香港城市大學
City University of Hong Kong

Department of Electronic Engineering

FINAL YEAR PROJECT REPORT

BEngECE-2008/09-<GRC>-<05>

<Delay Feedback Control of Flocks>

Student Name: Hu Ziyan
Student ID:
Supervisor: Prof. Ron, Guanrong Chen
Assessor: Dr. Wallace, K.S. Tang

Bachelor of Engineering (Honours) in
Electronic and Communication Engineering (Full-time)

Student Final Year Project Declaration

I have read the student handbook and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I declare that the work submitted for the final year project does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Student Handbook.

Project Title :

Delay Feedback Control of Flocks

Student Name : Hu Ziyan

Student ID:

Signature

Date :

Abstract

It is natural that we see flocks of birds flying over the sky when parents tell us that they are heading home. From the perspective of researchers, the seemingly random yet ordered formation is a fantastic and amazing phenomenon for them to investigate on—what we call—flocking.

Recent years a surging interest has been raised into the control of flocking behaviors. In this project, graph theory, complex network and multi-agent systems such related theories on flocking agents, systems and algorithms are studied. An algorithm to formulate flocking behavior is with a time delay is presented. Computer simulations incorporating separation, alignment, cohesion as well as obstacles avoidance and target following are carried out. The simulation result is discussed and some further development on the flocking algorithm is proposed.

Acknowledgements

I would like to take this opportunity to give my deep gratitude to people who have helped me throughout the year on my final year project.

I here express my earnest gratitude to my supervisor, Prof. Ron Chen, for his kind guidance, understanding support, and generous encouragement not only on my project but also on my study throughout the whole year.

I would also like to extend my appreciation to my professors in the department: Dr. Andy Chan, Dr. W. S. Chan and my assessor Dr. K.S. Tang, who gave me critical directions on the simulation result. They were all willing to spare time for me.

Further, special thanks to Dr. Housheng Su. He helped me out when I had difficulties in simulation.

A lot of my classmates and friends put inspirations into this project, to name a few: Shiyu, Allen and Queenie. Thanks for your accompany.

Finally, I owe a warmest gratitude to my parents, who have always believed in me and been there for me unconditionally.

Contents

List of figures and tables

Chapter 1 Introduction

1.1 Problems Formulation

Flocking behaviors impress people as the flocking agents can have a beautiful formation that steer generally toward the same direction with the same velocity. Inside the group there seems to be a central controller adjusting their positions and headings even though one bird does not know where the group is heading to.

Ever since Reynolds in 1986 demonstrated a computer simulation of flocking, computer scientists and mathematicians have been trying to analyze and model flocking behavior. One issue is raised and answered in this project: how to formulize the properties of flocking behavior to simulate the flocking model. The controlling factor is regulated by the acceleration of single agent; one general way to get that is by calculating a weighted average of the information from its neighbors; interactions between the agents form a feedback loop in between every two agents and every neighboring group.

There is a time delay when the information is being transmitted on the way. What is the effect of having this delay and what is the difference between a delayed feedback and an ideal feedback are also investigated in the project.

The simulation tool for analyzing and modeling the group formation also need to be agent-oriented. Java can be a good programming language for simulating the dynamic behavior of flocking agents. This project presents a computer simulation for an intuitive understanding of the flocking behavior.

Another interesting scene is that when the flocking agents meet with obstacles, they can steer away from the obstacles and form the collective formation again. In the Aquarian we often see fish schools take a detour around the corallites. In the simulation, circular entities acting as

obstacles are also added to make a lively animation.

Although flocking agents only have local perception, in real life there are usually several older members that lead the others toward some destination. A rule for setting up leader is created; the resulting flocking behavior can be more organized.

1.2 Motivation and Objective

Flocking happens in real life, I was intrigued when first introduced to the idea of mimicking the behavior live-beings in the natural world. With the fast development of the study on flocking behavior, the applications based on flocking are ever flourishing. The computer simulation built in here can be further developed and even applied into controlling of unmanned robots and computer games and movies.

My objectives on this final year project are:

1. To study theories, properties and application of complex network, multi-agent system and flocking behaviors.
2. To understand how to analyze the characteristics of single agent and local perception to control the group behavior.
3. To learn Java language to implement the concept to simulate the flocking behavior and carry out computer simulations.
4. To analyze the simulation result to get a deeper understanding of the flocking behavior.
5. To investigate on the possible improvement of the flocking algorithm and future development.

1.3 Organization of Report

This report has in total seven chapters, after introducing the flocking problem in this part, related theories on graph, complex network, complex systems and swarm intelligence are addressed in Chapter 2. In Chapter 3, the detailed flocking algorithms adopted in this project are explained in the form of formulae and pseudocode. The simulation result of flocking behavior is presented in Chapter 4, followed by analysis and investigation in Chapter 5. Chapter 6 came up with the possible improvement of the flocking behavior control by means of the potential function and the further tweaks in the algorithm. Lastly, Chapter 7 gives a conclusion of the project.

Chapter 2 Related Theories

2.1 Graph Theory

To study the interaction relations of the flocking agents and to simulate the flocking behavior, a mathematical tool is needed to describe and their properties. Theoretically, the study of graph can abstract flocking agents and their interactions into dots and lines connected.

A **graph** G (V, E, φ), is a collection of set $V(G)$ and set $E(G)$ with an **incidence function** φ, where $V = (1,2,\ldots,n)$ is the set of **vertices** (or somewhere called nodes/points), $E \in \{(i,j): i,j \in V\}$ is the set of **edges** (or somewhere called lines/links). The number of vertices ($|V|$) and edges ($|E|$) are called **order** and **size** of the graph respectively. If two vertices i and j are joined by an edge e, φ is represented as $\varphi(e) = ij$, i and j are called the **ends** of edge e. Each edge has two ends, for the self-connected edge, the two ends coincide, i.e. $i = j$.

If a graph is directional, (i, j) denotes an arrowed edge pointing from vertex i to vertex j, which means that vertex j can obtain information from vertex i, but not vice versa. In this project, edge (i, j) indicates that flocking agent i and agent j can obtain information from each other, which makes the graph undirected; and the study focuses on their connectionism rather than their positions or shapes.

Graphs can also be represented by drawing, the graphical representation helps understanding the properties of graphs.

Two undirected graph G_1 (V_1, E_1, φ_1) and G_2 (V_2, E_2, φ_2)

$V_1(G) = \{ v_1, v_2, v_3, v_4, v_5 \}$, $E_1(G) = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7, v_8 \}$

$\varphi_1(e_1) = v_1 v_2$, $\varphi_1(e_2) = v_2 v_3$, $\varphi_1(e_3) = v_3 v_3$, $\varphi_1(e_4) = v_3 v_4$,
$\varphi_1(e_5) = v_2 v_4$, $\varphi_1(e_6) = v_4 v_5$, $\varphi_1(e_7) = v_2 v_5$, $\varphi_1(e_8) = v_2 v_5$,
$V_2(G) = \{ v_1, v_2, v_3, v_4, v_5 \}$, $E_1(G) = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7, v_8 \}$
$\varphi_1(e_1) = v_1 v_2$, $\varphi_1(e_2) = v_1 v_1$, $\varphi_1(e_3) = v_2 v_3$, $\varphi_1(e_4) = v_3 v_4$,

$\varphi_1(e_5) = v_2 v_4$, $\varphi_1(e_6) = v_3 v_4$, $\varphi_1(e_7) = v_1 v_4$, $\varphi_1(e_8) = v_4 v_5$,

According to the incidence function, with edges and represented by lines and dots, two diagrams can be drawn to indicate G_1 and G_2.

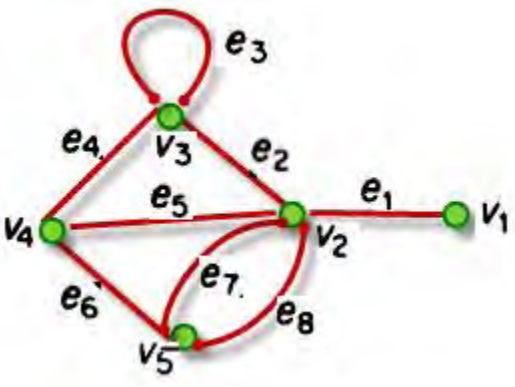


Figure 1.1 diagram of graph G_1

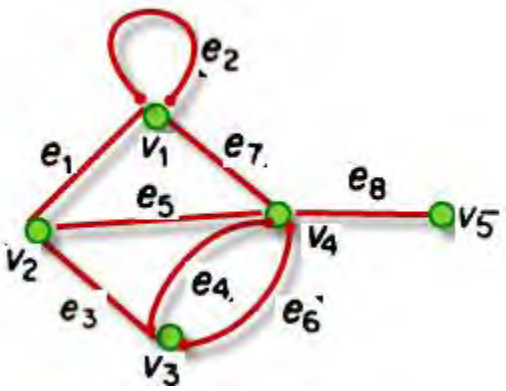


Figure 1.2 diagram of graph G_2

The two graphs are the same with only different **labels** (assignment of numbers on edge and vertex), but they are not identical since their incidence functions are different. Therefore it is not necessary for same diagrams having identical graphs. For the graphs G_1 and G_2 we call them **isomorphic**, written $G_1 \cong G_2$. A graph without any labels can represent a class of isomorphic graphs.

Edge e_3 is called a **loop** for its ends coincide, there are 4 (loop counted as 2) edges going through vertex v_3, this number is the **degree** of that vertex. A **path** is a list of vertices from which there is an edge connecting them as a sequence. The **distance** between two vertices is the number of edges that belong to the shortest path of two nodes. An **average path length** is the average value of the distances between any two vertices. $G(V, E, \varphi)$ is said to be a **connected** graph as there is at least one path between any two vertices.

2.2 Complex Network

"*A network is essentially anything which can be represented by a graph*"[1]

The study of network is part of graph theory; a network is same as a graph with a set of nodes connected with weighted edges.

In this project, a computer simulation of a flock of birds constitutes a network where the birds are the vertices and their interactions are weighted edges. Their arrangement and configuration make up of the network topology of the network.

2.2.1 Random Graph Theory

There are no strict definitions for complex network, one fundamental model is the random graph theory raised by *P. Erdős* and *A. Rényi*. The ER random graph states that for a graph *G* (*V, E,*), where $|V| = \mathrm{N}, |E| = \mathrm{M}$, the probability of any two vertices being connected by one edge is *p,* therefore the expected number of total vertices in the graph is $p[\mathrm{N}(\mathrm{N}-1)]/2$. The probability of having a graph with N vertices and M edges is

$$p(G_{N,M}) = p^{\mathrm{M}}(1-p)^{\frac{\mathrm{M}(\mathrm{N}-1)}{2}-\mathrm{M}}$$

One of the important findings is that the many significant properties of the ER model are emergent. If when $\mathrm{N} \to \infty$, the probability of one ER random graph having certain feature *Q* is 1, then almost every ER random graph contains this feature *Q*.

2.2.2 Complex network

Random graph opened up a new epoch in the study of complex network, people found out that in real life, most of the networks like the World Wide Web and the flocking agents in this project are neither regular nor following certain distribution like random graph. Complex network is

[1] Guido Caldarelli, Alessandro Vespignani, "Large Scale Structure and Dynamics of Complex Networks: From Information Technology to Finance and Natural Science", World Scientific, 2007 p2

distinguished from regular network and random graph in its structure diversity, connection diversity, dynamical diversity and network evolution. The complexity can be reflected in the following aspects: the complexity of structure, the complexity of nodes and the complexity of different impacts and interactions.

2.2.3 Properties of complex network

Despite all the complexity, many real complex networks do share some common features:

Clustering: Take social network for example, the clustering characteristics are like you have two friends A and B, it is possible that A and B know each other too. In real network, this probability of connecting will tend to be a constant value as the scale of the network enlarges, which means that some real complex network is not completely random, but follow a "Birds of a Feather Flock Together" property.

Small world: In 1960s, S. Milgram[18] did a famous experiment to test the probability of each node is connected to another node. He came to a „Six Degree of Separation" that in the world anyone can be related to any other person with no more than 5 intermediaries. Subsequence researches are proving a fact the average path length between two nodes can be very short even though the complex network contains a huge number of nodes.

Scale free: The degree of the nodes follows a power-law distribution, which means in a large scale complex network, most of the nodes have relatively small degrees with only a small number of nodes having large number of degrees. Without those nodes having small degrees the network can still work.

2.3 Complex System

„A *complex network forms the backbone of a complex system*“[2]

The study of complex systems focuses on the complex features exhibited by elements in the system. Several features of the complex system are:

1. Complex system has a large number of interacting elements, and their states are dynamic
2. The agents inside the system have direct or indirect feedback circle
3. Interactions are non-linear, which means the impact is more than assembling several parts together, superposition no longer applies. Sometimes a small variation may cause a large effect.

2.3.1 Intelligent agent

The study and simulation of an agent oriented system is the kernel topics in this project. We generally consider an agent as „intelligent“ if it holds the following properties:

1. Reactivity: Sense the change of the neighbors, and act accordingly, affects the neighbors at the same time
2. Autonomy: Control the self-behavior voluntarily
3. Proactivity: Perform prediction of the movement
4. Mobility: Move fluidly and smoothly

Intelligent agents are also called autonomous agents, or even intelligent autonomous agents. Figure 1.3 sketches a simple relation in between intelligent agents and with the environment.

[2] http://www.vs.uni-kassel.de/systems/index.php/Complex_Network

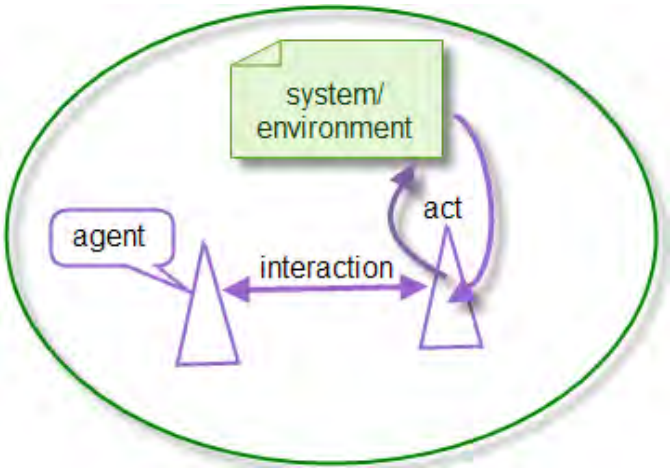


Figure 1.3 perception of an intelligent agent

2.3.2 Multi-agent system

The study of multi-agent system (MAS) helps understand the discrete system in flocking model. Agents inside MAS are decentralized, i.e. there is not any centralized control. In this project, the many interacting intelligent agents communicate directly or indirectly with only local perceptions and interactions make up for a multi-agent system. The system as a whole shows a collective movement behavior.

