# Simulation of Birds Flocking using Object Oriented Programming in C++


UNIVERSITY OF BIRMINGHAM

Evelyn Workman

I.D. 1435685

LH Y3 Object-Oriented Programming with C++

BSc Physics

Date: 19/01/2018

***Abstract***

Two flocks of two separate species of bird (starlings and larks) where simulated using object oriented programming in the C++ language. This was accomplished by implementing three simple rules on the birds in the flock, which together produced realistic flocking behaviour. The project was created in Qt creator, which allowed a graphical user interface to be created, with the user being able to input the number of birds in each flock.

## 1. Introduction

### Background

The task set was to create a simulation of birds flocking in C++ using object oriented programming.

Object oriented programming is based on the concept of objects. It contains classes, which are defined as objects of the same kind [1]. A class will contain data members which are attributes of the class, and member functions (methods) which define the behaviours of the class. An instance of a class can be created, these instances are usually referred to as objects. An important feature in C++ is the use of pointers and dynamic memory, which have been made use of throughout this code. A pointer is a variable which has the address of another variable as its value, it points to the memory location. They allow the use of dynamic memory allocation, and variables are put on the heap, rather than the stack.

The flocking behaviour some birds display is a natural phenomenon which has always struck wonder in humans, and although the reasons certain birds exhibit flocking behaviours is not currently fully understood, there are several proposed ideas. These include keeping safe from predation and successfully foraging for food [2]. Flocks can be beneficial for saving individual birds' energies, as the birds on the edges of the flock are the only ones which must look out for predators.

The first computer simulation of birds flocking was carried out in 1987 by Craig Reynolds [3]. He proposed that flocking behaviour is governed by three rules; separation, alignment and cohesion. Separation provides a short-range repulsion force, preventing crowding, alignment steers a bird towards the average heading of its neighbours and cohesion steers a bird towards the centre of mass of its neighbours, allowing the birds to remain in a group. Once these three rules had been implemented, Reynolds could see that flocking behaviour was modelled in a realistic way. Figure 1 demonstrates each rule, and the resulting direction which the green bird will travel in.

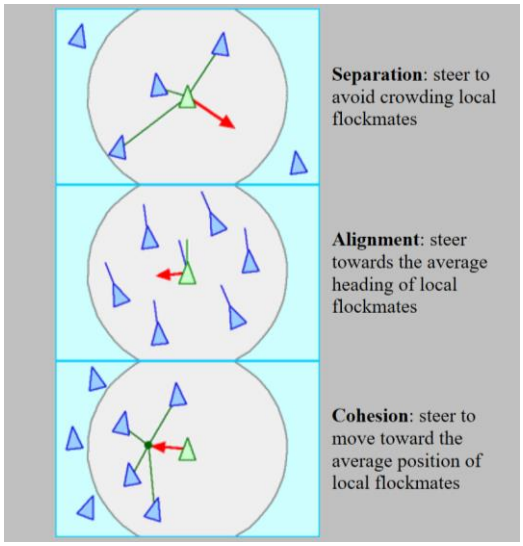


**Figure 1.** The flocking forces between a bird (green) and its flockmates (blue) [3].

### Problem

The aim of this project is to create a graphical user interface (GUI) which includes a display window where a simulation of two different species of birds flocking can be seen.

It is required that the project is created in Qt Creator. Qt is an excellent environment for this type of project to be carried out in as Qt widgets can create user interfaces. The QWidget class (base class) contains several widgets which were utilised in this project, i.e. labels, textboxes. The DisplayWindow class inherits from the QWidget class. Other base classes used included QTimer and QPainter. QPainter drew the birds on the display window, this class has several functions including drawPie and drawPoint which allows for the creation of shapes on the display window. In this project a starling was depicted by a sector using the drawPie function and a lark was depicted by a dot using the drawPoint function. The control window inherits from the base class QMainWindow and has buttons and textboxes, allowing the user to interact with the program, by inputting the number of each species and clicking the "Run" button to begin the simulation.

