Week 3 Exercise -readme

main.cpp

This code is the main function of a C++ program to start the OpenFrameworks application

```
#include "ofMain.h"
#include “testApp.h"
```

The first two header files are included. ofMain.h is the main header of the OpenFrameworks library, which contains a lot of OpenFrameworks functions and type declarations, while testApp.h is the header of the application we wrote ourselves.

```
int main() {
```

This is the main function of the program, the entry point to the program.

```
ofSetupOpenGL(1024, 768, OF_WINDOW);
```

This line calls the ofSetupOpenGL() function, which initializes the OpenGL context, sets the window size to 1024x768, and uses the normal window mode (OF_WINDOW).

```
ofRunApp(new testApp());
```

This line calls the ofSetupOpenGL() function, which initializes the OpenGL context, sets the window size to 1024x768, and uses the normal window mode (OF_WINDOW).

boid.cpp

This code implements a simple simulator of Boids, where Boids are virtual birds with clustering behavior and swarm intelligence.

```
#include "boid.h"
#include "ofMain.h"
```

Introduce header files boid.h and ofMain.h.

```
Boid::Boid()
{
    separationWeight = 1.0f;
    cohesionWeight = 0.2f;
    alignmentWeight = 0.2f;

    separationThreshold = 15;
    neighbourhoodSize = 100;
```

```
    position = ofVec3f(ofRandom(50, -100), ofRandom(-30, 50));
    velocity = ofVec3f(ofRandom(1, 2), ofRandom(-10, -2));
}
```

Defines the default constructor for the Boid class, initializing some of the properties of the Boid, such as weight, threshold, neighborhood size, and location and speed.Boid::Boid(ofVec3f &pos, ofVec3f &vel)

```
{
    separationWeight = 1.0f;
    cohesionWeight = 0.2f;
    alignmentWeight = 0.9f;

    separationThreshold = 10;
    neighbourhoodSize = 300;

    position = pos;
    velocity = vel;
}
```

Defines a parameterized constructor for the Boid class to create a Boid with the given position and speed.

```
Boid::~Boid()
{

}
```

Define the destructor for class Boid

```
float Boid::getSeparationWeight()
{
    return separationWeight;
}
float Boid::getCohesionWeight()
{
    return cohesionWeight;
}
```

```
float Boid::getAlignmentWeight()
{
    return alignmentWeight;
}
float Boid::getSeparationThreshold()
{
    return separationThreshold;
}
float Boid::getNeighbourhoodSize()
{
    return neighbourhoodSize;
}
void Boid::setSeparationWeight(float f)
{
    separationWeight = f;
}
void Boid::setCohesionWeight(float f)
{
    cohesionWeight = f;
}

void Boid::setAlignmentWeight(float f)
{
    alignmentWeight = f;
}
void Boid::setSeparationThreshold(float f)
{
    separationThreshold = f;
}

void Boid::setNeighbourhoodSize(float f)
{
    neighbourhoodSize = f;
}
```

Define functions that access and modify the properties of the Boid. These functions allow other classes to access and modify the Boid's properties.

```
ofVec3f Boid::separation(std::vector<Boid *> &otherBoids)
{
    // finds the first collision and avoids that
    // should probably find the nearest one
    // can you figure out how to do that?
    for (int i = 0; i < otherBoids.size(); i++)
    {
        if(position.distance(otherBoids[i]->getPosition()) < separationThreshold)
        {
            ofVec3f v = position -  otherBoids[i]->getPosition();
            v.normalize();
            return v;
        }
```

Calculate the separation, aggregation, and alignment of Boids from other Boids

boid.h

This is a header file for the Boid class, which is used to create objects that represent birds in a flocking simulation.

```
#ifndef _BOID
#define _BOID
```

These are preprocessor directives that ensure the Boid class is defined only once. If the header file is included multiple times in a project, the preprocessor directives prevent multiple definitions of the class.

```
#include <vector>
#include "ofMain.h"
```

These lines include the vector and ofMain header files in the current header file.

```
class Boid
{
// all the methods and variables after the
// private keyword can only be used inside
// the class
private:
```