2.4 Artificial Intelligence (AI)

People hear about the term AI all the time, concisely speaking, Artificial intelligence is a integrated science of designing machines that can have the ability to think like human beings.

2.4.1 Swarm Intelligence

Swarm intelligence (SI) is a branch of computational artificial intelligence, which is the intelligence of living organisms, such as bird flocks, animal herds, and fish schools. Their movement behavior can be simulated by computers model; representative algorithms include flocking ant colony optimization, and particle swarm optimization etc. Current applications

involve computer games, unmanned robots, routing and searching. The designing of agents with intelligence, which is the main focus in SI, is adopted in this project.

2.4.2 Flocking Behavior

Flocking is perceived in this project the emergences of the collective behavior of individuals. Reynolds summarized that flocking, which he described as a "general class of polarized, non-colliding, aggregate motion"[3] of a group of individuals, can be created as artificial swarms. He came up with 3 heuristic properties that flocking agents obey

1. Separation-steer to avoid collisions with agents in the neighborhood
2. Alignment-steer to match the velocities with agents in the neighborhood, thus make the whole group head toward the same direction.
3. Cohesion-steer to stay close with agents in the neighborhood

The three rules led to the first computer simulation of flocking behavior. In this project, the flocking rules are majored based on Reynolds three rules, with formulae implemented.

2.5 Interim Summary

Based on literature review, the essential fundamental theories are presented in this Chapter.

A good understanding of graph and network knowledge is essential. To simulate flocking behavior, the birds flock is abstracted as a multi-agent system, in which the bird has got the property of that of an intelligent agent. The common as well as distinct features of the complex system and swarm intelligence are implemented in designing and analyzing the simulation.

[3] Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.

Chapter 3 Flocking Algorithms

3.1 Overview

Separation, alignment and cohesion describe the general movement behavior of flocking agents, to apply these rules, a more clear and accurate mathematical statement is needed. In this project, the algorithm is an implementation of the three rules with two additional features added.

Suppose there are N agents moving in a 2 dimensional space, it is a multi-agent system. For each agent: position $\begin{cases} \dot{p}_i = v_i \\ \dot{v}_i = a_i \end{cases}$, where p, v, and a are the position, velocity and acceleration of agent i respectively, in which acceleration is the control. In this project the change of positions and velocities doesn't use differentiation, but just subtraction as discrete time is used.

Acceleration is the weighted summation of the component steering vectors generated from by Reynolds rules and two other features—obstacles avoidance and target following.

The velocity of the flocking agents is updated by change of acceleration and the position is updated by the change of the velocity.

3.2 Delayed Feedback

"*Feedback is a circular process of influence where action has effect on the actor*"[4]

While moving, the flocking agent passes its own information—velocity and position onto other agents in the neighborhood to influence its neighbors on their movement variation, at the same time the force acted on itself was calculated based on the neighbors' information. This makes up of a feedback circle.

The next time position of the flocking agent is determined on information feedback of its neighbors. There is a time delay when this information is transmitted from the neighbors to the agent. Therefore a delay time τ is added into the information of the neighbors, which means that

[4] http://necsi.org/guide/concepts/feedback.html

with a delayed feedback the sub-accelerations of agent *i* at time *t* is calculated not based on the positions and velocities of the its neighbors at time *t* but rather at *t*- τ. τ is the multiples of the step time Δt.

Another intuitive way to understand delay is that when the flocking agents move, they observe their neighbors, and the positions of the neighbors have a retentive image.

In this project, delay is implemented into Reynolds‟ rules but not into the two additional features.

3.3 Neighbors

3.3.1 Neighborhood definition

If a flocking agent has infinite field of sight, the system has got global control, the flocking formation can quickly form an entity and steer collectively. But in reality, one bird cannot see all the others in the whole group; it can only sense the few around.

In order to control the whole system using individual‟s local perception of the environment, every agent must be aware of who are its neighbors. Moreover, the position of the agents will change every time step, the agents need to check the neighbors once through the loop. The neighborhood is defined as a function of the radius r and field of perception α shown on figure … The circular sector is defined to be the neighborhood of agent 1.

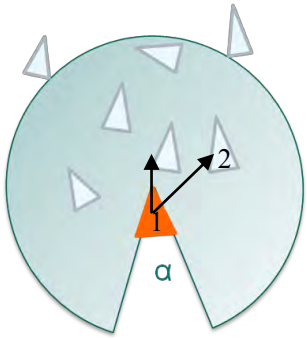


Figure 3.1Neighborhood

Pseudocode of determining the neighborhood

```
for every two agents a and b
   dist(a,b) = distance between agent a
   and b
   ϒ=angle between heading of a and dist
   vector
   if ( 0<dist(a,b)<radius && angle < ϒ )
       b is in the neighborhood of agent a
```

3.3.2 Field of sight

The magnitude of angle α in figure has different impact on the result. If the field of sight (360° - α) is an acute angle, the flocking agent has got a wide view. It can sense the neighbors from behind except a small part that is exactly behind its back. When the α is larger than π, i.e. is an obtuse angle, the flocking agent has got a narrow view and can only sense its neighbors in front of it. In the simulation the flocking agents have a wide field of sight, because using narrow field of sight the flocks may form a line formation which doesn't exactly obey the flocking rules. It is true that birds have their eyes on each side of their heads therefore can sense a larger group of neighbors.

3.3.3 Neighborhood size assignment

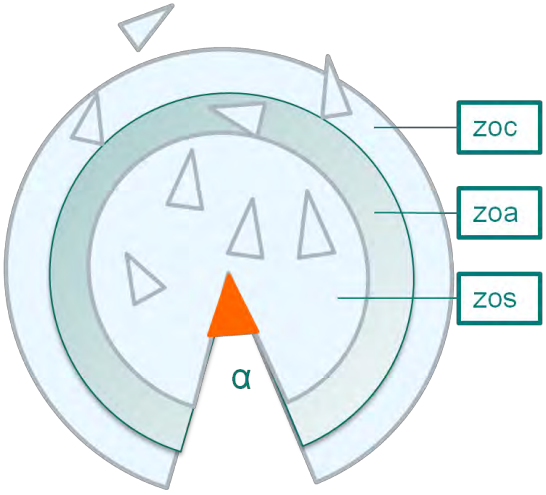


Figure3.2 Neighborhood size assignment

Zone of Cohesion (*ZOC*) is set to be largest, in this project 150. Agents tend to steer collectively with a larger group.

Zone of Separation (*ZOS*) is set to be smallest, in this project 50. It is understood that agents only have repulsion force with the closest neighbors.

Zone of Alignment (*ZOA*) adjusts the headings of the birds, in this project is set to be 100. There

are no strict rules for setting the zone radius in program simulation. But it is better we get a state where the agent doesn't apply separation or cohesion.

3.4 Flocking Rules

A simple algorithm to implement Reynolds rules is to steer toward the average positions and head toward the average headings. Separation and cohesion change the positions while alignment changes the headings. Acceleration is gernerally obtained by average sum with minor tweaks added for different rules.

3.4.1 Separation

Separation is the tendency to maintain a certain minimum distance between flocking agents. For every agent, a steering force is applied onto it to avoid collision with its neighbors.

The desired direction $\vec{d}_i^{sep}$ is the sum of the distance vectors between the agent and its neighbors, pointing away from the neighbors. And the summand vector is set to be inversely proportional to the magnitude of the distance vector in this project. We want the force to be larger when the two agents are closer, to better prevent colliding. $\vec{d}_i^{sep}$ is given below:

$$\vec{d}_i^{sep}(t+\Delta t) = \frac{-\sum_{j\neq i}^{j\in zos} \frac{\vec{p}_j(t-\Delta t)-\vec{p}_i(t)}{\left|\vec{p}_j(t-\Delta t)-\vec{p}_i(t)\right|}\frac{1}{\left|\vec{p}_j(t-\Delta t)-\vec{p}_i(t)\right|}}{N_{j\in zos}} \qquad (3.4-1)$$

The desired velocity $\overrightarrow{dv}_i^{sep}$ is the unit vector of desired direction multiplied with a fix parameter. This way any agent has the same desired velocity. The reason for doing this is that neighboring agents only perform the duty of giving directions to the agent; the actual acceleration is determined by the agent itself. Another explanation is that we want to make the animation stable and smooth, even though agents have different neighbors, the acceleration acted on them are almost the same as each other. Therefore we have:

$$\overrightarrow{dv}_i^{sep}(t+\Delta t) = \frac{\vec{d}_i^{sep}(t+\Delta t)}{\left|\vec{d}_i^{sep}(t+\Delta t)\right|} \times speedparameter \qquad (3.4-2)$$

Pay attention here the steering force is the desired velocity subtract the velocity vector of the agent, the steering force is actually the acceleration, which is obtained by the velocity difference. The separation steering vector can be represented as

$$\vec{s}_i^{sep}(t+\Delta t) = \overrightarrow{dv}_i^{sep}(t+\Delta t) - \vec{v}_i(t) \qquad (3.4-3)$$

Separation has the highest priority of the all the rules, we don't want flocking agents to bump into each other. This is adjusted later in the weighted summation.

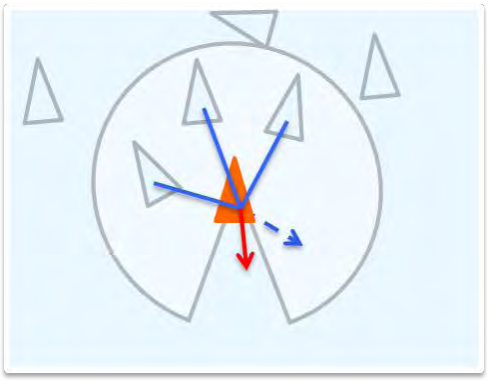


Figure 3.3
Separation
In wide field of sight, the centered agent has three neighbors, the blue dashed arrow represents the desired velocity and red real arrow is the steering vector

Pseudocode of Separation function:

```
Separation (agents list){

 GET the other agents

  for (i=0; i<number of agents; i++)

      if ( agent i is in ZOS)

      keep count++;

      get subvector(current.position-i.delayed position );

      subvector.normalize;

      subvector.divided by distance in between;
```

```
desired =+ subvector

end for

desired.divided by count;

desired.normalize;

desired.multiplied with a parameter;

steer vector = desired - current self velocity;

steer vecot.limit to maxsteer;

 RETURN steer vector;}
```

3.4.2 Alignment

Alignment can also be understood as headings matching, a tendency to adjust the flocking agents to head in generally the same direction. Within zone of alignment, an agent will try to align itself with the average orientation of its neighbors. To get the expected heading vector of an agent, say *i*, normalize the velocity vector of each other agents into a unit vector in the neighborhood, add them up and get the average. We get the desired heading direction:

$$\vec{d}_i^{ali}(t+\Delta t) = \frac{\sum_{j\neq i}^{j\in zoa} \frac{\vec{v}_j(t-\Delta t)}{\left|\vec{v}_j(t-\Delta t)\right|}}{N_{j\in zoa}} \qquad (3.4-4)$$

Another way to implement alignment is velocity matching. The steering vector is got from the average sum of the velocity vectors; there is no need to normalize. This way we can expect when time is long enough, velocities in *ZOA* will match. (i.e. $t\rightarrow\infty$, $\|v_i\text{-}v_{other}\|\rightarrow 0$)

$$\vec{d}_i^{ali}(t+\Delta t) = \frac{\sum_{j\neq i}^{j\in zoa} \vec{v}_j(t-\Delta t)}{N_{j\in zoa}} \qquad (3.4-5)$$

Same as separation, the magnitude of the desired heading direction is unified to a desired

velocity:

$$\overrightarrow{dv}_i^{ali}(t+\Delta t) = \frac{\vec{d}_i^{ali}(t+\Delta t)}{\left|\vec{d}_i^{ali}(t+\Delta t)\right|} \times speedparameter \quad (3.4-6)$$

The steering vector is the desired velocity subtracted by the self velocity, as follows:

$$\vec{s}_i^{ali}(t+\Delta t) = \overrightarrow{dv}_i^{ali}(t+\Delta t) - \vec{v}_i(t) \quad (3.4-7)$$

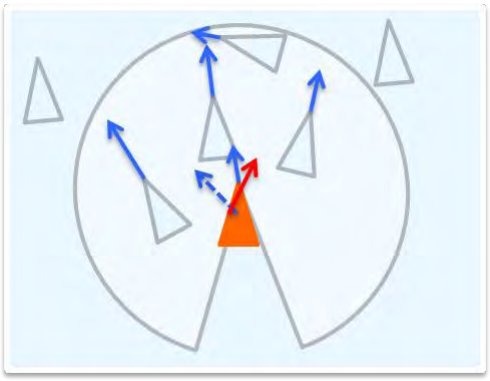


Figure 3.4
Alignment
In wide field of sight, the centered agent has four neighbors, blue dashed arrow represents the steering vectors, and red real arrow is the steering vector

Pseudocode of alignment function:

```
Alignment (agents list){

Get the other agents

  for (i=0; i<number of agents; i++)

      if ( agent i is in ZOA)

      keep count++;

      get delayed velocity of i;

      if (only adjust heading)velocity.normalize;

      desired =+ delayed velocity;

  end for

  desired.divided by count;
```

```
  desired.normalize;

  desired.multiplied by a parameter;

  steer vector = desired - self current velocity;

  steer vector.limit to maxsteer;

RETURN steer vector;}
```

3.2.3 Cohesion

Cohesion implies that all the flocking agents should stay together in a group, or else they would break up and go to separate ways. That is also why cohesion has got the largest neighborhood. To realize cohesion rule, the desired direction of agent *i* is the average position vector of its neighbors. The desired velocity and steering vector are obtained just like the two rules addressed above:

$$\vec{d}_i^{coh}(t+\Delta t) = \frac{\sum_{j\neq i}^{j\in zoc}\left[\vec{p}_j(t-\Delta t) - \vec{p}_i(t)\right]}{N_{j\in zoc}} \qquad (3.4-8)$$

$$\overrightarrow{dv}_i^{coh} = (t+\Delta t) = \frac{\vec{d}_i^{coh}(t+\Delta t)}{\left|\vec{d}_i^{sep}(t+\Delta t)\right|} \times speedparameter \qquad (3.4-9)$$

$$\vec{s}_i^{coh}(t+\Delta t) = \overrightarrow{dv}_i^{coh}(t+\Delta t) - \vec{v}_i(t) \qquad (3.4-10)$$

Cohesion is in some way similar with separation only the steering force is opposite.

