ななちゃんのIT教室

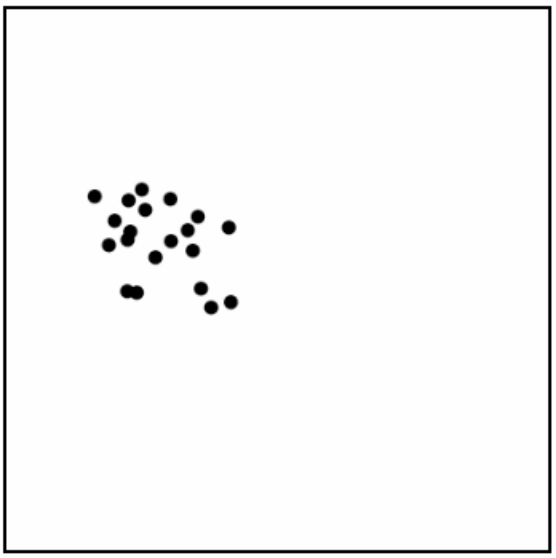
アニメーションプログラムを作ろうの巻

by nara.yasuhiro@gmail.com

プログラミングをまったく知らなかったななちゃんが
アニメーションプログラムを作れるようになるまでのお話

第 1.2 版　2017 年 5 月 7 日

sample



もくじ

第1回：ハローワールド

なな：これって、朝日新聞の「ののちゃんのDO科学」のパクリ？

先生：パロディって言ってちょうだい。家政婦のミタ（「家政婦は見た」のパロディ）、クレヨンしんちゃんのダズニーランド（「ディズニーランド」のパロディ）みたいなものよ。

フリー素材
http://freeillustration.net

なな：文部科学省が「2020年からの小学校での『プログラミング教育の必修化』を検討中」と聞いたけど、プログラミングをまったく知らなくて、とても不安なの。

先生：小学生に教えるくらいだから、とっても簡単よ。ななちゃんだったら、どんなプログラムを作ってみたい？

なな：ウェブページで見かける、アニメーションを作ってみたいな。どうやって作れば良いの？「鳥さん、動いてね」と書くの？

先生：人間がしゃべることばは不正確になりやすいし、コンピュータが理解するのは難しいので、**プログラミング言語**を使って表現するの。その代表選手が「JavaScript」という言語なの。

フリー素材
http://freeillustration.net

なな：ふうん。どうやってコンピュータに伝えるの？

先生：エディタ（Windows　メモ帳、Macintosh　テキストエディット、emacs、mi、Atomなど）を使い、右の内容のファイルを作ってみてね。ファイルの名前は「sample1.html」にしてね。

```
<script>
alert("Hello, world.");
</script>
```

なな：はい。できました。

先生：それでは、作ったファイルのアイコンをダブルクリックしてみてね。

なな：こんな画面になったわ！

先生：こういう画面を「アラート画面」というの。「OK」をクリックして、画面を閉じてね。これで、簡単だけど、JavaScript　のプログラムを実行したことになるのよ。今後、日本語の文字を表示したり、いろいろなブラウザで確実に表示させたり、普通のウェブページと組み合わせるには、次のような内容にしたほうが良いのよ。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>sample</title>
  </head>
  <body>
    <script>
alert("Hello, world.");
    </script>
  </body>
</html>
```

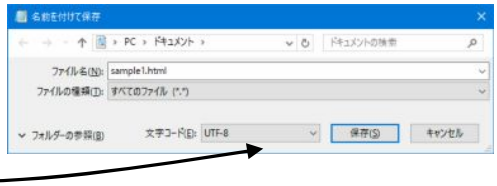
なな：ずいぶん長くなってしまうのね。毎回、こんなにたくさん入力するのは面倒ね。

先生：こういう内容のファイルを作っておいて、新しいプログラムを作るときには、このファイルをコピーして、<script>と</script>の間だけ書き換えるようにするといいのよ。どこか、1文字だけ書き間違えても正しく動作しなくなる可能性があるからね。

なな：コピーか！　はい、わかりました。ところで、「Hello, world.」って何？　「世界よこんにちは」？

先生：ひ・み・つ。気になったら、インターネット検索で調べてみてね。

JavaScript プログラムのファイルは「UTF-8」という文字コードで保存してください。



第2回：アニメーション用の画面（キャンバス）

先生：前回は、アラート画面に表示したけど、
アニメーションを表示するには不便なので、
今回は、ウェブ画面に直接表示を行う方法
を説明しましょう。右のような内容の
「sample2.html」という名前のファイルを
作って、アイコンをクリックしてみてね。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>sample</title>
  </head>
  <body>
    <h1>sample</h1>
    <script>
var canvas = document.createElement('canvas');
document.body.appendChild(canvas);
canvas.width  = 400;
canvas.height = 400;
canvas.style.border = "solid 2px";
var ctx = canvas.getContext('2d');
ctx.font = "24px sans-serif";

ctx.fillText("Hello, world.",100,100);
    </script>
  </body>
</html>
```

sample

Hello, world.

