Boids Simulation

Complex behavior arising from simple rules.

|  |  |  |
| --- | --- | --- |
| Huang Yuebin | M21303 | 5th October 5, 2021 |

# Project Overview

Boids Simulation is a program designed to simulate and visualize the concept of emergence, particularly in birds / flocks of birds. Emergence refers to the existence or formation of collective behaviors, what parts of a system do together that they would not do alone. In this program, the birds are parts of the system and the system is the bird population / flocks. Each bird is governed by a simple set of rules for its behavior (how it moves in the map), but collectively the birds behave in more complex ways and can be seen to form patterns and move elegantly across the map. Users are encouraged to play around with bird settings, generate maps or draw their own maps using the tools provided.

To run this program, make sure that the SimulationProject.jar file is in the same folder as default.csv, then double click SimulationProject.jar. Make sure java is installed on your computer as well.

## Features

Here are the features of the program. Key features will be highlighted in blue.

Some terminology used:

* 1. Map: refers to the Canvas object / the area where the birds fly.
  2. MapGrid: refers to the Map represented in a 2D array of Boolean values
  3. Bird: the triangle organism that moves
  4. Wall / Boundary: The areas where if the bird touches it, it dies (i.e. it crashed into a wall).
  5. Smoothen: refers to iterating over the MapGrid and implementing the cellular automata algorithm to “smoothen” the map.
  6. Main Thread: refers to the thread used to update the map.
  7. Vision Cone: The area which a bird can “see”. It is a cone shape, size determined by vision radius and vision angle of the bird.
  8. Adjacent Birds: Birds that are in a bird’s vision cone.

### Splash Screen

* Logo, company name

  Description automatically generatedLogo, company name

  Description automatically generatedThe splash screen features a self-designed logo from scratch that is animated, a loading label that refreshes, and a progress bar. All of these are updated via their own Threads.

### Tooltips

* Text

  Description automatically generated with medium confidenceFor most of the controls included in my program, there are tooltips provided to help make the program more intuitive and user friendly as they are included to explain the usage of the controls.

### A picture containing timeline Description automatically generatedMain Program (BOIDS SIMULATION)

* Number of Birds Slider:
* Slider to change the number of birds alive on the map. It adds birds which spawn in random locations on the map.
* Play/Pause button
  1. Button to pause the simulation of the birds (birds stop moving). This is done by skipping over the movement of birds inside the main thread.
* Next Step Button
* Clicking this button will cause the birds to move for 1 iteration. Done by letting the main thread run the updating of the bird’s movements once.
* Load Simulation Button
* Loads a FileChooser dialog for the user to choose the simulation settings file they want to load. Will produce an Error Alert if the file chosen is an invalid file or has invalid data inside the file. By default, default.csv is loaded during initialization.
* Save Simulation Button
* Saves the current configurations of the simulation, user can choose to include or exclude data such as the appearance, the map settings, and the bird settings to be saved. This is done using checkboxes for the user to have a wider variety of choices when saving a file.

### Graphical user interface Description automatically generatedAppearance

* Map Canvas
* A canvas object instead of a Pane or a Group is used for the map because canvas uses less processing power when rendering all the birds and walls on the map, since they are not rendered as objects but rather as pixels on the canvas. This is important for optimization in case the number of birds increases to 400.
* Theme ChoiceBox
* User can select a User Interface (UI) theme out of 4 preset themes (colors chosen by me). How this is done is that I made a base CSS file with 3 color variables, and the CSS properties of all the controls used is colored with the 3 colors. The 4 themes are additional CSS files stacked on top of the base CSS file, but only the new colors are included inside the secondary CSS files (the CSS files with their theme color). This is done to make the CSS file more readable, as well as more generalized, making the developer’s job easier when he or she wants to set the CSS for a certain control.
* Wall, Background and Bird Color ColorPicker
* Allows the user to set their own wall color, background color and bird color, updated as they choose their color in real time via the main thread.
* Bird Size and Bird Angle Slider
* Allows the user to change the bird’s appearance as much as they would like. Bird size is how big the bird is, bird angle is the angle of the bird isosceles triangle. These changes are purely aesthetic and have no impact on the birds’ behaviors.