## 2. Code Design

Initially a Bird class was created, which contained the position and velocity of a bird. A class then needed to be created which acted as a container for an array of birds, this is the flock class. The flock class can calculate the different interactions between all the birds, so that flocking behaviour can be observed.

A TwoVector class was created and contained mathematical functions, such as Mag() and Unit(), which were used to calculate the magnitude of a vector and its unit vector, respectively. This class was used to create vectors with x and y components.

Inheritance was used to create two different species of bird, starlings and larks. Inheritance allows a derived class to receive the properties and methods from the base class. In this case, the derived classes are Starling and Lark, which inherit the properties of the base class, Bird. This is because starlings and larks are both types of birds, so therefore share similar properties, i.e. position and velocity.

The flock class contains Bird, Starling and Lark and calculates the interactions between the birds in the flock and calls on the bird class to update the positions and velocities of all the birds in the flock, and flocking behaviour can be observed.

The graphical user interface (GUI) contains two classes, ControlWindow, where the user can input the number of birds and click the "Run" button to bring up the display window and start the simulation, and DisplayWindow which draws the birds and displays their movements.

The UML diagram (Figure 2) shows all the classes with their member variables and functions.

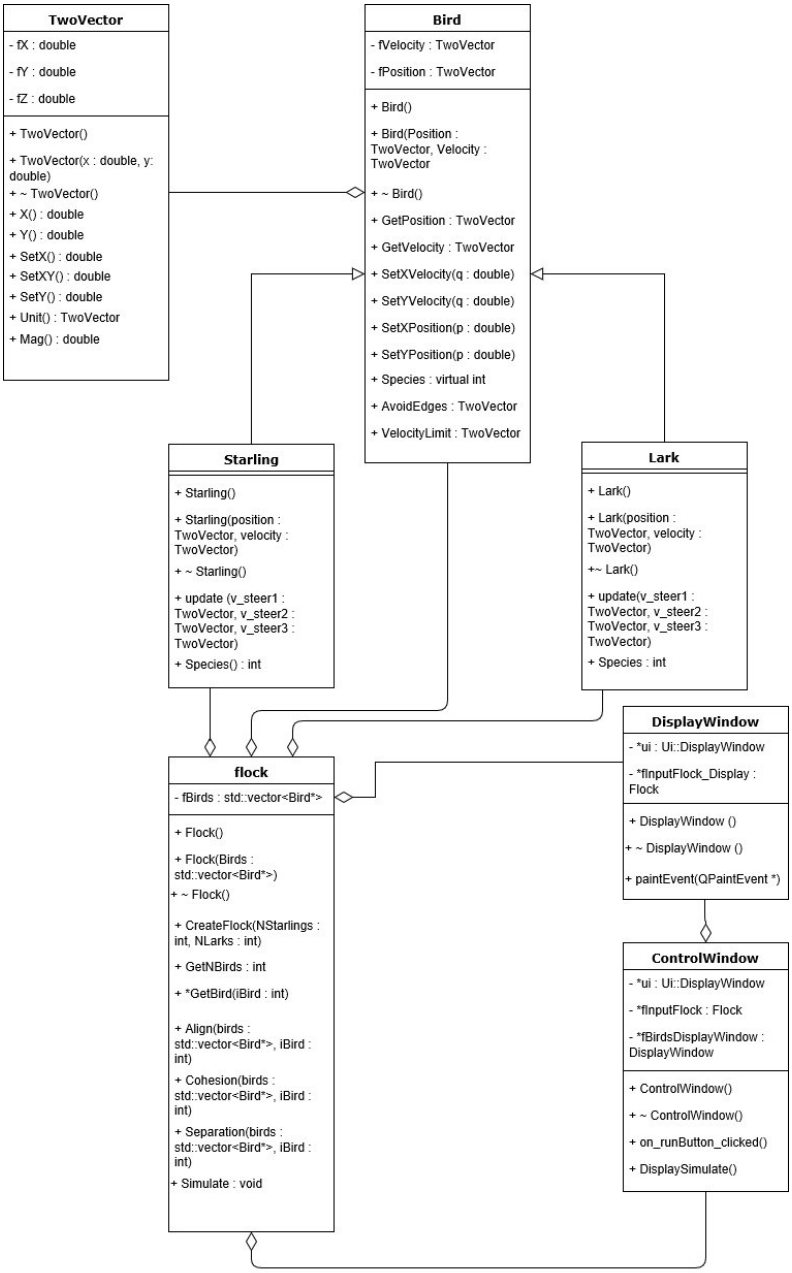


**Figure 2.** UML diagram for flocking birds code.

## 3. Implementation

### Overview

When the code is run, and the user enters a number into the textboxes and clicks the “Run” button on the control window of the graphical user interface (GUI), the display window will appear with a simulation of two flocks of birds.

When the “Run” button is pressed, the code runs through the on_runButton_clicked() function in the ControlWindow class. This takes the numbers entered by the user and calls on the CreateFlock() function in flock class. A timer (an instance of QTimer) is created on the heap and a signal-slot

mechanism causes DisplaySimulate() to be called every time the timer times out, which is every 15 ms in this case. The code for the signal-slot mechanism is shown below.

```
connect(timer, SIGNAL(timeout()), this, SLOT(DisplaySimulate()));
```

The function DisplaySimulate() calls the function Simulate() on the flock. This function is in the flock class and loops through all the birds in the flock, calling on the functions Separate(), Align() and Cohesion(), each of which return a steering velocity. Simulate() then calls the update() function in the Starling and Lark classes which calculate the new velocity of each starling and lark by adding the three steering velocities to the bird's current velocity, and calculates each bird's new position.

**TwoVector Class**

This class was used to calculate different operators and the SetX, SetY, SetXY, X and Y variables. Unit vectors and magnitudes of vectors are calculated in this class, both of which were useful for calculating the steering velocities.

