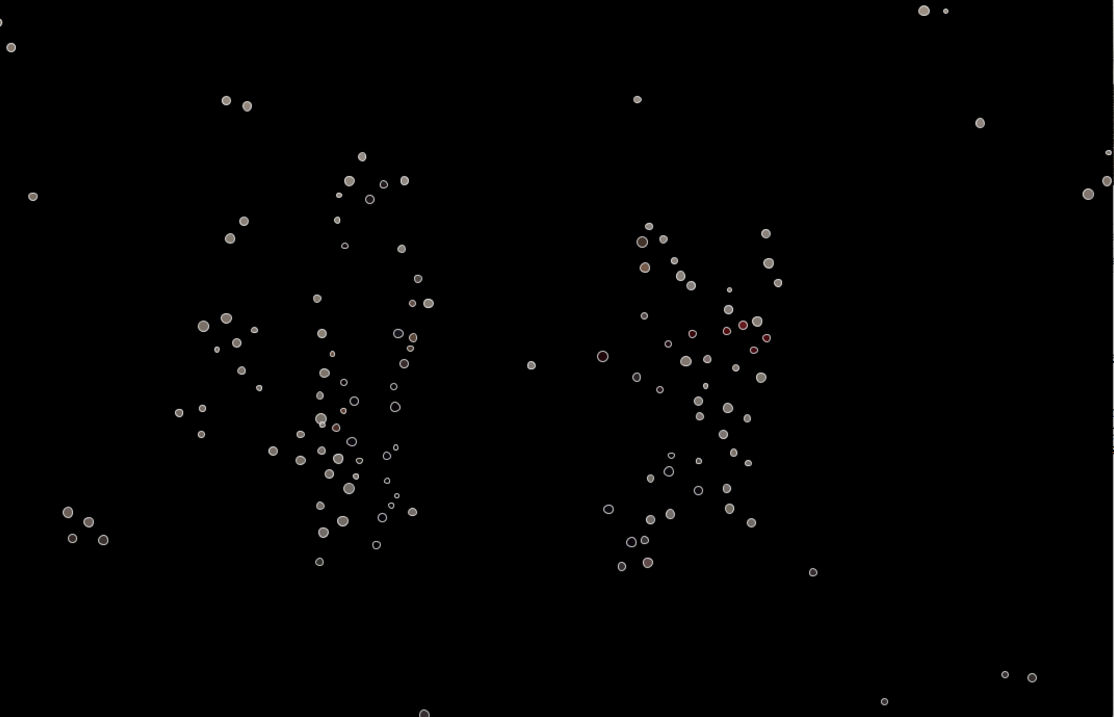
**Choreographic Swarm**

**Amy Cartwright**



This report shows the process undertaken to take a small project within Processing. It demonstrates my understanding of the fundamental principles of programming. I have included first the idea for each section, then the pseudocode in italics and how I translated that into code underneath. Any of the code seen in this report which has not been written by myself has been fully referenced and you can find links to original in the references section at the end.

Note, I have not included comments within the snippets of code scattered through this report. However, in the appendix you can find the full, functioning code for the project which is fully commented.

**My Idea**

I wanted to explore the possibility of using processing to create a dance performance. A loose computational choreography. I decided to try and achieve this by creating a choreographic loop, the dancer moves in front of the webcam and the program will take the data from the movement and use a swarm of particles to respond. The dancer continues to improvise but taking their movement cues from the swarm that the computer has created. I am interested in the change in the dancer’s movement style over the course of this performance. Will the movement move so far from what the dancer presented to begin with that we are able to say that the program is dictating the choreography? Will the dancer feel any noticeable changes in their own movement style as the performance progresses? Would an audience observing the performance be able to understand the loop and will it change their response to the performance knowing that the movement is dictated by a computational program?

These are fairly big questions to explore and will no doubt require a lengthy research process in order to attempt answer them. However, I decided to use this project as an opportunity to take the first step towards being able to conduct this research.

The program this report documents uses movement data gathered from a pre-recorded video to dictate the movement of a swarm of particles. The particles generate with each loop of the video and attempt to swarm. Added into the traditional swarm movement is a pull to the coordinates of the movement from the video.

