

Part2 アニメーション

Part1 では 2D の静止画を描くプログラムを作りました。

ここからは、動くものを作っていきます。単に動くだけでなく、マウスに反応したり、キーボードで操作できるようにします。

この手法はゲームプログラミングにも応用できます。

1. アニメーションの仕組み
2. マウス操作
3. if 文
4. 配列
5. 花火
6. 命令の作り方
7. translate、rotate、scale、ついでにキー操作

1. アニメーションの仕組み

テレビ、映画、そしてパソコンの画面は、いずれも絵が動いてるように見えます。しかし、実際に何かが動いているわけではなくて、人間が分からないくらい**速い速度で次々と静止画像を表示して、動いているように見せているのです。**

さっそく、アニメーションプログラムを作りましょう。ファイル名はPart2 から始めましょう。

```
float x; //ここに○の画面左からの距離を記録することにする
float y; //上からの距離
float vx; //この変数は何の意味があるだろうか?
float vy;

void setup(){
  size(500, 500);
  colorMode(HSB, 100, 100, 100);
  fill(0, 0, 100); //白
  rect(-1, -1, 501, 501); //画面全体を白い四角形で覆う
  x = 0.0;
  y = 0.0;
  vx = 1.0;
  vy = 0.5;
}

void draw(){
  ellipse(x, y, 10, 10); //(x, y)を中心に円を描く
  x += vx; //xをvxだけ増やす
  y += vy; //yをvyだけ増やす (つまり次に円を描く場所を変える)
}
```

このプログラムは、setup、draw、その外側、の3つの部分からなります。それぞれ、

setup : 最初に一回だけ実行される

draw : 一秒になんでも実行される

外側 : 変数を作るのはここ

という役割があります。

setup の中カッコの中身を見ると、画面の大きさの指定、色の指定、背景の指定などの、最初に一回だけ実行すればいい命令が書かれています。

draw の中身は、円の描画、そして、x, y の値を更新する命令が書かれています。

このプログラムを実行すると、次のような、移動する円とその残像が見えるはずです。





なぜ残像が見えるのかというと、draw という命令は、**今まで書かれている絵を消さないで、その上に重ねて新しい絵を描く**からです。

- ・ ○の進む方向や速度を変えてみよう
- ・ どうすればこの残像を消せるか考えて、残像が残らないようにプログラムを改良してみよう。

・ 問題

次の空欄を埋めて、中心から円が広がっていくようなアニメーションを表示してみよう。

```
float d;  
  
void setup(){  
    
}  
  
void draw(){  
  noFill();  
  ellipse(width/2, height/2, d, d);  
    
}
```

setup の中身は最初に一回だけ。draw の中身は一秒間になんども実行されるということを忘れないように。

なぜ float d; を外側に書いているのかというと、**中カッコ { }**の中で作った変数は、**その中カッコの中でしか使えない**という決まりがあるからです。例えば setup の中に float d; を書いてしまうと、draw のなかで d が使えなくなってしまうのです。

これは for 文などでも同じで、for (int i=0; i<10; i++) { } と書いた時、変数 i はこの中カッコの中でのみ使うことができます。

・ draw の最後に、「**delay(500);**」と書いてください。これは 500 ミリ秒=0.5 秒待てという命令で、動画は一秒間に 2 回のペースで動くようになります。

・ ellipse 命令の前の **noFill();** を消してください。何が起こりますか？

2. マウス操作

マウスに反応するプログラムを書きましょう。
マウス操作は、「位置」によるものと、「クリック」によるものがあります。

マウスが今画面上のどこにいるかを知るのは簡単で、width や height と同じように、

mouseX : マウスの画面左端からの距離
mouseY : 上

という名前の変数を使うだけです。

クリックの処理については、**mousePressed** という名前の命令を作ります。mousePressed 命令の中カッコの中に書いたものは、**マウスがクリックされたときに呼び出される**ことになります。

```
void setup(){ //これは最初に一回だけ呼ばれる
  size(500, 500);
  colorMode(HSB, 100, 100, 100);
  background(0, 0, 100); //背景：白
}

void draw(){ //これは一秒間に何回も呼ばれる
  rect(mouseX, mouseY, 10, 10); //マウスの位置に四角形を描く
}

void mousePressed(){ //マウスがクリックされたら
  ellipse(mouseX, mouseY, 50, 50); //マウスの位置に円を書く
}
```