なな：右のような画面になったわ。でも、とても長いプログラム
ね。めげてしまいそう。

先生：「ctx.fillText("Hello, world.",100,100);」以外の行は
これから、毎回、同じ形で変化しない、「おまじない」の
ようなものだから、気にしないで、毎回コピーしてくだ
さい。アニメ表示枠が、縦横 400 ドットで、文字の大きさ
は縦横 24 ドットというような意味なの。

大切なのは「ctx.fillText("Hello, world.",100,100);」だけ
で、上から 100 ドット、左から 100 ドットの位置に
「Hello, world.」と表示してね、という意味なの。

それじゃ、上から 130 ドット、160 ドットの
位置にも「Hello, world.」って表示してみてね。

なな：こんな感じ？

先生：正解！

```
 :
ctx.fillText("Hello, world.",100,100);
ctx.fillText("Hello, world.",100,130);
ctx.fillText("Hello, world.",100,160);
 :
```

sample

Hello, world.
Hello, world.
Hello, world.

第3回：くりかえし

先生：前回の、「Hello, world」を10行とか、20行とか、並べるにはどうすれば良いかしら？

なな：同じような行を10行とか、20行とか並べれば良いのでは？

先生：それじゃあプログラムが長くなって大変よね。
それでは、下のようなプログラムを実行してみてね。

なな：すごい！　たった 3 行のプログラムで、10 行も「Hello, world.」。

先生：y を、100 からはじめて、30 ずつ増やすことを、370 を越えない間　繰り返してね　という意味なの。こういうのを、「繰り返し文」とか「for 文」とかいうのよ。

```
 :
for (var y=100; y<=370; y=y+30) {
     ctx.fillText("Hello, world.",100,y);
}
 :
```

sample

Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.

第4回:ボールのアニメ

先生: 今回から、いよいよ、アニメに挑戦！ 「Hello, world.」を「●」に変えれば、ボールみたいに見えるわね。これをアニメーション的に動かしましょう。ひとつの絵を描くプログラムをひとまとめにして、それを、等間隔、たとえば、1/100 秒ごに呼び出すの。

なな: ひとまとめ？

先生: 「function 名前( ) { … } 」のような形でまとめるのよ。

なな: 名前をつけるの？

先生: かたまりがいくつもあったら、使うときに名前が必要でしょう。「そこの君」「あっちのあなた」じゃ不便でしょ。

なな: 「setInterval(step,10);」 は？

先生: 「step」 というプログラムのかたまりを、10 ミリ秒ごとに呼び出してね という意味。

なな: function step には ( ) がついているけど、ここの step には ( ) がつかないの？

先生: するどいわね。でも、今は ひ・み・つ。

なな: 「x = x + vx;」 は？

先生: 「x + vx」の足し算をして、その結果で、x を書き換えるということ。呼び出されるたびに、x は、30→31→32→33 と、増えていくということなの。

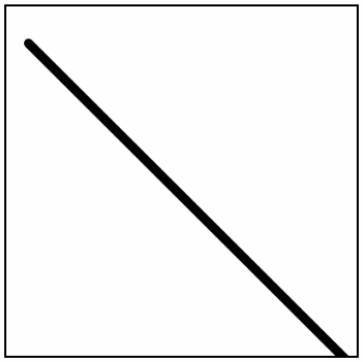
なな: だから、ボールの位置が変わってゆくのね！

```
var x = 30;
var y = 50;
var vx = 1;
var vy = 1;

setInterval(step,10);

function step( ) {
    ctx.fillText("●", x, y);
    x = x + vx;
    y = y + vy;
}
```

