

入門編テキスト 目次

Part1. グラフィックスとプログラミング入門.....	3
1. Processing を使ってみる.....	4
2. 図形を描く.....	6
3. 色をつける.....	8
4. 練習①.....	10
5. 計算させよう.....	12
6. 変数.....	14
7. 数学関数.....	17
Part2. 制御文.....	19
1. for 文の基本.....	20
2. for 文の練習.....	22
3. 乱数.....	24
4. if 文.....	26
5. 二重ループ.....	28
6. 総合練習.....	32
Part3. アニメーションと操作.....	35
1. アニメーションの仕組み.....	36
2. マウス操作1：マウスの位置を知る.....	38
3. マウス操作2：クリック.....	40
4. 変数で図形を動かす.....	42
5. if 文による制御.....	44
6. 練習.....	46
7. 簡単なゲーム.....	48
Part4. 配列.....	51
1. アニメーションと for 文.....	52
2. 配列の使い方.....	54
3. 配列と for 文.....	56
4. 総合練習.....	58
5. 二次元の配列.....	60
6. 二次元配列の練習.....	62

入門編 Part1. グラフィックスとプログラミング入門

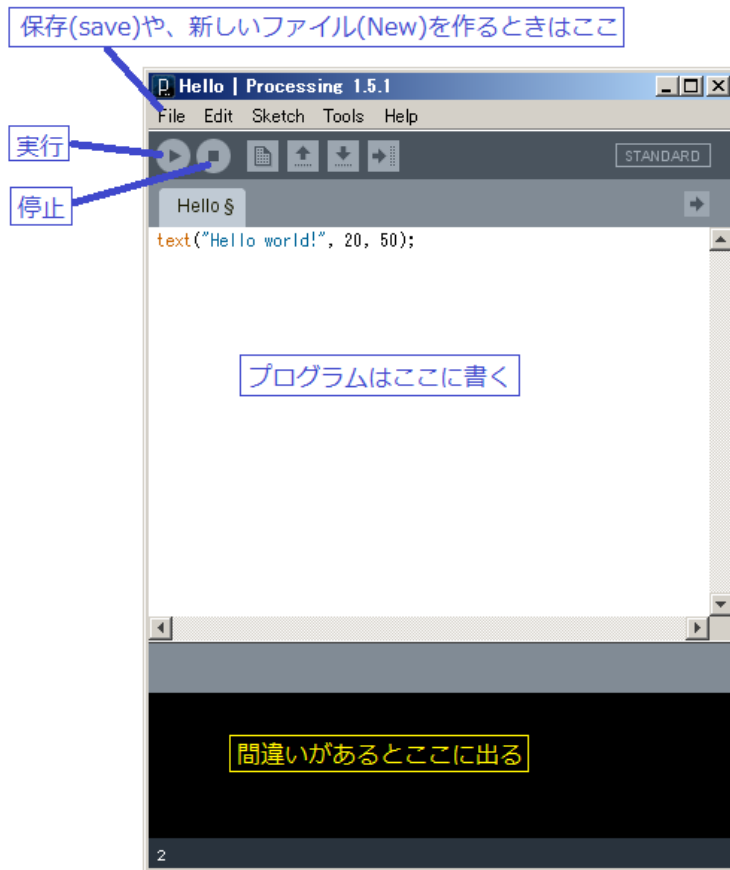
プログラムを書いてコンピュータに計算させたり、絵を描かせたりする方法を学ぶ。

1. Processing を使ってみる
2. 図形を描く
3. 色をつける
4. 練習①
5. 計算させよう
6. 変数
7. 数学関数

1-1. Processing を試してみる

- ・エディタ（プログラムを書くためのソフト）の使い方

デスクトップにある Processing をダブルクリックして起動しよう。



- (1) 上のメニューから「File」→「New」で新しいファイルを作る。
- (2) まんなかの白いところに、上のとおりにプログラムを書く。
文字や記号を間違えてはいけない。
- (3) 「File」→「Save」で保存。名前は通し番号を付けて
「Part1_1_Hello」のようにしよう。
- (4) 実行ボタンを押す。画面が開いてまんなかに
「Hello world!」と書いてあれば成功だ。

・このプログラムについて

プログラムとはコンピュータに与える指示である。

ここでは text という文を表示する命令が一つだけある。

命令	text(S, X, Y)
意味	文Sを左下が(X, Y)の位置になるよう表示
引数	S : 文。例: "Hello world!"
	X : 数。画面左端からの距離
	Y : 数。画面上端からの距離

命令に与えるデータを**引数**という。引数 S, X, Y を書きかえることで、文の内容や、表示する位置を変えることができる。

一つの指示の最後には、セミコロン(;)を書く。今は「行の終わりにはセミコロ
ン」と覚えておこう。

命令の書き方や、使う記号は英語の文法みたいなもので、間違えると意味が通じず、エラー（**バグ**という）になってしまう。

Processing にはいくつもの命令が用意されている。それをこれから学んでいくのだが、その前に…

・基本操作をマスターしよう

(1) text("Hello world!", 20, 50); をコピーして、次の行に貼り付けよう。

コピー : Ctrl キーを押しながら c。英語の大文字を打つ感覚で。

貼り付け : Ctrl 押しながら v

(2) 二行目の 3 番目の引数の 50 を 70 に書き換えよう。これで実行。

Hello world! がふたつ上下に表示されたら成功。

(3) 全てを選択 (Ctrl + a) して、BackSpace キーで消す。

そして元に戻す (Ctrl + z)。

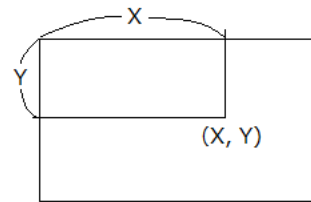
なにか操作を間違えた時には、これで元に戻せることを覚えておこう。また 1 から書きなおすよりもずっと速い。ちなみに 200 操作くらい前まで戻せる。

(4) 表示する文を変えてみよう。

1-2. 図形を描く

・画面上の位置

画面（ウィンドウ）上の位置を、2つの数字のペアで表す。（X, Y）で左端からX、上端からYの位置を表すことにする。



・図形を描く命令

次の命令で、点、線、長方形、だ円といった基本的な図形を描かせることができる。これらを組み合わせて、複雑な絵を描く。

図形を描く命令

```
point(X, Y)
    (X, Y)に点を書く
line(X0, Y0, X1, Y1)
    (X0, Y0)から(X1, Y1)に線を引く。
rect(X, Y, W, H)
    (X, Y)を左上の頂点とした、横W、高さHの長方形を描く。
ellipse(X, Y, W, H)
    (X, Y)を中心とした、横W、高さHのだ円を描く。
```

画面の大きさは100*100になっているが、次の命令で変えることができる。

命令

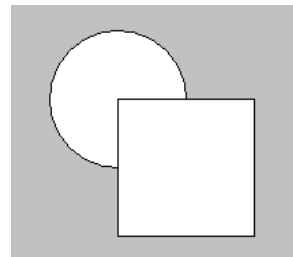
```
size(W, H)
    画面の横幅をW,縦幅をHにする。
```

・課題

- (1) 新しいファイルを作る。まず保存して「Part1_2_zukei」と名付けよう。
- (2) 画面を横400、縦300の大きさにして円を1個書こう。
- (3) ellipseの下の方に、長方形を描く命令を追加しよう。

うまく位置を調節して、円と少し重なるようにしよう。

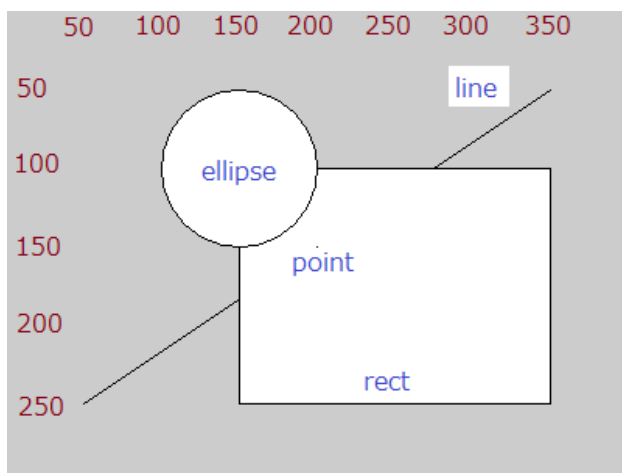
長方形のほうが上に来たら成功。processingでは下の行に書いたものほど（後に実行される命令ほど）上に重なる。



(例1)

- ・ 各命令と実行結果の図形との対応関係を確認しよう。
point はすごく小さい点になる。
- ・ ellipse と rect は黒いりんかく線に、白い塗りつぶしで描かれる。
- ・ 下に書いた命令ほど、図形は上になる。

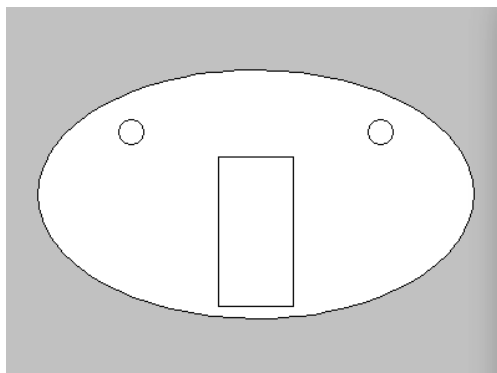
```
size(400, 300);  
line(350, 50, 50, 250);  
rect(150, 100, 200, 150);  
ellipse(150, 100, 100, 100);  
point(200, 150);
```



(例2)

- ・ ellipse 命令はだ円も描くことができる。

```
size(400, 300);           // 画面の大きさ  
ellipse(200, 150, 350, 200); // 顔  
ellipse(100, 100, 20, 20);  // 左側の目  
ellipse(300, 100, 20, 20);  // 右側の目  
rect(170, 120, 60, 120);    // 口
```



1-3. 色をつける

次は図形に色をつける。色の指定は次の 4 つの命令で行う。りんかく線と内部の塗りつぶしを別々に決める。

色に関する命令	
<code>stroke(R, G, B)</code>	りんかく線の色を R,G,B で指定
<code>fill(R, G, B)</code>	塗りつぶしの色を R,G,B で指定
<code>noStroke()</code>	りんかくなし
<code>noFill()</code>	塗りつぶしなし(透明)

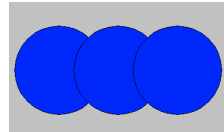
R, G, B には **0 から 255 の数** を入れる。R が大きいほど赤色に、G が大きいほど緑色に、B が大きいほど青色になる。（別紙の色見本を見よう）

これらの命令は、**それ以降の命令全てに影響する**。例えば、`stroke(255, 0, 0);` を実行すると、次に `stroke` か `noStroke` 命令が実行されるまで、描かれる図形のりんかく線は全て赤色になる。

・課題

(1) 新しいファイルを作り、「Part1_3_color」で保存。青い円を 3 つ描く。

```
size(400, 300);  
fill(0, 0, 255);  
ellipse(100, 150, 150, 150);  
ellipse(200, 150, 150, 150);  
ellipse(300, 150, 150, 150);
```



二行目の `fill` 命令で、塗りつぶしの色を青(0, 0, 255)にしている。

その後の 3 つの円全てが青になる。

(2) これを書き換えて、一番左の円だけ赤にせよ。

(3) 一番右の円をりんかくだけにせよ。

(4) 次のドイツの国旗を描こう。

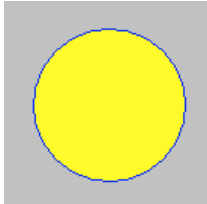
長方形を 3 つ描く。りんかく線は描かない。



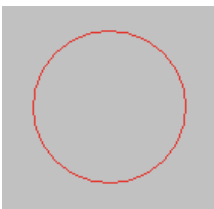
(例 1)

- ・ (255, 255, 0)は黄色、(0, 0, 255)は青、(255, 0, 0)は赤
(ellipse とかは省略)

```
fill(255, 255, 0);  
stroke(0, 0, 255);
```



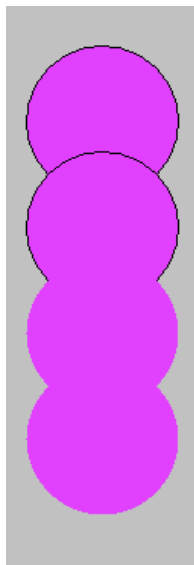
```
noFill();  
stroke(255, 0, 0);
```



(例 2)

- ・ fill などの命令は、それより下全てに影響する。
- ・ 1 行目の fill は、下の 4 つの ellipse すべての色を紫にしている。
- ・ 4 行目の noStroke は、下 2 つの円のりんかくを消している。