To begin, I split the project into smaller tasks

1. Write a program to play the video
2. Collect the co-ordinates of the movement within the video and store it ready for use
3. Write a separate particle system program
4. Write a new ‘seek’ algorithm for the swarm
5. Integrate the two programs
6. Appearance - Write an algorithm to color the pixels from the video
7. Data Hiding – protecting my variables

**Part 1 - Play Video**

This was a very simple step in the process however it is important to ensure that the setup for the video playback is correct before continuing to the next step. This program does not require any classes

* *Import Video library – giving access to the movie class*
* *Declare new instance of movie class*

*Setup:*

* *Initialize the movie object-detailing which file from the Data folder you would like it to play*

*Draw:*

* *Tell the video to play - in this case as I want the video to play continuously tell it to loop*
* *Read each frame from the movie*
* *Display each frame*

Movie video;

void setup(){

size(1080, 720);

video = new Movie(p, "try.mov");

video.loop();

}

void draw(){

if (video.available()) {

video.read();

video.loadPixels();

p.image(video, 0, 0);

video.volume(0);

}

}

**Part 2 - Frame Differencing**

Now that the video is set up I need to find a way to collect and store the movement data. I will do this using frame differencing. This method is adapted from Daniel Shiffman’s Motion Detection method in ‘Learning Processing’ Pg. 296

* *Create a new PImage*

*Setup:*

* *Initialize the image object*

*Draw:*

* *Save each video frame as an image*
* *Copy the previous frame onto the current one*
* *Use Shiffman’s distance squared method to compare the two frames and check for any pixels that have a difference over a set threshold*
* *Store pixels*

PImage prev;

int myWidth, myHeight;

float threshold, theta;

void setup(){

size(584, 424);

background(0);

myWidth = 584;

myHeight = 424;

video = new Movie(this, "BaharEditShort.mov");

prev = createImage(myWidth, myHeight, RGB);

video.loop();

threshold = 60;

}

void draw() {

if (video.available()) {

prev.copy(video, 0, 0, video.width, video.height, 0, 0, prev.width, prev.height);

prev.updatePixels();

video.read();

video.loadPixels();

prev.loadPixels();

image(video, 0, 0);

loadPixels();

ArrayList<Point> points= new ArrayList<Point>();

fdPoints.clear();

for (int x = 0; x < video.width; x++ ) {

for (int y = 0; y < video.height; y++ ) {

int loc = x + y \* video.width;

int currentColor = video.pixels[loc];

float r1 = p.red(currentColor);

float g1 = p.green(currentColor);

float b1 = p.blue(currentColor);

int prevColor = prev.pixels[loc];

float r2 = p.red(prevColor);

float g2 = p.green(prevColor);

float b2 = p.blue(prevColor);

float diff = distSq(r1, g1, b1, r2, g2, b2);

if (diff > threshold\*threshold) {

fdPoints.add(new PVector(x, y));

}

}

}

updatePixels();

}

float distSq(float x1, float y1, float z1, float x2, float y2, float z2) {

float diff = (x2-x1)\*(x2-x1) + (y2-y1)\*(y2-y1) +(z2-z1)\*(z2-z1);

return diff;

}

**Part 3 – Swarm Program**

Now that I have my video program producing movement data I need to write my swarm. To keep things a little simpler at this stage, I started this in a separate program. I knew that in order for the particles to move in a swarm I needed to use the three rules laid out by Craig Reynolds in his 1986 paper ‘Flocks, Herds and Schools: A distributed behavioral model’. The particles need to adhere to three algorithms, Separate, Align and Centre. For these algorithms I used code written by Lior Ben-Gai and influenced by Shiffman’s example in ‘The Nature of Code’ Pg. 308. I will explain one here, to see the full implementation of all three algorithms please refer to the full code in the appendix.