sample



なな: あれ？ ボールが移動するんじゃなくて、びよーん と線が出てくる！

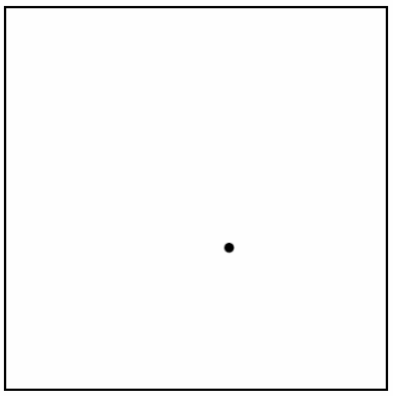
先生: 毎回、画面をクリアしてからボールを描く必要があるの。それが「ctx.clearRect(0,0,400,400);」。

```
var x = 30;
var y = 50;
var vx = 1;
var vy = 1;

setInterval(step,10);

function step() {
    ctx.clearRect(0,0,400,400);
    ctx.fillText("●",x-12,y);
    x = x + vx;
    y = y + vy;
}
```

sample



なな:動いた！ あれ？ でも、ボールが画面の右下に行ったら 消えてしまったわ！

---

setInterval は、小文字の set 大文字の I(アイ) 小文字の nterval

第5回：折り返し

なな：　前回のプログラムだと、ボイドが動いたのは良いけど、画面の右下を飛び出して消えてしまったわ。壁にぶつかったらはねかえるようにできないかしら？

先生：　壁にぶつかるというのは、x　や　y　がいくつになっているかを調べれば分かるわね。はねかえるというのは、vx　や　vy　の速度をマイナスにすれば実現できるわ。下のようなプログラムになるわ。

なな：　「400」や「0」はなんとなくわかるけど、「388」や「12」は？

先生：　文字、ここでは「●」だけど、そのサイズが縦横　12　ドットなので、その大きさを考慮しているの。

```
var x = 30;
var y = 50;
var vx = 1;
var vy = 1;

setInterval(step,10);

function step() {
    ctx.clearRect(0,0,400,400);
    ctx.fillText("●",x,y);
    if ((x > 388) && (vx > 0)) vx = -vx;
    if ((x < 0)   && (vx < 0)) vx = -vx;
    if ((y > 400) && (vy > 0)) vy = -vy;
    if ((y < 12)  && (vy < 0)) vy = -vy;
    x = x + vx;
    y = y + vy;
}
```

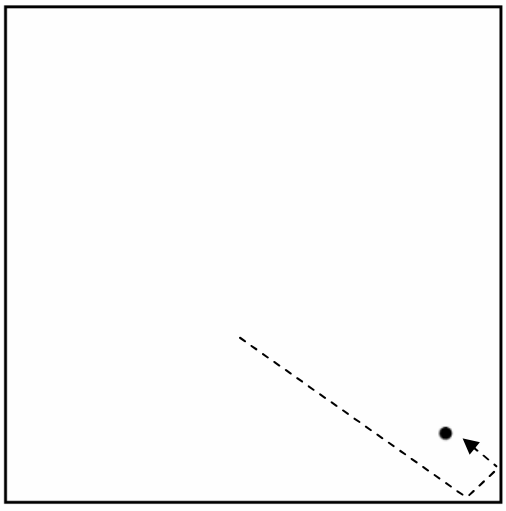
右の壁にぶつかったら、横方向の速度を反転

左

下

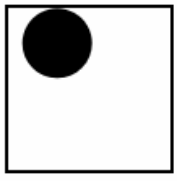
上

sample



「ctx.font = "24px sans-serif";」（24 ドットのサイズでひげ飾り無しフォント）を指定しているのに縦横 12 ドット？現実問題としてそうなります。

```
canvas.width  = 100;
canvas.height = 100;
:
ctx.font = "100px sans-serif";
ctx.fillText("●",0,50);
```



100px なのに縦横 50 ドット弱。
隣の文字とくっつかないように、さらに左と下に隙間がある。

第6回:ボールを複数に

先生: ボイドを 2 羽に増やしてみましょうね。

なな: var x1=30; var y1=50; var vx1=1; var vy=1;
var x2=30; var y2=70; var vx2=1; var vy2=1;
みたくするの？

先生: それでもできるけど、ボイドが 20 羽とかになったら、やってらんないわね。 そういう時には、「配列」や「オブジェクト」という、データのあつまりを使うと便利なの。
「var boids = [{x:30,y:50,vx:1,vy:1}, {x:30,y:70,vx:1,vy:1}];」
という形でデータのあつまりを作ると、「boids[0].x」で、ボイド 0 号の x 、「boids[1].vy」で、ボイド 1 号の vx を使うことができるの。

なな: ちょっと難しいけど、プログラムがとってもコンパクトになるのね。前回のプログラムは 17行だったけど、今回のプログラムも 17行。このテクニックを使わないと、倍近くの行数になってしまうのにね。

