JAVA programming Report

Student Exam ID: Y3852394

CONTENTS

1. Assignment abstract

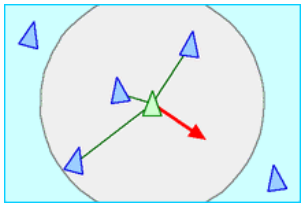
This java assignment requests to design and implement a flocking simulation in Java by means of object-oriented techniques in two dimensional world using Java libraries based on canvas and geometry class provided in labs. Meanwhile, each flock should react with neighbors by means of some simple rules which can cause flocking behavior. Besides, the number of flocks and the number of flock parameters should be able to vary in terms of users' inputs.

To achieve the assignment requirement, I choose to implement boids algorithms to simulate the flocking, which contains three main features including separation, alignment and cohesion, to be introduced in the next concept. At the same time, considering that users could decide the number of flocks and see different behaviors as extension of flockmates change, I use GUI elements including buttons and sliders to build user interface. Apart from these, avoiding predators and collision detection to improve the program complexities and the way of designing this simulation as well as playing illustration are going to be introduced in this report.

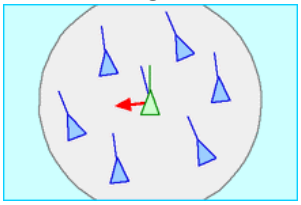
2. Explanation of flocking algorithms

To perfectly accomplish the flocking behavior, boids artificial life program which was created by Craig Reynolds in 1986 is implemented. [1] In natural world, lots of species such as birds, sheep, fish in addition to cows comply with this rule. This action describes the complexity of boids increases following the interaction of individual agents adhering to a set of simple rules which includes **separation**, **alignment** and **cohesion**.

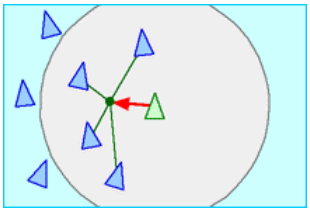
- Separation (Collision avoidance) [2]: Current flock steers to avoid crowding local flockmates. Normally this perception radius is defined smaller than the other two rules, which makes flocking birds aggregate but not crowd.



- Alignment (Velocity matching) [2]: Current flock keeps matching the velocity of neighbours. This helps the heading of flockmates turns to the same direction and the amount of velocity is set to be the same. Therefore, when the bird is in the area with perception radius, it steers towards the average heading of local flockmates but move together with a maximum velocity.

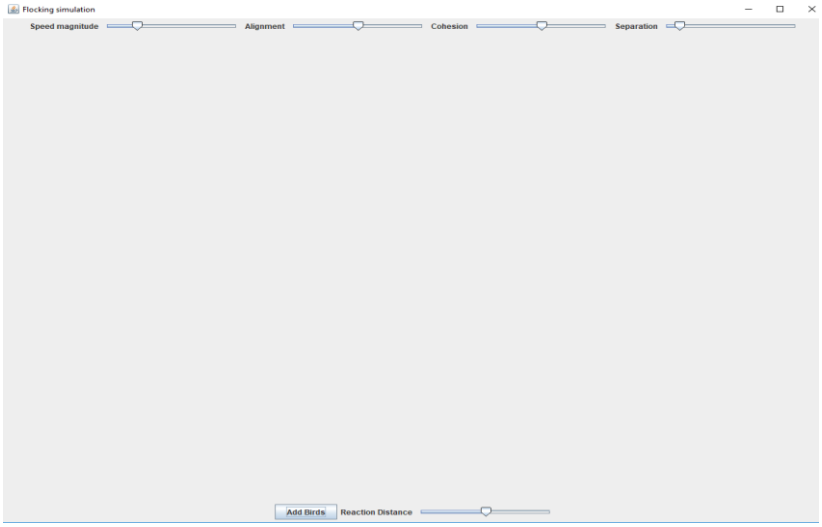


- Cohesion (Flocking centering) [2]: To make flockmates as close as possible. This takes the average position of local flockmates and forbid all steerings to move towards the center of mass. Assuming all birds have the same mass means it should be the average position of flockmates.