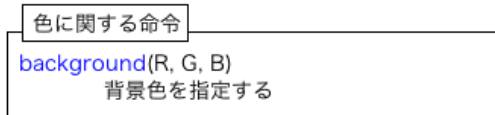
```
fill(255, 0, 255);  
ellipse(100, 100, 100, 100);  
ellipse(100, 170, 100, 100);  
noStroke();  
ellipse(100, 240, 100, 100);  
ellipse(100, 310, 100, 100);
```



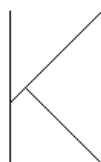
1-4. 練習①

ここで、もう少し絵を描く練習をしよう。それぞれ新しいファイルを作ると良い。日本語のファイル名は使えないので注意。

なお、次の命令を使うと、背景を灰色じゃなくて好きな色に出来るので便利。



(1) 直線で「K」



(2) スイカ

円と長方形で半円を作る。
種もつけてみましょう。

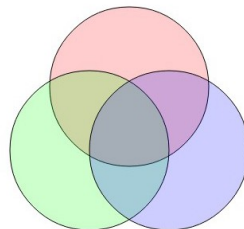


(3) 透明色

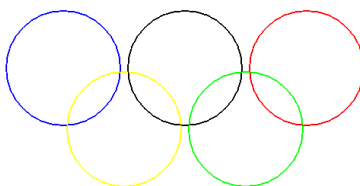
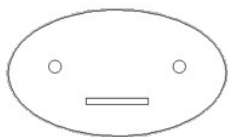
processingでは、半透明の色を使うことができる。今までのR, G, Bに加えて、**R, G, B, Aと4つの数を使う**。Aも0~255の数で、0が完全な透明、255が完全な不透明である。

例：`fill(255, 0, 0, 100);` 半透明の赤塗り

右のような、3つの半透明な円が重なっている絵を描こう。後ろの色が透けていることが分かる。



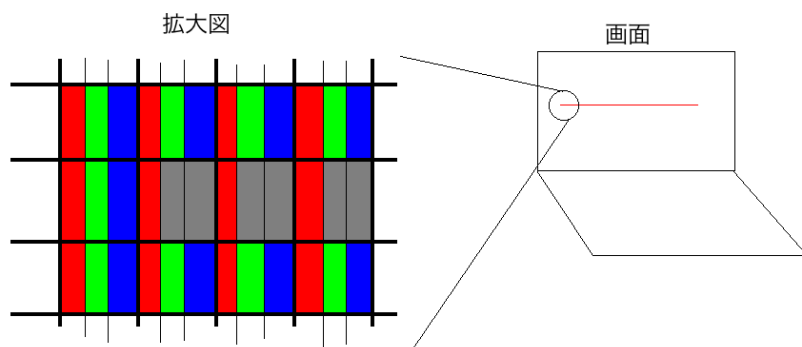
(4) 時間に余裕があったら。なにか好きな絵を作ってみてもいい。



(コラム 画面の仕組み)

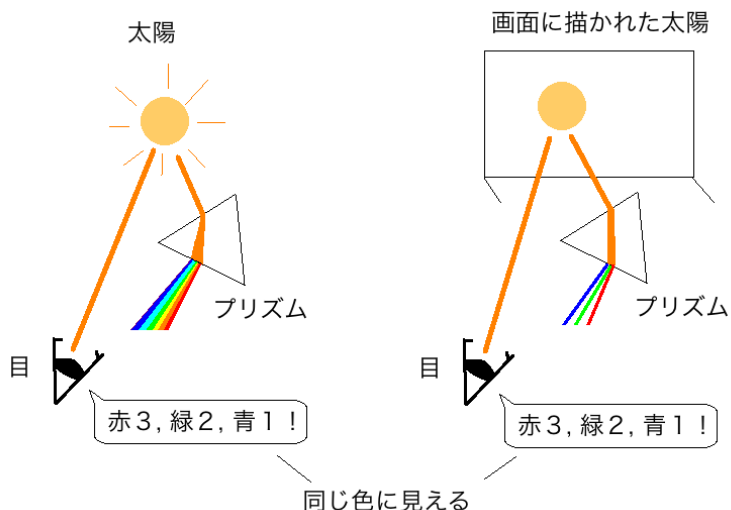
パソコンの画面を拡大すると、赤、緑、青の発光体がマス目状に並んでいる。下の図のように、画面の白いところでは赤、緑、青全てが光っていて、赤いところでは赤だけが光っている。

ちなみに画面上の位置(100, 200)などというのは、この小さいマス目で数えて何マス分のところか、ということである。



つまり下の絵のように、本物の太陽なら赤、橙、黄、…とあらゆる光が混ざっているが、画面に描かれた太陽は、赤と緑と青の3つの色だけで描かれているのである。

では、なぜ赤、緑、青の3つの色だけであらゆる色をつることができるのか？ それは、人間の目の中の色センサーは、赤、緑、青の3種類しかないからである。実際には違う光が混ざっていても、赤、緑、青の配合が同じなら、同じ色に見えてしまうのだ。



1-5. 計算させよう

ここではコンピュータに計算をさせる方法を学ぶ。コンピュータの強みはなんといってもその計算の速さと正確さなのだ。

まずは小学校の算数で習うような計算をさせてみよう。

計算と記号

足し算： +
引き算： -
かけ算： *
わり算： /
あまり： %
かっこ： ()

かけ算は×ではなく*、割り算は÷ではなく / を使う。なんで / なのかというと、4 割る 2 は $4/2$ と書くが、これは分数の 2 分の 4 のように見えるからである。

コンピュータでは整数と小数を区別する（違う電子回路で計算している）。小数点があるのが小数で、なければ整数となる。だから 1.0 は小数で、1 は整数。

整数と小数

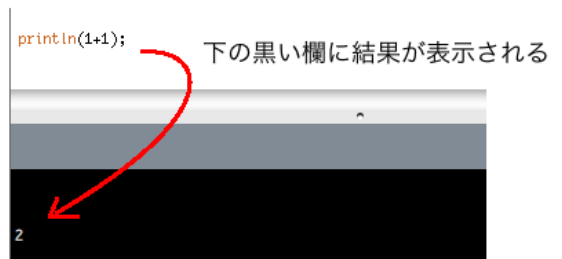
整数： 0, -1, 100	小数点なし
小数： -1000.1, 3.14, 1.0	小数点あり
整数 + 整数 → 整数	
整数 + 小数 → 小数	(-, *, / も同様)
小数 + 小数 → 小数	

これが問題になるのは割り算のときで、整数を整数で割ると、**四捨五入**した整数が結果になる。だから、 $5/2$ と書くと、これは 2.5 ではなく 2 になる。2.5 としたければ、 $5.0/2.0$ とか $5.0/2$ のように、どちらかを小数にしておかないといけない。

計算した結果を見るためには、次の命令を使う。

出力する命令

`println(X)`
X を下の黒い欄に表示する。



・ 課題

- (1) 新しいファイル「Part1_5_keisan」を作り、次のプログラムを書こう。
コピー&貼り付けを使って手速く書くこと。実行結果は右のようになるはずだ。

```
println(10+3);  
println(10-3);  
println(10*3);  
println(10/3);  
println(10%3);
```

```
13  
7  
30  
3  
1
```

- (2) 次の計算をさせてみよ。

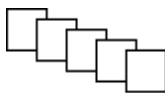
- ・ 7373×1507
- ・ 2 の 10 乗 (2 を 10 回かけたもの)
- ・ $1+2 \times 3+4$ と $(1+2) \times (3+4)$ 算数と同じで、かけ算が足し算よりも先
- ・ 12345 を 678 で割った商と余り。
- ・ 12345 を 678 で割った小数点付きの数。
- ・ $100/0$ ゼロで割ろうとするとどうなるか・・・?
- ・ 1234567×76543210 おかしな結果になる。

実は 2147483647 よりも大きい整数は扱えない。

- (3) 図形と計算

次のような等間隔にならんだ図形を描くときにも計算は便利である。

下の右のプログラムと左のプログラムどちらでも同じ絵が描けるが、計算を使った右のほうが、コピー&貼り付けを活用できるので優れている。



```
rect(0, 0, 20, 20);  
rect(15, 5, 20, 20);  
rect(30, 10, 20, 20);  
rect(45, 15, 20, 20);  
rect(60, 20, 20, 20);
```

←→
同じこと

```
rect(0*15, 0*5, 20, 20);  
rect(1*15, 1*5, 20, 20);  
rect(2*15, 2*5, 20, 20);  
rect(3*15, 3*5, 20, 20);  
rect(4*15, 4*5, 20, 20);
```

これを参考に、計算を使って次の絵を描け。ヒントは、円の中心は同じで、半径が 2 倍、3 倍…となっているということだ。



1-6. 変数

ここでは「**変数**」を学ぶ。変数とは変な数のことではなくて、「値を変えられる数」という意味である。値を変えようがない「100」のような数は**定数**という。

変数は… ひとつの数を覚えられる。
 数の代わりに使うことができる。
 新しい数を覚えさせると、前に覚えていた数を忘れる。
 好きな名前を付けられる。

文法 変数

変数を作る (整数)	<code>int hensu1;</code>	… hensu1 という名前の変数ができる。
	<code>int i, j, k;</code>	… i, j, k という名前の 3 つの変数ができる。
(小数)	<code>float hensu2;</code>	
変数に覚えさせる (代入)	<code>hensu1 = 100;</code> <code>hensu2 = hensu1/2.0;</code> <code>int x = 10;</code>	変数 = 式; と書くと、その変数は式の結果を覚える。 <code>int</code> で作った変数は整数を、 <code>float</code> で作った変数は小数を覚える。 作ると同時に代入する事もできる。
変数を使う	<code>int r=100, g=100, b=0;</code> <code>fill(r, g, b);</code>	… <code>fill(100, 100, 0);</code> が実行される

やや複雑だが、作って、代入して（覚えさせて）、使う、という 3 ステップを覚えておこう。

代入には `=` の記号を使う。これには次のような仲間がある。

代入の記号

<code>=</code>	左の変数に、右の式の内容を代入する
<code>+=, -=, *=, /=</code>	左の変数に、右の式の内容を足した（引、掛、割）ものを代入する
<code>++, --</code>	左の変数に 1 を足す、引く

たとえば、a という整数を覚える変数があったとして、

<code>a = 1;</code>	a は 1 を覚える。
<code>a += 2;</code>	a は 1 に 2 を足した 3 を覚える。
<code>a++;</code>	a は 3 に 1 を足した 4 を覚える。

このように使う。


・ 課題

(次のページの注意も参考にせよ。)

- (1) 新しいファイル (Part1_6_hensu) を作って、次のプログラムを書こう。
出力が右のようになったら成功だ。

```
int x;  
x = 1;  
println(x);  
x = 10;  
println(x);
```

xが覚えている数が増えている



4 行目の `x=10;` を、`x+=5;` に書き直してみよう。何が起きるか？
次に、それを `x++;` に書き直してみよう。今度はどうなるか？

- (2) 変数を使うと、同じ数を何度も書かなくて済む。例えば、
`3.14*3.14*3.14*3.14*3.14` を計算するのなら、このとおりに書くよりも、変数 `a` を
つくってそれに `3.14` を覚えさせ、`a*a*a*a*a` と書くほうが楽だ。

```
float a = 3.14;  
println(a*a*a*a*a);
```

これを参考にして、`1.122462` を 6 回かけたものを計算せよ。`2.0` になったら成功。

- (3) 変数を使うことで、同じプログラムで何種類もの計算ができる。
これは、底辺の長さ `a`、高さ `b` の長方形の面積を出力するプログラムである。

```
float a = 3.0;  
float b = 4.0;  
println(a*b);
```

このようにすることで、最初に `a` と `b` に覚えさせる数を変えることで、違う長方形の面積を答えることができる。

これを改良して、三角形の面積を計算するようにせよ。

また、これを参考に底面の半径 `a`、高さ `b` の円すいの体積を出力するプログラムを作れ。

注意

変数には次のような特性がある。

- ・タイプについて

整数と小数の 2 タイプがある。整数を覚える変数に小数を覚えさせようとするとエラーになる。（逆は可能）

- ・順序について

変数は作った後ならいつでも使えるが、作る前には使えない。

あとでもう一度言うが、ブロック（” { “と” } ” の間）内で作った変数は、そのブロック内でのみ使える。

- ・代入について

= 記号は数学で使う「両側が等しい」という意味ではなく、代入の指令を表す。何も代入しないと、0 を覚えている。

$a = a + 1;$ のように、同じ変数を = の両側に持ってこれる。この場合、a は古い a に 1 を足した数を新しく覚える。

- ・変数の名前について

変数の名前は英文字列で、1 文字でも 100 文字くらい長くてもいい。先頭以外は数字が使える。_ も使って良い。大文字、小文字は区別する（a と A は別の変数）。