**Bird Class**

This class contains the position and velocity of one bird. It contains functions which obtain its position and velocity (GetPosition() and GetVelocity()) and functions which set its x- and y- position and velocity components. It also contained the functions, AvoidEdges() and VelocityLimit(). AvoidEdges() makes sure the birds stayed within the display window by causing a bird who has moved beyond this boundary to be reflected. VelocityLimit() prevents the birds from moving too fast by preventing the velocity from going higher than a certain value. These two rules where inspired from Conrad Parker's pseudocode [4].

**Starling and Lark Class**

Both classes inherit from the bird class. They contain an update() function which updates the velocity and position of a starling/lark by adding on the three steering vectors calculated from Alignment(), Cohesion() and Separation() functions in the flock class. Their update functions differ, as the weighting of each rule differs for the two species, resulting in the two flocks having different behaviours and interacting differently. The Species() function returns a 1 in the Starling class and a 2 in the Lark class, and is a way to identify the birds.

**Flock Class**

This class contains a vector of birds from Bird class. It includes a function called CreateFlock() which creates the initial state of the flock, assigning random velocities and positions to each bird. The function GetNBirds() returns the number of birds in the flock and the function GetBird() returns a pointer to a particular bird (iBird) in the flock.

The three rules (alignment, cohesion and separation) are contained and calculated in the flock class. These functions are created in such as manner so as starlings and larks can't interact with each other, only with birds of the same species. These rules are implemented in the Simulate() function, which loops over all the birds in the flock. Simulate() calls on the update function in Lark and Starling classes with the steering velocities calculated from each rule. The update function in starling and lark causes the bird to update its velocity by adding on the three steering velocities to its current velocity, its position is then updated by adding the velocity to the position.

*Alignment()*

This function returns a unit vector which points in the average direction of the velocities of other birds of the same species within a 100 pixel radius.

This part of the code works by a bird checking if a bird in the flock is within 100 pixels of it, if it is it will then check if it is a starling or a lark and calculate its velocity. This is looped over all the birds in the flock and the total velocity of all the starling and all the larks in the flock which are within 100 pixels of the bird being considered is summed. If the bird considered is a starling, then a unit vector pointing in the direction of the total velocity of other starlings is returned from this function. If the bird is a lark, then a unit vector pointing in the direction of the total velocity of other larks is returned.