* *Create a new temporary PVector to store current particle position*
* *Begin a count to determine number of neighbours*
* *Loop over all particles in swarm*
* *Work out the distance between current particle and neighbor*
* *if the distance is greater than 0 and less than 30 add neighbours velocity to current position. Increase the count*
* *if count is 0, return nothing*
* *divide temp PVector by count*
* *increase the number*
* *add temp to the velocity of current particle*

void align(ArrayList<ParticleSwarm> swarm) {

PVector temp = new PVector(0, 0);

int count = 0;

for (int i = 0; i < swarm.size(); i++) {

ParticleSwarm neighbour = swarm.get(i);

float dist = loc.dist(neighbour.loc);

if (dist < 30 && dist > 0) {

temp.add(neighbour.vel);

count++;

}

}

if (count == 0) return;

temp.div(count);

temp.mult((float)0.3);

vel.add(temp);

}

All of this happens within a specific particle class. So what is happening within the main tab:

* *create a new ArrayList of particle objects*
* *set number of particles*

*setup:*

* *initialize the ArrayList of particle objects*

*draw:*

* *loop through particles*
* *call display function*
* *call move function*
* *if particles are dead remove them and add fresh ones*

ArrayList<ParticleSwarm> swarm;

int numPart = 50;

void setup() {

swarm = new ArrayList();

for (int i = 0; i < numPart; i++) {

swarm.add(new ParticleSwarm(random(width), random(height), this));

}

}

void draw() {

for (int i = 0; i < swarm.size(); i++) {

ParticleSwarm p = swarm.get(i);

p.display(this);

p.move(this, swarm);

if (p.dead()) {

swarm.remove(i);

swarm.add(new ParticleSwarm(random(width), random(height), this));

}

}

**Part 4 - New ‘seek’ algorithm**

A main part of this program is the use of the movement data from the video to power the swarm. To achieve this, I had to write a new behavior for the swarm. I want the particles to seek the points taken from the movement data.

* *Create a temporary empty vector*
* *Loop through all fPpoints*
* *Find location of all fdPoints*
* *Calculate distance between location of particles and location of fdPoints*
* *If distance is less than threshold move particle towards fdPoints*
* *Normalize the acceleration*
* *Scale the acceleration*
* *Add to velocity*

void seek(ArrayList<Particle> swarm, ArrayList<PVector> fdPoints) {

PVector temp = new PVector(0, 0);

for (PVector fd : fdPoints) {

float dist = loc.dist(fd);

if (dist < seekDist && dist > 0) {

PVector acc = fd.sub(loc);

acc.normalize();

acc.mult(seekVar);

temp.add(acc);