例：i, abc, x1, x2, a_b_c, nagaihensuu, ABC

同じ名前の変数を 2 つ作れない。

すでに何かに使われている名前の変数は作れない。（size とか int とか。たいていそういうのは書くと色が変わる）

1-7. 数学関数（高校数学を勉強している人向け）

数学関数
<code>sqrt(X)</code> ルートXを返す
<code>pow(X, Y)</code> XのY乗を返す（Yが小数でもよい）
<code>sin(X)</code> , <code>cos(X)</code> , <code>tan(X)</code> 三角関数（Xは360°法ではなくラジアン）
<code>exp(X)</code> , <code>log(X)</code> 指数、対数関数
※Xも戻り値もすべて小数

これらの命令により、数学を勉強した人ならおなじみの関数を計算させることができる。

例えば、変数 a にルート 10 を入れたかったら、

```
a = sqrt(10);
```

と書く。変数 x, があつたときに x の 3 乗を出力させたかったら、

```
println(pow(x, 3));
```

と書く。

・課題（わからなかったら飛ばして良い）

（1）二次方程式の解の公式を用いて、 $ax^2 + bx + c = 0$ の 2 解を出力せよ。

プログラムは

```
float a = 2.0;
```

```
float b = -11.0;
```

```
float c = 12.0;
```

から始めよ。ちなみにこの時の 2 つの解は 1.5 と 4 だ。代入する数を別の数に変えて、あつてるかどうか確かめてみよ。

（2）正三角形を描け。（ルートを使って座標を求める。三角関数でもできる）

（3）五角の星を描け。（ルートより三角関数で座標を求める方が速い）

（4）適当な関数の微分を計算してみよ。例えば $\sin'(1)$ だったら

$(\sin(1+h) - \sin(1))/h$ （h は小さい数）

で計算できる。これは $\cos(1)$ と一致するだろうか？

入門編 Part2. 制御文

「ここを何回繰り返せ」というループを実現する for 文と、
「もし・・・だったら」という条件分岐を実現する if 文を学ぶ。

1. for 文の基本
2. for 文の練習
3. 乱数
4. if 文
5. 二重ループ
6. 総合練習

2-1. for 文の基本

ここでは for 文という、コンピュータに繰り返しをさせる文法を学ぶ。コンピュータは非常に高速（一秒に数千万回）で繰り返しを行うことができる。

for文の基本

```
for(A; B; C){
    (繰り返し処理)
}
```

A: 変数用意	例 int i=0	最初にAが実行され、
B: 条件式	例 i<10	Bの条件が正しいかぎり繰り返し処理が実行される。
C: 変化分	例 i++	一回の繰り返しが終わるたびに、Cが実行される。

まず、for 文では、繰り返したい内容を中括弧 { } の中に書く。この囲まれた領域を**ブロック**と呼ぶ。

繰り返しというからには、繰り返しを終える条件が必要だ。これが B の部分である。条件としては、次のようなものを使える。

条件式の記号

```
X == Y :等しい
X != Y :等しくない
X < Y :XがYより小さい
X <= Y :XがY以下
X > Y :XがYより大きい
X >= Y :XがY以上
```

```
(X==Y) && (Y==Z) :XがYと等しく、かつ、YがZと等しい
(X==Y) || (Y==Z) :XがYと等しい、または、YがZと等しい
```

for 文の最も基本的な書き方は、「カウンター」方式である。カウンターというのはボタンを押す度に数字が 1 ずつ増えていく機械のことで、次のようになる。

```
for(int i=0; i<10; i++){ ... }
```

A の部分でカウンター用の変数 i を用意して、0 を入れる。

C の部分で、繰り返しの処理一回ごとに、カウンター i に 1 を足すようにする。

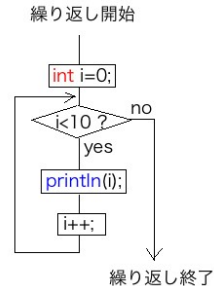
10 回繰り返したいなら、i が 10 未満でなくなったら終了すればいい。だから B のところには i<10 と書く。

・ 課題

(1) 新しいファイル (Part12_1_for) を作って、次のプログラムを書こう。

```
for(int i=0; i<10; i++){
    println(i);
}
```

```
0
1
2
3
4
5
6
7
8
9
```

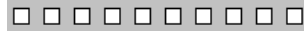


ここでの繰り返す処理は `println(i);`、つまり、カウンタ `i` の値を出力することだ。出力を見ると、カウンタ変数 `i` が 0 からはじまり、繰り返しの度に 1 ずつ足され、10 未満でなくなったところで繰り返しが終わっていることが分かる。

- ・ `i++` を `i+=2` に書き換えてみよ。何が起こるか？
- ・ `i=0` を `i=-10` に書き換えてみよ。どうなるか？
- ・ `i<10` は「`i` が 10 未満である限り繰り返す」という意味だったが、ここを書き換えて、実行すると -10, -8, -6, -4, -2, 0 となるようにせよ。

(2) 繰り返しといっても、例えば毎回同じ場所に四角形を書いても意味が無い。繰り返すたびに、すこしずつ違うことをしてこそ繰り返しの価値がある。

```
for(int i=0; i<10; i++){
    rect(i*20+5, 5, 10, 10);
}
```



このプログラムは、右のように等間隔で 10 個の四角形を描く。繰り返しの度に、カウンタ `i` が 0, 1, 2, ..., 9 となることで、`rect` の第一引数、すなわち四角形の左からの距離を、5, 25, 45, ... 185 と変化させている。こうすることで一回目の繰り返しでは一番左の四角形が、二回目では二番目が...となる。

これを書き換えて、四角形が縦に 8 個並ぶようにせよ。

(3) 次のプログラムは、1 から 10 までの合計を求めるプログラムである。これも、繰り返しの度にカウンタ `i` が 1 ずつ足されていくことを利用している。

```
int sum = 0;
for(int i=1; i<=10; i++){
    sum += i;
}
println(sum);
```

条件が `i<=10` になっていることに注意。これを書き換えて、1 から 1000 までの合計を求めよ。それができたら、 $1*2*3*4*5*6$ を求めるように書きかえよ。

2-2. for 文の練習

ここまで学んできた知識を使って、次のプログラムを書こう。

(1) 数列 (3 問)

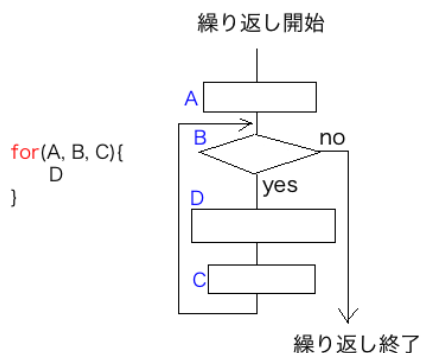
次のように出力されるように、for 文を書け。for 文のカッコの中を変える方法と、println のカッコの中を変える方法の二種類の解法がある。

```
10
9
8
7
6
5
4
3
2
1
```

```
10.0
10.5
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
```

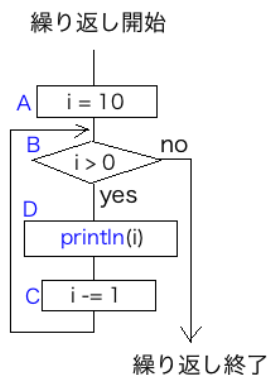
```
1
2
4
8
16
32
64
128
256
512
1024
```

わからなくなったら次の流れ図を書くといい。



例えば一番左の問題は次のようになる。

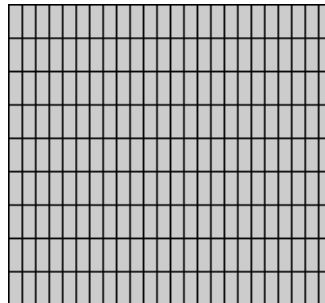
二番目の問題は小数なので int ではないことに注意しよう。



(2) 格子

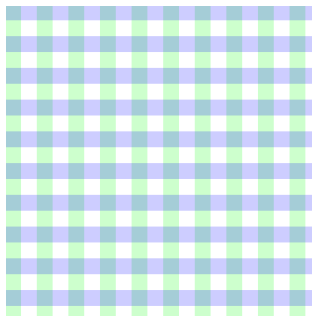
for 文で直線を等間隔でたくさん引く。空欄を埋めよ。一個目の for 文は縦線を、二個目の for 文は横線を引く。

```
size( );  
  
for( ) {  
    line( );  
}  
  
for( ) {  
    line( );  
}
```



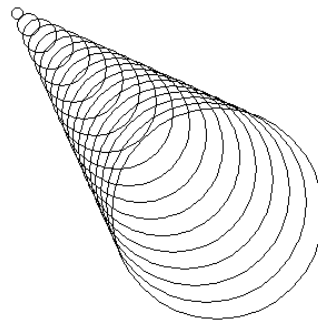
(3) チェック

格子の直線を長方形にするだけで、半透明の四角形にするとけっこう綺麗。



(4) 波紋

繰り返しの度に、円の位置と大きさを同時に変化させる。



(5) グラデーション

色を変えた直線をたくさん引く。位置と色を同時に変化させる。



2-3. 乱数

投げたコインの表裏、サイコロの目、適当に引いたトランプの数、のように、「何が出るかわからないもの」を**ランダム**な数、あるいは**乱数**という。これはゲームには必須の要素で、予測不能な敵の行動などは乱数をつかって決められている。

命令 乱数

`random(A, B)`
A以上B未満のランダムな小数を得る

この命令は今まででてきた `ellipse` や `println` とかとは少し違って、**値を持つ**（戻り値を持つという）。つまり、

```
float r = random(0, 1);
```

のように式の中で使い、この式では変数 `r` に 0 以上 1 未満のどれかの小数が入る。何が入るかは実行するたびに変わる。

ところで、ランダムな小数ではなくて、整数が欲しい時もある。そういうときは、次の方法で小数を整数に変換する（キャストするという）。整数へのキャストは、`int` にカッコをつけた、`(int)`で行う。

構文 キャスト

`(int)r`
`r`を整数に四捨五入する

```
int a = (int)random(0, 10);
```

のように書くと、`random(0, 10)`は 0 以上 10 未満のランダムな小数なので、変数 `a` にはそれを四捨五入した 0 以上 10 未満のランダムな整数が入る。

・課題

(1) 新しいファイル (`Part2_3_random`) を作って、次のプログラムを書こう。

```
float r = random(0, 10);  
println(r);
```

何回か実行して、毎回値が変わることと、0 以上 10 未満以外の値は出ないことを確かめよ。

(2) 上述のキャストを使って、毎回 0 以上 10 未満のランダムな**整数**が出力されるようにせよ。

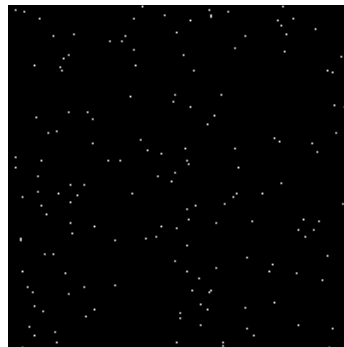
(3) 乱数と for 文

右のような星空を描こう。

for 文を使ってたくさんの点を描く。

点の位置は乱数を使って画面内のランダムな位置になるようにしている。

```
size(200, 200);  
background(0, 0, 0);  
stroke(255, 255, 255);  
for(int i=0; i<200; i++){  
  point(random(0, 200), random(0, 200));  
}
```

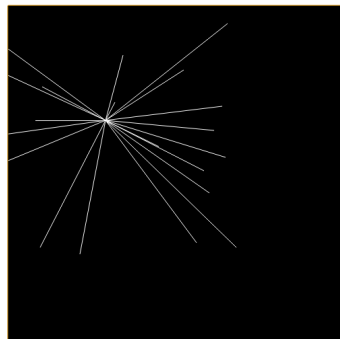


point 命令で描かれる点の色は fill でなく stroke 命令で決まることに注意しよう。

これができたら、星の数を増やしたり、画面の大きさを大きくしたりしてみよう。

(4) 乱数と for 文②

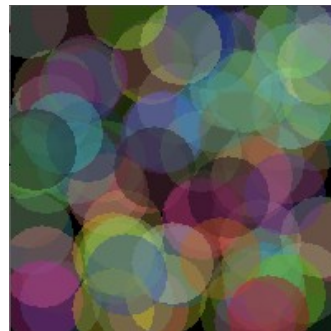
点の代わりに、ある点から真っ直ぐ伸びる直線を描くと、右のようにヒビのような模様が描ける。