3. Explanation of design

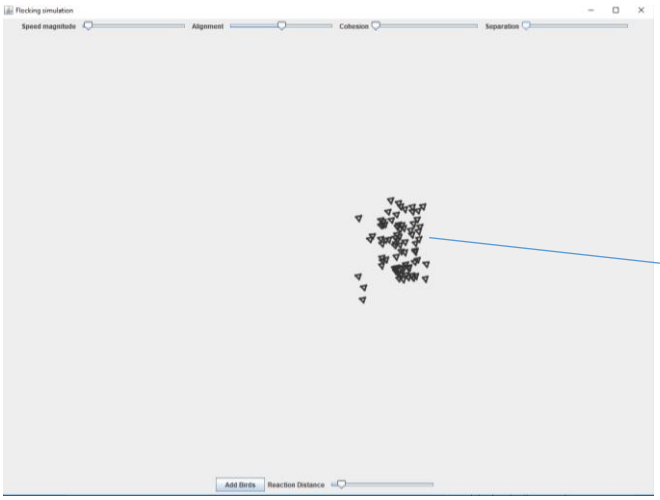
In main GUI interface which is 1200x900 big, users can use button at bottom called “Add Birds” to add 100 birds per time. Meanwhile, the slider next to the button is to control reacting radius from the mouse as a predator to flocking group. Besides, there are four sliders at top. The one at left can control speed magnitude of flockmates. Three remained can control the number of parameter of separation, cohesion and alignment, which influences flocking behaviors.



Main interactive interface GUI where user can change slider and use button

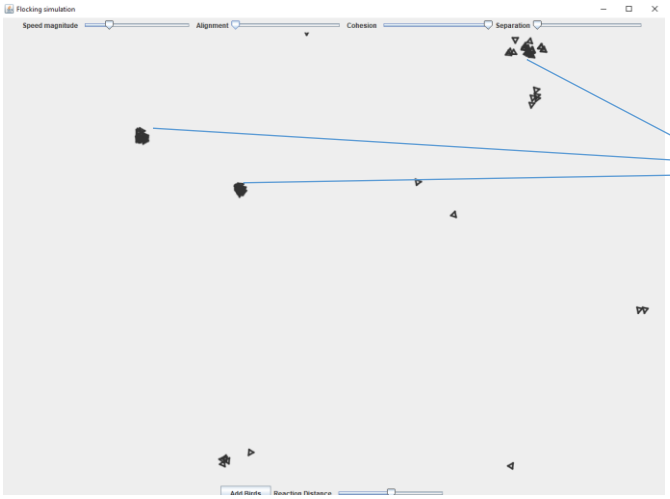
Considering about showing reasonable flocking behavior, initial speed magnitude of flockmates is set as 10 and the slider can adjust it between 1 and 40. Influence coefficients of alignment, separation and cohesion are set from 0.1 to 1 and the initial value of them are 0.5, 0.5 and 0.1 respectively. Screenshots below show the effects of each segment.

To show flocking behaviors more real, this program implements collision detection and predator avoidance. Instead of letting birds leaving edge come back to GUI from another edge, the program is designed to make birds hitting the edges turn 180 degrees in x or y direction to keep them always within the main interface. As for avoiding predator, assuming mouse cursor is the predator, once the distance from the mouse to one of the flockmates is smaller than the number set by slider *Reaction Distance*, flocking group is pushed away from mouse by a force by subtracting mouse position with their flockmates’ location. It is set 100 initially and could be modulated between 100 and 300.



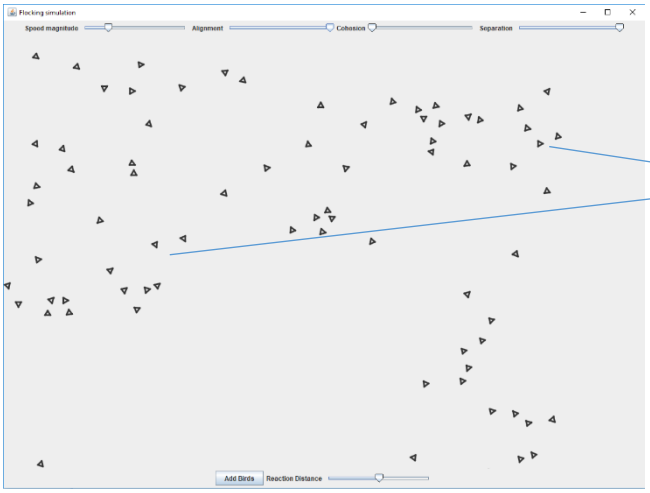
We can see birds flocking turn to move heading forward the average direction.

