC was equal to the surface area of three sub triangles ABM, BCM and CAM, where M is the location of the follower. I never got this to work despite various attempts.

I also made an implementation for a rectangle area which was similar to the implementation which I went with. It checked if the point was on the same side of two perpendicular vectors. Even though the performance was a bit better, I thought that it did not look as great as the triangle, so I chose the triangle implementation.

6. Data structures

I have chosen to use three data structures: mutable buffers from Scala's collection, observable Groups from ScalaFX and my own data structure VectorVariable. Buffers are used for storing data that is constantly accessed like the list of followers. This is because of my familiarity with them, their mutability and their good performance in accessing indices (Scala Documentation, 2021). I use the Group from ScalaFX to store the values of circles and lines that are drawn in the simulation. I chose the Group data structure because its children variable is an mutable observable list and it can be used easily in conjunction with other elements in ScalaFX. This makes it easier to update the simulation.

The VectorVariable class is used for representing points or vectors in a Cartesian coordinate system. I chose to make my own data structure because it made working with coordinates much easier. Its functions are

useful in operations between two points or vectors. These functions include for example the distance between two points, setting a vectors magnitude to some value, or turning a vector 90 degrees. VectorVariable is immutable.

7. Files and Internet access

The program can open .csv text files with a specific structure. The first line is reserved for user instructions for writing the second line and is ignored by the file reader. The second line contains the parameters of the simulation. The third line contains user instructions about how to write the rest of the lines and is ignored as well. The rest of the lines contain the information of the individuals starting with the leader. These are their x and y coordinates and their velocities in the x and y directions. If the individuals have only one coordinate or no velocities, the rest are filled with zeros.

The file reading requires that the file is in the correct form, has the right amount of parameters, and also that the specified values are in an allowed range. The files can be created in the editor. Attachment 4. has an example of the file structure.

8. Testing

In testing the leader and the followers, I focused especially on the forces that affected them since they were the hardest part to implement. Most of the testing for the leaders and the followers was done by running the program and seeing the effects on the user interface. I chose this approach because it usually gave me instant feedback on if my changes were working as intended. In the earlier stages of the program, I had implemented a test stub of the leader to test if the followers were working correctly by following the leader. Later in the project I made testing classes to see if the forces of the leader and the followers were giving correct values.

Testing of the user interface was handled by running the program and seeing if the user interface would respond correctly and look right. For the file reading and writing I used method stubs to see if the methods were working as intended before I moved on to fully implementing them. The testing process was like what I had planned in my technical plan. All of the tests that I ran passed. Testing with the user interface can be inaccurate especially for finding subtle bugs but for this projects purposes it seemed to work well.

9. Known bugs and missing features

The UserControl movement type has a bug where the leader does not stop when moving backwards when its maximum force is high. This can cause the leader to flip around or keep flipping around constantly which is not intended. I have reduced this problem a lot but at least flipping once is still possible with the leaders force set to the maximum, although it is a bit rare.

The editor does not warn the user if the parameters of the simulation have not been saved. It only does it if the leader or the followers have been altered. This is not necessarily a hard problem to fix and could be fixed if all the elements in the FollowerTab and EditorTab set the “saved” variable inside the EditorPane to false every time they were changed. I chose not to implement it because it did not seem like a very clean solution. Another solution could have been to always prompt the user with a warning when leaving the editor.

The gray area in EditorPane does not always update when it is opened. The reason for this is currently unknown. It is probably a relatively simple fix but requires a better understanding of ScalaFX.

When a file is read, the leader goes missing in the simulation if its location in the file is right on the border of the simulation.

10. 3 best sides and 3 weaknesses

I am happy with how the class and package structure of the program ended up. Before refactorization of the SimulationGUI class, it was very hard to read even for myself. Now it is clear and logical from my point of view.

The VectorVariable class was a good idea for the problem of figuring out how to represent vectors. This made it much easier to do the calculations I needed to do and in case I needed a new way to calculate something, I could add it to the class.

The biggest weakness of my program is the EditorPane class. The class uses the asInstanceOf method many times and has a lot of specific if clauses for distinguishing between the leader and the followers in its variables individualCircles and individualVelocities. These were caused because I was running out of time while making the class and had to get the class working in its entirety.

The classes of the user interface communicate between other classes in the user interface. In my original plan I had only one class called SimulationGui that handled all of the user interface and had implemented it but I later realized that it should be split into many parts. This caused some strange functions with side effects and made the implementation of the user interface a bit messier. I still feel that it was the right thing to do since the original SimulationGui class was very bloated and unclear. The current implementation is much better compared to it.