}

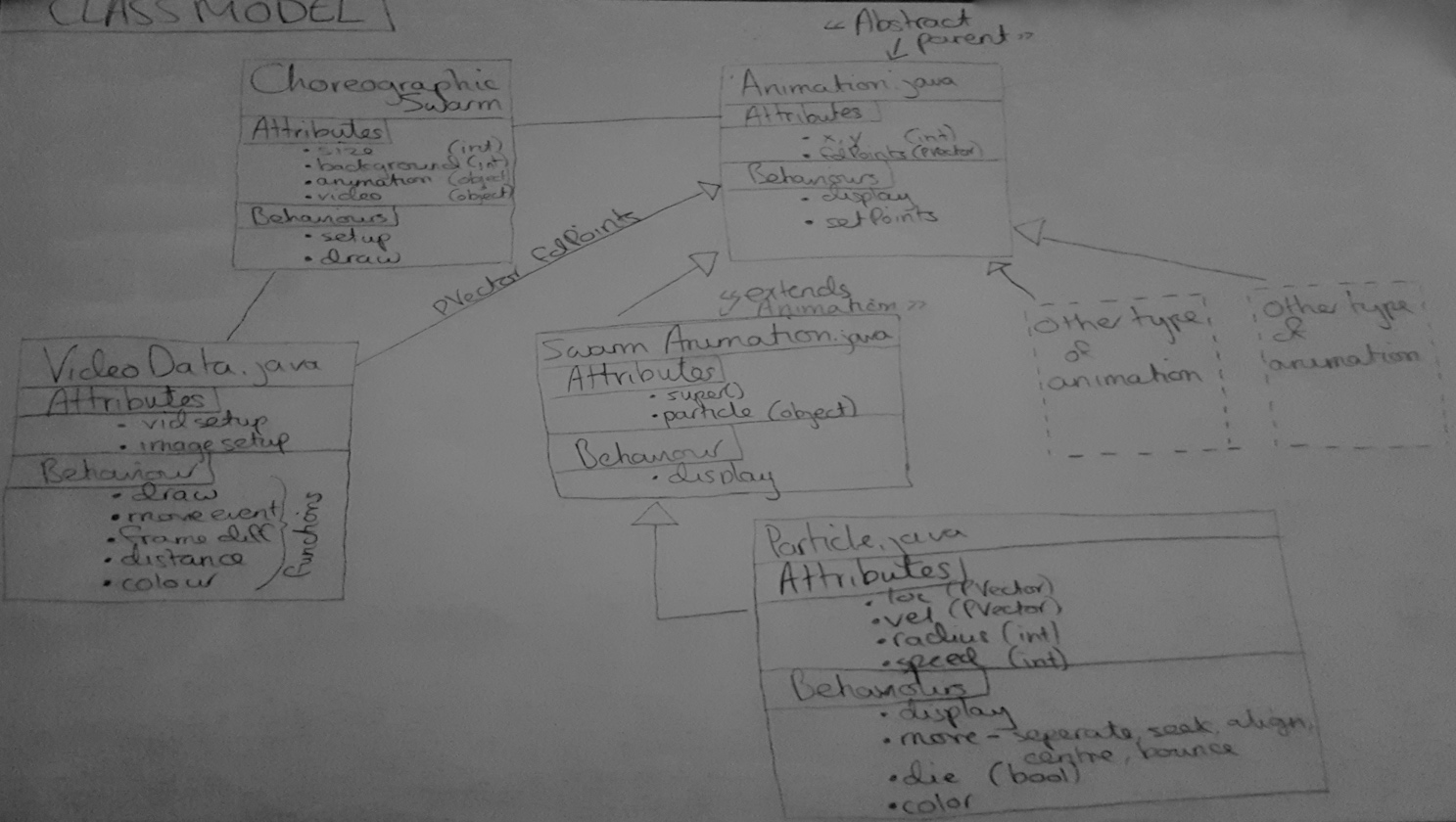
}

vel.add(temp);

}

**Part 5 - Integration and Class Model**

Before integrating the two programs, I wrote a class model to better understand the structure of the final program.



Many things needed to be changed when I integrated the two programs. I will discuss some of the more significant ones here.

As I am using .java tabs for all of my classes, I need to ensure that I have imported bothe the processing core library and the java util library. I also need to use PApplet whenever I use any syntax that comes from the processing library. Example below:

import processing.core.\*;

import java.util.\*;

Particle(float x, float y, PApplet p) {

velX = p.random(-1f, 1f);

velY = p.random(-1f, 1f);

}

One of the main focuses with the integration here was the use of inheritance. I made a new class ‘Animation which acts as the parent class from which SwarmAnimation extends. Animation is a functional class that sets the fdPoints to an x, y variable which can then be used for additional animations that may be added at a later date.

abstract class Animation {

public float x, y;

ArrayList<PVector> fdPoints;

public void display(PApplet p, VideoData video) {

for (PVector po : fdPoints) {

x = po.x;

y = po.y;

}

}

public void setPoints(ArrayList<PVector> po, PApplet p) {

fdPoints = po;

}

}

I also had to change the parameters which draw the initial particles within SwarmAnimation. As the SwarmAnimation class extends the Animation class, I can call super() in both the constructor and the display which allows me to use the parameters set within Animation. The line of code below shows how I can now use the x, y that are set within animation when calling a particle.

swarm.add(new Particle(x, y, p));

As the video processing required a larger amount of code I decided to move this into a separate class which can then be called in the main class.

**Part 6 - Improve the appearance**

Now that the program has been integrated and is running as expected I will work on the appearance. I would like the particles to take on the color of the pixels at the point of movement in the video underneath.

*Within the VideoData class:*

* *Function that returns the colour ofpixels in video at x, y*

*Within the Particle class:*

* *A function that sets color plus functions to access the location of particles*

*Within the SwarmAnimation class:*

* *Get the colors by taking x and y loc from particle class and set the colors in the display function*

VideoData

int getRed(int x, int y, PApplet p) {

return (int)p.red(video.get(x, y));

}

int getGreen(int x, int y, PApplet p) {

return (int)p.green(video.get(x, y));

}

int getBlue(int x, int y, PApplet p) {

return (int)p.blue(video.get(x, y));

}

Particle

public float getX() {

return loc.x;