・問題

- (1) draw の中で描かれる正方形を半透明で、りんかくのない黒い正方形にしてみましょう。
- (2) このプログラムを改良して、クリックされたら、今までに描いたものが全て消えるようにしましょう。
- (3) 2 ページ前に作ったプログラムを参考にして、
○はまっすぐ移動する
マウスがクリックされるたびに、○がマウスの位置に来て、進む方向はランダムに決定される
というプログラムを作ってください。

3. If 文

マウスがクリックされた時、マウスの位置によってすることを変えたい、のように、状況によって処理を変えるためには if 文というものを 사용합니다。for 文よりもずっと簡単ですのでご安心を。

```
if(条件 A) {  
    条件 A が正しい時実行される場所  
} else if(条件 B) {  
    A が正しくなくて、B が正しい時に実行される場所  
} else {  
    A も B も正しくないときに実行される場所  
}
```

if、else if、else を使って、上のように分岐を作ることができる。なお、else if はいくつ続いてもよいし、if の {} の中にまた if 文が入ってもよい。

「条件 A」の部分に入るのは、for 文の二番目に入るのと同じように、「 $X \leq Y$ (X は Y 以下)」や、「 $X == Y$ (X と Y は等しい)」のような、正しいか間違ってるか判断できる式です。このような記号は、Part1 の計算のところにまとめてあります。

```
void setup(){  
    size(500, 100);  
    colorMode(HSB, 100, 100, 100);  
    fill(0, 0, 100);  
    rect(-1, -1, 501, 501);  
}  
  
void draw(){  
}  
  
void mousePressed(){  
    //マウスがクリックされた時  
    if(mouseX < 200){  
        //マウスの左端からの距離が200より小さかったら  
        fill(0, 100, 100);  
        //塗りつぶしを赤に  
    } else if(mouseX < 300){  
        //そうでなくて、300より小さかったら  
        fill(33, 100, 100);  
        //緑に  
    } else {  
        //そうでもなかったら (300以上なら)  
        fill(66, 100, 100);  
        //青に  
    }  
    rect(-1, -1, 501, 501);  
    //画面全体を覆うような四角形を書く  
}
```

このプログラムは、画面の左のほうをクリックしたら赤、真ん中あたりで緑、右のほうで青に画面の色が変わるプログラムです。

・ 問題

(1) 画面の上のほうをクリックしたら、ランダムな場所にランダムな大きさの円が、下のほうをクリックしたら、ランダムな場所にランダムな大きさの正方形が描かれるプログラムを書いてください。長方形ではなく正方形にするにはどういう工夫が必要でしょうか。

(2) 2 ページ前の問題 3 で作ったプログラムを改良して、画面端に当たった○が跳ね返るようにしましょう。

ヒント（自信のある人は見ないでやってみてください）

まず、変数 vx は○が画面右方向に進む速度（draw が一回呼ばれるたびに x は vx 増えるから）、 vy は下方向に進む速度を表しています。そうすると、「跳ね返る」という処理は、左右方向の反射なら vx を、 $-vx$ にすればいいということが分かります。

「画面端に当たっている」という状態は、例えば右端なら、 $x - 5 > width$ ということです（5 は○の半径）。さらに、画面右端に当たっていても、○が左向きに進んでいるならもちろん跳ね返さなくていいので、 $vx > 0$ という条件も必要です。

この二つの条件をまとめると、

$x - 5 > width \ \&\& \ vx > 0$

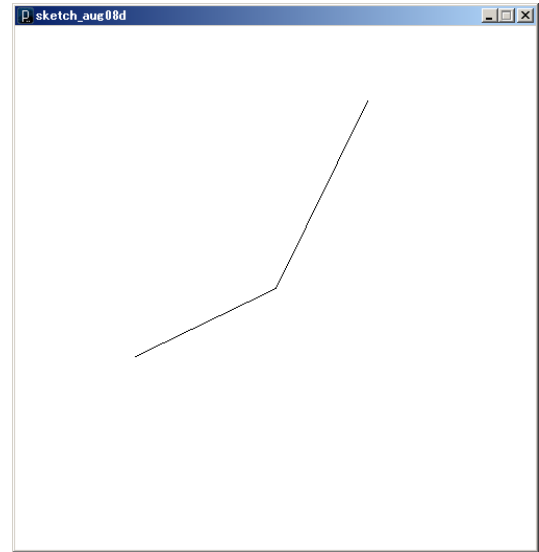
と書けます。

・アニメーション練習

(一旦飛ばして先に進んで下さい。時間が余るか、家に帰ってからやってみましょう)