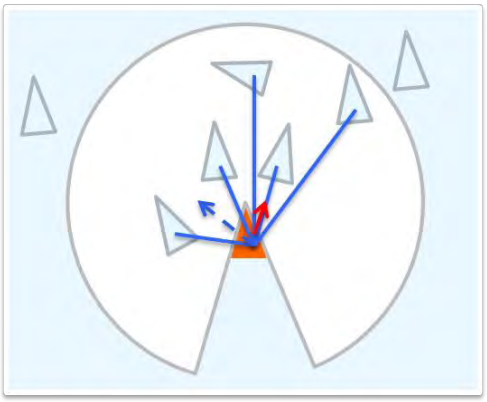


Figure 3.5
Cohesion
In wide field of sight, the centered agent has five neighbors, blue dashed lines represent the position difference vectors, and red real arrow is the steering vector

Pseudocode of cohesion function:

```
Cohesion (agents list){
 GET the other agents
  for (i=0; i<number of agents; i++)
      if ( agent i is in ZOC)
      keep count++;
      get subvector(current.position-i.delayed position);
      desired =+ subvector;
  end for
  desired.divided by count;
  desired.normalize;
  desired.multiplied by a parameter;
  steer vector = desired - self current velocity;
  steer vector.limit to maxsteer;

RETURN steer vector; }
```

3.5 Obstacles Avoidance

One feature added onto flocking behaviors is obstacles avoidance as it is natural to see fishes avoid corallites in the Aquarium.

In this project two types of obstacles are designed. One is fixed obstacle, the other is moving obstacle.

The rules for both types of obstacles are the same. We need to compare the magnitude of vector $\vec{p}$ with the obstacle radius r to check whether it is necessary to add a sub-steering force onto the agent. To get vector $\vec{p}$, project the distance vector $\vec{d}$ onto the velocity vector using dot product. If $|\vec{p}| < r$, a steering vector in the direction of $\vec{p}$ is acted upon the agent, the steering magnitude is proportional to the magnitude of $\vec{p}$ and inversely proportional to the magnitude of $\vec{d}$, where there is an emergency to avoid the obstacles.

To make the rule more precise, a comparison of the velocity magnitude with the magnitude of the projected $\vec{v}$ can be added. This can eliminate the case when an agent is at somewhere far away from the but is heading toward the obstacle, the velocity of the agent is too small to bump into the obstacle, thus not necessarily has this steering force.

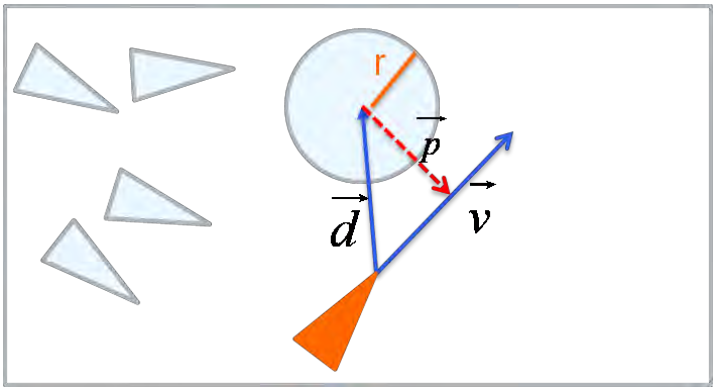


Figure 3.6
Obstacle Avoidance
The steering vector is in the direction of $\vec{p}$

Pseudocode of Obstacle Avoidance:

```
Avoid (agents list){
  for (i=0; i<Obstacle Number; i++)
     get d⃗ = distance(circular center, position)
     dp = dotproduct(d⃗,v⃗) / |v⃗|;
     x⃗ = multiple(dp,unit v⃗);
     p⃗ = subtract(x⃗, d⃗);
  end for
  if |p⃗|<r
    steer vector = p⃗
    steer vector.normalize;
    steer vector.multiplied by a parameter;
    steer vector.divied by |d⃗|;
 RETURN steer vector;}
```

3.6 Target Following

When an agent doesn"t have any neighbors around, it doesn"t have external force applied. It will perform uniform motion on a straight line. It seems that the group is just wandering without any targets or leaders to follow.

There can be different rules for determining which is the target and who are the leaders, the feature added in this project is that an agent will sense the position of the target when it doesn"t have any neighbor around to make up for this. In this project, the target is set to be where the mouse browser is. When an agent does not have any neighbors in the three zones, a steering

vector pointing toward the browser is generated upon the agent.

As it is moving toward the browser, this information is passed onto its nearby flocking agents, which makes the agent practically a leader. For one thing, there may be agents outside the agent‟s field of perception, but it may belong to other agents‟ field of perception, another is that there is a delayed feedback in the transmission. Even when the agent has got neighbors while it is following the browser, and will not sense the browser anymore, this browser information can still be known by some flocking agents and thus the flocking group will steer collectively toward the browser. In this case, the mouse browser acts as the target and agents that do not have neighbors firstly obtain the initial target information and act as leaders.

Once an agent has neighbors, the steering force toward the browser will act on it anymore, i.e. it is no longer a leader. So the flocking group has and always has changeable leaders to lead the group.

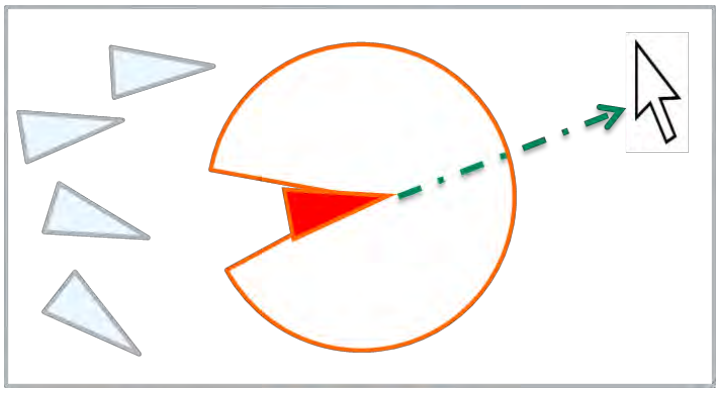


Figure 3.7 Target Following

Pseudocode of Target Following:

```
GET current agent i
```

```
if (in Cohesion(agent list) count == 0)
  isLeader = True;
Leader  (agent list){
 if (isLeader = = true)
 steer vector = substract (mouse, current position)
RETURN steer vector;
}
```

3.7 Position Update

After getting the steering vectors from Reynolds rule and the two features respectively. Remember to limit their value to a max steering force, for sometimes the steering force is too large for flocking agents to bear, the many strong steering forces added upon the agents can cause the agents swaying.

In this project the weights for separation and obstacle avoidance are set to be 1.5, alignment and cohesion are set to be 1.0. The target following weight is set from to 10; the different weights assignment will be discussed in chapter 5.

Pseudocode for getting the acceleration:

```
GetAcceleration (Agent list) {
 For each flocking agent:
    Get steer vector sep = Separate(Agent list);
    Get steer vector ali = Alignment(Agent list)
    Get steer vector coh = Cohesion(Agent list);
    Get steer vector avi = Avoid(Agent list);
```

```
    Get steer vector led = Leader(Agent list);
     sep.mult(weight 1.5);
     ali.mult(weight 1);
     coh.mult(weight 1);
     avi.mult(weight 1.5);
     led.mult(weight 10);
     acceleration.add(sep);
     acceleration.add(ali);
     acceleration.add(coh );
     acceleration.add(avi);
    if (isLeader) acceleration.add(led);
  }
```

Pseudocode for updating position:

```
Update (Agent list) {
  For each flocking agent:
   velocity.add(acceleration);
   velocity.limit(maxsvelocity);// limit to a maximum
   position.add(velocity);
   acc.mult(0);// clear acceleration
   }
```

In mathematical formulation, the update is performed every step time as follows:

$$\overrightarrow{sum}(t+\Delta t) = w_{sep} \times \vec{a}_{sep}(t+\Delta t) + w_{ali} \times \vec{a}_{ali}(t+\Delta t) + w_{coh} \times \vec{a}_{coh}(t+\Delta t) + w_{avoi} \times \vec{a}_{avoi}(t+\Delta t) + w_{lead} \times \vec{a}_{lead}(t+\Delta t) \quad (3.11)$$

$$\vec{a}_s(t+\Delta t) = \frac{\overrightarrow{sum}(t+\Delta t)}{w_{sep} + w_{ali} + w_{coh} + w_{avi} + w_{lead}} \quad (3.12)$$

$$|\vec{a}_s(t+\Delta t)|_{max} = a_{max} \quad (3.13)$$

$$\Delta\vec{v}_s(t) = \vec{a}_s(t+\Delta t) \quad (3.14)$$

$$\vec{v}_s(t+\Delta t) = \vec{v}_s(t) + \Delta\vec{v}_s(t) \quad (3.15)$$

$$\Delta\vec{p}_s(t) = \vec{v}_s(t+\Delta t) \quad (3.16)$$

$$\vec{p}_s(t+\Delta t) = \vec{p}_s(t) + \Delta\vec{p}_s(t) \quad (3.17)$$

3.8 Borders definition

In the computer simulation, a screen window with fixed width and length is generated, which leads to the question of handling the agents when they approach and hit the borders. Two types of borders are designed in this project.

The first one is that the window is set to be connected from left to right, from top to bottom. For example, when an agent reaches out the right side of the wall of the window, it will appear on the left side of the wall of the window.

Pseudocode :

```
Borders1( ){
   if (position of x < -size of agent)
   position of x = width+size of agent;
```

```
  if (position of x > width+ size of agent)
        position of x = -size of agent;
    if (position of y < -size of agent)
        position of y = height+ size of agent;
    if (position of y > height+size of agent)
      position of y = -size of agent;  }
```

The advantage of this setting is that the animation result is fluid as it doesn‟t bump onto the wall, on the other side leads to the drawback that the animation seems separate and unordered, the flocking agents may not aggregate for a long time, the expected flocking behavior is not clearly and truly reflected.

Having this problem, another type of fixed border is designed. When an agent hits on the wall, its velocity will be turned to its opposite direction with the same speed. This time, the flocking agents are bounced back from the screen border and are only moving inside the window. Observing their movement characteristics can be more clear and easier.

Pseudocode:

```
Borders2( ){
  if (position.x < 0 or position.x > height)
     velocity of x = -velocity of x;
  if (position.y < 0 or position.y > height)
     velocity of y = -velocity of y;
  velocity.limit to 3;}
```

The fixed border can give agents an opposite force which doesn‟t follow any flocking

rules; therefore there are times the flocking agents would form a small group nearby the border as they constantly bump into the wall, not able to go out, unless there are leaders outside the group having a strong force to steer them out.

In this project, when firstly constructing the flocks with only Reynolds rules, connected border was used to better observe the effect of the flocking rules. When obstacle avoidance and target following were added, the border was set to be fix as a collective behavior was better presented in this way.

3.9 Interim Summary

Algorithm is based on three rules: Separation, Alignment, and Cohesion, in which alignment can adjust and headings or velocities, separation and cohesion adjust the positions. A delayed feedback is added onto the neighbors‟ information when calculating the respective steering vectors.

Static and moving obstacles use the same formulation to avoid. The flocking group has changeable leaders, agents who do not have any neighbors change their positions by following the target of the mouse browser, thus make them changeable leaders.

Chapter 4 Simulation

Simulation is recognized as a proper and vivid tool in designing and simulating complex and dynamic systems. With the aid of simulation, we can better understand the flocking behavior, and even make predictive assumptions about the performance.

4.1 Simulation Platform

4.1.1 Simulation environment

Processing is an open-source programming language initiated by Ben Fry and Casey Reas[6] and developed by a group of hobbyists for programming.

The processing environment consists of java virtual machine and processing's own libraries. I started with coordinate systems and arrays till trigonometry and objects, processing language is easy to develop images and animations thus it is widely used by media designers and students.

The processing application is free for downloading and easy to install.

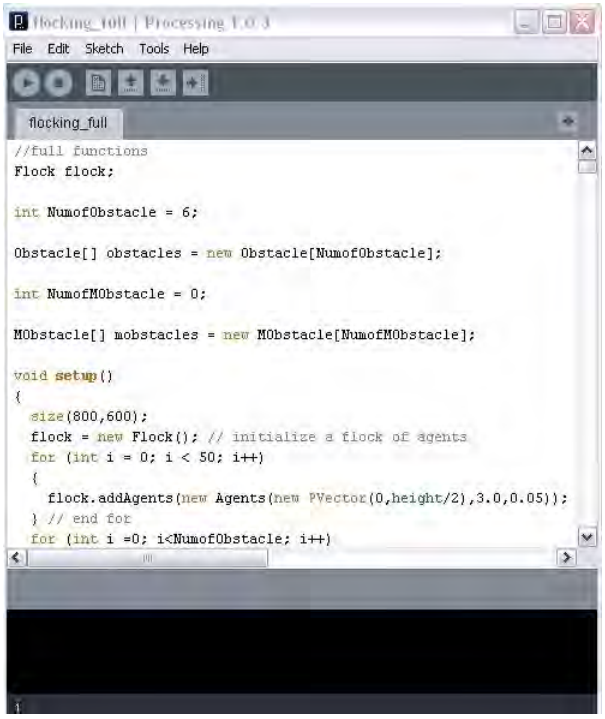


Figure 4.1: *Processing development environment*
This is a capture of the application. Code is written inside the processing window, click button „run", animation will be generated in a new window with the size set in setup( )

4.1.2 Object-oriented programming

The language used is the same as Java. There are online references containing most of the processing keywords that can construct a 2-D animation.

All the programs in *Processing* start with two functions setup( ) and draw( ). setup( ) is used to initialize the environment properties like window size, There are only one setup( ) function in each program and it is executed once in the whole program; draw( ) is right after setup( ) in the program, unlike setup( ), draw( ) is executed continuously in the program, it is especially useful when designing fluid animation. For example inside draw( ), background( ) is used for setting the background color, it is loaded with a frame rate that can set by oneself. Otherwise background( ) is only loaded once and screen will not be cleared.

PVector: The acceleration, a delay position and delay velocity are set to be a PVector, which contains its magnitude and direction. Basic operations of PVectors include subtract, add, multiply, divide and get magnitude. Other than these, anglebetween() is a useful tool to get the angle between two vectors from 0 to 180°, which is used in determining the field of sight.

ArrayList: The position and velocity of every agent is defined as an ArrayList so as to store the values in the current state as well as the last several time steps. To generate a group of flocking agents, flocking group is also an ArrayList. The delayed velocities and positions are also stored as lists of PVectors in the ArrayList so that different delayed time lengths can be fetched.

Class: One of the important reasons to use processing is that the object-oriented programming can store all the variables and functions inside a class; the class thus contains the properties such as position, velocity and acceleration. The functions inside the class can tell the class what to do. The advantage of using class lies in that class can wrap up and imitate an object that has many properties; the properties can be called out at any other time and place.