}

public float getY() {

return loc.y;

}

void setColor(int red, int green, int blue) {

r = red;

g = green;

b = blue;

}

SwarmAnimation

for (int i = 0; i < swarm.size(); i++) {

Particle pa = swarm.get(i);

int x = (int)pa.getX();

int y = (int)pa.getY();

int red = video.getRed(x, y, p);

int green = video.getGreen(x, y, p);

int blue = video.getBlue(x, y, p);

pa.setColor(red, green, blue);

pa.display(p);

}

**Part 7 - Data Hiding**

Data hiding allows me to protect certain variables and methods and make them inaccessible to the client. This means that the client will not be able to make changes that may be crippling to the functionality of the code. As you can see in the code below I have labeled each one of my functions as public, private or protected. Private if they are only used within the class they are written, protected if I need to be able to access them within another class, public if it is a function that the client may need to adapt.

In some cases, I have labeled a variable as private but have used a ‘getters’ function in order to access it. An example:

I need to keep my PVector of locations private as if this were to be adjusted my swarm would not work in the desired way. I do however need to access these locations when setting the color, as seen above. To do this, I use this line of code:

public float getX() { return loc.x;}

public float getY() { return loc.y;}

I use the same technique to control the number of particles.

**References**

* Shiffman, D. (2008) Learning Processing Burlington: Elsevier
* Shiffman, D. (2012) The Nature of Code USA: Shiffman
* Processing online reference - https://processing.org/reference/
* Lior Ben-Gai -https://drive.google.com/open?id=0B2ZtRje\_7ZHYT2tMNXhReWtWdjg

**Appendix**

The full source code for this project:

/////////////////////////////////////////////// Choreographic Swarm //////////////////////////////////////////////

/\*Created by: Amy Cartwright

Year: 2017

A program that creates a swarm of particles that are dictated by movement data from a pre-recorded video. A starting point for research into computational choreography.

Known bugs:

1. The particles always first appear in the top left corner of the screen. They should appear at location of movement.

\*/

import processing.video.\*;

Animation a; //declare new instance of Animation class

VideoData video; //declare new instance of VideoData class

ArrayList<PVector> fdPoints = new ArrayList<PVector>(); //New arrayList of PVectors storing movement data

void setup() {

size(1080, 720); //for best results set to size of video

background(0); //black

a = new SwarmAnimation(this); //initialize Animation object

video = new VideoData(this); //initialize VideoData object

}

void draw() {

video.draw(this, fdPoints); //call the draw function from the video class

a.setPoints(fdPoints, this); //call to set the points from the array list

video.update(this); //call the update function from the VideoData class

background(0); //draws over the video

a.display(this, video); //display the animation

}

//Class for running the video and extracting the movement data

import processing.core.\*;

import java.util.\*;

import processing.video.\*;

public class VideoData {

Movie video; //new instance of Movie class

PImage prev; //new instance of Image class

public int myWidth, myHeight;

private float threshold;

VideoData(PApplet p) {

myWidth = 1080;

myHeight = 720;

video = new Movie(p, "try.mov"); //Initialize video object

//create an image using the previous frame

prev = p.createImage(myWidth, myHeight, p.RGB);

video.loop();

threshold = 60;

}

public void draw(PApplet p, ArrayList<PVector> fdPoints) {

if (video.available()) {

//copy the prev frame onto current frame

prev.copy(video, 0, 0, video.width, video.height, 0, 0, prev.width, prev.height);

prev.updatePixels(); //update image pixels

video.read();

video.loadPixels();

prev.loadPixels();

p.image(video, 0, 0); //display video

p.loadPixels();

video.volume(0);

frameDiff(fdPoints, p); //run the frameDiff function

}

}

void update(PApplet p) {

p.updatePixels(); //update video pixels

}

//----------------------------------------------------------------------

//Method to calculate the difference between current frame and previous and store in an array. Method is only useful within this class so has been declared private

private void frameDiff(ArrayList <PVector> fdPoints, PApplet p) {

fdPoints.clear();

// Loop through every pixel

for (int x = 0; x < video.width; x++ ) {

for (int y = 0; y < video.height; y++ ) {

int loc = x + y \* video.width;