The sector force never looked that great in the end even though I spent a lot of time on it. I tried many different shapes and implementations and got it to work multiple different ways. The final product was underwhelming but since it was in the project requirements, I left it as a toggleable option.

11. Deviations from the plan, realized process and schedule

In the first two weeks I got a good start and followed the plan I had made, even getting a bit ahead. I created almost all the main classes and their basic functionality. At times made some good changes to the plan where needed like making the VectorVariable class. I also started looking a bit into the user interface and working with threads.

In the next two weeks I implemented more of the functions of the classes as well as threading. At this point I started realizing that making the GUI was going to be a much bigger job than I initially realized. Getting familiar with ScalaFX took a lot of time. I had never made a GUI before, so I did not even know the basics.

During the next two weeks I had exams, so I did not have as much time to work on the project. I worked on the functions and pretty much finished the requirements of the project.

In the last two weeks I knew that I had a lot of time so I could still make some bigger changes and features. This is where I remade the entire GUI structure and at the same time implemented the editor. Remaking the GUI did not take very much time since I already had the needed methods ready. I just needed to spread them to their respective classes. However, making the editor was much more time consuming than I expected.

Making of the editor was a mistake at least at the time I chose to do it. I still had not added all the elements to the GUI and fixed the smaller problems which take some time. I should have made the program complete without the editor and after being ready with the original plan, evaluated if I still had the time to make it. The reason why I did not choose this option was because I thought that making the editor in the end would require much more refactoring. This really cut down on my time to test, comment, cleanup and optimize my code.

I learned a lot during the process of making my project. This was my first big programming project, and it taught me many new things not just about programming but also project management. Things like planning a project, implementing the plan, and finishing the project will be crucial in the future. On the programming side of things, I learned many things from bigger subjects like threading, working with files and working with a GUI in code but also many much smaller details to make my code or the program I am working on look a bit nicer.

12. Final evaluation

I feel like the program is what I envision when I look at the requirements of the program. It fulfills all the requirements and has many extra features that were not mentioned in the original project description.

The class structure is logical to me. The user interface part of the class structure and the simulation part are split well, and the simulation part has a clear API. Most things happen where they should happen. Some methods inside the classes of the user interface could be changed to have less side effects and maybe intake parameters. This was something I did not realize when making the program and I think it would have made the interactions between and inside classes clearer and made it easier to add extensions to the program in the future.

I am happy with my choice of data structures. I could have used the observability and bounding of values in ScalaFX better. These were things that I learned later in the project, so I did not implement them very often.

As a summary, if I were to do the project again, I would focus on making the interfaces of the classes in the user interface clearer, used more of the features of ScalaFX and optimized the program better. Additionally, I would finish the initial plan before starting to add other extra features.

13. References

Red3d.com. (2021). Steering Behaviors For Autonomous Characters. [online] Available at: http://www.red3d.com/cwr/steer/gdc99/ [Accessed 25 Apr. 2021].

Scala Documentation. (n.d.). Performance Characteristics. [online] Available at: https://docs.scala-lang.org/overviews/collections/performance-characteristics.html [Accessed 27 Apr. 2021].

Other materials I used:

Stack Overflow. (2021). Stack Overflow - Where Developers Learn, Share, & Build Careers. [online] Available at: https://stackoverflow.com/ [Accessed 28 Apr. 2021].

Mark Lewis' Youtube channel (2021). Mark Lewis. YouTube. Available at: https://www.youtube.com/user/DrMarkCLewis [Accessed 28 Apr. 2021].