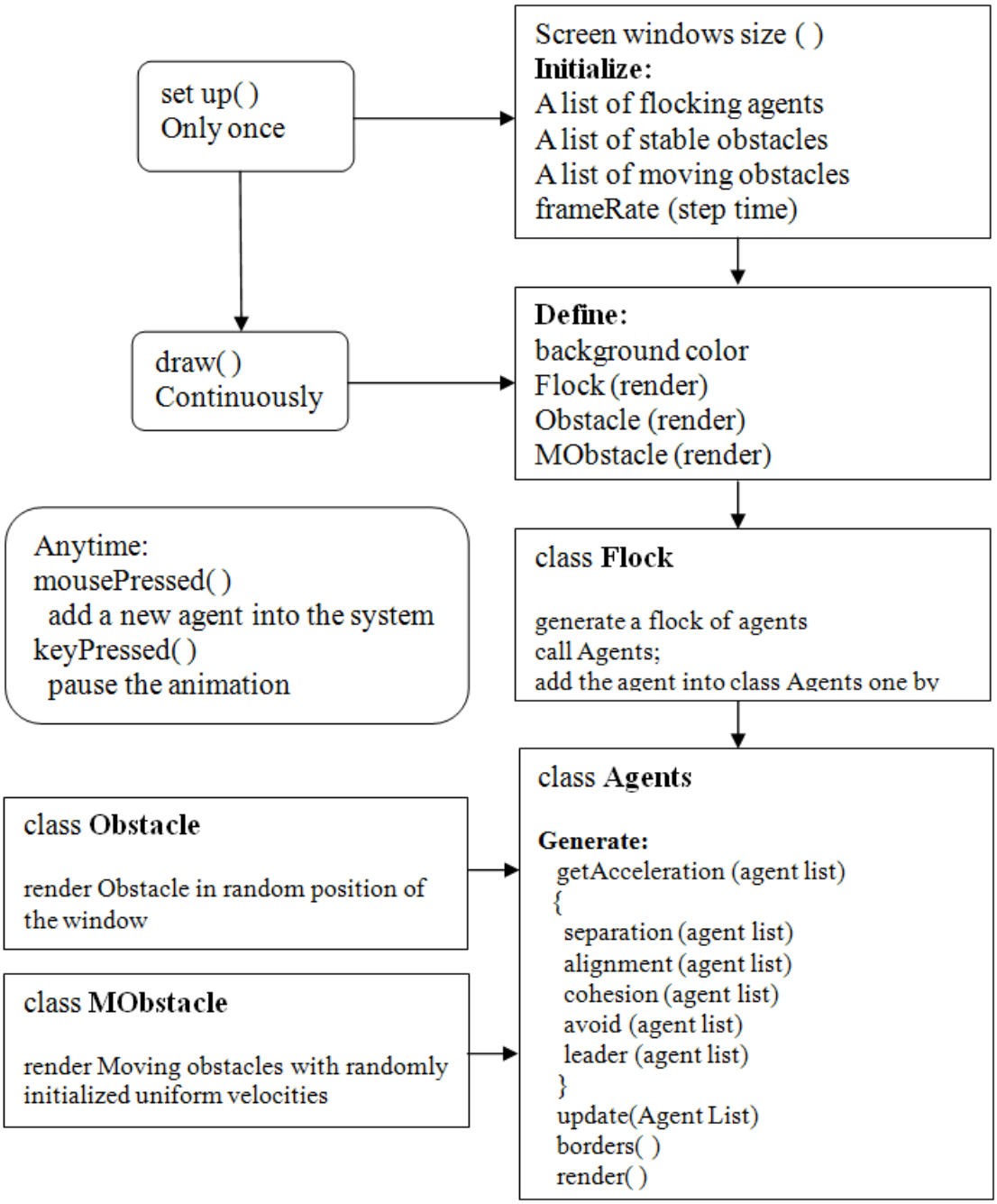
Four classes are set in the program: class **Agents** contains all the rules that applying onto the flocking agents, inside Agents class, the acceleration, velocity and position of the flocking agents are calculated and updated. Class **Flock** generates a list of flocking agents into the system and it is used in Flocks class. Two other classes **Obstacle** and **MObstacle** are to design static obstacles and moving obstacles. Inside the class, the color, size, and shape of the obstacle are defined. The moving obstacles also have velocities and borders situation.

4.2 Methodology

A simple model with only Separation( ), Alignment( ), and Cohesion( ) was first constructed, with only the flocking agents an ArrayList, and then position and velocity were modified into ArrayList to store their past state information, a delayed feedback was added into the system. After these, class Obstacle and MObstacle were created to draw the obstacles, function Avoid( ) was used to avoid obstacles. A boolean value isLeader determines which agent becomes the leader, and function Leader( ) added a steering vector onto those leaders.

To make the model more real life like, the flocking agents were generated with random size within a range; agents that became leader changed their color to red immediately. Click mouse browser can generate a new flocking agent into the screen window, and press the blank key would ask the animation to pause, press again would resume the animation.

The complete structure of the program:



The flocking rules are added into Agents class with separate functions. Adding more rules or removing rules will change the according function, not the program structure.

4.3 Simulation Result

The parameters in the simulation result are given below:

Parameter	Values explored	Parameter	Values explored
Windows size	500x600	Step times	1/60 sec
Number of flocking agents	50	Steering weight of separation	1.5
Zone of separation	30	Steering weight of alignment	1
Zone of alignment	50	Steering weight of cohesion	1
Zone of cohesion	50	Steering weight of avoidance	0/1.5
Field of perception	5/3 π	Steering weight of leader	0/10
Size of obstacle	25	Size of agents	2.5-6
Number of static obstacle	0/5	Number of moving obstacle	0/3

Table 4.1 Simulation parameters

4.3.1Simple flocking

First, a simple flocking simulation with only three rules and no obstacles were constructed,

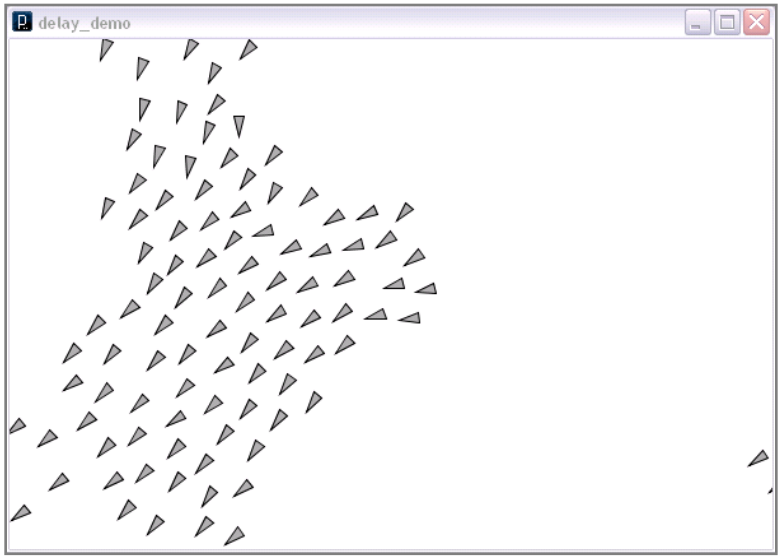


Figure 4.2 Simple flocking

and then 5 obstacles are added, Figure 4.3 shows obstacle avoidance result:

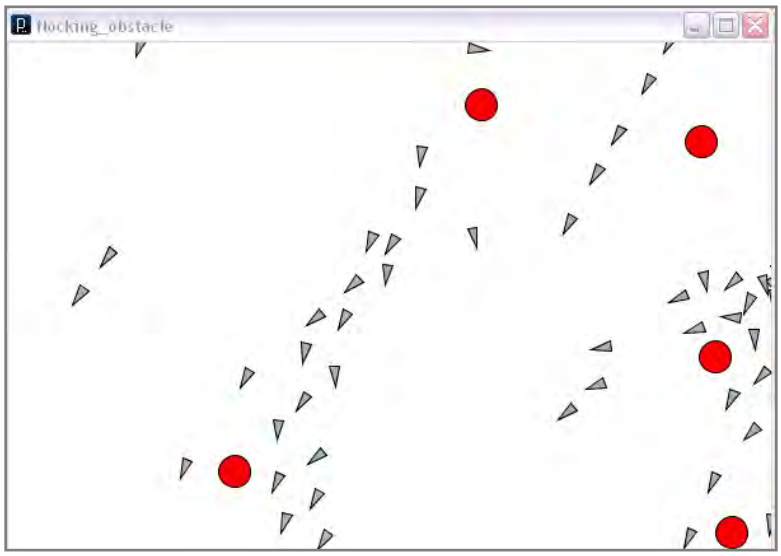


Figure 4.3 Simple flocking with obstacles

4.3.2 Target following added

With target following algorithm added, a group of agents with various sizes and colors are simulated. 6 snapshots are presented as follows from the beginning till the flocking agents form a collective formation.

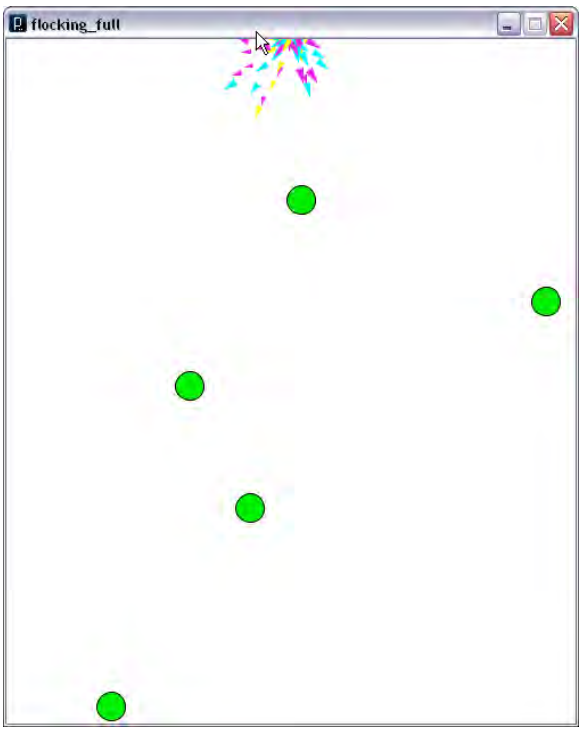


Figure 4.4 initialization (Random velocities)

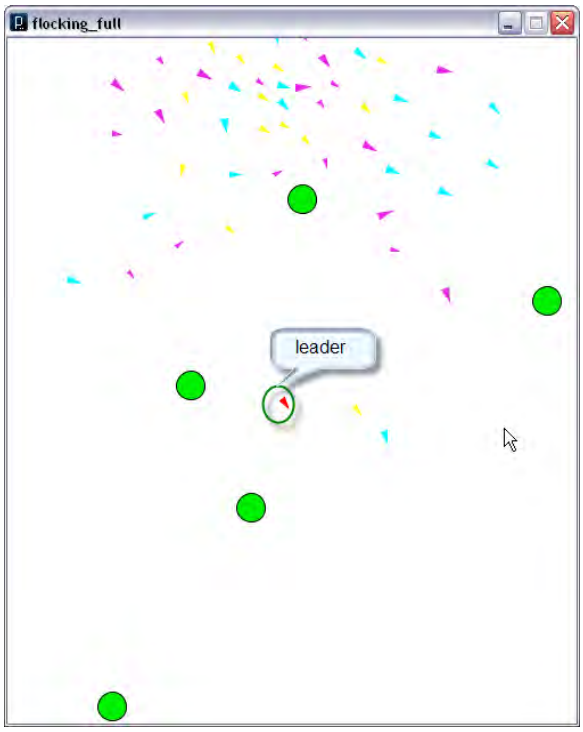


Figure 4.5 one agent becomes leader

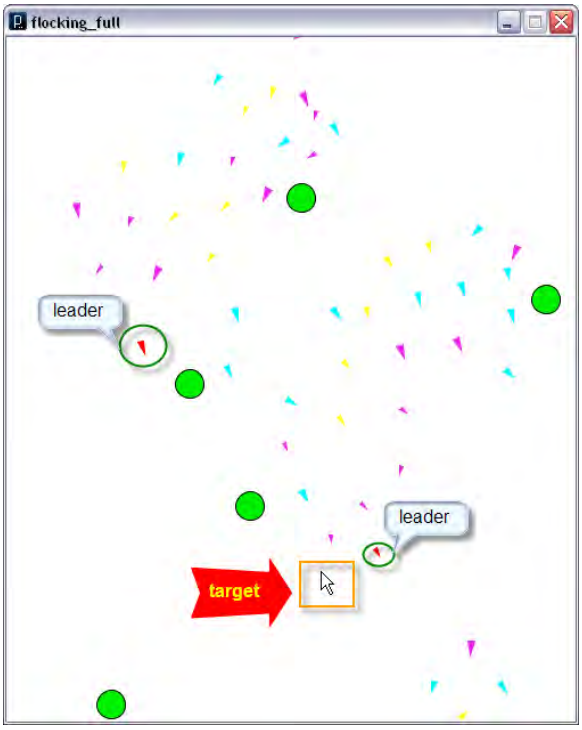


Figure 4.6 following the target

Figure 4.7 forming collective behavior

After four snapshots, leaders emrged and led the group toward the mouse browser, the headings and relative positions of the flocking agents gradually steer consistently. Two more are below:

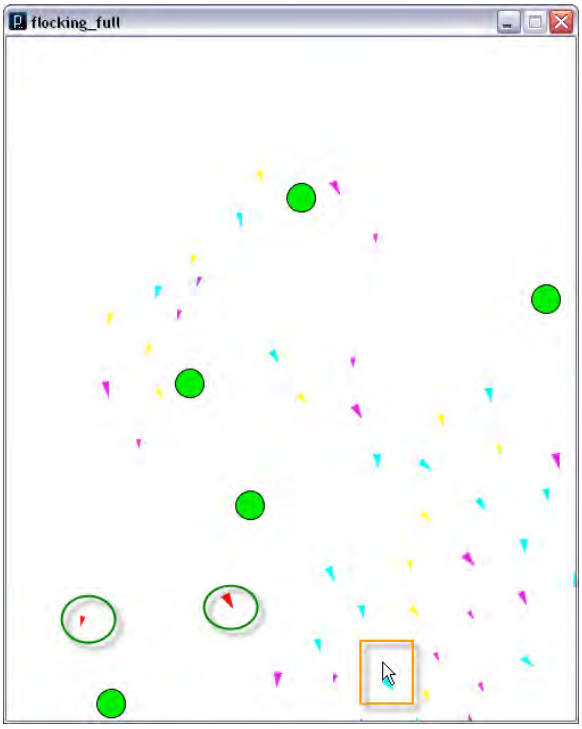


Figure 4.8 avoiding obstacles

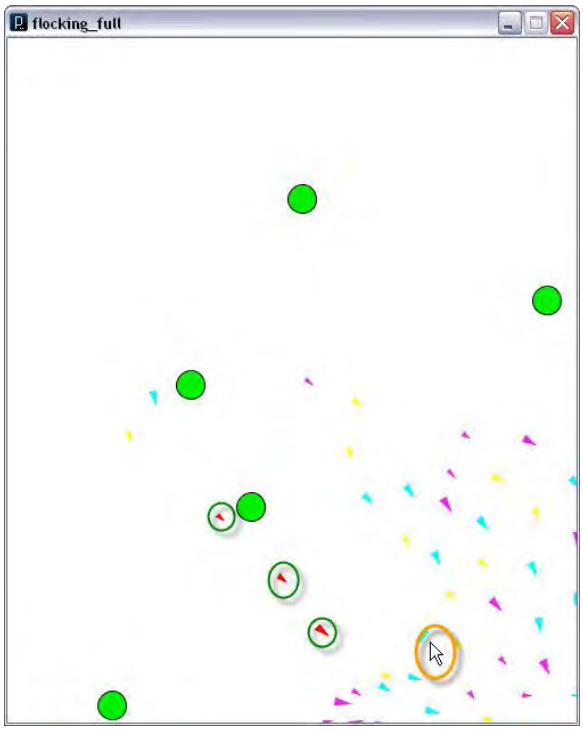


Figure 4.9 aligning headings

From their transition of positions and headings we observe that flocking agents spread out in the screen window, the formation and their headings are not yet consistent. After several snapshots the collective behavior can be observed and leaders become effective.