*Cohesion()*

Cohesion() returns a unit vector pointing towards the centre of mass of the flock. This function loops over all the birds in the flock and calculates the total position of all the starlings and the total position of all the larks. A unit vector is return from this function which points to the position of the centre of mass of the other birds which are members of the same species.

*Separation()*

Separation() returns a unit vector pointing away from birds within a certain radius. It loops over all the birds in the flock and if the bird is a starling and less than 30 pixels away from the bird being considered then a vector is calculated which points away from the bird in the flock. If the bird is a starling and less than 20 pixels away from the bird being considered then a vector is calculated which points away from the bird in the flock. If the bird considered is a starling then the function returns a unit vector pointing away from neighbouring starling, while if it is a lark a unit vector pointing away from neighbouring larks is returned. The separation distance is less for larks, as they are drawn on the display window as a single point, which is smaller than the sector representing a starling.

*Simulate()*

Simulate() loops through all the birds and calculates the steering velocity associated with each rule. Three instances of the TwoVector class are created (v_steer1, v_steer2 and v_steer3), and they are set to the alignment, cohesion and separation velocities. The update() function from the bird class is then called and causes the larks and starlings to update their positions by adding on the three steering velocities to their current velocities. The Simulate() function loops over all the birds in the flock.

**ControlWindow Class**

The main function in the control window is on_runButton_clicked() which determines what happens when the “Run” button is clicked. A private data member, fInputFlock is a pointer to the flock, In on_runButton_clicked() it is set equal to a flock on the heap and it accesses the CreateFlock() function, with variables equal to the number of starlings (NStarlings) and the number of larks (NLarks). The values of NStarlings and NLarks are set by the user. An instance of QTimer is created on the heap and a signal-slot mechanism connects the timer to the DisplaySimulate function. Every time the timer times out DisplaySimulate is called. The timer times out every 15 ms. The private data member fBirdsDisplayWindow is set equal to an instance of DisplayWindow on the heap, which is a function of fInputFlock (a flock on the heap). fBirdsDisplayWindow then accesses the function show() which shows the display window. The function DisplaySimulate() allows the fInputFlock data member to access the Simulate function from flock.

**DisplayWindow Class**

The DisplayWindow class includes the flock class. An instance of QTimer on the heap is set up in the constructor. It is connected to the update() function by a signal-slot mechanism, when the timer times out (every 15 ms), the update() function is called. This means that the birds update their position every 15 ms. The DisplayWindow class also has a paintEvent() function which is a function of a pointer to the Qt class QPaintEvent. In this function an instance of QPainter, painter, is created. Painter accesses the function fillRect() from the QPainter class and sets the background to black. Painter then accesses the function setPen(), which sets the colour of the objects to white. A for loop loops over all the birds in the flock and draws them on the display window. The x coordinate is set to the x position of the bird, and the y coordinate is set to the y position of the bird.

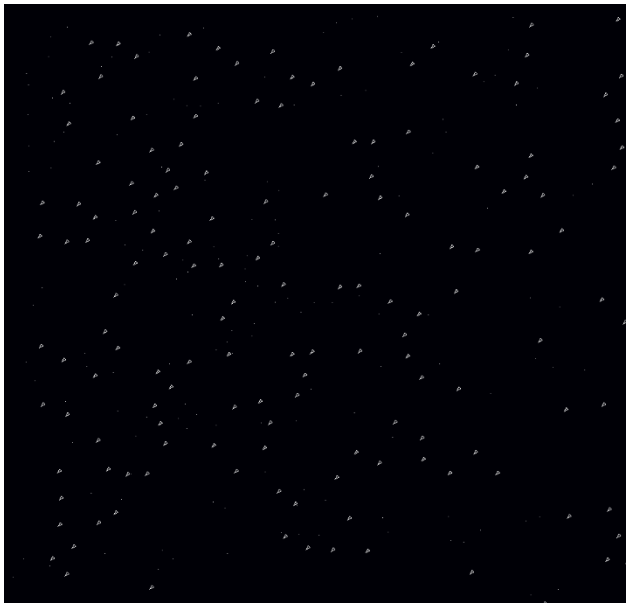

**Figure 3.** The display window shortly after simulation has started. Both species are spread out and in random positions. Starlings are depicted by triangles and larks are depicted by dots.

