FlockingBirds2

```
// It know it doesn&apos;t work proporly, but I can&apos;t understand why.
// I still hope it shows that I understand the concept

float randomX;
float randomY;

float r = 4; // radius
int numObj = 30;

// +1 becouse of the predator
PVector[] acc = new PVector[numObj+1];
PVector[] pos = new PVector[numObj+1];
PVector[] vel = new PVector[numObj+1];
float[] theta = new float[numObj+1];
PVector target = new PVector(0, 0);

PVector rule1 = new PVector(0, 0);
PVector rule2 = new PVector(0, 0);
PVector rule3 = new PVector(0, 0);
float avgPosWeight = 1;
float avgVelWeight = 1;
float seperationWeight = 1.5;



//PVector acc = new PVector(0,0);
//PVector pos = new PVector(width/2,height/2);
//PVector vel = new PVector(0,-2);

float visionField = 50;
float seperationField = 25;

float maxSpeed = 3;
float predatorMaxSpeed = 7;
float maxForce = 0.05;

boolean once = false;

void setup() {
 size(800, 600);
 background(10, 10, 50);


 // sets acceleration, and random start velocity and positions for birds
 for(int i=0; i<numObj+1; i++){
   randomX = random(0, width);
   randomY = random(0, height);
   pos[i] = new PVector(randomX, randomY);

   randomX = random(-1, 1);
   randomY = random(-1, 1);
   vel[i] = new PVector(randomX, randomY);

   acc[i] = new PVector(0, 0);
 }


}


void draw() {
 background(10, 10, 50);

 if(once == false){

 }

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

   // rules to create target
   rule3 = seperation(i);
   rule2 = avgVel(i);
   rule1 = avgPos(i);

   rule3.mult(seperationWeight);
   rule2.mult(avgVelWeight);
   rule1.mult(avgPosWeight);

   acc[i].add(rule3);
   acc[i].add(rule2);
   acc[i].add(rule1);
```

```
  //PVector test = PVector.sub(rule2, rule3);

  // movment




  if(checkForPredator(i)){
    target = avoidPredator(i);
    target.mult(20);
  }

  update(i);
  checkEdges(i);
  display(i);
 }



/*


  //predator
  target = avgPos(numObj);
  steer(numObj);
  update(numObj);
  checkEdges(numObj);
  display(numObj);

  */

 once = true;

}



PVector avgPos(int i){
 int counter = 0;
 float sumX = 0;
 float sumY = 0;
 PVector sum;
 for(int a=0; a<numObj; a++){
  if(dist(pos[i].x, pos[i].y, pos[a].x, pos[a].y) <= visionField && a != i){
    sumX += pos[a].x;
```

```
     sumY += pos[a].y;
   counter++;
   }
 }

  if(counter == 0){
   return new PVector(0, 0);
 }

 sum = new PVector(sumX/counter, sumY/counter);
 sum = PVector.sub(sum, pos[i]);
 sum.normalize();
 sum.mult(maxSpeed);

 sum = PVector.sub(sum, vel[i]);
 sum.limit(maxForce);
 return sum;

 /*
 return new PVector(avgPos.x-pos[i].x, avgPos.y-pos[i].y).normalize();
 */



}

PVector avgVel(int i){
 int counter = 0;
 float sumX = 0;
 float sumY = 0;
 PVector sum;
 for(int a=0; a<numObj; a++){

   if(dist(pos[i].x, pos[i].y, pos[a].x, pos[a].y) <= visionField && i != a){
   sumX += vel[a].x;
   sumY += vel[a].y;
   counter++;
   }
 }
```

```
  if(counter == 0){
   return new PVector(0, 0);
  }

 sum = new PVector(sumX/counter, sumY/counter).normalize();
 sum.mult(maxSpeed);
 sum = PVector.sub(sum, vel[i]);
 sum.limit(maxForce);
 return sum;



}

PVector seperation(int i){
 PVector avgPos;
 int counter = 0;
 float sumX = 0;
 float sumY = 0;
 PVector sum = new PVector(0, 0);
 PVector sum2 = new PVector(0, 0);

 for(int a=0; a<numObj; a++){
   // 0 is self
   if(dist(pos[i].x, pos[i].y, pos[a].x, pos[a].y) <= seperationField && dist(pos[i].x, pos[i].y, pos[a].x,
pos[a].y) != 0){
     sum = PVector.sub(pos[i], pos[a]);
     sum.normalize();
     sum.div(dist(pos[i].x, pos[i].y, pos[a].x, pos[a].y));
     sum2.add(sum);
     counter++;
   }
 }



 if(counter != 0){
   sum2 = new PVector(sumX/counter, sumY/counter);
 }

 if (sum2.mag() > 0) {
     // Implement Reynolds: Steering = Desired - Velocity
     sum2.normalize();
```

```
    sum2.mult(maxSpeed);
    sum2.sub(vel[i]);
    sum2.limit(maxForce);
   }

 return sum2;

}

boolean checkForPredator(int i){
 if(dist(pos[i].x, pos[i].y, pos[numObj].x, pos[numObj].y) <= visionField){
   return true;
 }

 return false;
}

PVector avoidPredator(int i){
 float vecDegree = atan2(vel[i].y, vel[i].x)*180/PI;
 float predatorDegree = atan2(vel[numObj].y, vel[numObj].x)*180/PI;

  if(vecDegree >= predatorDegree){
   return new PVector(vel[numObj].y, -vel[numObj].x);
 } else {
   return new PVector(-vel[numObj].y, vel[numObj].x);
 }
}





void update(int i) {
 vel[i].add(acc[i]);
 if(i == numObj){
   vel[i].limit(predatorMaxSpeed);
 } else {
   vel[i].limit(maxSpeed);
 }
 pos[i].add(vel[i]);
 acc[i].mult(0); // reset acceleration
}
```

```
void display(int i){
 float extra = 1;
 if(i == numObj){ // if its predator
   extra = 2;
 };

 theta[i] = vel[i].heading2D() + PI/2;
  if(i == 0){
   // print("\n#... theta: " + theta[i]);
 }
 if(i==0){
   fill(255, 100, 50);
 } else if(i == numObj){
   fill(255, 0, 0);
 } else {
   fill(240, 240, 240);
 }
 strokeWeight(1);
 stroke(0);
 pushMatrix();
 translate(pos[i].x, pos[i].y);
 rotate(theta[i]);
 beginShape();
 vertex(r+extra, r*2+extra);
 vertex(-r-extra, r*2+extra);
 vertex(0, -r*2*extra);
 endShape(CLOSE);
 popMatrix();
}

void checkEdges(int i){
 if(pos[i].x > width){
   pos[i].x = pos[i].x = 0;
 } else if (pos[i].x < 0) {
   pos[i].x = pos[i].x = width;
 }

 if(pos[i].y > height){
   pos[i].y = pos[i].y = 0;
 } else if (pos[i].y < 0) {
  pos[i].y = pos[i].y = height;
 }



}
```