(5) 乱数と for 文③

右のような模様を描こう。

ランダムな位置に、ランダムな色の半透明な円をたくさん描く。



2-4. if 文

「もし ならば…せよ」といった条件分岐には if 文を使う。

if文

A, Bは条件式
(for文のページに詳述)

```

if(A){
    (Aが正しければ実行される処理)
}

if(A){
    (Aが正しければ実行される処理)
}else{
    (Aが間違っていれば実行される処理)
}

if(A){
    (Aが正しければ実行される処理)
}else if(B){
    (Aが間違っていて、Bが正しければ実行される処理)
}else{
    (AもBも間違っていれば実行される処理)
}

```

上の「A が正しければ」というのは、例えば `if(x<3)` で `x` が 3 未満という状況を指す。条件式の種類と記号は 16 ページを見てほしい。ちなみに上の 3 番めの書き方では `else if` が何段階も続いても大丈夫で、その数だけ分岐を増やせる。

・課題

(1) おみくじ

新しいファイル (Part2_4_if) を作って、次のプログラムを書こう。

```

float r = random(0, 3);
println(r);
if(r<1){
    println("small");
}else if(r<2){
    println("medium");
}else{
    println("big");
}

```

このプログラムは、まず 0 以上 3 未満の乱数を変数 `r` に代入し、`r` が 1 未満なら `small`, 1 以上 2 未満なら `medium`, 2 以上なら `big` と出力する。

(2) おみくじ 2

これを参考にして、まず変数 r に 0 以上 1 未満の乱数を代入して、 r が 0.5 未満なら青い円、0.5 以上なら赤い円を表示するようにせよ。

(3) for 文の中の if 文

```
size(200, 200);
background(0, 0, 0);
stroke(255, 255, 255);
for(int i=0; i<200; i++){
    float x = random(0, 200);
    float y = random(0, 200);
    point(x, y);
}
```

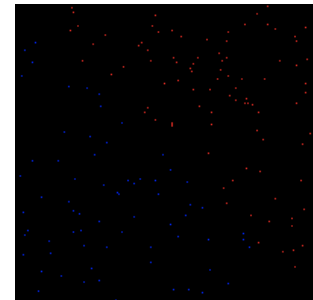
このプログラムは、21 ページの星空のプログラムを少し変えたものである。これに if 文を付け加えて、 x が y より大きい時だけ点を描くようにせよ。

成功すれば、右図のように右上だけに点が描かれる。



(4) for 文の中の if 文②

上のプログラムをさらに書き換えて、右図のように、右上が赤、左下が青になるようにせよ。



(5) 最大値（やや難）

次のプログラムは、左のように、10 個のランダムな数を作って表示するプログラムである。これを書き換えて、右のように 10 個の乱数を表示したあとで、その 10 個の中で一番大きい数を出力するようにせよ。（下の例では 0.9926714 が一番大きい）

最大値を記録する変数 mx を用意して、繰り返しの度に r が mx より大きかったら、 mx を r に置き換える（ mx に r を代入する）ようにすればいい。

```
for(int i=0; i<10; i++){
    float r = random(0, 1);
    println(r);
}
```

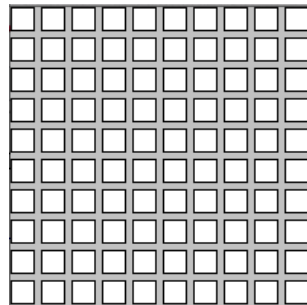
```
0.8101876
0.87151057
0.26559848
0.9926714
0.61440974
0.63663745
0.40953827
0.8945355
0.748191
0.22402161
```

```
0.8101876
0.87151057
0.26559848
0.9926714
0.61440974
0.63663745
0.40953827
0.8945355
0.748191
0.22402161
0.9926714
```

2-5. 二重ループ

for 文の中に for 文を入れることを多重ループという。

```
size(200, 200);
for(int x=0; x<200; x+=20){
    for(int y=0; y<200; y+=20){
        rect(x, y, 15, 15);
    }
}
```

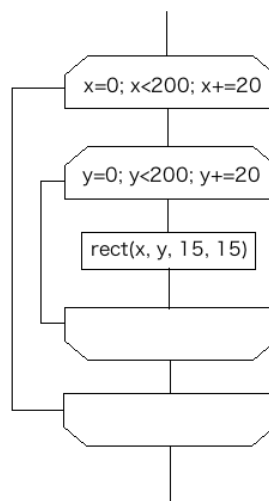
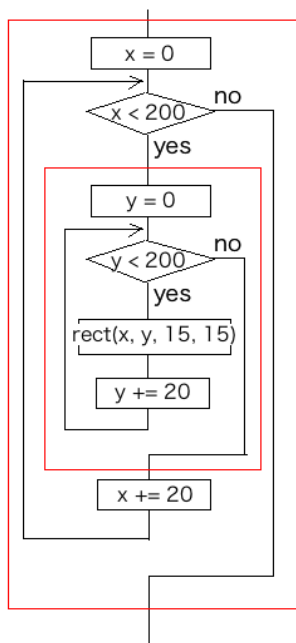


このように書くと、一番内側の rect では (x, y) が

(0, 0) → (0, 20) → … → (0, 180)
 → (20, 0) → (20, 20) → … → (20, 180)
 → … →
 → (180, 0) → … → (180, 180)

と変化する。これによって、マス目状に四角形を描くことができる。

流れ図を書くと、左のようになる。内外の赤い長方形が、それぞれの for 文を表している。ただ、ごちゃごちゃになるので慣れてきたら右のように略して書く。



多重ループを使うときは、次のインデントと変数の有効範囲に注意しよう。

・ インデント

もう気づいているかもしれないが、ブロック（{ }の中）では、行の先頭に空白を入れて、少し右にずらす。ブロックの中にブロックが入るときは、その深さだけ空白を入れる。入れなくてもエラーにはならないが、こうすることでプログラムが見やすくなる。

長いプログラムの例。中括弧が開かれる度に、それが閉じるまで、その中身が空白2個（あるいはTabキー1個）分右にずれていることがわかるだろう。

Tabキー あるいは 空白2個

```
void draw(){
  background(255, 255, 255);
  er.idou();
  te.idou();
  tp.idou();
  pl.idou();
  for(int i=0; i<en.n; i++){
    for(int j=tp.n-1; j>=0; j--){
      if(isCollided(en.get(i), tp.get(j))==1){
        Enemy e = (Enemy)en.get(i);
        e.hp--;
        tp.sakujo(j);
      }
    }
  }
  for(int i=0; i<pl.n; i++){
    for(int j=te.n-1; j>=0; j--){
      if(isCollided(pl.get(i), te.get(j))==1){
        Jiki p = (Jiki)pl.get(i);
        p.hp--;
        te.sakujo(j);
      }
    }
  }
  noStroke();
  fill(255, 0, 0, 100);
  rect(0, 0, width, height);
}
}
```

・ 変数の有効範囲

ブロックの中で作られた変数は、そのブロックが終わると使えなくなる。だから下の例では、変数 c は一番内側の if ブロックの中だけで、変数 j は二番目に内側の for ブロックの中だけで使える。ブロックを出るとその変数は消滅する。

```
for(int i=0; i<10; i++){
  int a;
  for(int j=0; j<20; j++){
    if(...){
      int c;
      ...
    }
    ...
  }
  ...
}
```

変数 i の有効範囲
変数 a の有効範囲

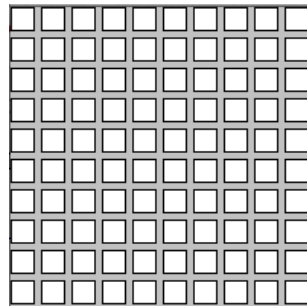
変数 j の有効範囲
変数 c の有効範囲

・ 課題

(1) 28 ページの例を実装せよ。

右のように表示されたら、次に、

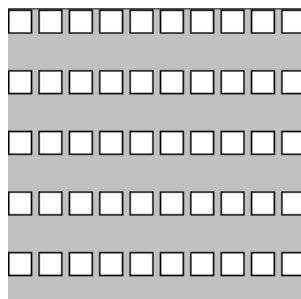
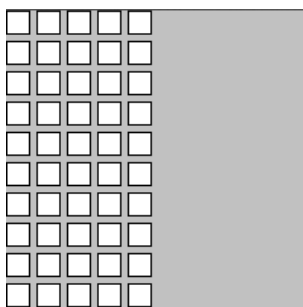
```
for(int i=0; i<10; i++){
  for(int j=0; j<10; j++){
    rect(, 15, 15);
  }
}
```



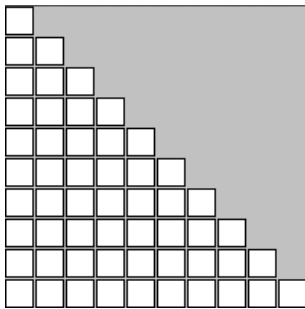
この空欄を埋めて、全く同じように表示されるようにせよ。

変数名が x, y から i, j に変わっていることに注意しよう。28 ページの例では変数 x, y はどちらも 0, 20, 40, ... 200 というふうに変化したが、今度は変数 i, j が 0, 1, ... 10 というふうに変化するので、それに合わせて `rect` の中身を書き換えなければならない。

(2) `for` 文を書き換えて、次のように表示されるようにせよ。



- (3) if 文を使って、次のように表示されるようにせよ。
if 文を使う方法と、for 文の変数 j の条件を変える方法がある。
ヒント：対角線上にある四角形たちは i と j の値が等しい。



- (4) 右のようなグラデーションを描け。
point を使って 255*255 個の色の違う点を描く。
point の色指定は stroke を使う。

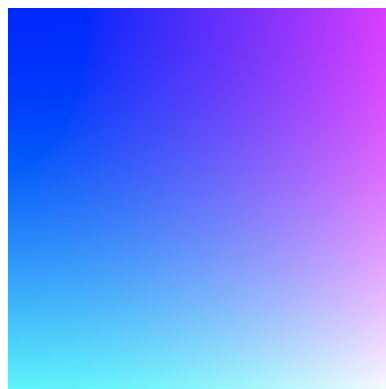
(ヒント)

左上の色は (0, 0, 255)

右上は (255, 0, 255)

左下は (0, 255, 255)

右下は (255, 255, 255)

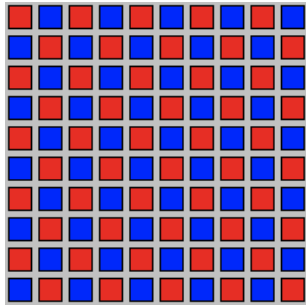


2-6. 総合練習

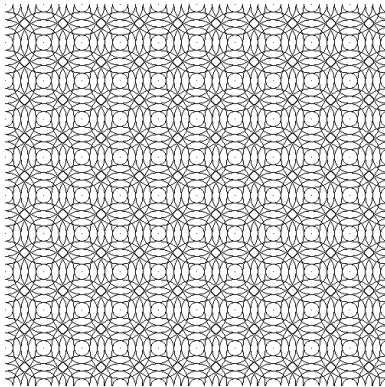
(1) 二重ループと if 文

和が偶数 (0, 0) (0, 2) (2, 2) ... → 赤

和が奇数 (0, 1) (1, 0) (2, 1) ... → 青



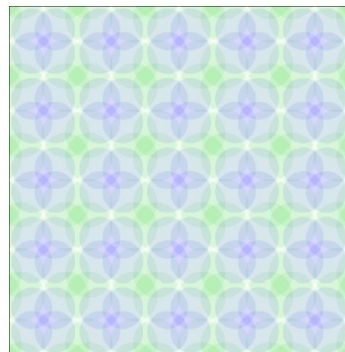
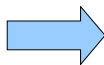
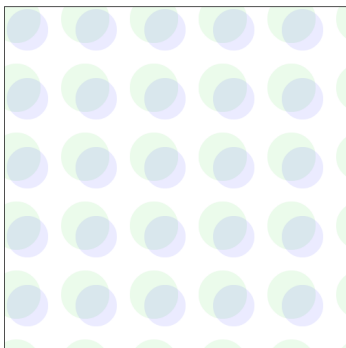
(2) 三重ループ