### Bird Settings

* Alignment, Coherence, Separation sliders
* Graphical user interface, text

  Description automatically generatedSlider to change the alignment/coherence/separation factor of a bird. Alignment means how much the bird tends to follow the direction of adjacent birds. In programming terms, it means the bird tries to match the other birds’ velocity vectors, provided that the birds are in its vision cone. Coherence is how much the bird tends to move towards the center of birds in its vision cone (tries to match the position vector of the other birds). Separation is equally distancing itself from other birds in a vision cone to avoid overcrowding.
* Vision radius and Vision angle sliders
* Vision radius changes the distance of the vision cone, how far the bird can see. Vision angle changes how far around itself the bird can see, changing the angle of the vision cone.
* Max speed and Max acceleration sliders
* Max speed changes the maximum speed of the birds in pixels traversed on the map. Max acceleration changes the maximum acceleration of the birds and is represented as a percentage of the current max speed.
* Boundary Margin slider
* Changes how much the birds tend to avoid the boundaries/walls, in code it is represented as how many pixels do the birds need to be within the walls to start avoiding them.

### A picture containing diagram Description automatically generatedMap Settings

* Map seed
* This sets the seed for the random function. User can empty the text field and press ENTER to generate a random map seed. This seed is used to randomly choose whether a wall spawns in a certain MapGrid coordinate, dependent on the wall spawn chance. This is included so that when the wall spawn chance is increase, all the previous MapGrid coordinates that were walls remain walls and its not the entire MapGrid wall positions changing, so it looks cooler and nicer when you increase the wall spawn chance.
* Grid square size
* Changes the size of one grid square / MapGrid coordinate square. Effectively MapGrid array will be smaller because the sides are shorter. Larger grid square sizes produce better maps generated from the cellular automata smoothing algorithm (ideally around size 8).
* Wall spawn chance
* Chance of a wall spawning in each grid square / MapGrid coordinate.
* Neighbor threshold
* Used primarily in the cellular automata algorithm. Number of neighbors is the number of neighboring cells which are a wall within a 3x3 grid centered on the cell being evaluated itself. If the number of neighbors is greater than or equal to the threshold, a wall is placed on the current cell, else the cell becomes an empty space.
* Smoothening
* User can increment or decrement a smoothen value. Smoothen value is the number of iterations of the cellular automata algorithm that the MapGrid will go through once the Smoothen button is pressed.
* Map editor tools
* There are 3 tools, fill bucket, brush and eraser, inspiration taken from drawing apps such as Autodesk Sketchbook. If the Flood Fill checkbox is ticked, and the user clicks on anywhere in the map, any cells connected to the cell that was clicked and is of the same value (true or false) will be filled with the current tool selected. If Draw walls radio button is selected, then the flood fill will fill the area with walls. If Erase walls radio button is selected, then the flood fill will fill the area with empty space. Editor tools are implemented by taking the position of the mouse when it is clicked/dragged and then finding the MapGrid coordinate that corresponded to the X and Y positions of the mouse.

## Key Algorithms

### A picture containing light, traffic, dark, night sky Description automatically generatedBoids algorithm

* Namely the alignment, coherence and separation factors is to be calculated each iteration as a vector representing the force vector acting on the bird, which in turn affects the acceleration vector of the bird, which changes the velocity vector and then finally the position of the bird.

### Cellular automata algorithm

* Based off on Conway’s Game of Life, the cellular automata algorithm is also an example of emergence, how complex behavior can arise from simple rules, the rules being how many neighboring cells can be “alive” (in my case is the neighboring cells being a wall).
* A picture containing dark, night sky

  Description automatically generatedThe algorithm goes through every single cell in the MapGrid array and saves its new cell value in a different 2D array based off on the neighbor threshold mentioned earlier. The cell value is not directly set in the original MapGrid array as the cells adjacent to it need its old value to process its own new value. If the value is directly set in the original MapGrid array, there will be a strong diagonal bias in the maps generated and smoothened using the random seed.