This line begins the declaration of the Boid class. It defines a private section that can only be accessed by methods inside the class.

```
ofVec3f position;
ofVec3f velocity;

float separationWeight;
float cohesionWeight;
float alignmentWeight;

float separationThreshold;
float neighbourhoodSize;
```
These lines declare variables that store the position, velocity, and behavior weights for each boid. The separationWeight, cohesionWeight, and alignmentWeight variables determine how strongly the boid is influenced by each behavior. The separationThreshold and neighbourhoodSize variables are used to determine which other boids are considered neighbors.

```
ofVec3f separation(std::vector<Boid *> &otherBoids);
ofVec3f cohesion(std::vector<Boid *> &otherBoids);
ofVec3f alignment(std::vector<Boid *> &otherBoids);
```
These lines declare three private methods that calculate the separation, cohesion, and alignment behaviors for each boid.

```
public:
Boid();
Boid(ofVec3f &pos, ofVec3f &vel);

~Boid();

ofVec3f getPosition();
ofVec3f getVelocity();


float getSeparationWeight();
float getCohesionWeight();
float getAlignmentWeight();

float getSeparationThreshold();
float getNeighbourhoodSize();
```

```
void setSeparationWeight(float f);
void setCohesionWeight(float f);
void setAlignmentWeight(float f);

void setSeparationThreshold(float f);
void setNeighbourhoodSize(float f);

void update(std::vector<Boid *> &otherBoids, ofVec3f &min, ofVec3f &max);

void walls(ofVec3f &min, ofVec3f &max);

void draw();
void drawnew();
```

These lines declare the public methods for the Boid class. The constructor and destructor methods are declared, along with several methods to get and set the behavior weights and other parameters of the boid. The update method is used to update the boid's position and velocity based on its behavior and the positions of nearby boids. The walls method is used to keep the boids within the boundaries of the simulation. The draw and drawnew methods are used to draw the boid on the screen.

```
#endif
```

This line marks the end of the header file and ensures that the preprocessor directives are closed properly.

testApp.cpp

This code is an OpenFrameworks application source code, used to simulate the "Boids" algorithm of group behavior in nature.

```
#include "testApp.h"
```

This statement refers to the testApp.h file, which contains the class definitions for the application.

```
testApp::~testApp()
{
    for (int i = 0; i < boids.size(); i++)
    {
        delete boids[i];
    }
```

```
}
```

This function is a destructor of class testApp that frees memory allocated to the boids object.

```
void testApp::setup(){
    int screenW = ofGetScreenWidth();
    int screenH = ofGetScreenHeight();
    ofBackground(0,0,25);
    for (int i = 0; i < 50; i++)
        boids.push_back(new Boid());
    for (int i = 0; i < 50; i++)
        myboids.push_back(new Boid());
}
```

This is the TestApp-class setup function, which is called when the application starts running. It first gets the width and height of the screen, then sets the background color to dark blue. Then it creates 50 Boid objects and adds them to a vector named boids, and another 50 Boid objects and adds them to a vector named myboids.

```
void testApp::update(){
    ofVec3f min(0, 0);
    ofVec3f max(ofGetWidth(), ofGetHeight());
    for (int i = 0; i < boids.size(); i++)
    {
        boids[i]->update(boids, min, max);
    }
    for (int i = 0; i < myboids.size(); i++)
    {
        myboids[i]->update(myboids, min, max);
    }
}
```

This is the TestApp-like update function, which is called every frame of the application. It creates two ofVec3f objects, min and max, to represent the boundary. The update function is then called on each Boid object in the boids and myboids vectors to update their position and speed.

```
void testApp::draw(){
    for (int i = 0; i < boids.size(); i++)
    {
        boids[i]->draw();
```

```
    }
    for (int i = 0; i < myboids.size(); i++)
    {
        myboids[i]->drawnew();
    }
}
```

This is the TestApp-like draw function, which is called every frame of the application. It calls the draw function on each Boid object in the boids and myboids vectors, drawing them to the screen.