ボイド 0 号のデータ

ボイド 1 号のデータ

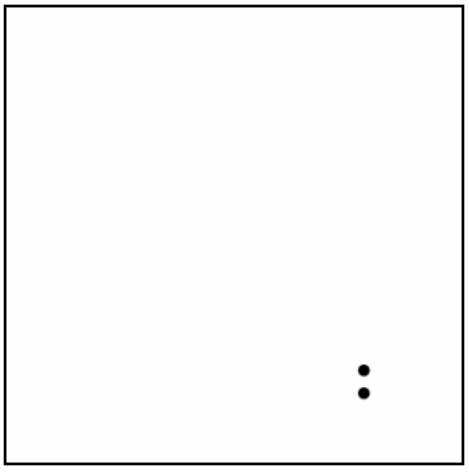
```
var boids = [ {x:30,y:50,vx:1,vy:1}, {x:30,y:70,vx:1,vy:1} ];

setInterval(step,10);

function step() {
    ctx.clearRect(0,0,400,400);
    for (var i in boids) {
        var boid = boids[i];
        ctx.fillText("●",boid.x,boid.y);
        if ((boid.x > 388) && (boid.vx > 0)) boid.vx = -boid.vx;
        if ((boid.x < 0)   && (boid.vx < 0)) boid.vx = -boid.vx;
        if ((boid.y > 400) && (boid.vy > 0)) boid.vy = -boid.vy;
        if ((boid.y < 12)  && (boid.vy < 0)) boid.vy = -boid.vy;
        boid.x = boid.x + boid.vx;
        boid.y = boid.y + boid.vy;
    }
}
```

for (var i=0; i<boids.length; i++) {
の 省略記法！

sample



第7回：勝手にシンドバッド（古い！）

先生：ボイドをたくさん（たとえば20個）用意する場合、ひとつひとつに位置と速度を設定するのは手間がかかるわね。
そういう場合、コンピュータに適当に設定させると楽になるわ。

なな：「適当にやれ」なんていう命令があるの？

先生：Math.random() という命令は、0 から 1 の間の数、0.3 とか、0.81001 とかの数を作ってくれるの。さいころをふるような感じで。こういうのを「ランダムな数」というの。厳密には、1 のちょっと手前の数まで。

なな：0 から 1 だけだと不便ね。0 から、キャンバスのサイズの 400 までを作りたいのに。

先生：そういう場合は、Math.random()*400 と書けば良いのよ。

なな：なーるほど！

空っぽの配列を用意

ボイド 0 号から 19 号までを作る

x と y は 0 から 399.99… の間の乱数に

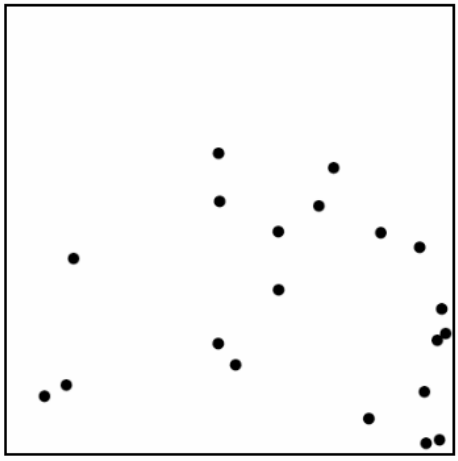
```
var boids = [ ];
for (var i=0; i<20; i++) {
    boids[i] = { x:Math.random()*400, y:Math.random()*400,
                 vx:Math.random()*4-2, vy:Math.random()*4-2};
}

setInterval(step,10);

function step() {
    ctx.clearRect(0,0,400,400);
    for (var i in boids) {
        var boid = boids[i];
        ctx.fillText("●",boid.x,boid.y);
        if ((boid.x >= 388) && (boid.vx > 0)) boid.vx = -boid.vx;
        if ((boid.x < 0)       && (boid.vx < 0)) boid.vx = -boid.vx;
        if ((boid.y >= 400) && (boid.vy > 0)) boid.vy = -boid.vy;
        if ((boid.y < 12)     && (boid.vy < 0)) boid.vy = -boid.vy;
        boid.x = boid.x + boid.vx;
        boid.y = boid.y + boid.vy;
    }
}
```