### Flood Fill algorithm

* A picture containing logo

  Description automatically generatedInitially, I applied the flood fill algorithm using recursion to recursively call the flood fill function on adjacent cells to process them and perform the flood fill, but due to the stack limit, this was not a feasible idea as I ran into errors if I tried to fill an area that is too large, causing a stack overflow error.
* I changed the way I implemented the flood fill algorithm to using Breadth-First Search (BFS) and a Queue data structure to make use if its FIFO function (First in First Out). The 2D MapGrid array can be imagined as a graph with each cell as a vertex and each edge in the map as an edge in the graph. The edges will all be unweighted.
* The BFS works like this: the first vertex which is the cell that is clicked will be manually added to a Queue. Then you start iterating through the queue and pop the first element in the queue to process it. If the first cell is the same color as the color you are filling right now (call it color C1, which could be true or false, representing the presence of a wall in the MapGrid), then you can stop processing the element and move on to the next element in the Queue. If it’s a different color from C1, it changes its own value to C1, then adds all the adjacent vertexes/cells to the end of the Queue to be processed. Then it goes on to the next element in the Queue. This process is repeated until the queue is empty, then the map is updated with the new MapGrid values. This is an improvement over the recursive method as it is not in danger of a stack overflow error and can be used anywhere on the map without fear of the fill area being too big.

### Avoiding Collision

* As mentioned earlier, the birds avoid collision based on the Boundary Margin. How the boundary margin works is that all the wall edges which are inside the vision cone of the bird is avoided by the bird, except that the vision cone in this scenario is different from the original vision cone. This vision cone is 360 degrees and its radius is determined by the boundary margin. Then the separation function mentioned earlier as well, originally used to separate from birds is used here to separate from walls. The avoiding collision function returns a unit vector representing the force of how much the birds avoid the walls. This vector is higher priority than the other 3-unit vectors (alignment, coherence, and separation) so that the birds would prefer to not crash into a wall and die than follow the adjacent birds.
* A picture containing text, device, gauge

  Description automatically generatedTo optimize my algorithm, every time the MapGrid is updated, I update an array list of wall edges. Wall edges are walls that are adjacent to an empty cell. Wall edges are found by iterating through the walls array list which contain vector positions of all the walls in the MapGrid array. Then this array list of wall edges is passed to the collision function so that each bird only needs to check through a small number of walls every iteration, as compared to every cell in MapGrid or every wall in the wall array list, thus optimizing the code.
* All the vector calculations are done through a Vector class I implemented myself, and not through any physics package. Collision is also not through a physics package but coded by me as well. Having calculations done through a Vector class gives a certain level of abstraction as every time I needed to do a Vector calculation such as scalar multiplication or addition of vectors or getting the magnitude of a vector (used in situations such as calculation of the alignment coherence and separation force vectors) I could just call a function on the Vector object, and I do not need to hardcode calculations.
* Using this type of collision system is better than the standard hit and turn back as birds in real life do not fly headfirst into a wall and turn back as if nothing happened. They swerve to avoid them and if they don’t swerve in time (which happens when the boundary margin is too low or if the max speed is too high and the acceleration is too low) they crash and die in the program.

## MVC design

### Model-View-Controller structure

* Graphical user interface, text, application

  Description automatically generatedThis is the MVC design/structure I used:

### Controller package

* Inside the Controller package, I have the 2 controllers for the 2 FXML files. Splash Screen is the splash/loading screen that starts when the application is opened at first. You can close the splash screen via the X button on the top right. The splash screen uses 3 threads to animate the logo, the “loading…” label and the progress bar. After the progress bar is loaded, it loads the Simulation Screen FXML file, which is the main application.