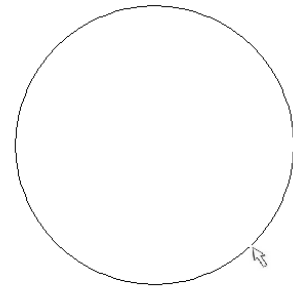
(1) 時計

長針と短針のある時計を描きましょう。
12:00 で針が重なるように、長針が短針の 12 倍の速度で動くようにしましょう。
sin、cos を使います。draw が実行されるたびに角度が進むようにするとよいでしょう。



(2) 泡

画面の中心を中心とし、マウスのカーソルに接するような円を描こう。
マウスを移動させたら、円もつられて大きくなったり小さくなったりする。
ちなみに、二点 (x_0, y_0) 、 (x_1, y_1) の距離は、
 $\text{dist}(x_0, y_0, x_1, y_1)$
命令で得ることができる。(三平方の定理から解いても良い。
ルート X は $\text{sqrt}(X)$)



(3) 泡 2

(2) のプログラムを改良して、クリックすると円の大きさがそこで固定される⇔もう一度クリックすると円がまたマウスに接するようになる、と切り替わるようにしよう。

4. 配列

さて、ここではたくさんのものを扱う方法をやります。たとえば、今まで1つだった○の数を、100個にしてみるといった話です。

100個の○を動かすには、100個の○の場所と速度を覚えておかないといけません。ということはx, y, vx, vyが100個で、あわせて400個の変数が必要です。いくらコピー、貼りつけを使っても、これは面倒くさいです。

ここで登場するのが**配列**です。大量の変数をまとめて作ることができます。

```
float[] x = new float[100];
```

と書くことで、100個の変数を作ります。変数の名前は、

```
x[0], x[1], ..., x[99]
```

となります。0から始まっていることに注意しましょう。便利なことに、iを整数が入る変数とすると、x[i]でi番目の変数を呼び出すことができます。

```
float[] x = new float[100];
float[] y = new float[100];
float[] vx = new float[100];
float[] vy = new float[100];

void setup(){
  size(500, 500);
  colorMode(HSB, 100, 100, 100);
  fill(0, 0, 100); //白
  rect(-1, -1, 501, 501); //画面全体を白い四角形で覆う

  for(int i=0; i<100; i++){ //変数iを0から99まで動かす
    x[i] = random(0, width); //それぞれの○の初期位置、速度はランダム
    y[i] = random(0, height);
    vx[i] = random(-1, 1);
    vy[i] = random(-1, 1);
  }
}

void draw(){
  fill(0, 0, 100); //白
  rect(-1, -1, 501, 501); //画面全体を白い四角形で覆って残像を消す

  for(int i=0; i<100; i++){ //変数iを0から99まで動かす
    ellipse(x[i], y[i], 10, 10); //i番目の○を描く
    x[i] += vx[i];
    y[i] += vy[i];
  }
}
```

このプログラムでは100個の○をランダムな位置から、ランダムな方向に動かします。x, y, vx, vyの4種類の配列を作って、それぞれの○の位置と速度を記録しています。for文を使って、100個の○を描いたり動かしたりする命令をまとめて記述しています。最初のページで作ったプログラムとの違いをよくよく確認してみましょう。

ここからしばらくは、このプログラムを改良していくことにします。

・問題

(1) マウスで○を追加する

最初の改良は、マウスでクリックすると、クリックしたところに新しい○を追加して、ランダムな方向に動いていくようにするというものです。4 ページ前の問題3に近いですが、新しい○を作るという点で違います。

まず、1~4 行目の配列を作っているところは変える必要はありません。5 行目に、

```
int N = 0;
```

と書いてください。ここに「今存在する○の数」を記録することにします。**最初是一个も無い**ように、Nに0を入れておきます。

mousePressed 命令を作って、そのなかに、**○を一つ追加する**命令を書きましょう。今5個の○が存在するとします。このときNの値は5です。配列xは、x[0], x[1], x[2], x[3], x[4]が今存在する5個の○のために使われていて、x[5]からx[99]までは使われていません。ということは、新しく作る○はx[5]、つまりx[N]を使えばいいことが分かります。

```
x[N] = mouseX;
```