x と y は -2 から +1.99.99… の間の乱数に

sample



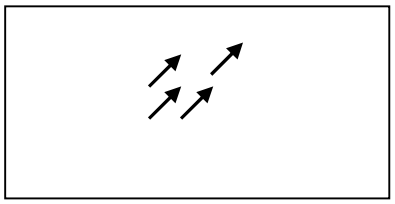
ボイド（Boids）

たくさんの鳥たちが群れをなして飛んでいるのを見たことがあるかも知れない。かつて、生態学者たちはこのような群れにはリーダーがいて、それに他の鳥たちが追従しているのではないかと考えていた。しかし、動画撮影して、細かく解析してみると、先頭にいる鳥が、時々刻々と変化していることが分かってきた。そんな中、1987年にアメリカのアニメーション・プログラマ、クレイグ・レイノルズ（Craig Raynolds）は、生態学とはまったく別の興味で、CGで、リアルな鳥軍団を再現しようと、アルゴリズムを考案した。たった 3 つのルールを規定するだけで鳥の群れをシミュレーションできるというもの。Boid という名の由来は、鳥もどきという意味の言葉 birdoid（bird-oid、鳥 (bird) に似たもの (-oid)、バードイド）が短くなったもの。実際に、『バットマン・リターンズ』の ペンギンの集団のシーン、『ライオンキング』の バッファローの集団が移動するシーンなどは、このような方法で作られたとのこと。今では、実際に、鳥軍団にはリーダは存在せず、簡単なルールで成り立っていると考えられている。多細胞生物でも、各細胞が似たようなルールで同調しているのではないかと考える人もいる。

Boids の3つのルール：

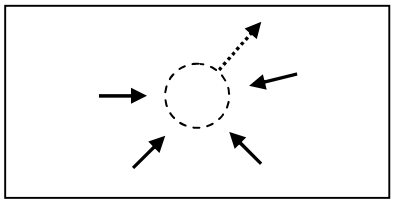
(1) Alingment（整列）、整列（Velocity Matching）

近くの鳥たちと飛ぶスピードや方向を合わせようとするルール。同じ方向にあまり距離を空けないように飛ぶようにする。ある一定の距離より遠ざかりすぎてしまったら前を飛んでいるボイドはスピードを遅くし、後ろを飛んでいるボイドはスピードを速くするようにすることで実現できる。



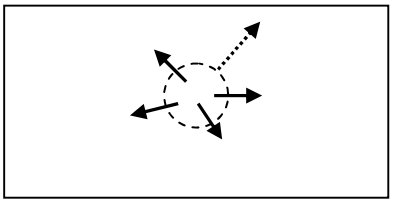
(2) Cohesion（結合）、接近（Flock Centering）

鳥たちが多くいる方へ向かって飛ぶルール。鳥が多くいる方向というのは、群れの中心（重心）方向ということ。ボイドに群れの中心の方向へ飛んでいくことを指示。群れの中心は全ボイドの位置（座標）の平均として求める。



(3) Separation（引き離し）、衝突回避（Collision Avoidance）

近くの鳥に近づきすぎたらぶつからないように離れるルール。もし、ボイド同士が近づきすぎてしまったら、前を飛んでいるボイドはスピードを速くし、後ろを飛んでいるボイドはスピードを遅くするようにする。



参考資料： http://coderecipe.jp/recipe/gRrj53OPQF/
http://members.jcom.home.ne.jp/ibot/boid.html

第8回：ボイドにルールを埋め込む　ボイドルール1（整列）

```
var boids = [];
for (var i=0; i<20; i++) {
    boids[i] = { x:Math.random()*400, y:Math.random()*400,
                 vx:Math.random()*4-2, vy:Math.random()*4-2};
}

setInterval(step,50);

function step() {
    ctx.clearRect(0,0,400,400);
    for (var i in boids) {
        var boid = boids[i];
        rule1(i);
        ctx.fillText("●",boid.x,boid.y);
        if ((boid.x >= 388) && (boid.vx > 0)) boid.vx = -boid.vx;
        if ((boid.x < 0)       && (boid.vx < 0)) boid.vx = -boid.vx;
        if ((boid.y >= 400) && (boid.vy > 0)) boid.vy = -boid.vy;
        if ((boid.y < 12)     && (boid.vy < 0)) boid.vy = -boid.vy;
        boid.x = boid.x + boid.vx;
        boid.y = boid.y + boid.vy;
    }
}
// ルール 1: ボイドは近くのボイドの平均速度に合わせようとする
function rule1(index) {
    var px = 0, py = 0;
    for (var i in boids) {
        if (i != index) {
            px = px + boids[i].vx;
            py = py + boids[i].vy;
    }    }
    px = px / (boids.length - 1);
    py = py / (boids.length - 1);
    boids[index].vx = boids[index].vx + (px-boids[index].vx) / 8;
    boids[index].vy = boids[index].vy + (py-boids[index].vy) / 8;
}
```