### Model Package

* In the Model package, I have all the classes for the objects used in the Controller classes. Inside the Organisms package, I have the Organism class that is the parent class of Bird. It has all the properties and functions that an organism in my program would need, for example position, velocity and acceleration Vector fields, distance to another organism function, etc. The Bird class extends Organism and includes the features of a Bird, such as functions needed to calculate the alignment, coherence, and separation unit force vectors.
* The Boids class is the class that is responsible for controlling all the birds and has functions to perform the Boids algorithm on all the birds among other functions. MapGrid controls the state of the map and has the functions to perform the flood fill algorithm and the cellular automata algorithm.
* FileIO is a class used to read CSV files and write Strings to files so that if I must perform file I/O in different classes I can call the static methods from FileIO class to read files and write files. This creates a layer of abstraction as I would only need to be concerned with the content of the data that is extracted from the files and the String that I am writing to the files. Lastly there is the Vector class, which as mentioned earlier does all the vector calculations. It is used for the position, velocity, and acceleration of the birds as well as map positions.

### View Package

* In the View package, there are the 2 FXML files already explained just now. There is also the logo images, this is the main logo of my application:
* Icon

  Description automatically generated with medium confidence
* This logo is designed in Inkscape, all on my own. It shows a bird in flight. The original image file is in the .SVG format as the image is drawn using Bezier curves and not stored in a bitmap format.
* This is the second part to the logo, used in the splash screen to create a simple flapping animation while the application is loading:
* Icon

  Description automatically generated
* There are 5 .CSS files, 1 base CSS file and 4 themes of the base CSS file.

## OOP design

### UML diagram

* A picture containing text

  Description automatically generatedThis is the UML diagram for my application:
* Here is the same diagram but without the controllers and Main.java included for more readability:

A picture containing text

Description automatically generated

### Abstraction

* Abstraction is achieved in my program through grouping various blocks of code together into a function so that the code is more readable, and is easier to debug. One example is the abstraction of certain algorithms like the flood fill algorithm. I just need to pass in the position of the mouse click and what value it is settings the cells to and the function will do its work, I do not need to worry about the contents of the function outside of it if I did a good job implementing the function.

### Encapsulation

* Encapsulation is achieved through encapsulating the Vector class. All Vector functions are encapsulated inside the Vector class and I do not need to perform any Vector manipulation manually outside of the class itself as I can use the methods in the Vector class to perform the calculations, such as scalar multiplication and division, Vector addition and subtraction, getting the angle of a Vector, and getting the magnitude of a Vector.
* It is also achieved via the Stats classes. Certain fields of a class such as bird alignment factor is encapsulated inside this class so that all the birds can have the same alignment factor, as the Stat class is passed by reference and not by value (which some fields are as they are a primitive type like Double).

### Polymorphism

* There is not much application of polymorphism in my program yet, however there are some examples. For instance, the inVisionCone function is overloaded by having 2 of the same functions but with different parameters, demonstrating polymorphism. In future I plan to add more organisms other than birds, such as ants, to further demonstrate the concept of polymorphism as the ant will inherit the organism class.

### Inheritance

* Inheritance is achieved through the Bird Class inheriting from the Organism class, and the BirdStats class inheriting from the OrganismStats class. This is done because birds are organisms, so it is a IS-A relationship.

## Testing

### First round

* For the first round of testing, I try to do weird stuff within my application. To test the robustness of my flood-fill algorithm, instead of filling small wall regions like what someone would normally do, I tried filling the entire map/a large portion of the map with walls which resulted in a stack overflow error like I mentioned earlier, which made me rethink my recursive approach to the flood-fill algorithm.
* When drawing the walls using the draw tool, I tried dragging the cursor outside of the map, which resulted in an ArrayIndexOutOfBounds Exception. This is because the index calculated from the cursor being outside of the map is bigger than the array itself, causing an error. To fix this I performed a check on the validity of the indexes obtained before using the indexes.

### Second round

* In the second round, I tested the File I/O of my program. What if I for some reason chose to edit the saved CSV files? Or if I chose to load a CSV file that is not saved from my application? The program will throw an error but continue to run as if a file was not chosen. So I decided to add an Error Alert to alert the user that something is wrong with the file that they chose to load.