4.3.3 Moving obstacles added

Two moving obstacles are now added into the screen window. They have a uniform velocity randomly initialized from 0 to 3.

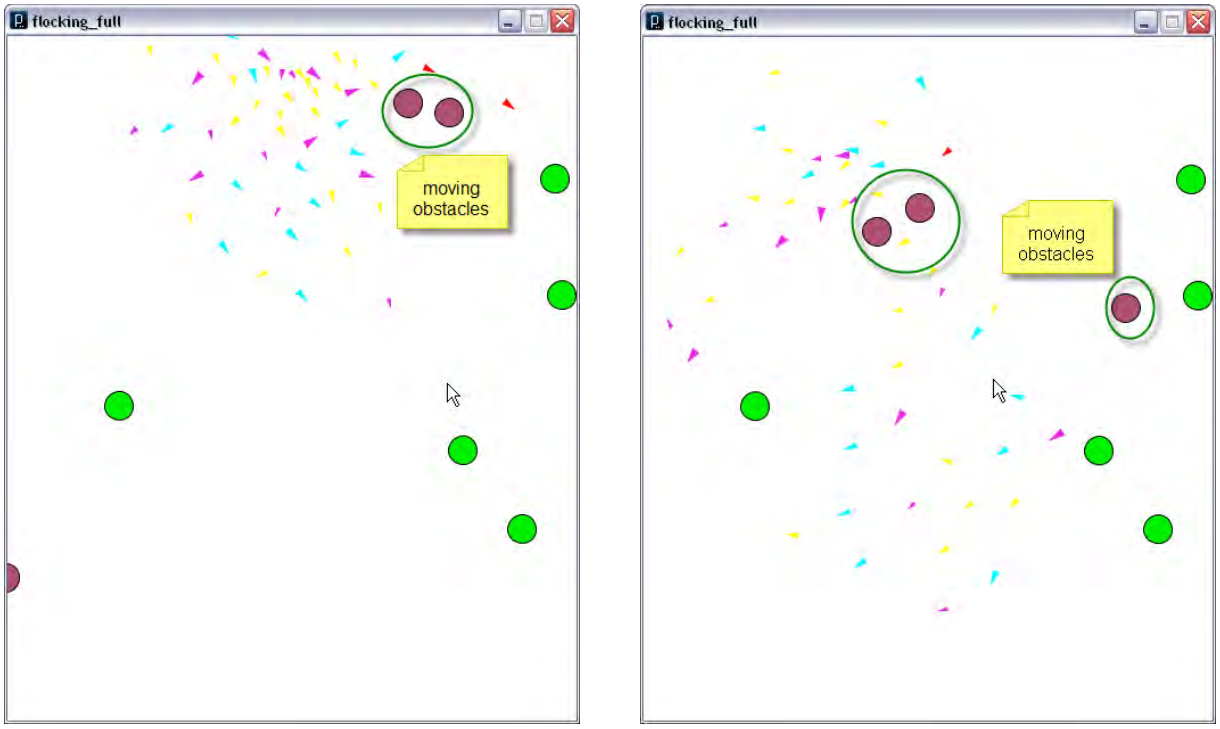


Figure 4.10 with moving obstacles

4.4 Interim Summary

This chapter introduces the simulation environment and languages for flocking behaviors. Java language is used for the convenience of applying object-oriented concept. A series of simulation result from simple flocking to fully-functioned result is captured. Analysis and discussions on the simulation result will be delivered in Chapter 5.

Chapter 5 Result Analysis

Table 5.1 shows the range of parameters explored. Different consequences of flocking behavior happen when changing the values of the parameters.

Table 5.1 summary of model parameters

Parameter	Values explored	Parameter	Values explored
Windows size	400x400-800x800	Step times	1/30s-1/60s
Number of agents	10-100	Steering weight of separation	1-10
Zone of separation	25-60	Steering weight of alignment	1-10
Zone of alignment	50-60	Steering weight of cohesion	1-10
Zone of cohesion	50-60	Steering weight of avoidance	1-10
Field of sight	π-2π	Steering weight of leader	5-20
Size of obstacle	20-25	Size of agents	2.5-6

5.1 Discussion on parameters

5.1.1 Different field of sight

Change the field of sight to π, flocking agents can only sense the neighbors in front of them. The flocking behavior will be like a colony of ants sneaking up one by one in a line.

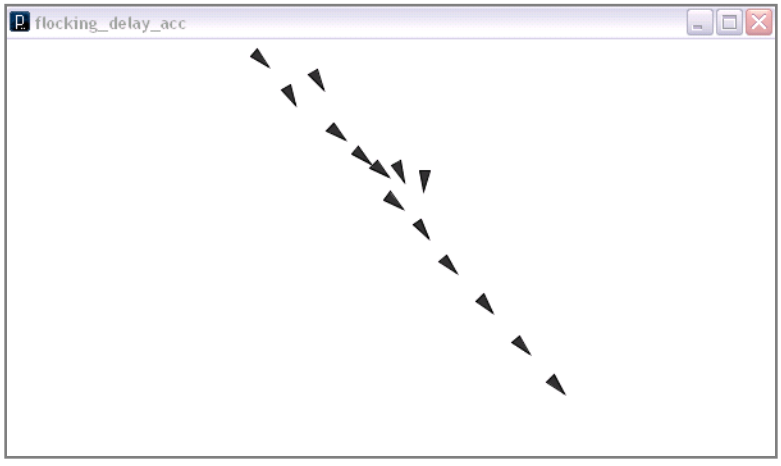


Figure 5.1 Result of plane field of sight

If the field of sight is less than a right angle, i.e. flocking agents have a narrow field of sight, the simulation result would be much less vivid, and sometimes agents collide with each other. It is easy to understand this happening as for a major part of the circle is not counted as neighborhood.

In the simulation result in Chapter 4 the field of sight is set to be 5/3 π, which is closer with real birds, and the result started to be normal. From observing their behavior, a complete field of sight reflects the best result.

5.1.2 Different neighborhood assignment

The assignment of neighborhood size was discussed in chapter 3.3.3. The collective behavior actually exhibits sharp difference when the relative radii of Reynolds rules change.

A highly cohesive yet with low level of parallel alignment behavior would occur if zone of alignment is set to be so small to equal zone of separation. Compare two transition behaviors in Figure 5.2.

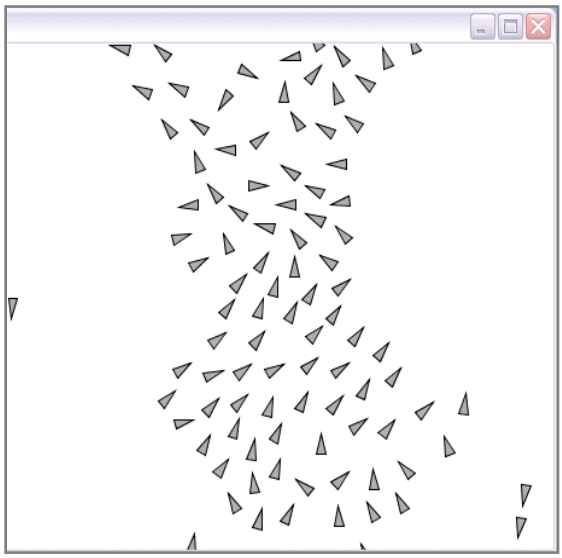


Figure 5.2-1 *ZOS=ZOA=ZOC/2*

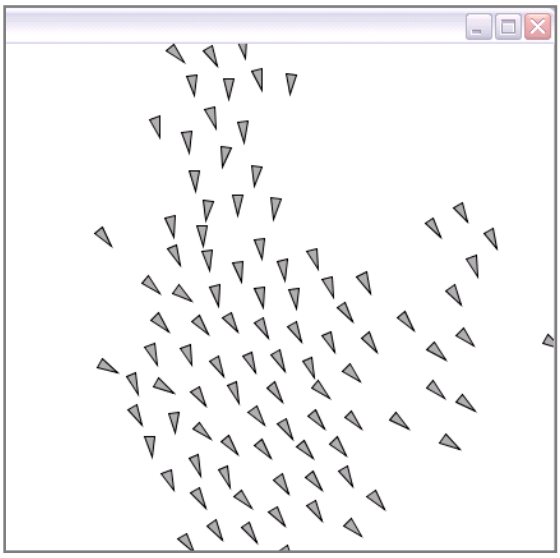


Figure 5.2-2 *ZOS*2=ZOA=ZOC*

Figure 5.2 simulation result of different neighborhood assignment

Nevertheless, this kind of cluster is not permanent, within 10 seconds; the group behavior in Figure 5.1 would adjust the headings and head generally in parallel. This transition is captured as below:

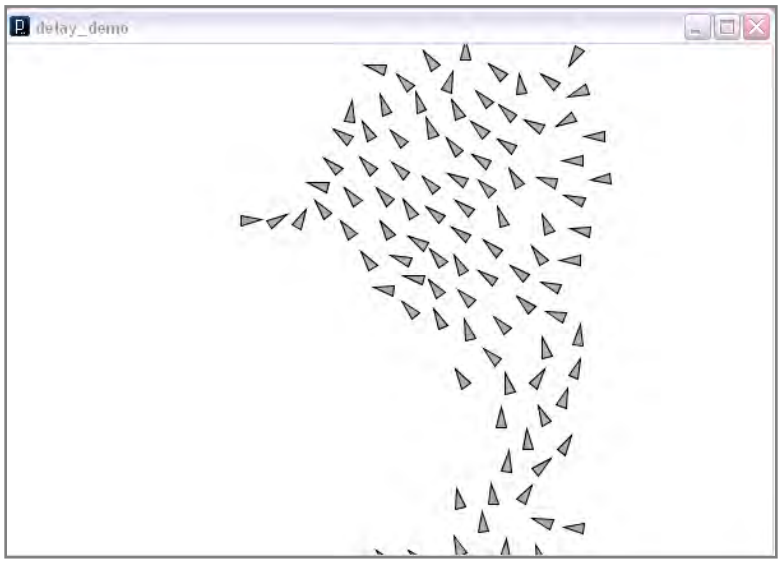


Figure 5.3 collective behaviors after a while

5.1.3 Different weight

The different steering weights might be the most critical and demanding adjustment. The collective flocking behavior may not be reflected given inappropriate weights. We have to be careful that if one weight is too small, the effect ceases to work, while on the other hand, too large weight may cause other effects disappear.

After a series of trial and error, the settings in Figure 4.3 in chapter 4 are used here; which was adjusted so that the collective behavior of flocking agents can be best exhibited. One principle is that the separation steering weight should be a little bit larger than that of alignment and cohesion. The weight effect is analyzed by setting one of the weights to be very large; the result from this setting is practically the same when only one of the steering rules applying on the flocking agents.

Table 5.2 Default settings for Reynolds rules

Parameter	Values explored	Parameter	Values explored
Windows size	600x400	Step times	1/60s
Number of agents	100	Steering weight of separation	1.5
Zone of separation	25	Steering weight of alignment	1
Zone of alignment	50	Steering weight of cohesion	1
Zone of cohesion	50	Steering weight of avoidance	NA
Field of sight	$5/3\pi$	Steering weight of leader	NA

The steering weight of separation is set to 10, others remains the same. We can see that the separation rule „rules" thus make the other two rules lose efficacy.

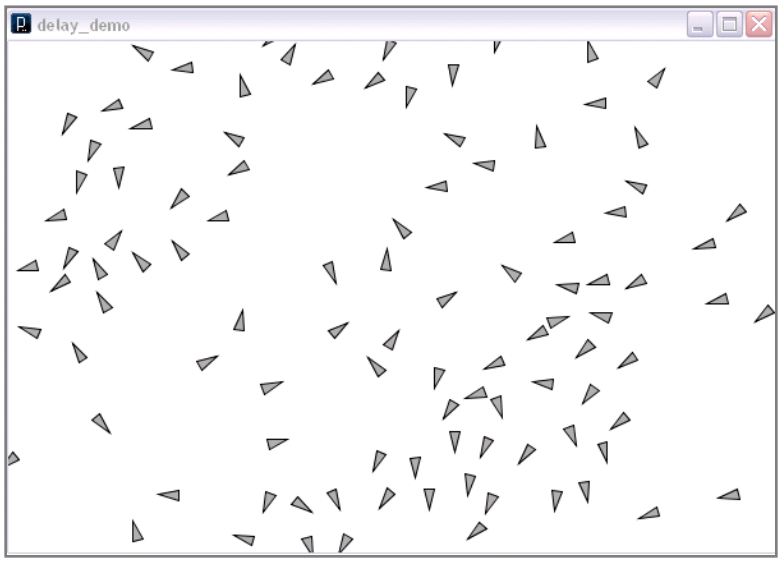


Figure 5.4 Simulation result with large separation weight

When tweaking the weight of alignment to 10, the result is just as expected: the headings of the flocking agent align with each perfectly but the collision is unavoidable.