Interactive interface with alignment = 1, cohesion = 0, separation = 0



Cohesion makes birds move together close to the average position of a group. However, cohesion coefficient was set as 1 after setting alignment coefficient as 1, birds heading to the same direction.

Interactive interface with cohesion = 1, alignment = 0, separation = 0



Different from using alignment to influence flocking only, separation affect flockmates to stay certain distance with neighbors. To show the influence of separation more obvious, alignment is turned on for this reason.

Interactive interface with separation = 1, cohesion = 0, alignment = 1

4. Explanation of program implementation

To implement boids algorithms, three features are imported into DynamicBirds.java file. There are several similarities among these three features when implementing. First of all, *perception* is radius in pixels of a circle and within it, movements of birds should be influenced by neighbours. Secondly, *steering* is a force but assuming mass of every bird is unit, *steering* is regarded as an acceleration. It pushes flockmates to steer certain angles according to three rules and is returned from three methods. Moreover, these three methods all take the same parameter, which is *List<DynamicBirds> birds,* which is the only one group of flocking birds. At last, *count* is an integer variable to store the number of flocking birds and when averaging velocities or locations, it is used as denominator.

Alignment() is a method that returns *steering,* a vector which has direction and amount. This method allows flockmates which are items inside *List<DynamicBirds>* heading towards the average direction with same speed. Thus, in the program, when the birds are close to one another, the speed vectors of each bird are added and then divided the number of birds to get average velocity direction. The amount of the velocity is set to be *maxSpeed*, which can be controlled by the slider at far left. Because the steering force is defined by desired velocity subtracting current velocity of each flocking mates, vector subtraction is used.

Similarly, *Cohesion()* returns *steering* vector. It pushes birds inside *List<DynamicBirds>* towards the average position among all birds flocked. Other than summing birds' velocity in alignment, cohesion adds up birds' current location and then divides it by *count* which is the number of flockmates to get average location. Having calculated the perceived centre, the program applies the same steps as alignment to let flockmates steer different angle towards the average location but limit them to max speed.

However, *Separation()* is quite different from cohesion and alignment. According to how far from each neighbour to our target bird, a proper force *Difference* is generated as a vector, recording the force pushing target away. Because when the neighbour is close to the target, we need a big force, on the contrary, we need a small force, the magnitude of the vector is inversely proportional to the distance of the local flockmate so *Difference* is divided by distance to get the proportion and then we add *Difference* to *steering* force. After getting the proportion, the same step to subtract desired velocity with current actual velocity and then limit *steering* to *maxAcceleration*.

Nevertheless, flocking behaves strangely if cohesion and alignment are both implemented fully to influence flocking because steering force generated by both of them conflicts with it generated by separation. To solve this, by looking at pseudocode of boids algorithms, [3] a small portion is multiplied by the *steering* vector. Hence, *steering* in alignment is multiplied by an eighth and in cohesion, related to distant goal, moves boids 1% of the way towards the goal at each step.

Flocking() method is imported into DynamicBirds.java file. It is used to multiply three steering forces with coefficients controlled by sliders and then add them together to compute the total acceleration

in order to calculate the next velocity of movement according to *Newton's second Law* [4].

Because the acceleration, velocity and location of flockmates are all controlled by two variables following x and y direction, it is more convenient to declare them as vectors in program so that I extend CartesianCoordinate.java to include vector calculation such as addition, subtraction, multiplication etc, instead of calculating variables in x and y separately. Methods doing vector computations are shown in the sheet below.