(3) 壁紙模様

半透明の二色の円を縦横等間隔に二重ループで描く（左図）。

円の種類を増やすと右図のようなきれいな模様が描ける。



(4) 素数の判定

ある整数の変数 a (2 以上とする) が、素数 (1 と a 以外に割り切る数がない) であるか判定するプログラムを書け。

for 文で 2 から $a-1$ まで、その数が a の約数か (割った余りが 0 か) チェックし、約数の個数が 0 ならば、 a は素数である。

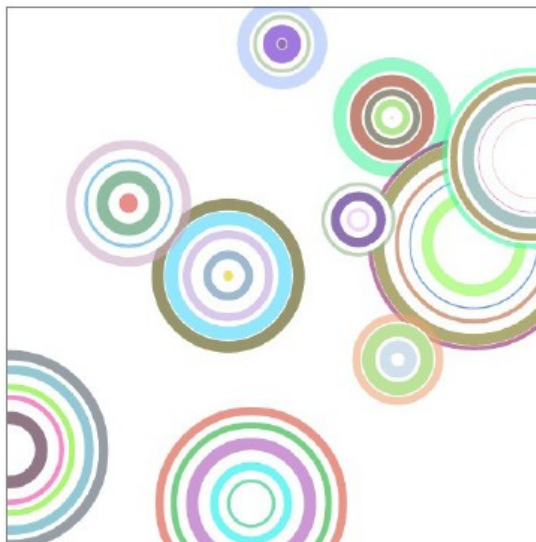
(5) 素数の判定②

上のプログラムを改良して、1000 以下の素数を全て出力せよ (二重ループ)。

(6) 二重ループと乱数

for 文で 10 回のループをする。1 回のループにつき、乱数で決まった (x, y) を中心とする同心円たちを描画する。

同心円は for ループで外側の円から描いていき、色と半径の変化は乱数で決める。



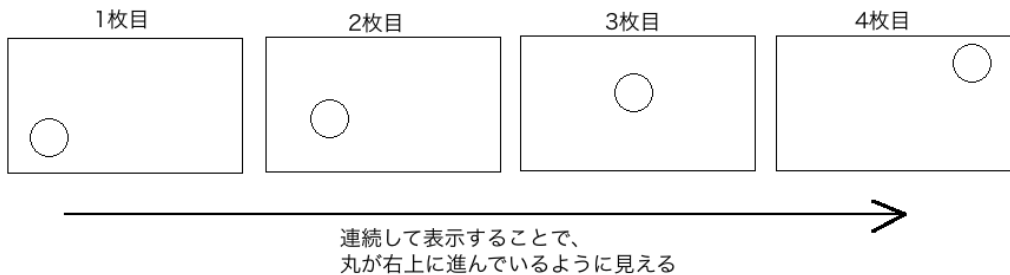
入門編 Part3. アニメーションと操作

動画を作る方法と、マウスによる操作を学んでゲームを作る。

1. アニメーションの仕組み
2. マウス操作 1 : マウスの位置を知る
3. マウス操作 2 : クリック
4. 変数で図形を動かす
5. if 文による制御
6. 練習
7. 簡単なゲーム

3-1. アニメーションの仕組み

パソコンの画面は、実は一秒間に 60 回くらい書き変わっている。素早く書き変わっているために、人間の目にはそうとはわからない。書きかえるたびに少しずつ絵を変えることで、その絵を動いているように見せることができる。これがアニメーションの仕組みだ。



いままでは動かない静止画を表示する方法を学んできた。アニメーションを作るには、静止画を表示する処理を、一秒間に何度も実行させれば良い。そのためには次の構文を使う。

アニメーションの構文

```

void setup(){
    (最初に一回だけ実行する処理)
}

void draw(){
    (画面を書きなおす処理。一秒に何度も実行される)
}
    
```

この書き方は、「setup と draw という名前の命令を作る」ということなのだが、命令の作り方については時間があったら扱うとして、今は、

void setup() { ... } の中括弧の中に書いたものは、最初に一回実行され、
 void draw() { ... } の中括弧の中に書いたものは、一秒間に何度も実行される。
 ということを覚えておこう。この draw の中の処理を、時間とともにちよつとずつ
 変えることで、アニメーションさせることができるという戦法である。

ちなみに、一秒間に draw 命令を呼び出す回数は、次の命令で設定できる。

命令 frameRate(X)
 一秒間に X 回 draw 命令を呼ぶように設定する。
 標準では 60 になっている。
 この命令は setup 命令の中に書く。

(1) 次のプログラム (Part3_1_animation) を書こう。

```
int step;

void setup(){
  size(200, 200);
  step = 0;
  frameRate(10);
}

void draw(){
  ellipse(random(0, 200), random(0, 200), 20, 20);
  step++;
  println(step);
}
```

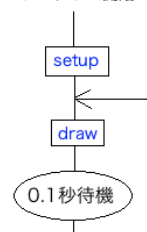
```
284
285
286
287
288
289
290
291
292
293
294
295
296
297
```

円がどんどん増えていくアニメーションが表示されて、黒い画面に上のような1ずつ増えていく数が表示されたら成功だ。

このプログラムは、外側、setup ブロック、draw ブロックの3つに分かれている。前のページに書いたように、setup は最初に一回、draw はそれから一定の時間ごとに何回も実行される。外側には、setup と draw で共通に使う変数を書く。

```
外側  < int step;
      <
      < void setup(){
      <   size(200, 200);
      <   step = 0;
      <   frameRate(10);
      < }
      <
      < void draw(){
      <   ellipse(random(0, 200), random(0, 200), 20, 20);
      <   step++;
      <   println(step);
      < }
```

プログラム開始



(2) setup ブロックの中にある frameRate 命令は、一秒間に draw 命令が呼ばれる回数を設定しているのだった。この値を10でない数値にかえて、実際その通りになっている (円の表示される速度、黒い画面に数が表示される速度が変わる) ことを確認しよう。ただし PC の性能によって、限界がある (200 くらいかな)。

(3) background(255, 255, 255); という命令は、画面全体を白く塗りつぶす、という意味を持つ。これを次の3つの場所に書いた時にそれぞれ何が起こるか考え、確かめよ。

- ・ setup ブロックの中 (最初に一回だけ白く塗りつぶす)
- ・ draw ブロックの中の、ellipse 命令の前 (円を描く前に白く塗りつぶす)
- ・ draw ブロックの中の、ellipse 命令の後 (円を描いた後に白く塗りつぶす)

(4) 次に、変数 step に注目しよう。この変数は「外側」で作られているので、setup, draw 両方の中で使うことができる (逆に、draw の中で作った変数は draw ブロックの中でしか使えない)。変数 step には最初 0 が代入され、draw 実行のたびに、1 足されて、そのあと黒い画面に println 命令で出力される。

- ・ 今黒い画面には 1, 2... と出力されるが、100, 101... となるようにせよ。
- ・ 10, 20, 30, ... と 10 ずつ増えるようにせよ。

3-2. マウス操作①：マウスの位置を知る

ここからマウスによる操作方法を学んでいこう。

マウスの画面上の位置は、最初から名前の決められた次のような変数に記憶されている。

マウスの位置	
<code>mouseX</code>	... マウスの画面左端からの距離
<code>mouseY</code>	... マウスの画面上端からの距離

これらの変数は、マウスが移動したならばそれに合わせて値も変わる。

このような「はじめから役割が決まっている変数」には他にも次のようなものもあるので、ついでに覚えておこう。

画面サイズ	
<code>width</code>	... 画面の横幅
<code>height</code>	... 画面の縦幅
これらは <code>size</code> 命令で指定した値と同じになる	

これらの変数を使うと、「画面の半分」といったときに(`width/2`, `height/2`)と書けるので便利である。特に、プログラムを作ってる途中で画面のサイズを変更したくなった時などに、これを使ってみると、その部分を変更しなくて済むので楽である。

さて、それではマウスの位置を活用したプログラムを作ってみよう。

(1) 次のプログラムを作れ (Part3_2_mouse1)。

```
void setup(){
  size(200, 200);
}

void draw(){
  ellipse(mouseX, mouseY, 10, 10);
}
```

`ellipse`の第一、第二引数が`mouseX`, `mouseY`となっているので、円はマウスの位置と同じ場所に描かれる。`draw`命令は一秒間に何度も何度も実行されているということを頭に入れておこう。

`mouseX`と`mouseY`を入れ替えてみよう。なかなか気持ちわるいことになる。左上から右下への対角線を軸に、マウスと線対称な位置に円が描かれる。

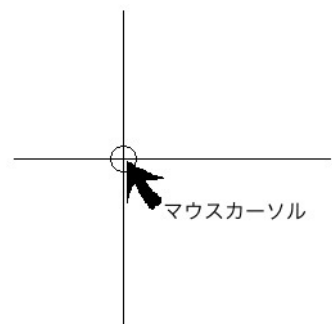
さらに、`println`文を使って、`draw`命令の中で`mouseX`と`mouseY`を出力させてみよう。マウスの現在位置がひたすら表示されるようになるはずである。

(2) マウスの位置によって描かれる円の色も変わるようにしよう。mouseX を赤色の強さ、mouseY を緑色の強さにすると右のような感じになる。



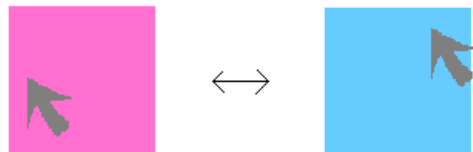
(3) 照準器

右のようにマウスのまわりに縦横の直線と円を描いて、照準器（銃の狙いをつけるところ）のようにしよう。残像を消すには白く塗りつぶす：background(255, 255, 255); をどこかに入れる。



(4) センサー①

マウスの位置によって画面の色が変わるようにしよう。if 文を使って、マウスが画面中央より右にあったら (mouseX > width/2) 青、左にあったら赤にしよう。



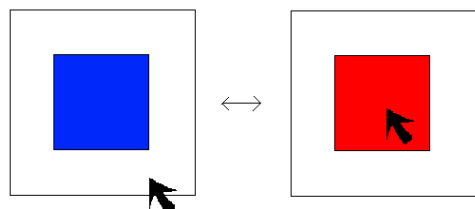
(5) センサー②

右のように画面内に四角形を描くのだが、マウスがその四角形の外にあったら青、四角形の中に入ったら赤で描くようにしよう。

if 文の中の条件「mouseX が 50 以上 150 未満」は、

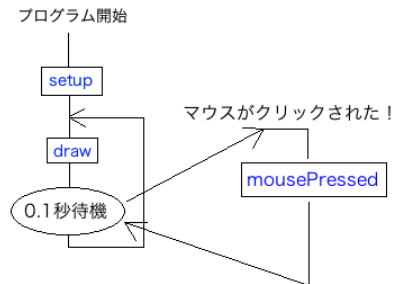
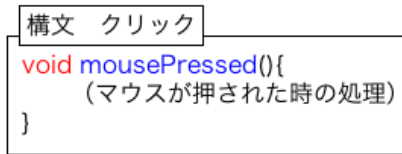
(mouseX >= 50) && (mouseX < 150)

と書く。



3-3. マウス操作②：クリック

「マウスがクリックされたら何かをする」という処理は、次の構文を使う。



この `mousePressed` ブロックは、`setup` や `draw` の外側の領域に書く。このブロックの中に書かれた処理は、マウスがクリックされた時だけ実行される。

(1) 次のプログラムを書こう (Part3_3_mouse2_1)。

```

void setup(){
  size(200, 200);
  background(255);  ————— background(255, 255, 255);の省略
}

void draw(){
}

void mousePressed(){
  ellipse(mouseX, mouseY, 20, 20);
}
```

これはマウスがクリックされたら、その位置に円を描くプログラムである。`draw` ブロックには何も書かれていないので毎ステップ描かれるものはないが、`mousePressed` ブロックの中に `ellipse` 命令があるので、マウスが押された時に円が描かれるというしくみになっている。

・マウスがクリックされたら、画面がランダムな色で塗りつぶされるようにせよ。

(2) 移動

右のプログラムを書こう (Part3_3_mouse2_2)。

まず、最初の行でつくられた `x` という変数は描画する円の左端からの距離を表す変数である。今の状態では、クリックする度にこれに 10 が足され、従って円も右に 10 動くはずである。

・クリックしたら円がランダムな方向に動くようにせよ。

```

float x;

void setup(){
  size(200, 200);
  background(255);
  x = 0;
}

void draw(){
  ellipse(x, 100, 20, 20);
}

void mousePressed(){
  x += 10;
}
```