### Final round

* I sent my JAR file to a few close friends for them to attempt to break it, but unfortunately only one of them can successfully install JDK to run my program with my help in debugging why his java version is not the right one. After playing around with the program for a bit, he feedbacked to me that the colors I chose for the walls and background (as well as UI) were too similar and it is hard to tell the difference, so I changed that. He also remarked that it would be helpful if there was more explanation for what each control does, so I included Tooltips for the harder-to-understand controls.
* Another problem is that he is using 1280x720 resolution while I was using 1920x1080 resolution, so the UI design of my application is quite messy on his screen so I had to redesign the size of the UI to accommodate for users using 1280x720 resolution + 100% windows scaling and 1920x720 resolution + 150% windows scaling. A sacrifice had to be made for users on 1920x720 resolution + 100% windows scaling as they would see more empty spaces that are not useful, but it is a compromise I have to make for users on the other screen sizes so that they can still get the full experience.

## Reflections

### Obstacles faced

* One obstacle faced is the difference in resolutions as mentioned just now. I was completely unaware of this problem and that it existed prior to testing my program with my friend. As I cannot be sure of what is the resolution that my user is using, such as the resolution of the device that my teacher is using to grade my project, I had to decide and choose which resolution to design my app for. In the end I chose to design for the smaller screen resolution as if it’s the other way around, the smaller screen resolution users will miss out of controls outside of their screen, but this way the users with bigger screens will only have additional blank space that only affects the aesthetics of the program a little bit.
* Another obstacle is the problem of optimizing my program. If I had used a Pane object or a Group object from JavaFX to render my birds, it would be much slower and take a lot more processing power. So I decided to use a Canvas object and manually draw the shapes using GraphicsContext in Canvas. I also wanted to cut down on unnecessary operations or large calculations, such as the wall edges array list being used for collision instead of the entire wall array list being used, reducing the power needed for computing. Of course, I could just reduce the maximum number of birds available for the user to play around with, but I believe that the full experience of Boids simulation can be achieved with a larger number of birds being simulated at once. (It is much more beautiful in my opinion to have large flocks interacting.)

### Stuff I learnt from the Project

* Firstly, I learned the beauty of nature and basic animals like birds. Birds can be found so commonly around us and after I did this project, I started looking up at the sky more and observing the flocks of birds moving, then compare it to the Boids algorithm. I learnt the concept of emergence and would very likely do projects related to emergence in the future as I find it captivating.
* Secondly, I learnt how to use recursion, a queue and BFS.
* Thirdly, I learnt how to use a lot more JavaFX controls, like slider, accordion, and a menu.

### Improvements to application

* The biggest regret I have is that I did not have time to implement more organisms such as ants, or the concept of a predator and food to interact with the birds because I wanted to refine the birds algorithm and the user-friendliness as well as functionality of the map. If given more time, I would implement a predator and a food source for the birds to have an objective. This could lead to a reproduction system where predators can reproduce to form more predators, and birds could do the same, thus simulating an ecosystem within the map.
* An ambitious goal, likely to not be done with Java and JavaFX but with C# and Unity instead if I find rendering 3D graphics using Canvas and GraphicsContext too hard is to place the birds in a 3D map, where they can fly up down left right front and back. Maybe then I would change the name of the bird to be a fish instead because there aren’t complex cave systems in the middle of the sky.

### Improvements to task

* This is by far my favorite CS task so far since year 1. Mainly because we are allowed to explore our own interests and propose our own projects, where we can be creative in what we want to do and how we implement features.
* In my opinion, I find the other CS tasks/projects to be extremely boring, as they are just applications which manage logistics and are not interesting at all, because I do not ever see myself opening it in the future just to run the program and playing around with it. This makes doing the project a drag. But in this task I chose something I like to do, so I naturally spend a lot more time on it including time I normally use for leisure because I genuinely find it fun to do. (the programming part was fun, not the 4000 word count report part).