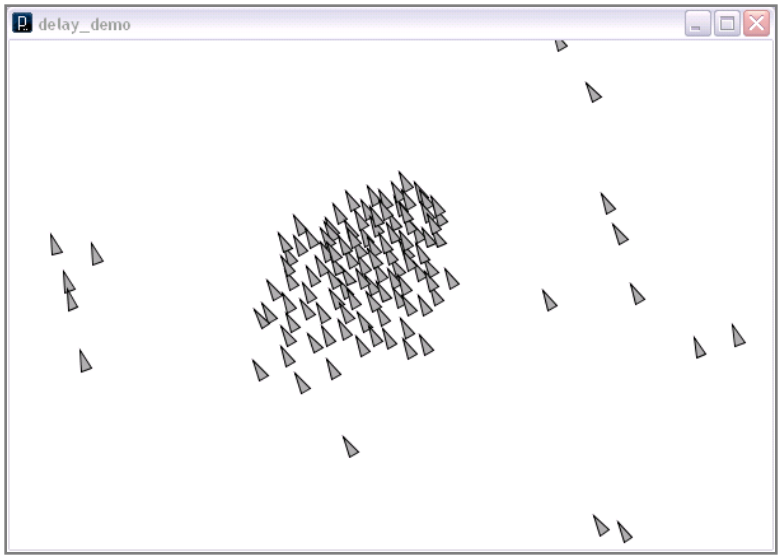


Figure 5.5 Simulation result with large alignment weight

Lastly, weight of cohesion is set to 10; figure 5.6 is a series of snapshot of the flocking behavior from beginning to the stable state. The flocking agents quickly congregate and overlap one another, all happen within 10 seconds.

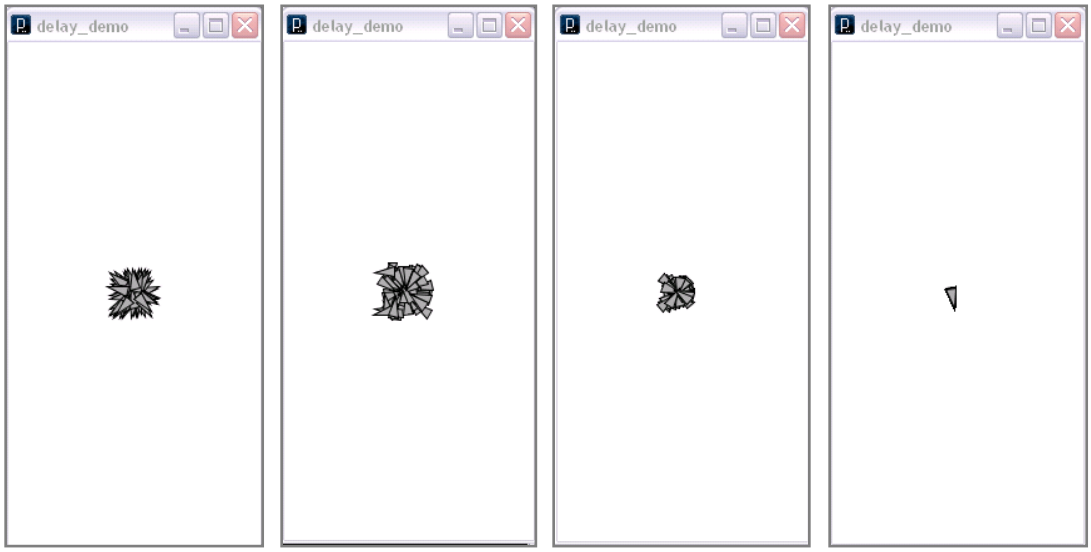


Figure 5.6 Simulation result with large cohesion weight

Actually just a little variation on the parameters is already enough to make a distortion on the flocking behavior. Extreme cases were used to represent a quick typical tendency.

5.2 Fragmentation

When the number of flocking agents increases an issue is addressed: the flocking agents scatter out, like being chased by the enemy thus dispersed. We call it fragmentation, i.e. the movement of the flocking agents might follow the flocking rules but they form separate groups, or sometimes a small portion of the group members are left out and cannot follow the formation of the group but rather interact among themselves. Another reason for fragmentation is the random initialized velocities and positions.

In Figure 5.8 and 5.9 when the flocking number is 100, a minor fragmentation happens not every time the program runs, and the fragmentation can converge to a collective group soon. But the step time is only 1/60seconds, the flocking agents have actually moved for a large number of steps to reach a relative organized formation.

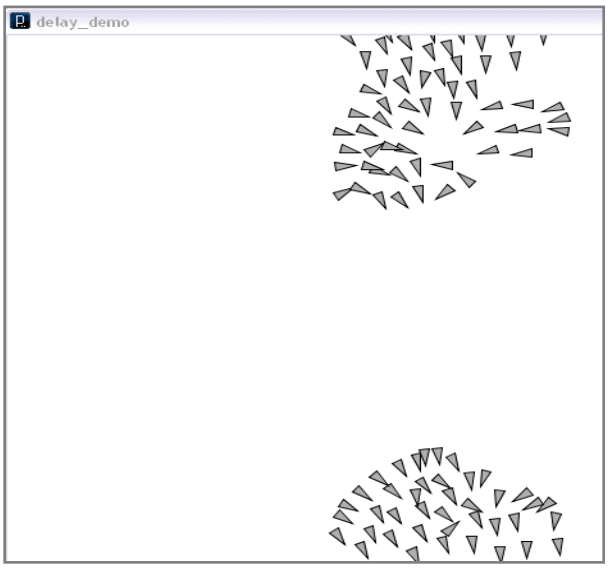


Figure 5.7 Fragmentation

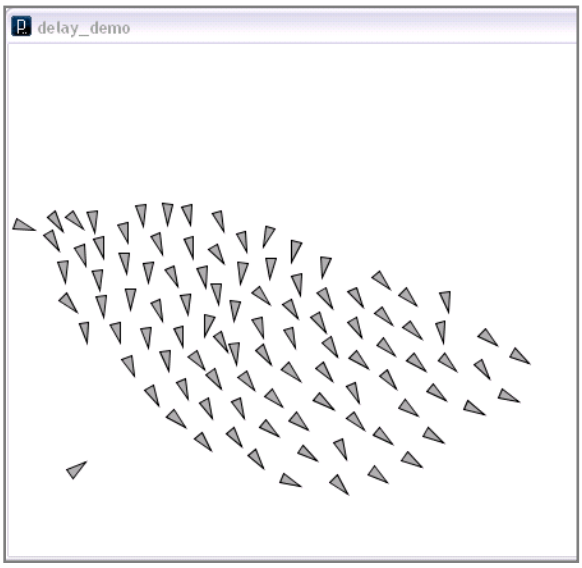


Figure 5.8 Flocking over 10 seconds

Also the connected border eases the fragmentation as those dropped out agents can rejoin the group after appearance from the other side of the screen window. Figure 5.10 presents 200 flocking agents with fixed border. This time the fragmentation phenomenon is obvious and it

took some time for the group to convergence, or sometimes there are minor agents falling behind.

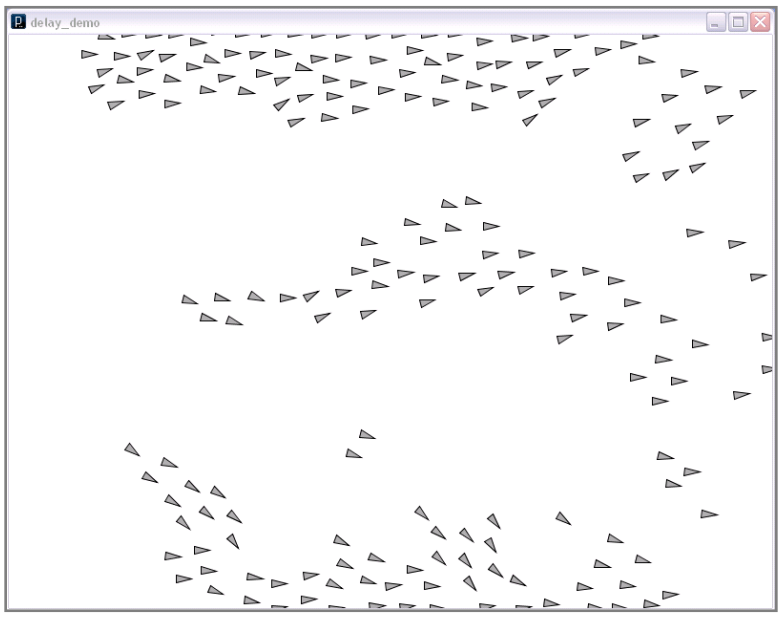


Figure 5.9 Fragmentation for large number of agents

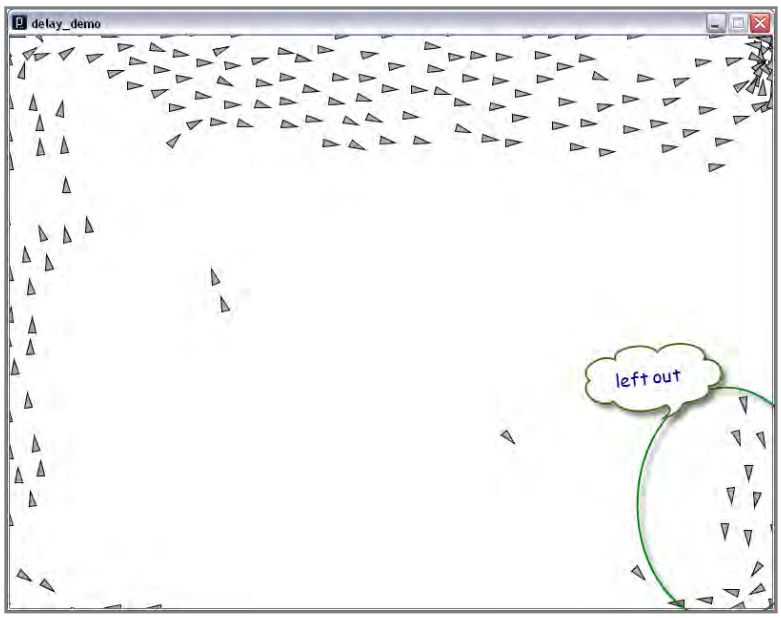


Figure 5.10 Minor group left out

Figure 5.11 was captured for a time interval of 20 seconds after Figure 5.10. Still, the flocking agents could not move outside the border, if that is possible, fragmentation will be more obvious.

Fragmentation is clearly not expected in the simulation result. One of the ways to improve this situation is to assign leaders to the agents and let the whole group head toward some goal, which can effectively remedy the drawback of fragmentation. The leader gives a feedback to its neighbors at the same time neighbors give feedback to neighbors; the whole group can have better performance. It is also proved that implementing Reynolds rules alone cannot create a complete flocking behavior. In Figure 5.12, even the agents scattered out upon initialization, with the emergence of the leaders, they tend to stay together, not going separate ways.

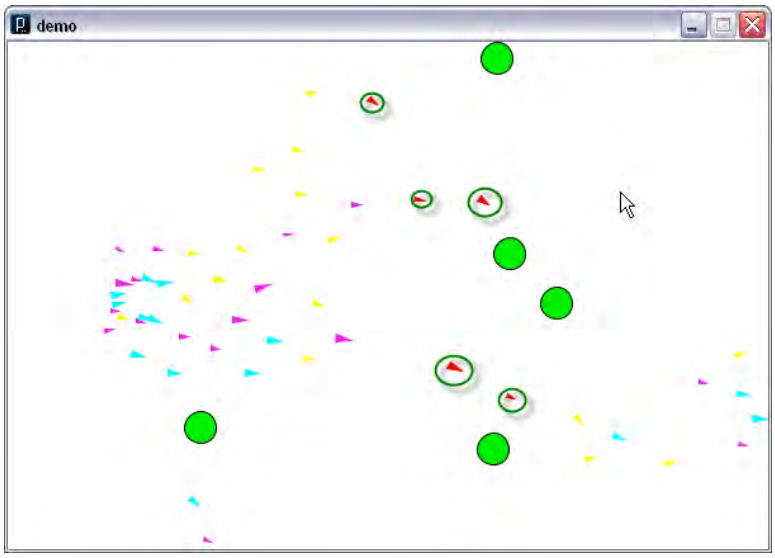


Figure 5.11 Leaders easing fragmentation

As it is noticed that the weight of the leader steering vector is very large compared with the other rules, the purpose of large weight is to enforce the leader head toward target more quickly thus make the leading more effective.

5.3 Delay Effect

The original intention to use a delayed feedback on the neighbors‟ information is that this delay exists in between the transmission. Besides, the vision of birds can have an image persistency in the brain; a delay is a good explanation for that.

However, comparing the simulation result on pure feedback with delayed feedback, there seems to be no significant difference. That is because in this project, the default step time is 1/60 seconds, which is very subtle for consideration. To get obvious result, more steps of time are delayed, in this project the delayed time can be modified up to 5τ. After the delay time is set large one thing is observed: the time from initialization to the flocking agents reaching a stable state is longer if delay is added, i.e. better result is from the one without any delays.

The comparison is made easier when the border is fixed and the screen window is relatively large. A 100 number of agents are generated with/without delay under the same condition. From Figure 5.12 we can observe that fragmentation happens in both cases. However as it goes, the group without delay will gradually flock toward a collective behavior with most of the agents while the group with delay keep separating into different small groups, even they have the tendency to congregate, there are always a small portion left out.