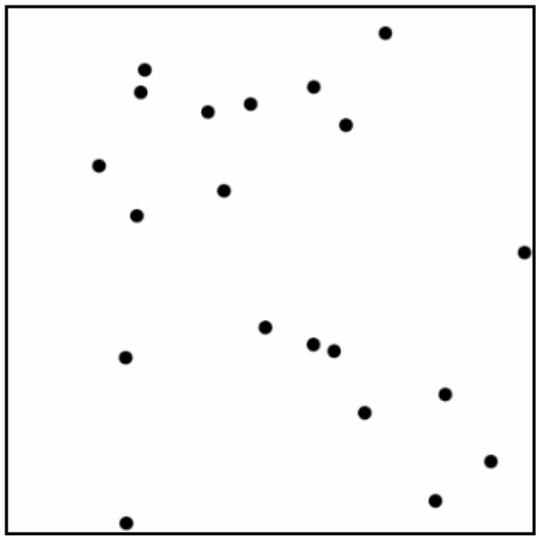
自分以外の全ボイドの速度の平均値を求める

その平均速度に近づくように速度調節

このルールだと、
ランダムの平均はゼロなので
停滞してしまう。

この段階では　これで良い

sample

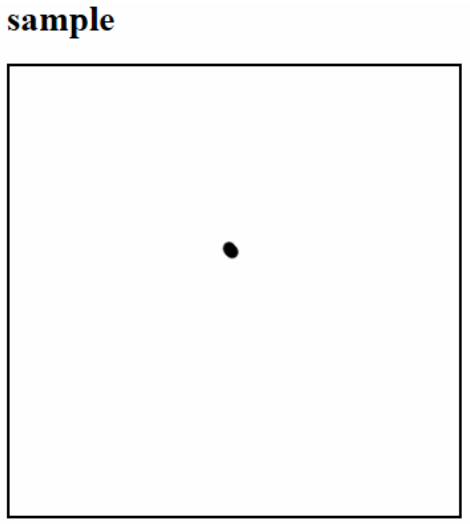


第9回:ボイドにルールを埋め込むボイドルール2(結合)

```
:
function step() {
    ctx.clearRect(0,0,400,400);
    for (var i in boids) {
        var boid = boids[i];
        rule1(i); rule2(i);
        ctx.fillText("●",boid.x-12,boid.y);
        if ((boid.x >= 388) && (boid.vx > 0)) boid.vx = -boid.vx;
        if ((boid.x < 0)    && (boid.vx < 0)) boid.vx = -boid.vx;
        if ((boid.y >= 400) && (boid.vy > 0)) boid.vy = -boid.vy;
        if ((boid.y < 12)   && (boid.vy < 0)) boid.vy = -boid.vy;
        boid.x = boid.x + boid.vx;
        boid.y = boid.y + boid.vy;
    }
}
// ルール 1: ボイドは近くのボイドの平均速度に合わせようとする
:
// ルール 2: ボイドは近くに存在する群れの中心に向かおうとする
function rule2(index) {
    var cx = 0, cy = 0;
    for (var i in boids) {
        if (i != index) {
            cx = cx + boids[i].x;
            cy = cy + boids[i].y;
    }   }
    cx = cx / (boids.length - 1);
    cy = cy / (boids.length - 1);
    boids[index].vx = boids[index].vx + (cx-boids[index].x) / 100;
    boids[index].vy = boids[index].vy + (cy-boids[index].y) / 100;
}
```