Methods Name	**Parameters**	**Description**
Public void VectorAdd ()	CartesianCoordinate v	For *v.VectorAdd(v),* update variable by adding x,y parts of vector v to them of variable(x,y) respectively.
Public void VectorSub ()	CartesianCoordinate v	For *v.VectorSub(v),* update variable by subtracting x,y parts of vector v to them of variable(x,y) respectively.
Public void Division()	Double value	For *v.Division(value),* update variable by dividing x,y parts of variable by value in parameter.
Public void Multiplication()	Double value	For *variable.Multiplication(value)*, update variable by multiplying x,y parts of variable with value in parameter.
Public double Magnitude()	/	For *v.Magnitude()*, return magnitude of v using distance equation: $\sqrt{x^2 + y^2}$
Public void normalise()	/	For *v.normalise(),* update v by dividing x,y parts of v by the magnitude of vector v respectively. Normalizing a vector using equation: $\frac{(x,y)}{\sqrt{x^2+y^2}}$
Public void limit()	Double max	This method is used to limit birds to have the same speed when they join the flocking. *v.limit(max)* should change x,y parts of v according to the radio of magnitude
Public double heading()	/	For *v.heading(),* calculate inclined angle between x and y direction of v.
Public static CartesianCoordinate SubVectorBetween()	CartesianCoordinate v1 CartesianCoordiante v2	Return a vector by subtracting two vectors v1, v2 using vector subtraction.

To realize wrapping position, *wrapPosition()* method in Birds.java is implemented. *wrapPosition()* takes *JFrame j* as the parameter. It uses vector multiplication by -1 on *speed* along x or y direction when hitting horizontal or vertical edge respectively and then *myLocation* is updated by adding *speed* vector. As for avoiding predator, *avoidPredator()* in DynamicBirds.java is designed to let all flockmates avoid the mouse. *avoidPredator()* takes *List<DynamicBirds>* birds as parameter. Due to similar principle to *Separation()* method, after acquiring mouse location which is stored in *mouseLocation*, a vector *steering* pointing inversely to *steering* is vested and updated only save the proportion according to the distance between mouse and every bird in x and y direction. Next, *steering* is limited to be max acceleration and added to *speed*, affecting flocking movement to move towards the same direction together.

5. Assessment and comments

To satisfy assessment requirement, multiple classes including extended class, compositions and inheritances are designed so that different classes in different packages can use methods and various types of variables such as *public, private, private static, protected* are written properly in case of errors. Simultaneously, polymorphism is used to call and draw several birds with the help of *List*. The interface is organized by GUI segments involving printed letters, button, sliders, panels and frames two classes.

Regarding to the shortcomings and parts to be developed, although the program implements avoiding predator, the predator cannot break up the flock, which shows flockmates react insensitively. To use steering force to control the speed cannot affect single bird's movement in the flocking group, which may cause certain bird to move closer to mouse and once this bird is inside the circle with the reaction radius, it turns to keep away from the mouse. To improve this, we can break the cohesion of single bird to negate it, which could make the boids move away from the centre of mass.

Apart from this, when the number of birds is increasing up to 1000, the program is possibly to be stuck due to memory storage and release. Some birds are stuck and start flashing, which can be a problem, affecting to watch the phenomenon of flocking. Finally, the size of interface has been settled so that any enlarging or shrinking the window can cause birds out of edge and won't come back to the main window.

To solve the other two problems above, in the future, because there is no need to reserve all locations of birds in the list and the memory is able to be released every time a bird moves. Moreover, as regards the resize, locations of items should be changed proportionally to the altered edges when interface is resized.

6. References

[1] *Wikipedia, the free encyclopedia. 2019. Boids. [ONLINE]*

Available at: https://en.wikipedia.org/wiki/Boids [Accessed 3 March 2019].

[2] *Craig Reynolds. 2001.Boids Background and Update. [ONLINE]*

Available at: https://www.red3d.com/cwr/boids/ [Accessed 3 March 2019]

[3] *Kfish.org. (2019). Boids Pseudocode. [ONLINE]*

Available at: http://www.kfish.org/boids/pseudocode.html [Accessed 15 March 2019]

[4] *Wikipedia, the free encyclopedia. 2019. Newton's law of motion. [ONLINE]*

Available at: http://en.wikipedia.org/wiki/Newton's_laws_of_motion [Accessed 29 March 2019]