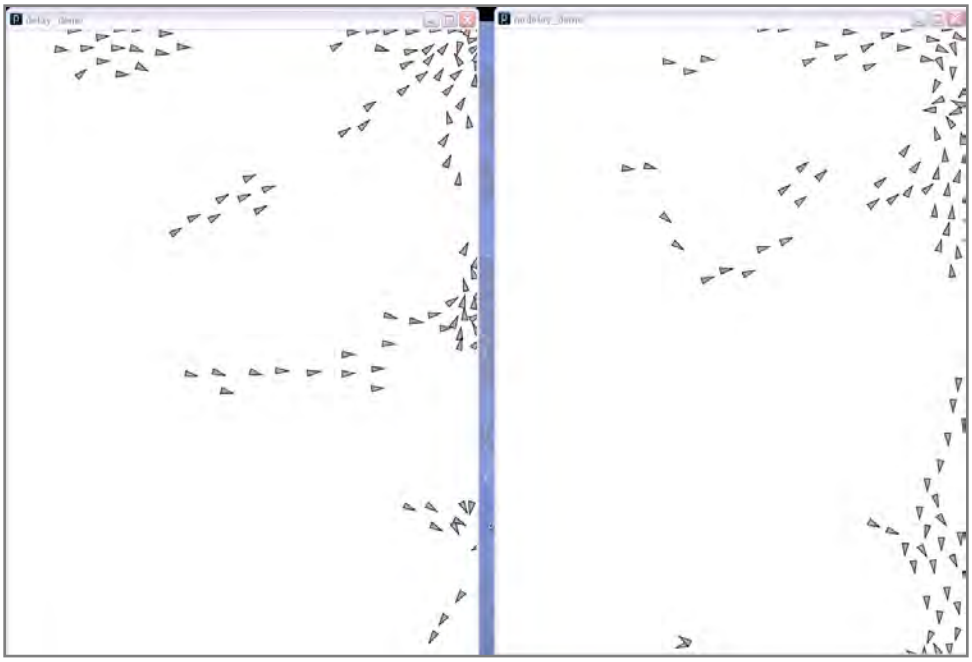


Figure 5.12 comparison on initialization with delay/without delay

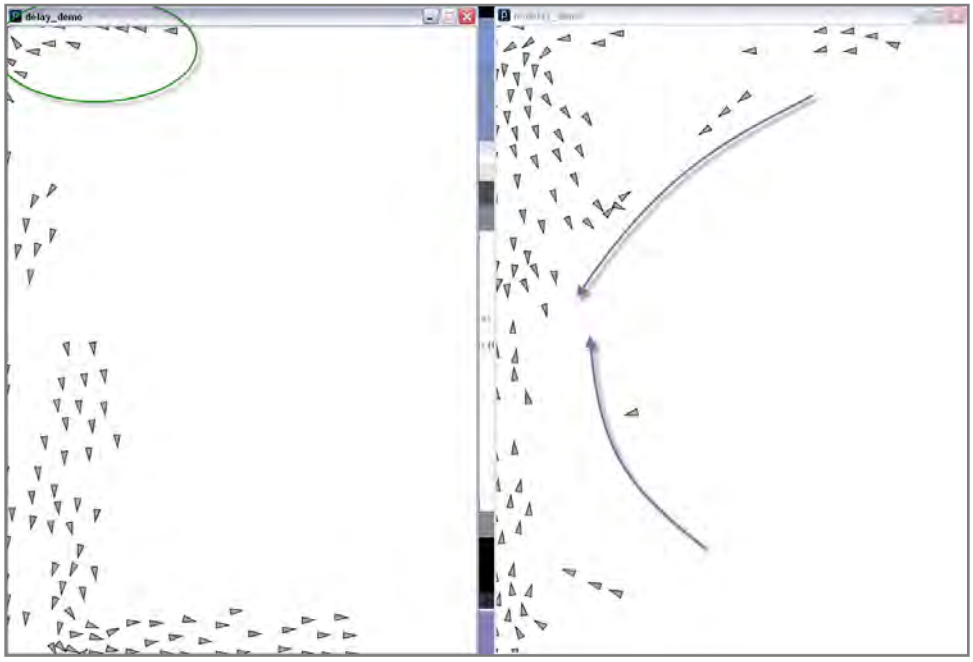


Figure 5.13 flocking after 5 seconds

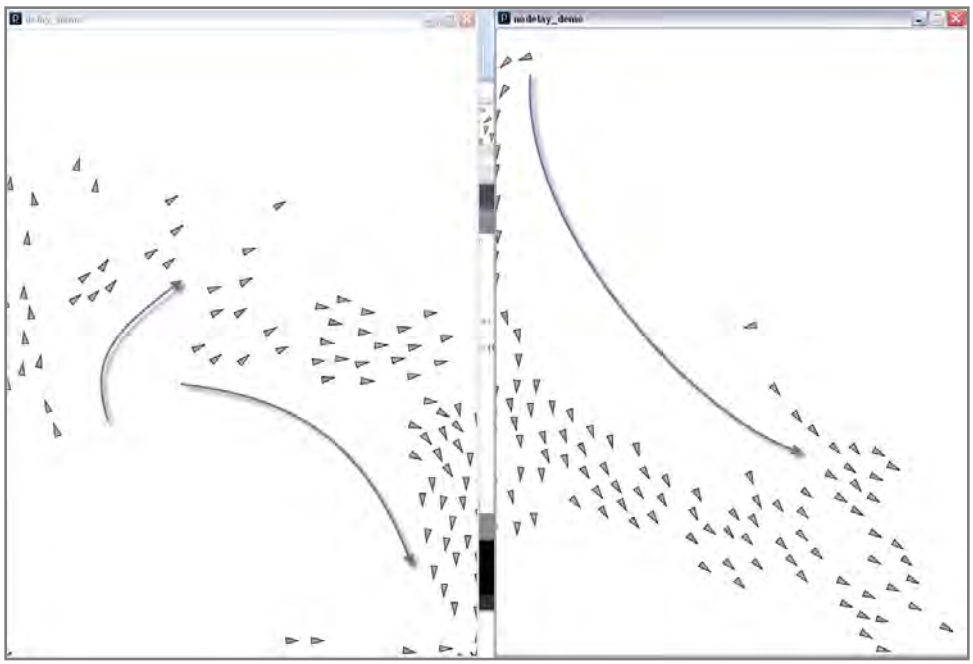


Figure 5.14 comparison on flocking formation with/without delay effect

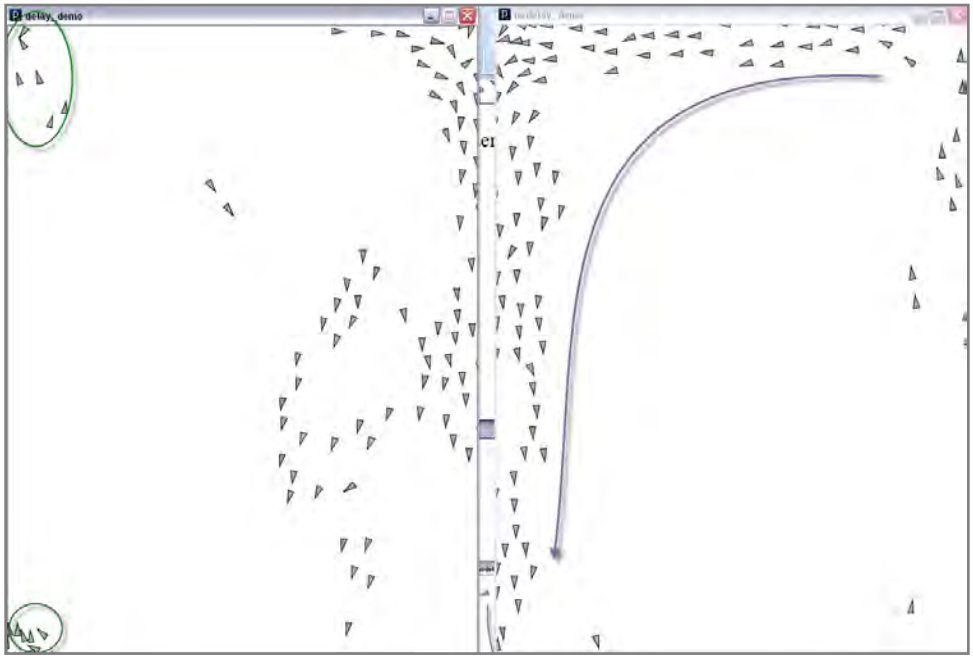


Figure 5.15 comparison on lasting state with/without delay effect

This finding is not a happening by chance, almost every time we run the program, flocking with delay is resulted with a little fragmentation effect. To think about it, delay effect is not desirable in the simulation even it may exist in real life.

5.4 Problems occurred

With all the functions enabled, some phenomena that we do not expect can happen, sometimes it is reasonable and yet cannot be solved by simply adjusting the parameters.

5.4.1 Collision

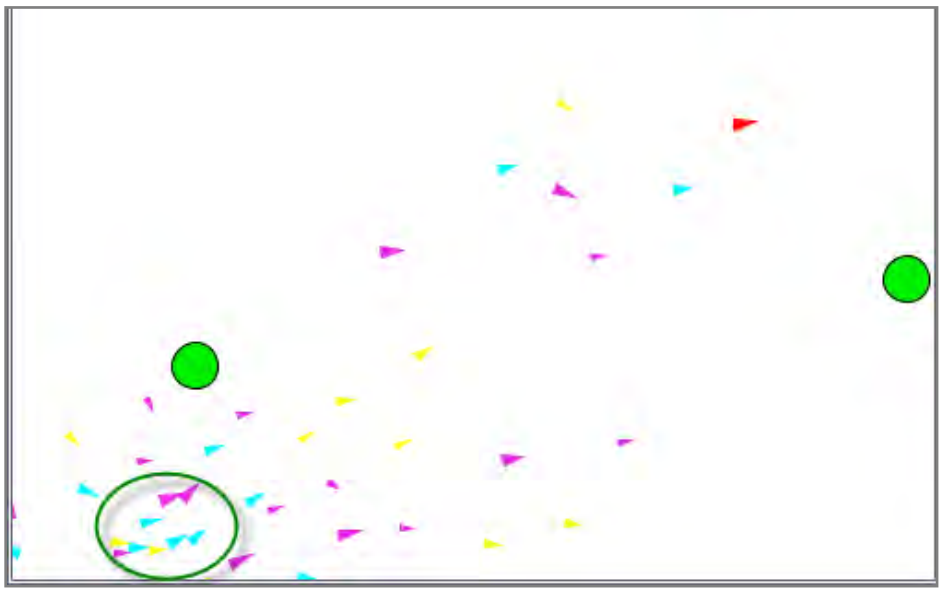


Figure5.16 Agents colliding

The circled out agents adhere to each other.

There are times the distances between agents are very small even the separation rule doesn't apply. Figure 5.1 is captured when executed for about 30 seconds. It also happens in initialization state, but that is due to the initial positions are the same, hence is beyond discussion here. This is a reasonable situation in the program. I assume there are two possible cases when collision happens:

Case 1: when agents hit the border of the screen window, their velocities will toward immediately to the opposite, which doesn't follow any flocking rules. It is possible that they bump into their neighbors when they are bounced back. When the borders are set to be connected

in the screen window, this case would not happen, as agents approaching one side of the border will appear on the other side of the border.

Case 2: For every step time the neighbors of the agents can be in different positions, different headings and different numbers; their acceleration is changing every single step. The calculation is hardly feasible. Also the final acceleration applied on one agent is the weighted summation of several rules including separation and cohesion; the separation force may be weaken by other forces or even canceled out with cohesion.

The problem of agents colliding was actually found after obstacle avoidance and target following rules were added. Therefore Better result can be obtained if there are so many steering forces applied onto the flocking agents.

Following is a simulation result of flocking agents with only delay feedback and Reynolds‟ three rules. Borders are connected.

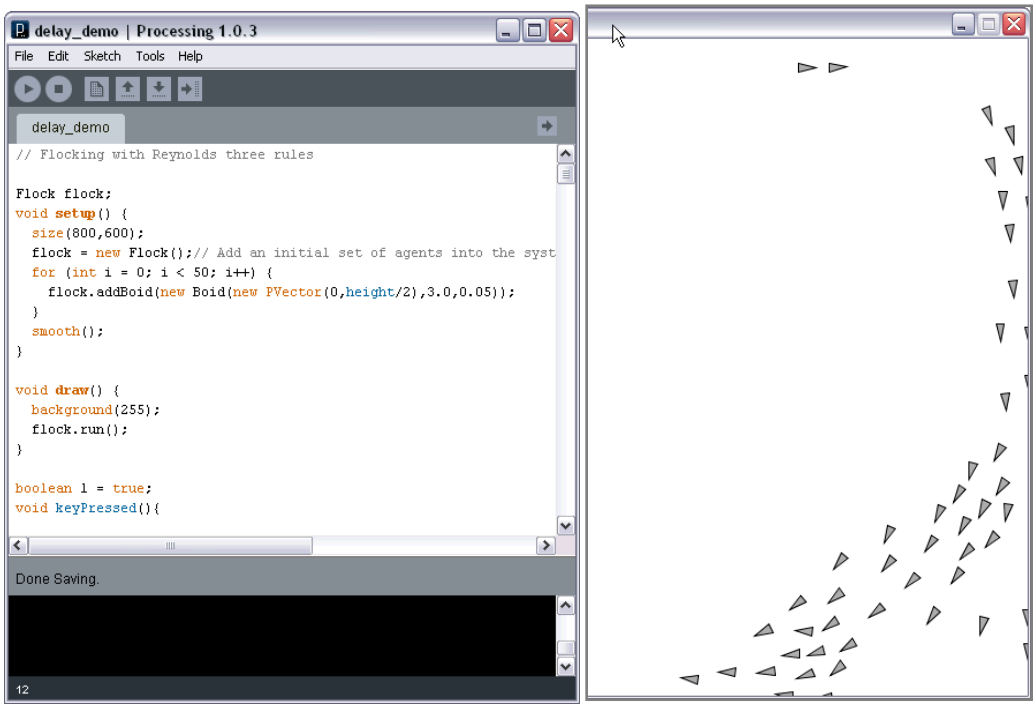


Figure 5.17 simple flocking with connected borders

Similar with the problem of agents colliding, agents missing avoiding obstacles can happen, especially to moving obstacles. Moreover, the impact of the moving force onto the agents impedes the collective flocking behavior. As in figure.. agents are somehow bounced by the force of moving agents thus their positions seem to be disarrayed.

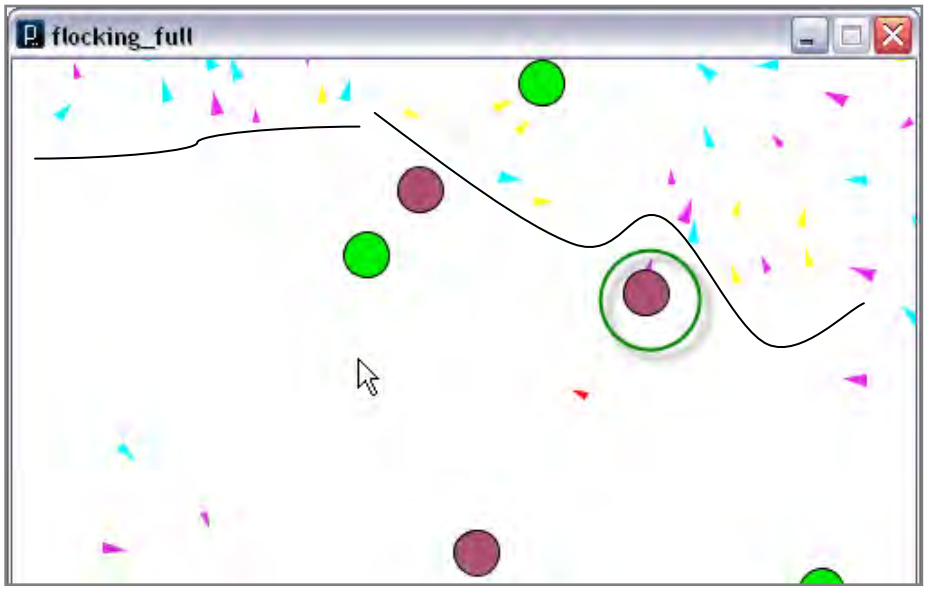


Figure5.18 Colliding with obstacles

5.4.2 Oscillation

Another problem is that the flocking agents tend to sway from side to side while moving. The shaking phenomenon cannot be captured using figures but we can conclude from Figure 5 That their disordered headings are resulted from the oscillation, which is also due to the many steering forces acted upon the agents.