(3) スイッチ①


右のプログラムの draw ブロックの中は、mode という変数が 0 なら赤、それ以外なら青で画面を塗りつぶす。

mousePressed ブロックの空欄を埋めて、クリックする度に画面の赤青が切り替わるようにせよ。

```
int mode;

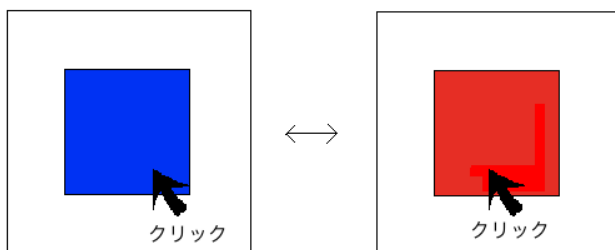
void setup(){
  size(200, 200);
  mode = 0;
}

void draw(){
  if(mode==0){
    background(255, 0, 0);
  }else{
    background(0, 0, 255);
  }
}

void mousePressed(){
  
}
```

(4) スイッチ②

2 ページ前のセンサーの問題を改良して、四角形の内部でクリックしたら、その四角形の色が赤 \leftrightarrow 青と切り替わるようにせよ。



3-4. 変数で図形を動かす

3-1 では draw 命令でランダムな位置に円を描いた。ここでは図形の位置を変数で決められた場所にして、それを draw 命令のたびに变化させることで、図形が動いているようにみせることを考えよう。

(1) 次のプログラムを書いて (Part3_4_hensu) 実行してみよう。

```
float x;

void setup(){
  size(200, 200);
  x = 0;
  frameRate(10);
}

void draw(){
  ellipse(x, 100, 10, 10);
  x += 3;
}
```

円が右に移動しているように見えるだろうか？

setup は最初に一回、draw はその後何度も実行されるということを思い出すとこのプログラムは次のような順序で命令を実行する。

外側	float x;	
setup	size(200, 200); x = 0; frameRate(10);	円の描かれる位置
draw 1回目 (0.1秒待機)	ellipse(x, 100, 10, 10); x += 3;	(0, 100)
draw 2回目 (0.1秒待機)	ellipse(x, 100, 10, 10); x += 3;	(3, 100)
draw 3回目 (0.1秒待機)	ellipse(x, 100, 10, 10); x += 3;	(6, 100)
draw 4回目	ellipse(x, 100, 10, 10); x += 3;	(9, 100)
...

draw が実行される度に、ellipse 命令の 1 番目の引数（画面左端からの位置）に入っている変数 x に 3 が足されているので、結果的に 0.1 秒ごとに 3 ずつ右にずれた位置に円が描かれ、円が右に移動しているように見えるわけだ。

(2) 残像を消してみよう。

(3) 右図のように、円が右下に移動するようにしたい。次の手順で改良してみよう。

(i) 今、外側で変数 x を作っているが、ここでもう一つ変数 y を作る。

```
float y;
```

(ii) setup ブロックの中で、 y に 0 を代入する。

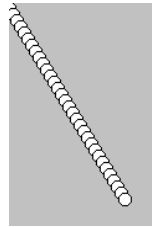
```
y = 0;
```

(iii) draw ブロックで、円を描く位置を $(x, 100)$ ではなく (x, y) にする。

```
ellipse(x, y, 10, 10);
```

(iv) draw ブロックで、毎回 y に 5 を足すようにする。

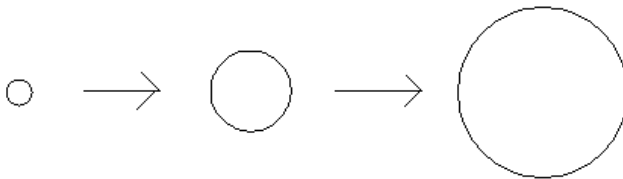
```
y += 5;
```



円を描く位置は、 $(0, 0) \rightarrow (3, 5) \rightarrow (6, 10) \rightarrow (9, 15) \rightarrow \dots$ となるはずである。

(4) だんだん円が大きくなるアニメーションを作れ。

いままでは `ellipse` の第一、第二引数（画面上の位置）を変数で変化させてきたが、今回は第三、第四（円の大きさ）引数をいじる。



(5) だんだん色が変わっていくアニメーションを作れ。

(i) 黒から白



$(0, 0, 0) \quad \dots \quad (120, 120, 120) \quad \dots \quad (240, 240, 240)$

(ii) 赤から青



$(255, 0, 0) \quad \dots \quad (125, 0, 130) \quad \dots \quad (0, 0, 255)$

3-5. if 文による制御

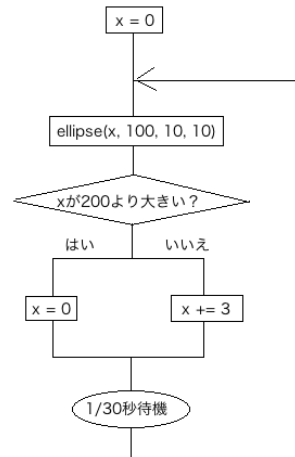
if 文を使って動きに変化をつけてみよう。少しずつ複雑になっていくので、ちゃんと理解してから先に進むようにしよう。

(1) 次のプログラムを作れ (Part3_5_if1)。3-4 で作ったプログラムをコピーすると楽である。

```
float x;

void setup(){
  size(200, 200);
  frameRate(30);
  x = 0;
}

void draw(){
  background(255, 255, 255);
  ellipse(x, 100, 10, 10);
  if(x > 200){
    x = 0;
  }else{
    x += 3;
  }
}
```



右端に行った円が左端に戻れば成功だ。

このプログラムは、右の図のような流れで実行される（主要な部分以外省いてある）。draw 中の if 文は、x が 200 を超えていたら x に 0 を代入し、200 以下なら x に 3 を足す。すると x は

0, 3, 6, ... 195, 198, 201, 0, 3, ...

↑ ここで if 文は else の側が実行されるように、200 を超えたところで 0 に戻るようになる。

(2) このプログラムを書き換えて、円は左方向に進み、左端に到達したら右端に戻るようにせよ。

(3) 前のページの (5) で作った黒から白に色が変わるプログラムを改良して、完全に白になったら黒に戻るようにせよ。

(4) もう少し複雑な制御に挑戦してみよう。今度は右方向に進む円が右端に到達したら、左方向に引き返すようにしてみたい。

次のプログラムを作れ (Part3_5_if2)。

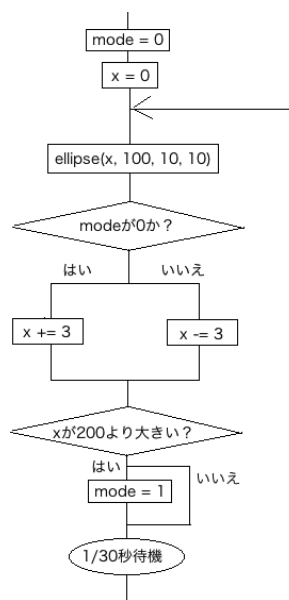
```
float x;
int mode;

void setup(){
  size(200, 200);
  frameRate(30);
  x = 0;
  mode = 0;
}

void draw(){
  background(255, 255, 255);
  ellipse(x, 100, 10, 10);

  if(mode==0){
    x += 3;
  }else{
    x -= 3;
  }

  if(x > 200){
    mode = 1;
  }
}
```

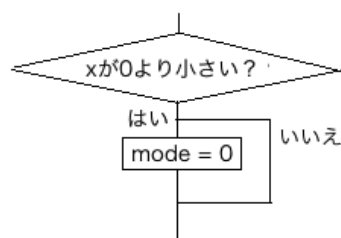


このプログラムのポイントは、「mode」という変数が0なら右向きに、そうでなければ(1なら)左向きに進む、としている点である。**xが200を超えたところでmodeを0から1に切り替えることによって、そこで突然進行方向が逆転し、右端で跳ね返るように見せることができる。**これに納得できたら、次に進もう。

(5) このプログラムを改良して、左端でも跳ね返るようにしよう。

右の図のような働きをする if 文を追加すれば良い。

つまり、左端に到達したら mode を 1 から 0 に戻して、再び右向きに進むようにするのである。



3-6. 練習

(1) 下に加速

重力に引かれているかのように、下向きに加速していく円を描きたい。

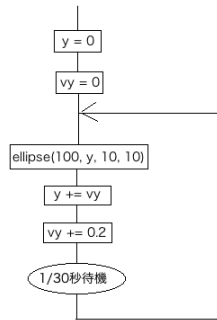
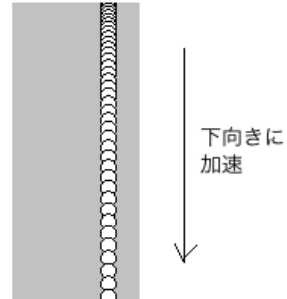
これを実現するには、2つの変数 y と vy を用意し、
円の上端からの距離 y が

```
y: 0.0 0.0 0.2 0.6 1.2 2.0 3.0 4.2
vy: 0.0 0.2 0.4 0.6 0.8 1.0 1.2
```

の一行目のように変化するといい。

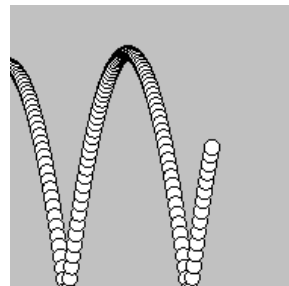
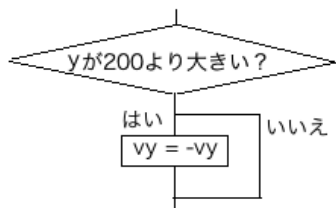
この二行目の vy と書いた変数は一行目の隣り合った2つの差をとったもので、これが繰り返しごとに0.2ずつ増えていることがわかる。これは下方向への「速度」を表している。なぜなら繰り返しのたびに y が vy ずつ増えていくからである。

次の図を参考にして、下方向に加速する円のアニメーションを書こう。



(2) 下に加速②

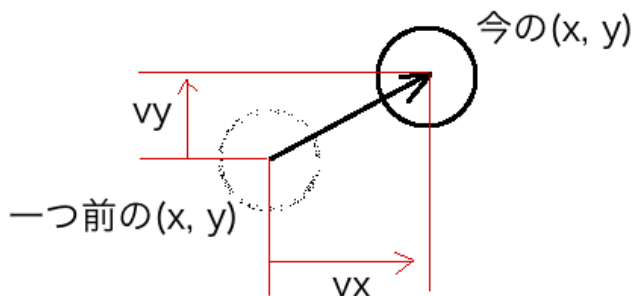
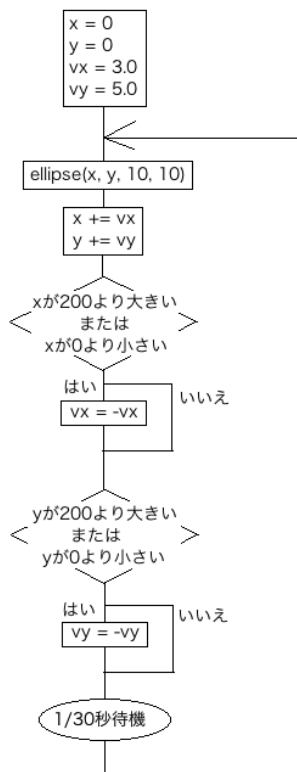
(1) のプログラムを改良して、下の端で跳ね返るようにしたい。前のページでは $mode$ という変数を使って進む向きを記憶したが、今回は vy という変数でそれが記憶されているので、もう少し簡単にかける。次の図のような条件分岐を入れてみよう。これは「速度」の正負を変えていることから、進む方向を180度反転することにつながる。さらに横方向にも一定速度で動くようにすると、右図のようなボールが弾むアニメーションが描ける。



