**sketch12.js**

// This example of Craig Reynolds'steering formula in action

// (steering = desired-velocity) is from Dan Shiffman's

// Processing book, The Nature of Code and modified for p5.js by al.

// al 14 November 2016

var gPop = 100;

var gAliWt = 2;

var gSepWt = 2;

var gStaWt = 2;

var gTarWt = 6;

var gLocalityRad = 100;

var gSepMult = 4.5;

var gThings = [];

function setup() {

createCanvas(1000,600);

background("#fefefe")

gThings.length = gPop;

for (var i = 0; i < gPop; i++) {

gThings[i] = new Thing(random(width),random(height));

}

createP("Flocking with Reynolds' three rules - separation (avoid), alignment (copy), cohesion (centre) (");

createP("sketch12.js");

}

function draw() {

gTarget = createVector(mouseX,mouseY);

background("#fefefe");

for (var i = 0; i < gThings.length; i++) {

gThings[i].applyBehaviours(gThings);

gThings[i].update();

gThings[i].render();

}

}

function Thing(startX,startY) { // thing constructor

this.d = 5;

this.maxSpeed = 7;

this.maxForce = 0.01;

this.radar = gLocalityRad; // neighbour sensing - set to global value

// euler integration physics engine

this.acc = createVector(0,0);

this.vel = createVector(0,0);

this.pos = createVector(startX,startY);

// vectors for Reynolds' steering formula

this.desired = createVector(0,0);

this.steering = createVector(0,0);

this.update = function() {

this.vel.add(this.acc);

this.vel.limit(this.maxSpeed);

this.pos.add(this.vel);

this.acc.mult(0); // reset acc after each update

} // update

this.applyForce = function(aForce) {

this.acc.add(aForce);

} // applyForce

this.applyBehaviours = function(thingArray) {

this.sepVect = this.separate(thingArray);

this.sepVect.mult(gSepWt);

this.aliVect = this.align(thingArray);

this.aliVect.mult(gAliWt);

this.staVect = this.stayTogether(thingArray);

this.staVect.mult(gStaWt);

this.tarVect = this.seekTarget(gTarget);

this.tarVect.mult(gTarWt);

this.applyForce(this.sepVect);

this.applyForce(this.aliVect);

this.applyForce(this.staVect);

this.applyForce(this.tarVect);

} // applyBehaviours

this.align = function(thingArray) {

closeSum = createVector();

closeCount = 0;

for (var i = 0; i < thingArray.length; i++) {

locSep = p5.Vector.dist(this.pos,thingArray[i].pos);

if (locSep>0 && locSep<this.radar) {

closeSum.add(thingArray[i].vel);

closeCount++

}

}

if (closeCount>0) {

closeSum.div(closeCount);

closeSum.normalize();

closeSum.mult(this.maxSpeed);

this.steering = p5.Vector.sub(closeSum,this.vel);

this.steering.limit(this.maxForce);

} else {

this.steering.mult(0);

}

return this.steering;

} // align

this.seekTarget = function(target) {

this.desired = p5.Vector.sub(target,this.pos);

this.distTarget = this.desired.mag();

this.desired.normalize();

this.desired.mult(this.maxSpeed);

this.steering = p5.Vector.sub(this.desired,this.vel);

this.steering.limit(this.maxForce);

return this.steering;

} // seekTarget

this.separate = function(thingArray) {

this.desSep = this.d\*gSepMult;// desired separation in body widths

closeSum = createVector();

closeCount = 0;

for (var i = 0; i < thingArray.length; i++) {

locSep = p5.Vector.dist(this.pos,thingArray[i].pos);

if (locSep>0 && locSep<this.desSep){

diffVect = p5.Vector.sub(this.pos,thingArray[i].pos);

diffVect.normalize();

diffVect.div(locSep);

closeSum.add(diffVect);

closeCount++;

} //if

} //for

if (closeCount>0) {

closeSum.div(closeCount);

closeSum.normalize();

closeSum.mult(this.maxSpeed);

this.steering = p5.Vector.sub(closeSum,this.vel);

this.steering.limit(this.maxForce);

} else {

this.steering.mult(0);

} //else

return this.steering;

} // separate

this.stayTogether = function(thingArray) {

closeSum = createVector();

closeCount = 0;

for (var i = 0; i < thingArray.length; i++) {

locSep = p5.Vector.dist(this.pos,thingArray[i].pos);

if (locSep>0 && locSep<this.radar) {

closeSum.add(thingArray[i].pos);

closeCount++

}

}

if (closeCount>0) {

closeSum.div(closeCount);

closeSum.normalize();

closeSum.mult(this.maxSpeed);

this.steering = p5.Vector.sub(closeSum,this.loc);

this.steering.limit(this.maxForce);

} else {

this.steering.mult(0);

} // else

return this.steering;

} //stayTogether

this.renderTemp = function() {

noStroke();

fill(127,127,0,72);

ellipse(this.pos.x,this.pos.y,this.d);

} // render

this.render = function() {

// Draw a triangle rotated in the direction of velocity

theta = this.vel.heading() + radians(90);

fill(127,127,0,127);

noStroke();

push();

translate(this.pos.x,this.pos.y);

rotate(theta);

beginShape();

vertex(0, -this.d\*2);

vertex(-this.d, this.d\*2);

vertex(this.d, this.d\*2);

endShape(CLOSE);

pop();

} // update

} // Thing