Oscillation was a very common problem when I began to construct the simulation model.

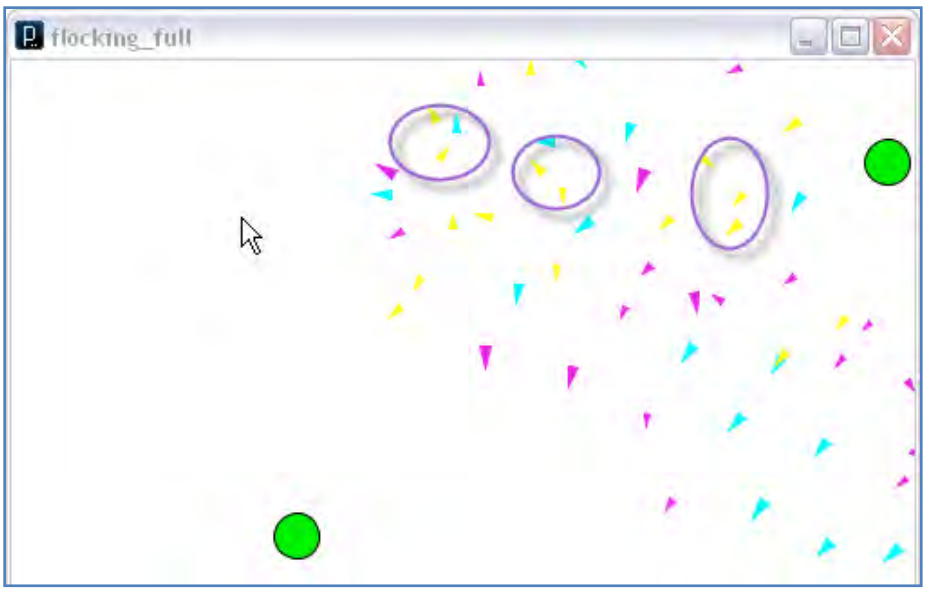


Figure 5.19 Oscillation

5.5 Interim Summary

In this chapter a series of comparisons and discussions on different impacts due to the setting of parameters are presented. The significant of having target following is that leader and target can help ease the pitfall of fragmentation when there are a large number of flocking agents under Reynolds condition alone and their initial states are randomized.

The simulation model is not totally flawless, by adjusting the parameters many of the problems occurs due to the lack of numerical calculation, which leads to an open question for improving this situation.

Chapter 6 Further Development

Followed by the last chapter, a good improvement of the project would be to make all the rules calculable. A simple yet effective improvement is using potential function to control the positions changes of the flocking agents.

6.1 Potential Function

From the rules we previously addressed, the similarity of separation and cohesion suggest that two agents can have a force of repulsion when they are close within some distance, and attract to each other when the distance becomes large, which reminds us that we might know the interatomic force between molecules have the same property. Therefore, the use of a potential function is brought on the table.

To construct a function that can show attraction and repulsion with respect to different distances in between, we take Lenard-Jones potential function to illustrate. A potential energy exists in between the agents, given that we have used acceleration to control their behavior. The intermediate force between the two agents is the negative derivative of the potential function.

The equation of Lenard-Jones and its derivative is shown below:

$$U\,(d) = -\frac{A}{d^m} + \frac{B}{d^n} \qquad (6.1)$$

$$F\,(d) = -\frac{d(U)}{d(d)} - \frac{mA}{d^{m+1}} + \frac{nB}{d^{n+1}} \qquad (6.2)$$

Following is a plot of Lennerd-Jones function with A = B = 10000, m = 1, n = 2.

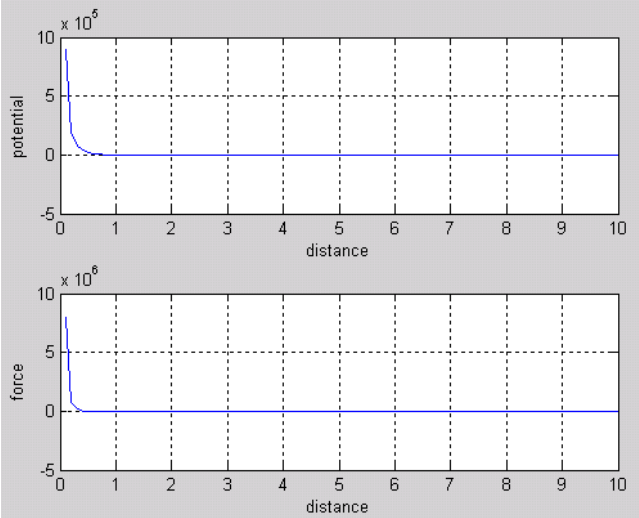


Figure 6.1 example of a potential function and its derivative

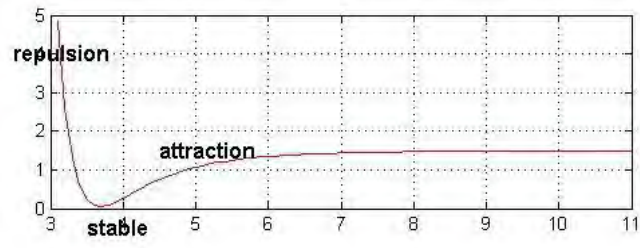


Figure 6.2 Potential function

A simple plot of the potential function explains the meaning of using Lennard-Jones potential function: when the distance between two agents are samller than a certain value, the potential function gives a sharp increase toward infinity, a repulsion force is generated which is inversely proportional to the distance; thus the problem of collision can be solved. When the distance in between goes to infinity, there is no force between the agents, which is consistent with the local perception.

The significance of using potential function is that it is differentiable everywhere, and there is one point where the derivative of the potential function is zero, i.e. no force is acted on either of the agent if they are that distance away. Once two flocking agents have reached this state, the acceleration applied onto them does not contain separation and cohesion anymore, only

headings being adjusted. Eventually the flocking group can have a stable behavior.

Pseudocode for adding this potential function:

```
Potential (Agent List){
A = 2000; B = 1000; m = 1;n = 2;
GET the other agents
  for (i=0; i<number of agents; i++)
     get d = distance between(current.position, i.position);
     get subvector(current.position,i.position);
     subvector.normalize( );
     get force F  = -dU = -mA/power(d, m+1) + nB/power(d, n+1);
     subvector.mult(F);
     steervector = subvector;
  end for
   RETURN steer vector;}
```

6.2 Path Finding

First, the concept of path finding is presented below:

One simplest path finding algorithm:

```
Pathfinding (Agent List){
Get Agent List
       for (i=0;i<number of agents;i++)
       if(i.position.x > destination.x)  i.position.x--;
       if(i.position.x < destination.x)  i.position.x++;
```

```
      if(i.position.y > destination.y)  i.position.y--;
      if(i.position.y < destination.y)  i.position.y++;
   end for;}
```

Path finding was used in target following when the agents know where the target is. In this project, the advantage of using path finding is not that obvious but the result could be better and realistic if a optimization path finding algorithm is implemented.

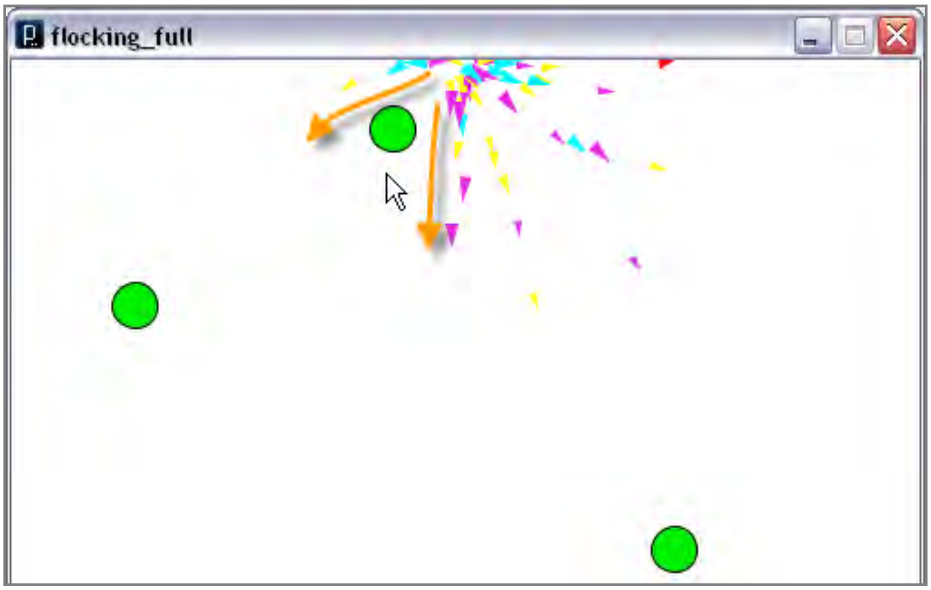


Figure 6.3 pathfinding with obstacle avoidance

When the obstacle is generated too close to the flocking agents‟ initial position, the flocking agents will try to avoid the obstacles thus behave dispersedly. The agents would take a detour around the obstacle but not following the major direction of the flocking group, which is correct according to the algorithms presented in chapter 3.

However, this may not be the case for real bird flocks; look at figure 6.3, the left group of birds would rather avoid the obstacle by moving along the right path than the left path. This cannot be accomplished in flocking algorithm. The possible improvement is to have a better path finding algorithm. Path finding can be related with optimization which is also a hot topic in

swarm intelligence.

This path finding improvement is just my assumption. The algorithm that fits for flocking behavior needs to consider the flocking direction of the major flocking agents, at the same time not bumping into others or obstacles.

6.3 Current achievement

A good algorithm of flocking simulation was prompt by Reza.O.Saber [10]; he constructed a flocking model that can flock without leader and no fragmentation phenomenon by means of a collective potential function in a Eulician space. A Java simulation platform based on Reza"s theory was development by H.Su, Y.Z, and X.W [4]. The study of flocking behaviors control is everlasting and the applications are substantial in the field of network control, computer games, and robots intelligence.

6.4 Interim Summary

This chapter mainly came up with a possible improvement into the flocking model—a potential function can be implemented to combine separation and cohesion and solve the problem of collision. A tentative further development on optimum path finding was discussed, and some innovative work on the flocking algorithm was addressed.

Chapter 7 Conclusion

7.1 Conclusion

The flying behaviors of migrating birds embodies complex and yet organized principles. Through learning the characteristics of the flocking birds and their movement, we can get to the conclusion that global information is not necessary in controlling the flocking behavior. With local perception of each agent, individual‟s actions and interactions can make a large scale system steer into collective movements. A small number of agents can lead the whole group toward a target.

An intuitive way to present the flocking behavior is by means of abstracting the group into a multi-agent system model and carry out agent-based computer simulation. Object-oriented designing methods make the program classifiable thus convenient. *Processing* is a good environment encapsulating Java virtual machines to get the simulation result.

Separation, Alignment and cohesion best describe the distinguished characteristics of flocking behavior; however, they alone cannot create a complete flocking model given a large number of randomly initialized agents. Leaders are needed to prevent the happening of fragmentation.

One drawback in the algorithm is that calculation is difficult to perform; inserting an appropriate potential function into the get to an equilibrium state where they keep a certain distance away and only adjust headings.

The complex properties of the flocking agents can be reflected on the simulation, such that a small variation on the states of agents can cause a distinct difference on the flocking behavior.

7.2 Contribution

The major contributions of this project are:

1. A computer program simulating the collective behaviors of flocking agents was written in Java. The flocking agents follow the classic flocking rules with avoidance of static and dynamic obstacles. In certain condition, system can have changeable leaders that steer toward the mouse browser.
2. Besides using current feedback, a delay is counted as a last step time information feedback, and is added into the program. The user can select to delay velocity or position or both, the weights of the delayed feedback and the step time are modifiable. Effect and comparison of using delayed feedback were analyzed and discussed.
3. Further improvement and feasibility in the flocking algorithm were proposed and analyzed.

It is almost done here on the report; this final year project has opened a door to the interesting and challenging flocking behavior to me. With the tool of mathematics and programming, real life behaviors are reflected by computer simulation. I got the chance to learn about researches and applications in the area of multi-agent systems and artificial intelligence. During the last year, the first time I explored the research and science world on my own; the learning process can be time consuming and frustrating when I headed for the wrong direction or could not come to the expected result. And yet all those cannot beat the delight and contentment on the completion of this final year project. The gain through all the trial and error has equipped me for a further dig into the fantastic artificial world.

Bibliography

[1] Chen, G. (n.d.). Consensus and control over complex networks.

[2] David M.Bourg, Glenn Seemann. (2004). *AI for Game Developers.*

[3] Housheng Su. (2006). Flocking Control of Multi-agent Networks. In *Complex Networks.*

[4] Housheng Su, Yan Zhou and Xiaofan Wang. (2008). Simulation Platform for Flocking in Multi-Agent Systems with A Virtual Leader.

[5] I.Couzin,J.Krause,R.James,G.Ruxton and N.Frank. (2002). Collective Memory and Spatial Sorting in Animal Groups.

[6] Reynolds, C. Retrieved from http://www.red3d.com/cwr/boids/.

[7] *Lennard-Jones potential*. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Lennard-Jones_potential.

[8] Murty.J.A. and U.S.R.Bondy. (1976). *Graph Theory with Applications.*

[9] Parker, C. *Boids*. Retrieved from http://www.vergenet.net/~conrad/boids/index.html.

[10] Olfati-Saber, R. (2006). Flocking for Multi-agent Dynamic Systems: Algorithms and Theory.

[11] *Processing 1.0*. Retrieved from http://www.processing.org/.

[12] Schumacher, M. (2001). *Objective Coordination in Multi-Agent System Engineering* (Vol. 2039/2001).

[13] Tu, X. (1999). *Artificial animal for computer animation.*

[14] Wang, Z. (n.d.). *Zongyao's home*. Retrieved from http://privatewww.essex.ac.uk/~zwangf/.

[15] Xiaofan Wang, Guanrong Chen. (2003). Complex Networks: Small-World and Beyond.

[16] Xiaofan Wang, Xiang Li, Gruanrong Chen. (2005). *"fu za wang luo: li lun ji qi ying yong"*.

[17] Yoav Shoham,Kevin Leyton-Brown. (2009). *Multiagent systems-algorithmic,game-theoretic and logical foundations.*

[18] *Milgram*. (n.d.). Retrieved from http://www.cba.uri.edu/Faculty/dellabitta/mr415s98/EthicEtcLinks/Milgram.htm.

[19] Jinhuan Wang, Jianping Hu, Daizhan Cheng. (2006). Consensus Problem of Multi-Agent Systems with an Active Leader and Time Delay.