(3) 上下左右跳ね返り

この vy という「 y 方向の速度を表す変数」を使う方法を x 方向にも適用して、上下左右すべての端で円が跳ね返るようにしよう。

ちなみに $y > 200$ または $y < 0$ という条件は「 $(y > 200) \ || \ (y < 0)$ 」と書く。

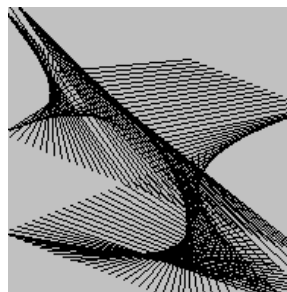


(4) 2 個のボール

2 個のボールが跳ね返るようにしよう。

変数が二倍に増えるのでとても面倒くさいが、コピー貼付けを駆使してがんばろう。

これを二つの円ではなく、2つの点を結ぶ直線にすると、右のようなちょっとカッコいいアニメーションが描ける。



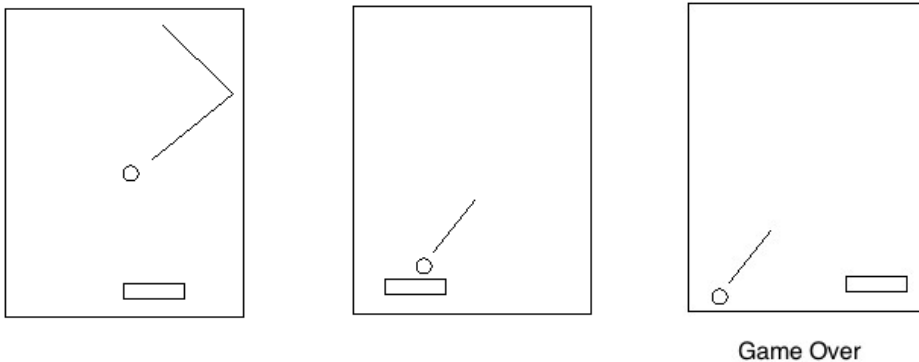
3-7. 簡単なゲーム

ここまで学んできたことを使って、簡単なゲームを作ろう。

(1) テニスゲーム

マウスで下のバーを動かして、下に来たボールを打ち返す。

ボールは右、左、上端では跳ね返ってくる。



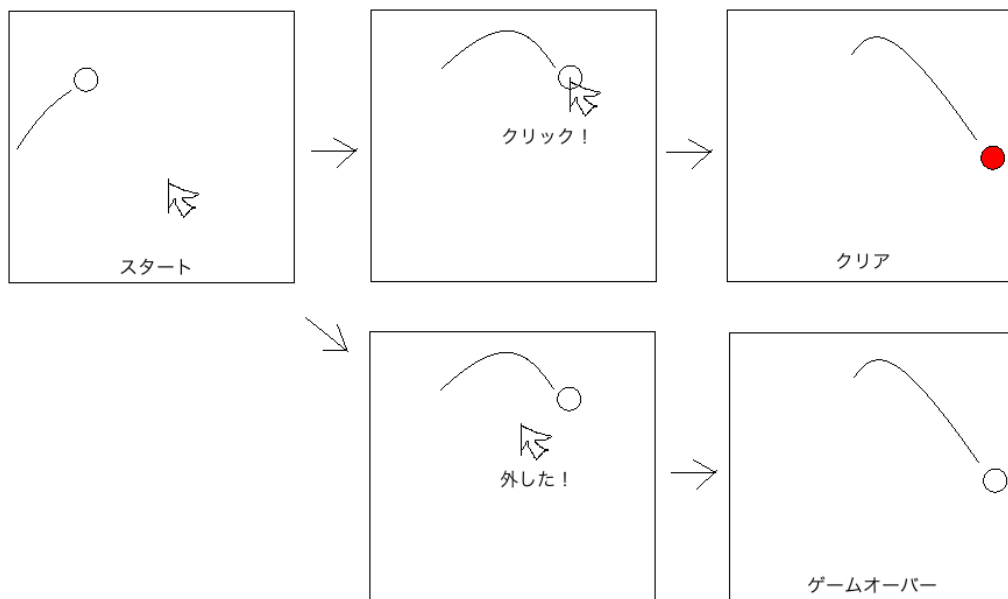
ボールを打ち返せなかったらゲームオーバー。

敵を作ってみたり、ボールを増やしてみたり、打ち返す位置によって跳ね返り方を変えてみたりすると面白い。



(2) 射撃

左から飛んでくる的をクリックできたら勝ち。



8 ページでやったように、円を下方方向に加速すると、カーブを描く。

マウスをクリックした時に、それが円の中かどうか、は円の中心からマウスまでの距離が、円の半径以下かどうかを調べれば良い。二点 $(x1, y1)$ と $(x2, y2)$ 間の距離は、次の命令で計ることができる。

命令

`dist(x1, y1, x2, y2)`

戻り値：小数

点 $(x1, y1)$ から $(x2, y2)$ までの距離を返す

この命令は `random` 命令と同じように、これ自体が小数の変数のように使うことができる。だから、「 $(x1, y1)$ と $(x2, y2)$ の距離が 10 以下」という if 文は、

```
if(dist(x1, y1, x2, y2) < 10){ ... }
```

のように書く。

的の飛ぶ方向や速度をランダムにしたり、的が画面外に出たらまた左から飛んでくるようにしたり、工夫してみよう。

入門編 Part4. 配列

大量の変数をまとめて扱う、「配列」という技術を学ぶ。

1. アニメーションと for 文
2. 配列の使い方
3. 配列と for 文
4. 総合練習
5. 二次元の配列
6. 二次元配列の練習

4-1. アニメーションと for 文

Part2 と Part3 の復習でアニメーションと for 文を組み合わせたプログラムを作る。

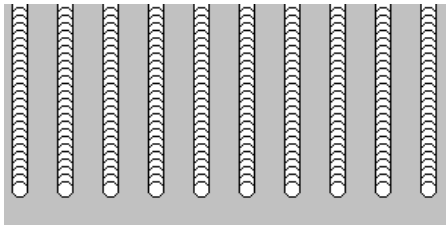
(1) for 文の復習

10 個横に並んだ円を描こう。



(2) アニメーションとの組み合わせ

今の 10 個の円を全部一斉に下向きに動かそう。draw ブロックの中に (1) の for 文を入れる。下向きに動かすためには、円の上からの距離を表す変数を用意して円をその位置に描くことにし、それを毎ステップ加算する。



(3) 残像と軌跡

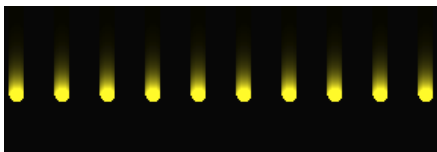
`background(255, 255, 255);`を書き加えて、(2) の残像を消そう。
次に、その `background` 命令の代わりに、

```
fill(255, 255, 255, 10);  
rect(0, 0, width, height);
```

の 2 行を書き加えよう。どうなるだろうか？

この命令は、半透明の白い四角形を画面全体を覆うように描くという意味になるから、ちょっとずつ前の色が薄くなっていき、軌跡を残すことができる。

`fill` の色を変えて次のようにしてみよう。

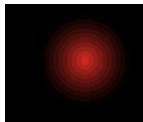


(4) for 文とアニメーション

Part3 で作った円が跳ね返るアニメーションがあったが、あの円を多重円にしてみよう。



これを内側の円ほど明るく、外側ほど暗く描くと、次のような光ってる玉が描ける。



さて、今の(3)で10個の円が一斉に動いたが、これらをばらばらに動かすことはできないだろうか？そのためには10個の円の縦横の位置を記憶しておくための変数20個が必要になりそうだ。

```
float x0, y0, x1, y1, x2, y2, x3, y3 ...
```

と頑張って20個書くこともできるが、「配列」を使うとたくさんの変数をまとめて作ることができる。それでは次のページに行ってみよう。

4-2. 配列の使い方

配列を使うことで、たくさんの変数をまとめて作ることが出来る。こうして作られた変数たちは、同じ名前を持ち、番号によって区別されるという悲しい運命にある…。

配列

(1) 宣言 `int[] hoge;`

(2) 確保 `hoge = new int[100];`

(3) 代入 `hoge[3] = 10; hoge[0] = 5;`

(4) 使用 `println(hoge[99]);`

配列名

配列に含まれる変数の数

「i番目の変数」を使いたかったら、配列名の後に[i]と書く。ただしiは0から始まることに注意。100個の変数を持った配列なら、[0]から[99]まで使うことができる

いままでの変数と違うのは、四角いカッコ [] を使うという点と、(2)の確保の命令が必要になったという点である。上のようにint[]で宣言した配列は、それに含まれる変数は全てint型（整数が入る）になる。

変数が試験管なら、配列は「試験管立て」である。(1)の宣言で試験管立てを持ってきて、(2)の確保で、試験管立てに指定した本数の空の試験管が確保される。あとは「どの試験管立ての何番目の試験管」というふうに指定して、配列の変数を使うことになる。

(1) 次のプログラムを書こう。

```
int[] hoge;
hoge = new int[3];
hoge[0] = 100;
hoge[1] = hoge[0]*2;
hoge[2] = hoge[0] + hoge[1];
println(hoge[0]);
println(hoge[1]);
println(hoge[2]);
```

実行する前に、最後の3つのprintlnで何が出力されるか考えよう。

配列といっても、それぞれの要素hoge[0], hoge[1], hoge[2]は別々の変数として考えればいいということが分かるだろう。

(2) このプログラムの配列hogeは、今要素数3で確保されているが、要素数を5に増やそう。次に、hoge[3]には5を、次にhoge[4]にはhoge[3]の二乗を代入して、最後にそれらをprintlnで出力させよ。5と25が出力できれば良い。

(3) 次のプログラムを書こう。コピー&貼り付けを使うと簡単に書けるだろう。

```
float[] x, y;           (1) 宣言

void setup(){
  size(300, 300);       (2) 確保
  x = new float[2];
  y = new float[2];
  x[0] = random(0, width); (3) 代入
  y[0] = random(0, height);
  x[1] = random(0, width);
  y[1] = random(0, height);
}

void draw(){
  ellipse(x[0], y[0], 10, 10); (4) 使用
  ellipse(x[1], y[1], 10, 10);
  x[0] += 3;
  y[0] += 1;
  x[1] += -4;
  y[1] += 1;
}
```

2つの円が、実行する度に変わるランダムな場所から、右下と左上に飛んで行ったら成功だ。

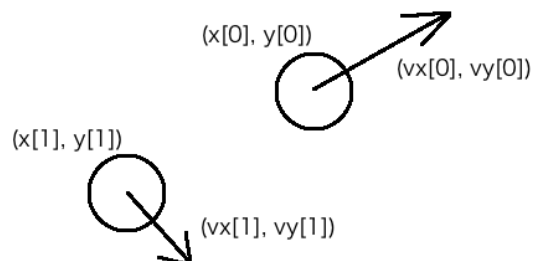
x, y は二つの要素から成る配列である。このプログラムのように、宣言は外側の領域で、配列の確保と初期値の代入は setup ブロック内で、そして draw ブロックで配列を使う、というのが普通だ。

(4) 前のプログラムでは0番目の x, y は毎ステップ(3, 1)ずつ変化し、1番目の方は(-4, 1)と変化していた。この値は固定されているので何度実行しても円が同じ方向に飛んでいくが、ここも、円のスタート地点と同じように、実行する度に方向が変わるようにしたい。

Part3 で vx, vy という円の速度を表す変数を使ったことを思い出そう。これらも配列にすることで、2つの円がランダムな方向に飛んでいくように改良しよう。

- (i) 新しい配列 vx, vy を宣言する
- (ii) vx, vy を要素数2で確保する
- (iii) vx[0], vy[0], vx[1], vy[1]にそれぞれ-1 から 1 までの乱数を代入する
- (iv) x[0] += 3;を x[0] += vx[0];に変える。ほかの3つも同様

それぞれを書く位置は、今ある配列 x, y の書き方を参考にしよう。



4-3. 配列と for 文

配列がその真価を発揮するのは、for 文と組み合わせたときだ。
配列の[]の中には数字だけでなく変数も入れるので、int 型の変数 i があつたとき、
hoge[i]
と書くと、hoge という名前の配列の、i 番目の変数を得ることができる。この i を
for 文で 0 から 99 まで変えれば、まとめて 100 個の変数を操作できるのである。

(1) 次のプログラムを書こう

```
int[] a;  
a = new int[10];  
  
for(int i=0; i<10; i++){  
    a[i] = i;  
}  
for(int i=0; i<10; i++){  
    println(a[i]);  
}
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 と表示されるはずだ。

4~6 行目の for 文で配列 a に、a[i]なら i を代入し、7~9 行目の for 文で a[0]から a[9]まで 1 つずつ出力している。

(2) このプログラムの 6 行目と 7 行目の間に新しい for ブロックを付け加えて、
a[0]から a[5]までを全て二倍せよ。出力結果は、

0, 2, 4, 6, 8, 10, 6, 7, 8, 9
となるはずだ。

(3) 新しいプログラムを作る。次の手順でプログラムを書こう。

- (i) float 型の配列 x を宣言する。
- (ii) 配列 x を要素数を 100 で確保する。
- (iii) for 文を使って x[0]から x[99]までに、ランダムな小数 (0 から 10 までとしよう) を代入する。
- (iv) 変数と for 文を使って、x[0]から x[99]までの平均 (合計を 100 で割ったもの) を求め、printlnを使って出力する。5 に近い数になるだろうか？
- (v) 次に、一番大きい数を求めて出力せよ。10 に近い数になるだろうか？

(4) 55 ページの (4) で作ったプログラムを for 文を使ったものに改良したい。

(i) 宣言はそのまま

(ii) 確保するところを、x, y, vx, vy すべて要素数 100 にする。

(iii) 代入は for 文を使って、全ての要素に代入する。

for 文の中身の例