同じように、y, vx, vy も書きましょう。

最後に、○の数が増えるので、N++;と書きましょう。これでNが1足されます。

draw 命令の中の for 文は、x[0]からx[N-1] (y, vx, vy も同様) までをみるように書き換えましょう。

setup 命令では、いままでは100個の○の初期値を決めていたのですが、今回は最初は○が0個なので、for 文まるごと消してしまいましょう。

完成したら、マウスをクリックした回数を数えながら、いくつも○を追加してみましょう。101個目の○を追加しようとするとうどうなるかな？

(2) (1)を改良して、端に来た○が跳ね返るようにしよう

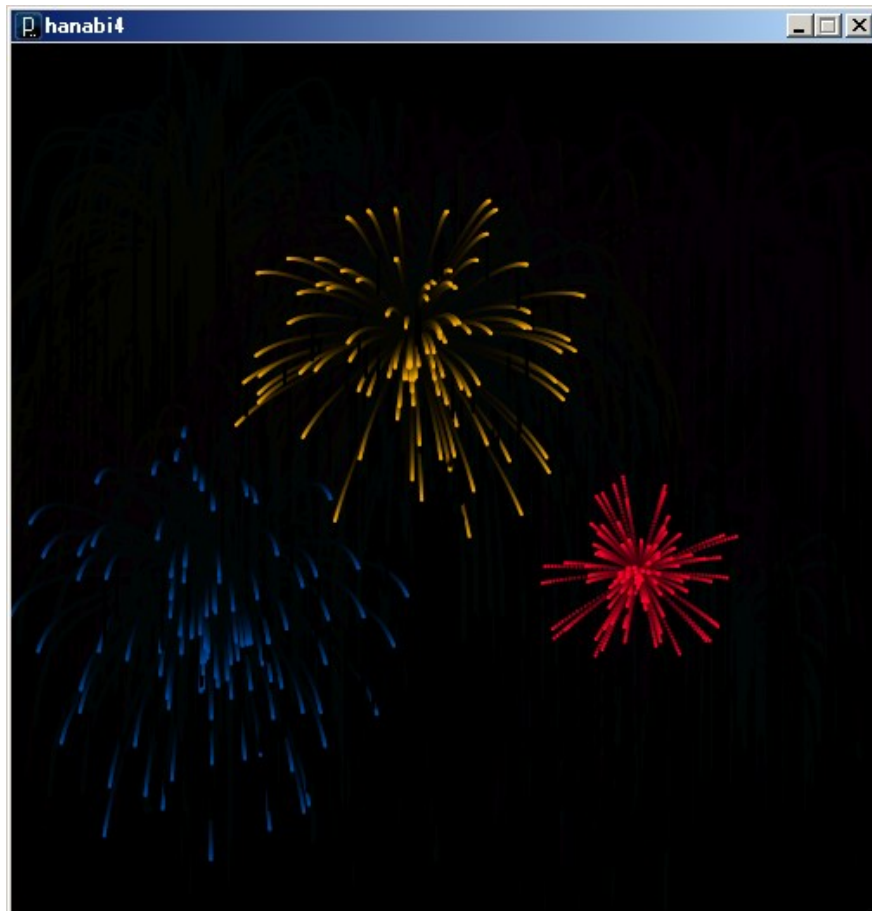
(3) (2)を改良して、○に色をつけよう

色を表す新しい配列hを作ろう。i 番目の○の色の、色相の部分をh[i]で表すことにする。つまり、i 番目の○を描く前にfill(h[i], 100, 100);と書けばよい。新しい○を追加するたびに、その○の色をランダムに決めてh[N]に記録する。

(4) 今までは残像を消すために、draw が実行されるたびに白く塗りつぶした四角形で画面全体を覆って、それまでに描かれていたものを消していた。では、○をnoStroke で描くようにして、残像を消すために白半透明の四角形を使うようにしたらどうなるだろう？