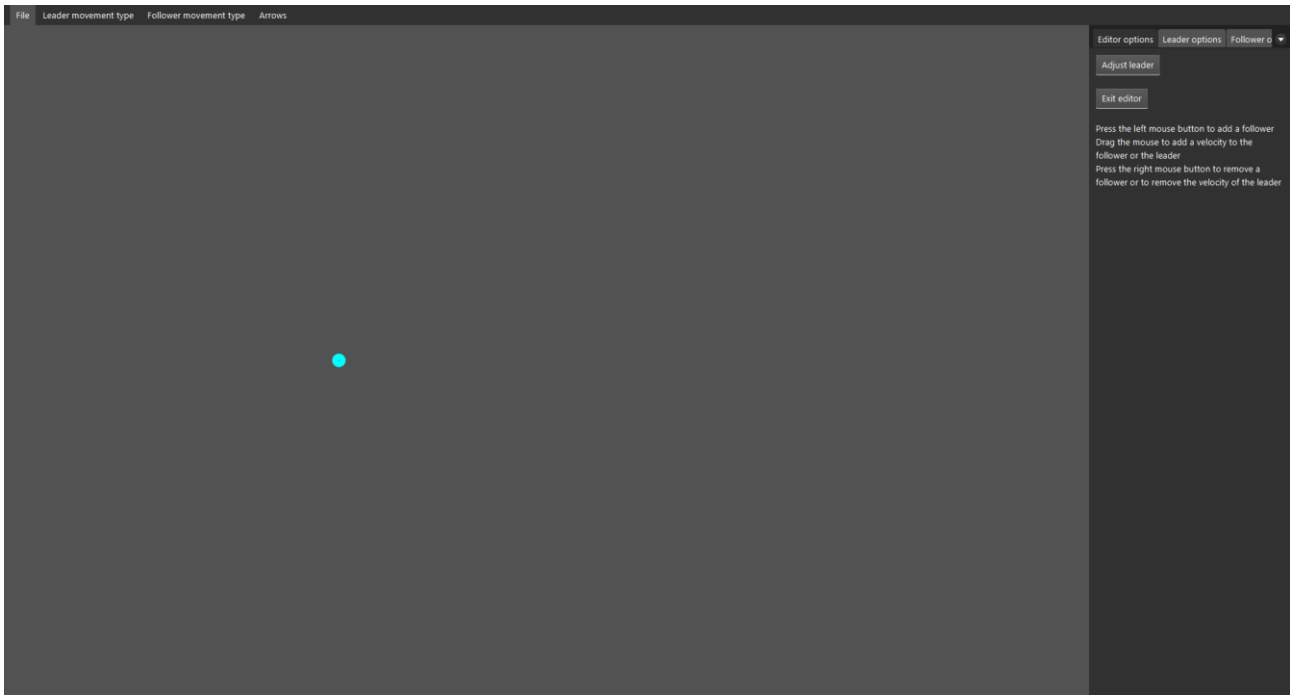
The codin Train Youtube channel (2021). The Coding Train. YouTube. Available at: https://www.youtube.com/user/shiffman [Accessed 28 Apr. 2021].

GeeksforGeeks. (2021). GeeksforGeeks | A computer science portal for geeks. [online] Available at: https://www.geeksforgeeks.org/ [Accessed 28 Apr. 2021].