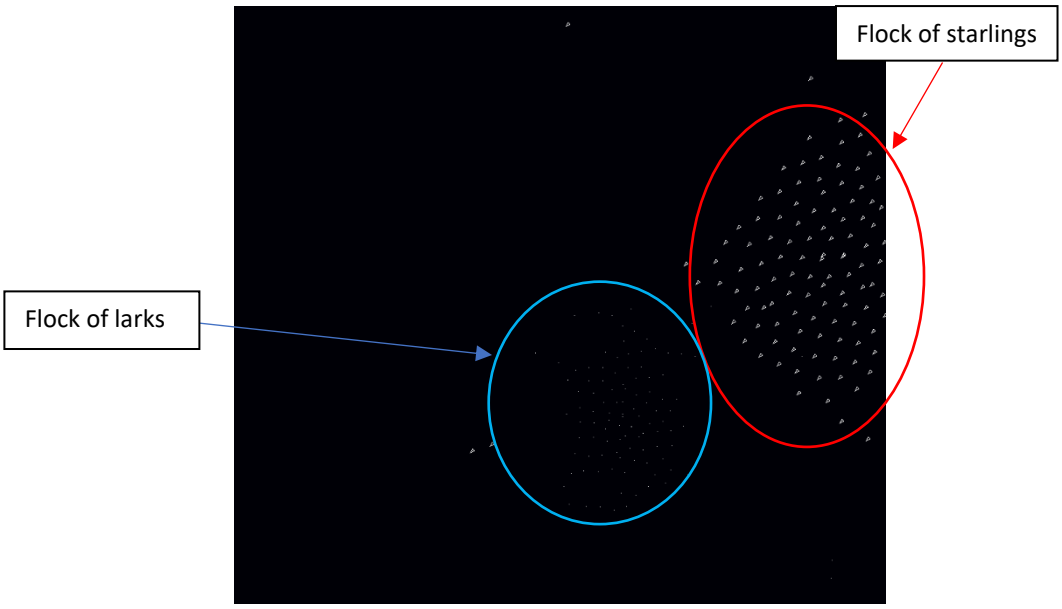


**Figure 4.** The display window at a time later than in Figure 3. The birds have formed flocks.

## 4. Improvements

An improvement which could be made is to make the birds flock in three dimensions, rather than two. An addition step that could be added to the code is to make one of the flocks hunt the other flock, an example of how this could be done by making the starling follow the larks, perhaps by altering the cohesion function between starlings and larks. Another idea for an extension onto the project presented is to have the flock split in two, and then re-join, when faced with an obstacle.

## 5. Conclusion

It was found that two flocks of birds of different species could be simulated in C++ using object oriented techniques. Three relatively simple rules (alignment, cohesion and separation) were implemented, and realistic flocking behaviour was able to be produced. Birds only interacted and applied the rules to other birds of the same species, and this was done using one flock object. A graphical user interface was used so that the user could interact with the program and a display window showing the motion of the birds could be seen.

## References

[1] Object-oriented Programming (OOP) in C++ [Internet]. Www3.ntu.edu.sg. 2018 [cited 19 January 2018]. Available from: http://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

[2] Why Birds Flock and Why It Matters [Internet]. The Spruce. 2018 [cited 19 January 2018]. Available from: https://www.thespruce.com/what-does-a-flock-of-birds-mean-386452

[3] Boids (Flocks, Herds, and Schools: a Distributed BehavioralModel) [Internet]. Red3d.com. 2018 [cited 19 January 2018]. Available from: https://www.red3d.com/cwr/boids/

[4] Boids Pseudocode [Internet]. Macs.hw.ac.uk. 2018 [cited 19 January 2018]. Available from: https://www.macs.hw.ac.uk/~dwcorne/Teaching/Boids%20Pseudocode.htm