自分以外全員の中心位置を計算

その中心に向かうように速度調節



ひとつにまとまろうとするが、くっついてしまう。

この段階では これで良い。

第10回：ボイドにルールを埋め込むボイドルール3（引き離し）

```
var boids = [];
for (var i=0; i<20; i++) {
    boids[i] = { x:Math.random()*400, y:Math.random()*400,
                 vx:Math.random()*4-2, vy:Math.random()*4-2};
}

setInterval(step,50);

function step() {
    ctx.clearRect(0,0,400,400);
    for (var i in boids) {
        var boid = boids[i];
        rule1(i); rule2(i); rule3(i);
        ctx.fillText("●",boid.x-12,boid.y);
        if ((boid.x >= 388) && (boid.vx > 0)) boid.vx = -boid.vx;
        if ((boid.x < 0)       && (boid.vx < 0)) boid.vx = -boid.vx;
        if ((boid.y >= 400) && (boid.vy > 0)) boid.vy = -boid.vy;
        if ((boid.y < 12)     && (boid.vy < 0)) boid.vy = -boid.vy;
        boid.x = boid.x + boid.vx;
        boid.y = boid.y + boid.vy;
    }
}
// ルール 1: ボイドは近くのボイドの平均速度に合わせようとする
:
// ルール 2: ボイドは近くに存在する群れの中心に向かおうとする
:
// ルール 3: ボイドは隣のボイドと少しだけ距離をとろうとする
function rule3(index) {
    for (var i in boids) {
        if (i != index) {
            var d = dist(boids[i], boids[index]);
            if (d < 7) {
                boids[index].vx = boids[index].vx - (boids[i].x - boids[index].x);
                boids[index].vy = boids[index].vy - (boids[i].y - boids[index].y);
}   }   }   }

function dist(b1, b2) {
    return Math.sqrt(Math.pow((b1.x-b2.x),2) + Math.pow((b1.y-b2.y),2));
}
```

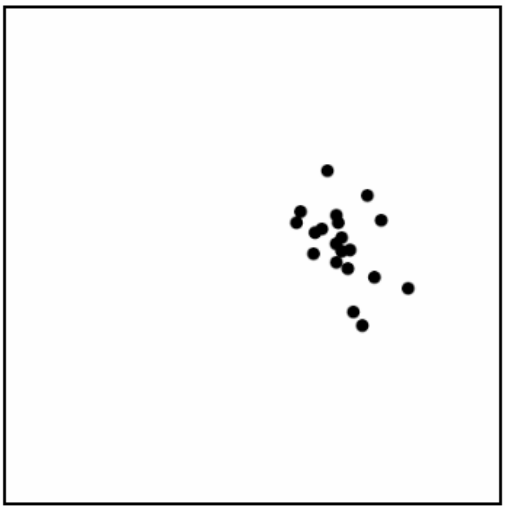
距離が近ければ離れる方向に速度調節

距離計算

sample

集まろうとするが
衝突は避ける。

完成！！！！！



まとめ： 以上のプログラムをまとめ、少しコンパクトに書き改めると、下記のようになります。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>sample</title>
    <style>
        canvas { border:solid 2px; }
    </style>
  </head>
  <body>
    <h1>boids</h1>
    <script>
var canvas = document.createElement('canvas');
document.body.appendChild(canvas);
canvas.width  = canvas.height = 400;
var ctx = canvas.getContext('2d');
ctx.font = "24px sans-serif";
var boids = [];
for (var i=0; i<20; i++)
    boids[i] = { x:Math.random()*400, y:Math.random()*400,
                 vx:Math.random()*4-2, vy:Math.random()*4-2 };
setInterval(step,50);