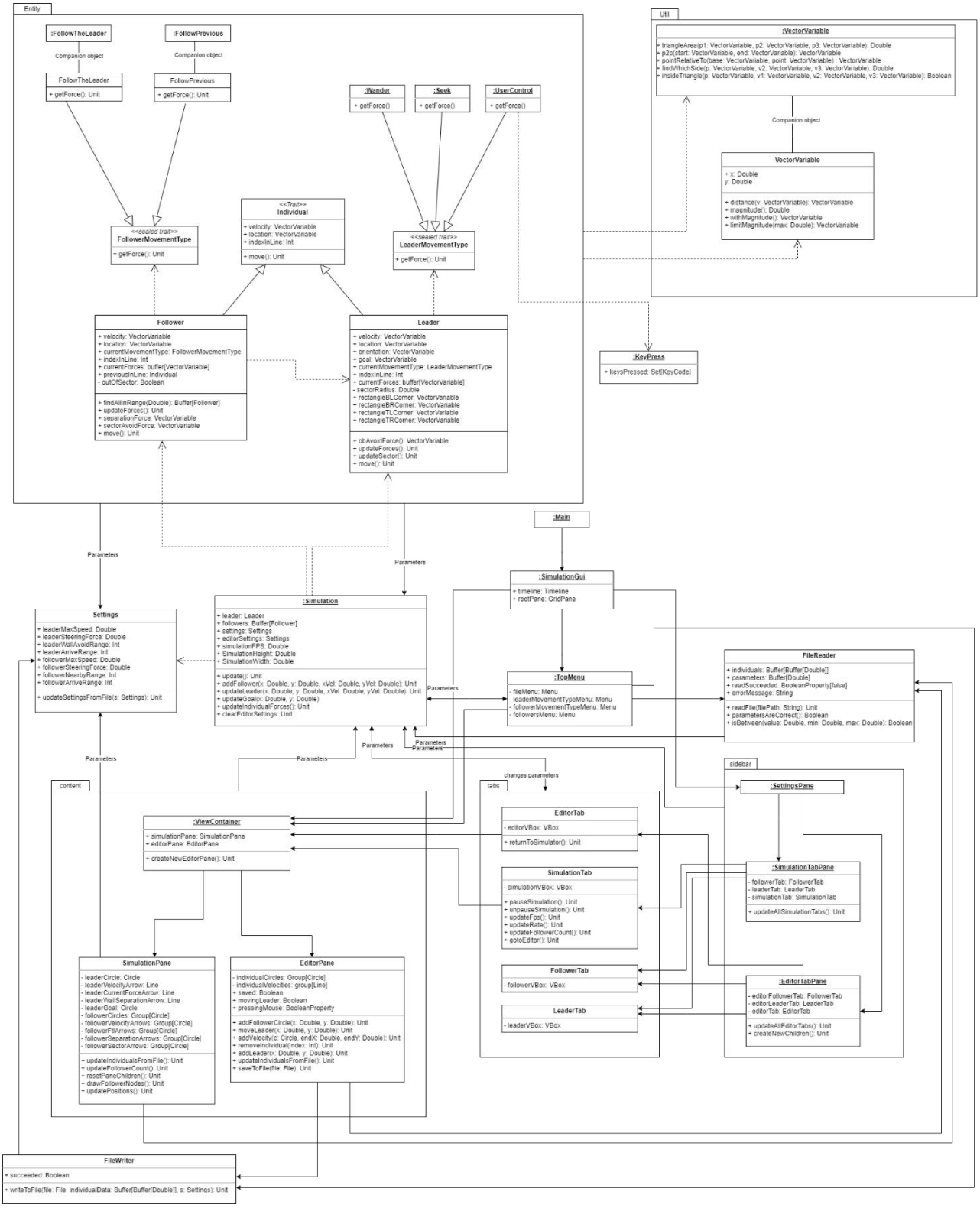
14. Appendixes

1. Picture of the simulation when it is running.

2. Picture of the editor mode.



3. UML diagram of the project structure.



4. An example of the file structure

```
LeaderMaxSpeed, LeaderSteeringForce, LeaderWallAvoidRange, LeaderArriveRange, FollowerMaxSpeed, FollowerSteeringForce, FollowerNearbyRange, FollowerArrivalRange
5.0, 0.5, 50, 50, 5.0, 0.7, 200, 50
xPos, yPos, xVel, yVel -- First one is the leader
500.0, 500.0, 0.0, 0.0
468.0, 253.0, -19.74881263833413, -3.159810022133456
404.0, 255.0, -19.29527642475466, 5.262348115842201
355.0, 270.0, -17.88854381999829, 8.944271909999145
309.0, 300.0, -12.1016653506711169, 15.923243882462032
279.0, 348.0, -5.15325301120663, 19.324698792024947
264.0, 413.0, -0.6662966046527572, 19.988898139583057
263.0, 480.0, 2.9668090586048947, 19.778727057365927
280.0, 556.0, 8.515940726597591, 18.096374044019853
330.0, 628.0, 16.959966080101765, 10.599978800063582
412.0, 668.0, 19.352345447936898, 5.048437942940041
512.0, 689.0, 20.0, 0.0
586.0, 683.0, 18.66691212406124, -7.179581586177392
648.0, 654.0, 13.015827469119358, -15.185132047305956
696.0, 589.0, 4.923076923076906, -19.38461538461536
715.0, 490.0, -0.6892455167956086, -19.98811998707174
704.0, 391.0, -11.384199576606193, -16.443843828875586
642.0, 325.0, -15.726366776448458, -12.35643103863805
566.0, 274.0, -17.662314389148264, -9.383104519234962
494.0, 249.0, -19.90990945187906, 1.89618185255992
382.0, 360.0, -2.828427124746213, -19.79898987322332
400.0, 353.0, 1.1744404390294108, -19.965487463499926
422.0, 349.0, 6.8348612617340905, -18.79586846976872
446.0, 354.0, 13.232432741736943, -14.996757107301846
459.0, 369.0, 16.2135845679976, -11.70981107688715
522.0, 371.0, -9.92277876713672, -17.364862842489174
533.0, 361.0, -3.922322702763722, -19.611613513818384
551.0, 355.0, 3.742242157799865, -19.646771328449518
576.0, 357.0, 10.679859827759628, -16.909778060619443
592.0, 370.0, 17.88854381999829, -8.944271909999145
382.0, 549.0, 18.306300648455306, 8.054772285320382
420.0, 566.0, 19.657443738686425, 3.685770701003662
```