// current color

int currentColor = video.pixels[loc]; //pixel array

float r1 = p.red(currentColor);

float g1 = p.green(currentColor);

float b1 = p.blue(currentColor);

// previous frame color

int prevColor = prev.pixels[loc];

float r2 = p.red(prevColor);

float g2 = p.green(prevColor);

float b2 = p.blue(prevColor);

//calculate distance2 between previous and current color

float diff = distSq(r1, g1, b1, r2, g2, b2);

// if the difference is larger than threshold2 add the x, y point so fdPoints PVector

if (diff > threshold\*threshold) {

fdPoints.add(new PVector(x, y));

}

}

}

}

//----------------------------------------------------------------------

//functions to get the r,g,b color from the x, y locations within the video

int getRed(int x, int y, PApplet p) {

return (int)p.red(video.get(x, y));

}

int getGreen(int x, int y, PApplet p) {

return (int)p.green(video.get(x, y));

}

int getBlue(int x, int y, PApplet p) {

return (int)p.blue(video.get(x, y));

}

//----------------------------------------------------------------------

//Method to calculate the differences between frames

private float distSq(float x1, float y1, float z1, float x2, float y2, float z2) {

float diff = (x2-x1)\*(x2-x1) + (y2-y1)\*(y2-y1) +(z2-z1)\*(z2-z1);

return diff;

}

}

//Functional parent class which takes the data from frame differencing and allocated it for use in child classes

import processing.core.\*;

import java.util.\*;

//abstract class as this class contains methods that are declared but contain no implementation

abstract class Animation {

public float x, y;

ArrayList<PVector> fdPoints;

public void display(PApplet p, VideoData video) {

//loop through arrayList holding movement data and assign each an x and y

for (PVector po : fdPoints) {

x = po.x;

y = po.y;

}

}

public void setPoints(ArrayList<PVector> po, PApplet p) {

//set points from the po arraylist to the fdPoints arraylist

fdPoints = po;

}

}

//Child class extending from Animation. It creates new instances of the Particle.

import processing.core.\*;

import java.util.\*;

public class SwarmAnimation extends Animation {

ArrayList<Particle> swarm;

private int numPart;

public int maxPart = 150;

//----------------------------------------------------------------------

//'getter' & 'setter' functions to access the number of parts

public int getNumPart() {

return numPart;

}

public void setNumPart(int num) {

numPart = num;

}

//----------------------------------------------------------------------

SwarmAnimation(PApplet p) {

super(); //do not override animation constructor, add to it

swarm = new ArrayList<Particle>();

setNumPart(getNumPart() + maxPart); //set num oof parts using functions above

//add particles to swarm arrayList

for (int i = 0; i < numPart; i++) {

swarm.add(new Particle(x, y, p));

}

}

public void display(PApplet p, VideoData video) {

super.display(p, video); //do not override animation display, add to it

//loop over particles addressing each one

for (int i = 0; i < swarm.size(); i++) {

Particle pa = swarm.get(i);

//get x & y locations of particles

int x = (int)pa.getX();

int y = (int)pa.getY();

//get r,g,b colours of video at x and y locations of particles

int red = video.getRed(x, y, p);

int green = video.getGreen(x, y, p);

int blue = video.getBlue(x, y, p);

pa.setColor(red, green, blue); //set colours of video

pa.display(p);

pa.move(p, swarm, fdPoints);

//If the boolean returns true remove particles and add new ones in their place

if (pa.dead()) {

swarm.remove(i);

swarm.add(new Particle(x, y, p));

}

}

}

}

//A particle class containing behaviours based on flocking algorithms written by Craig Reynolds and seen in 'The Nature of Code' pg.308

import processing.core.\*;

import java.util.\*;

final class Particle {

private PVector loc = new PVector(0, 0);

private PVector vel = new PVector(0, 0);

private float velX, velY;

public float radius, speed, life;

private float seekVar, alignVar, centreVar, sepVar;

private float seekDist, alignDist, centreDist, sepDist;

int r, b, g;