5. 花火

今まで練習してきたことを使って、花火を作ろう！



クリックしたら、その場所で花火が爆発する。花火の火の玉はそのまま周りに飛んでいき、一定時間後に消える。

方針

(1) マウスをクリックすると、その場所からランダムな方向に 100 個くらいの○が飛んでいくようにする。配列は余裕を持って、

```
float[] x = new float[10000];
```

のように大きなサイズで作っておく。

(2) 背景を黒、○の大きさを適当な大きさ、クリックごとにランダムに色を決めて、そのとき出現する○は全て同じ色になるようにする。

(3) このままでは四角形に広がっていくので、 \sin 、 \cos をうまく使って円形に広がるようにする。

(4) ○が徐々に暗くなって消えるようにする。配列 b を作って、ここに○の明るさを記録しよう。

(5) 火の玉は最初素早く広がって、すぐに減速する。毎ステップで $vx[i]$ と $vy[i]$ が小さくなるようにすると、このような挙動が再現できる。

(6) 火の玉は重力に従って落ちる。毎ステップで下方向に少し加速すると、リアルになる。

(7) 好きなように改良しよう。形を変えたり、火の玉の見え方を変えたり。

6. 命令の作り方

今までいろんな「命令」を扱ってきました。ここでは自分で命令を作る方法を学びます。よく使う命令の流れをひとつの命令にまとめたり、クラスの値を操作するために命令を作る必要があるときもあります。

命令を「引数（カッコの中の数）」と「戻り値」という観点から見てみましょう。

```
r = random(0, 100);    ... 引数は小数 2 つ（ここでは 0 と 100）、戻り値は小数
line(x0, y0, x1, y1); ... 引数は小数 4 つ、戻り値なし
noFill(); setup();     ... 引数なし、戻り値なし
```

命令を作るときは、

```
<戻り値の型> 命令の名前(引数たち){
    命令の内容
    return 戻り値;
}
```

の形に書きます。戻り値がないときは、戻り値の型は void と書きます。

命令を作る例です。

この 3 つの命令がそれぞれ何をする命令なのか考えてみてください。

```
float ookiihou(float a, float b){
    if(a > b){                引数：小数 2 つ
        return a;            戻り値：小数
    }else{
        return b;
    }
}

void sankaku(float x0, float y0, float x1, float y1, float x2, float y2){
    line(x0, y0, x1, y1);
    line(x1, y1, x2, y2);    引数：小数 6 つ
    line(x2, y2, x0, y0);    戻り値：なし (void)
}

void hello(){
    fill(random(0, 100), 100, 100);
    text("Hello!", random(0, width), random(0, height));  引数：なし
}                                                            戻り値：なし (void)
```

ところで、今までなんども書いてきた setup() や draw()、mousePressed() などは、もともと役割の決まっている命令です。これらは違う名前にはできないし、戻り値や引数を変えることもできません。

一方で、Ball クラスの run() 命令は、私が勝手につけた名前です。こういう命令については、名前を自由に決められて、引数、戻り値も好きなように決められます。

7. translate、rotate、scale、ついでにキー操作

wsad キーで上下左右移動

qe キーで回転

rf キーで拡大縮小します

translate(x, y)はその後に書かれるすべての絵を左に x、下に y ずらします。

rotate(t)は、t 回転します。

scale(s)は s 倍に拡大します。

すべて、その後全てに影響するので、そのままとやばいんですが、

pushMatrix(); と popMatrix(); にはさむと、影響をその間だけにできるのです。

```
float x, y;
float t;
float s;

void setup(){
  size(500, 500);
  colorMode(HSB, 100, 100, 100);
  x=0;
  y=0;
  t=0;
  s=1;
}

void draw(){
  background(0, 0, 100);

  pushMatrix();
  translate(x, y);
  rotate(t);
  scale(s);

  int N = 5;
  int M = 2;
  float T = 2 * PI / N;
  for(int i=0; i<N; i++){
    line(100*sin(i*T), 100*cos(i*T), 100*sin((i-M)*T), 100*cos((i-M)*T));
  }

  popMatrix();
}

void keyPressed(){
  if(key == 'a') x-=1;
  if(key == 'd') x+=1;
  if(key == 'w') y-=1;
  if(key == 's') y+=1;
  if(key == 'q') t-=0.1;
  if(key == 'e') t+=0.1;
  if(key == 'r') s+=0.3;
  if(key == 'f') s-=0.3;
}
```