function step() {
    ctx.clearRect(0,0,400,400);
    for (var i in boids) {
        var boid = boids[i];
        rule1(boid); rule2(boid); rule3(boid);
        ctx.fillText("●",boid.x-12,boid.y);
        if ((boid.x >= 388) && (boid.vx > 0)) boid.vx = -boid.vx;
        if ((boid.x < 0)    && (boid.vx < 0)) boid.vx = -boid.vx;
        if ((boid.y >= 400) && (boid.vy > 0)) boid.vy = -boid.vy;
        if ((boid.y < 12)   && (boid.vy < 0)) boid.vy = -boid.vy;
        boid.x = boid.x + boid.vx;
        boid.y = boid.y + boid.vy;
}   }
function rule1(b) { //  ルール 1:  ボイドは近くのボイドの平均速度に合わせようとする
     var px = 0, py = 0, bi;
     for (var i in boids)
          if ((bi = boids[i]) != b) { px += bi.vx; py += bi.vy; }
     px /= (boids.length - 1);
     py /= (boids.length - 1);
     b.vx += (px-b.vx) / 8;
     b.vy += (py-b.vy) / 8;
}
function rule2(b) { //  ルール 2:  ボイドは近くに存在する群れの中心に向かおうとする
    var cx = 0, cy = 0, bi;
    for (var i in boids) if ((bi = boids[i]) != b) { cx += bi.x; cy += bi.y; }
    cx /= (boids.length - 1);
    cy /= (boids.length - 1);
    b.vx += (cx-b.x) / 100;
    b.vy += (cy-b.y) / 100;
}
function rule3(b) { //  ルール 3:  ボイドは隣のボイドと少しだけ距離をとろうとする
    for (var i in boids) {
        var bi;
        if ((bi = boids[i]) != b)
            if (dist(bi, b) < 7) { b.vx -= (bi.x - b.x); b.vy -= (bi.y - b.y); }
}   }
function dist(b1, b2) {
     return Math.sqrt(Math.pow((b1.x-b2.x),2) + Math.pow((b1.y-b2.y),2)); }
    </script>
  </body>
</html>
```

いろいろなプロフェッショナル・テクニックを使っています。研究してみましょう。

for 文や、if 文の中身が 1 行（1 実行文） の場合は、「{」、「}」 を省略できます

i ではなく、boid を送っています

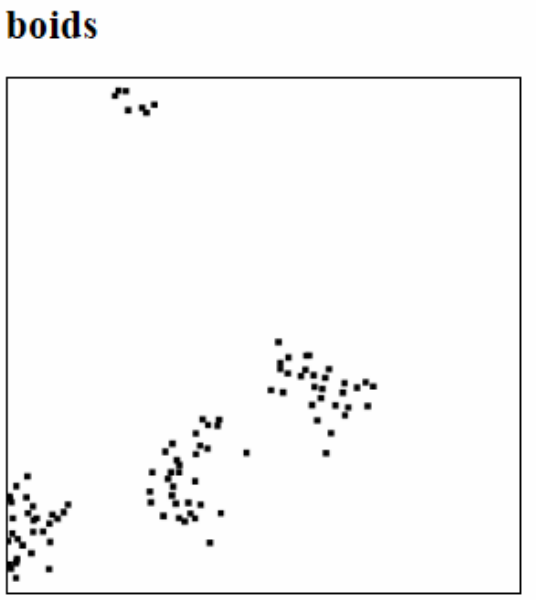
bi = boids[i];
if (bi != b) {… }
の短縮系

(bi = boids[i]) は、
bi に boids[i] の値を代入するとともに、bi の値をもちます

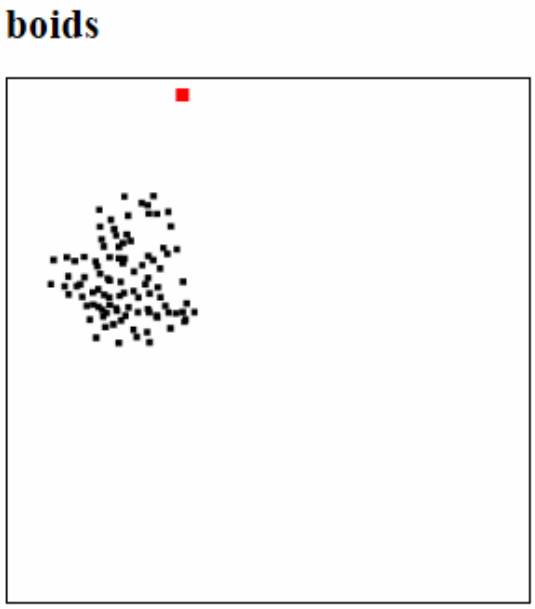
研究課題

ボイドのルールを追加、改良してみよう。

近いボイドだけで
サブグループ

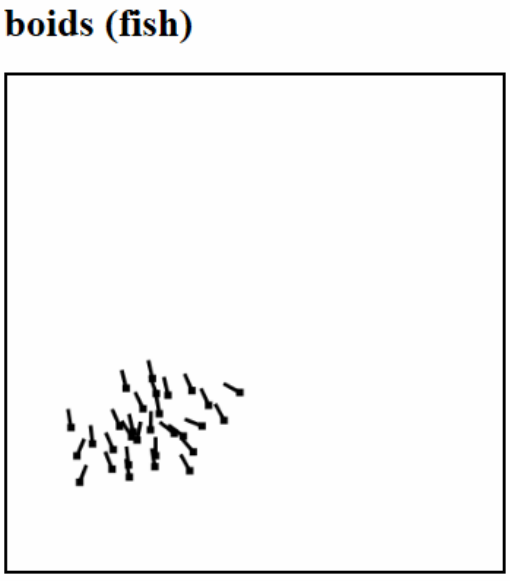


逃げるボイドと
それを追跡するボイド群



鳥ではなくて
魚に

ヒント：vx、vy　から
Math.atan2　で角度を求める



移動しない植物、それを食べる草食動物、それを捕食する肉食動物