Particle(float x, float y, PApplet p) {

loc.set(x, y);

velX = p.random(-1f, 1f);

velY = p.random(-1f, 1f);

speed = 2;

vel.set((speed \* velX), (speed \* velY));

radius = p.random(2, 5);

life = 255; //set to 255 so can reduce slowly and particles will fade

//variables for use in the individual 'rule' algorithms

seekVar = 2.0f;

alignVar = 0.3f;

centreVar = 0.001f;

sepVar = 0.07f;

seekDist = 20;

alignDist = 30;

centreDist = 40;

sepDist = 20;

}

//----------------------------------------------------------------------

//'getter'functions to access the location of particles

public float getX() {

return loc.x;

}

public float getY() {

return loc.y;

}

//----------------------------------------------------------------------

public void move(PApplet p, ArrayList<Particle> swarm, ArrayList<PVector> fdPoints) {

vel.limit(speed);//limit the speed

loc.add(vel);//add velocity to location to intiialise movement

life -=0.5; //Reduce lifespan

//call each of the algorithms here to reduce code in the swarmAnimation class

seek(p, swarm, fdPoints);

seperate(p, swarm);

centre(p, swarm);

align(p, swarm);

bounce(p);

}

//----------------------------------------------------------------------

public void display(PApplet p) {

p.stroke(255, life);

p.fill(r, g, b, life); //set color from the set color function below

p.ellipseMode(p.RADIUS);

p.ellipse(loc.x, loc.y, radius, radius);

}

//----------------------------------------------------------------------

//Find out if the particles is dead or alive. If dead return true, else return false. Needs to

//be accessed by another class but I dont want to be accessable to the client

protected boolean dead() {

if (life < 0.0) return true;

else return false;

}

//----------------------------------------------------------------------

// Particles are pulled towards points of movement in video

private void seek(PApplet p, ArrayList<Particle> swarm, ArrayList<PVector> fdPoints) {

PVector temp = new PVector(0, 0);

//Loop through all fdpoints and find location

for (PVector fd : fdPoints) {

float dist = loc.dist(fd);

//If distance is less than threshold move particles towards

if (dist < seekDist && dist > 0) {

PVector acc = fd.sub(loc);

acc.normalize();

acc.mult(seekVar);

temp.add(acc);

}

}

vel.add(temp);

}

//----------------------------------------------------------------------

//Particles seperate if they are too close to others

private void seperate(PApplet p, ArrayList<Particle> swarm) {

PVector temp = new PVector(0, 0);

for (Particle neighbour : swarm) {

float dist = loc.dist(neighbour.loc);

//if the distance is greater than 0 and less than 20 minus neighbours position from current position

if (dist < sepDist && dist > 0) {

temp.sub(PVector.sub(neighbour.loc, loc));

}

}

temp.mult(sepVar);

vel.add(temp);

}

//----------------------------------------------------------------------

//Particles move away from neighbour if they are too close

private void avoid(PApplet p, ArrayList<Particle> swarm) {

PVector temp = new PVector(0, 0);

int count = 0;

for (Particle neighbour : swarm) {

float dist = loc.dist(neighbour.loc);

//if the distance is greater than 0 and less than 20 minus neighbours position from current position

if (dist < centreDist && dist > 0) {

temp.add(neighbour.loc);

count++;

}

}

if (count == 0) return;

temp.div(count);

temp.sub(loc);

temp.mult(centreVar);

vel.add(temp);

}

//----------------------------------------------------------------------

//Particles attempt to take the same path as others they are close to

private void align(PApplet p, ArrayList<Particle> swarm) {

PVector temp = new PVector(0, 0);

int count = 0;

for (Particle neighbour : swarm) {

float dist = loc.dist(neighbour.loc);

//if the distance is greater than 0 and less than 20 minus neighbours position from current position

if (dist < alignDist && dist > 0) {

temp.add(neighbour.vel);

count++;

}

}

if (count == 0) return;

temp.div(count);

temp.mult(alignVar);

vel.add(temp);

}

//----------------------------------------------------------------------

//Method to allow particles to bounce when they hit edge of window

private void bounce(PApplet p) {

int width = p.width;

int height = p.height;

if (loc.x > (width - radius) || loc.x < (0 + radius)) vel.x \*= -1;

if (loc.y > (height - radius) || loc.y < (0 + radius)) vel.y \*= -1;

}

//----------------------------------------------------------------------

//Method to setColor of particles

void setColor(int red, int green, int blue) {

r = red;

g = green;

b = blue;

}

}