```
x[i] = random(0, width);
```

```
y[i] = random(0, height);
```

```
vx[i] = random(-1, 1);
```

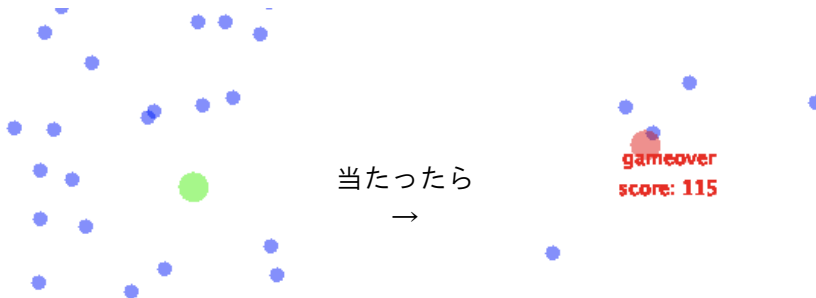
```
vy[i] = random(-1, 1);
```

(iv) 使用も for 文を使って、円を 100 個書いた後、位置を更新する。

100 個の円がランダムな位置から、ランダムな方向に飛んで行ったら成功だ。

これによって、たくさんの物体が画面内を動きまわるようなプログラムが書けるようになった。これはシューティングゲームなど、いろんなゲームの基礎になる。

(5) 避けるだけのシューティングゲーム（ちょっと大変かも）



ランダムに飛んでくる弾から避け続ける。自機（上の図の緑）はマウスで操作する。

円と円とがぶつかっているかどうかは、中心間の距離が半径の和以下となってるかどうかで判定できる。 (x_0, y_0) から (x_1, y_1) までの距離は $\text{dist}(x_0, y_0, x_1, y_1)$ で得られる。

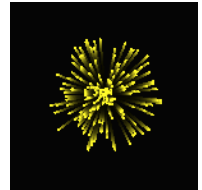
最初は弾に当たったら自機の色を変えることにしよう。

次にゲームオーバーになるようにして、さらに余裕があったらスコアを表示したり、自分を狙ってくる弾を作ったりするといい。

4-4. 練習

(1) 花火 1

画面中央から色のついた大量の小さな円が周囲に広がって進む。
円の数 は 200 くらいあるといい。
きれいに円形に広がるようにするのが難しい。

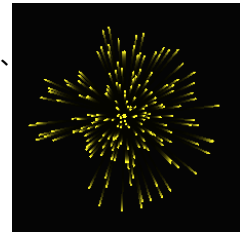


速度 v , 角度 t で移動する物体の vx , vy (x 方向と y 方向速度) は、

$$vx[i] = v \cdot \cos(t);$$

$$vy[i] = v \cdot \sin(t);$$

と書ける。 v を小さめのランダムな数、 t を 0 から 2π までのランダムな数にすると、右の図のようにきれいに広げられる。



(2) 花火 2

マウスをクリックしたら、その地点から広がるようにして、一定時間経ったら消えるようにしよう。

もう一つ配列 a を作って、それを使って、円の表示／非表示を切り替えられるようにする。

$a[i]$ が 100 未満で円を表示、100 以上で非表示とするなら、

setup の中: $a[i] = 100$; 最初は非表示にしたいから 100 以上の数を。

draw の中: $a[i]++$; 毎回 1 を足して、時間をカウントする。

mousePressed の中: $a[i] = 0$; マウスをクリックされたら表示モードに。

のようにする。さらに、if 文で $a[i]$ の値によって ellipse を実行するかどうか分岐させる。

(3) 花火 3

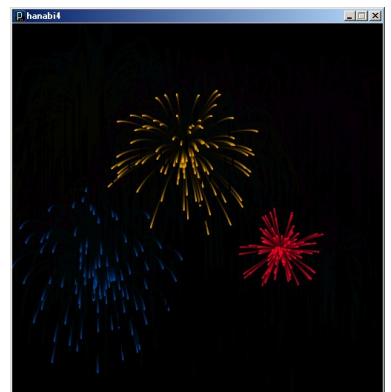
マウスをクリックした数だけ、カーソルの位置に花火が出てくるようにしよう。配列の要素数を 20000 くらいにして、クリックする度に、200 個ずつ表示モードにしていく。できたら次の 3 つの改造もしたい。

- ・ 毎回色をランダムに (色を記録する配列が必要になる)。

- ・ 空気抵抗で速度が減速するようにする。

draw の中で $vx[i]$, $vy[i]$ をちょっとずつ減らすといい。

- ・ 重力によってちょっと落ちるようにするとリアル。



(5) 並び替え

右のプログラムの 4~6 行目は、a[0]から a[99] までにランダムな小数を代入している。

空欄の部分に a の中身を小さい順に並び替えるプログラムを書き足せ。

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

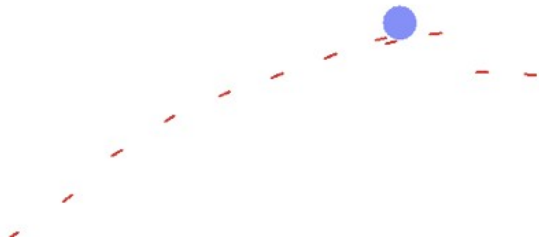
```
for(int i=0; i<100; i++){  
    a[i] = random(0, 100);  
}
```



```
for(int i=0; i<100; i++){  
    println(a[i]);  
}
```

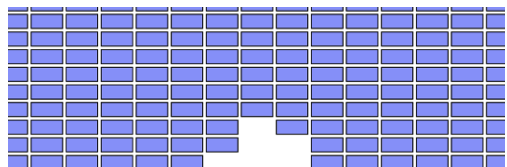
(6) シューティング 2

- ・ 右の丸が敵
- ・ 敵は適当に動く
- ・ 弾を配列で記録
- ・ マウスで撃つ角度を調整



(7) ブロック崩し (難)

- ・ ボール (白い玉) をバー (下の緑の長方形) で跳ね返してブロックに当てる
- ・ 当たったブロックは消える
- ・ ボールを落とさず全部消したら勝ち



4-5. 二次元配列

二次元配列

- (1) 宣言 `int[] a;`
- (2) 確保 `a = new int[10][10];`
- (3) 代入 `a[3][4] = 10;`
- (4) 使用 `println(a[0][0]);`

二次元配列は、今までの配列の添字を一つ追加したものである。配列の配列（`a[10]`が10個並んだ配列）と考えてもいいし、表のようなもの（`a[i][j]`は*i*, *j*の要素を表す）と考えても良い。添字が2つあるので二重ループとの相性がいい。ちなみに[]を追加していくことでもっと多次元の配列を作ることできる。

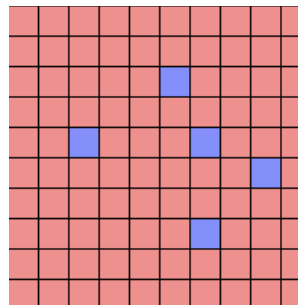
(1) 次のプログラムを書こう。

```
int[][] a;

void setup(){
  size(200, 200);
  a = new int[10][10];
  for(int i=0; i<10; i++){
    for(int j=0; j<10; j++){
      a[i][j] = 0;
    }
  }
}

void draw(){
  background(255, 255, 255);
  for(int i=0; i<10; i++){
    for(int j=0; j<10; j++){
      if(a[i][j]==0){
        fill(255, 0, 0, 100);
      }else{
        fill(0, 0, 255, 100);
      }
      rect(i*20, j*20, 20, 20);
    }
  }
}

void mousePressed(){
  a[mouseX/20][mouseY/20] = 1;
}
```



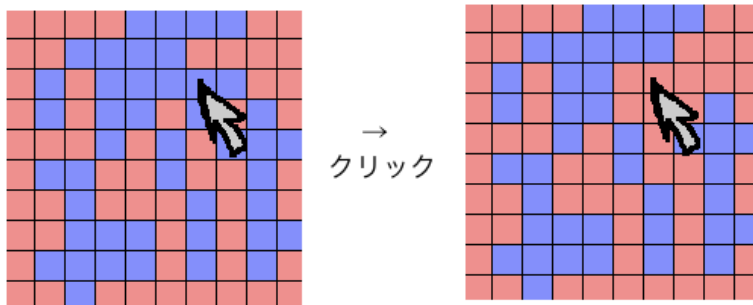
```
a[0][0] a[1][0] a[2][0]
a[0][1] a[1][1] a[2][1] ...
a[0][2] a[1][2] a[2][2]
a[0][3] a[1][3] a[2][3]
...           ...
```

このプログラムを実行すると、右上のようなマス目が表示されて、クリックするとクリックしたマスが青色になる。ここで、各マスと配列の要素 `a[i][j]` の対応関係は右下の図のようになっている。

※表や行列で `a[i][j]` と言ったときは、*i* を縦方向、*j* を横方向（*i* 行 *j* 列）と考えるのが普通だが、今回は画面上の位置(*x*, *y*)が、*x* が横、*y* が縦を表しているため、それに合わせて一つ目の添字を横方向、二つ目を縦にとった。

(2) これを改良して、青になったマスをもう一回クリックしたら赤に戻るよう
にしよう。

(3) さらに次のようなゲームにしよう。



クリックしたところの上下左右が反転。最初はランダムな赤青の配置から始めて、
全部同じ色に揃えられたら勝ち。(10*10 は揃えるのが非常に難しいので 4*4 と
かでやってみよう)

あるマス $a[i][j]$ の右隣のマスは $a[i+1][j]$

左 $a[i-1][j]$

上 $a[i][j-1]$

下 $a[i][j+1]$

盤面の端っこに気をつけよう (例えば右上角のマスは、上隣と右隣がない)。

4-6. 二次元配列の練習

二次元配列は次のような場面でよく使われる。

- ・ ゲームのマップのような二次元空間の情報を表すとき。

山森草道草	32010
森草草道草	20010
草道道道草	01110
草草草道草	00010

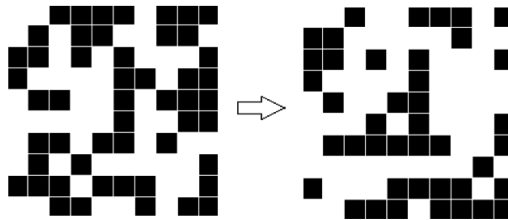
- ・ 対応関係を表として表すとき

(リーグ戦の勝敗表や、駅間の運賃表、人間同士の相性表など)

- ・ ゲームやパズルの盤面を表すとき。

(将棋や囲碁のようなマス目で戦うゲームや、数独のようなパズル)

(1) ライフゲーム



ライフゲームは、生物の繁殖と絶滅のシミュレーションである。

黒マスが生物がいることを、白マスがいないことを表す。

1ステップごとに、次の規則で生物が生まれたり消えたりする。

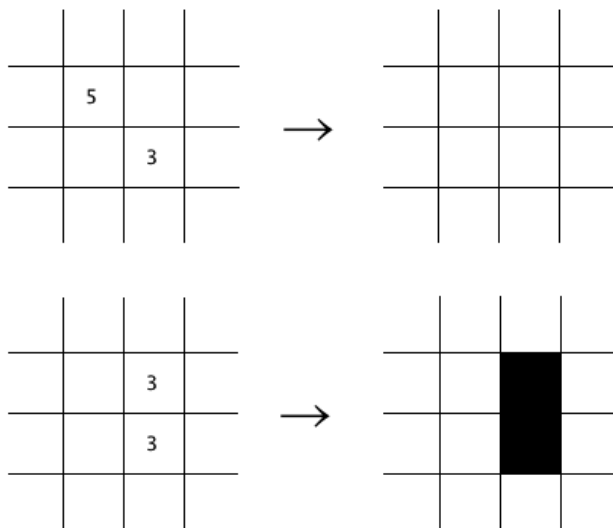
① すでに生物がいるマス（黒マス）について、そのマスの上下左右斜めに 2 匹か 3 匹他の生物がいたら、次のステップではそのマスは黒のままで、その他の場合は過疎すぎるか過密すぎてそのマスの生物は死滅し、白になる。

② 生物がいないマス（白マス）について、そのマスの上下左右斜めに 3 匹他の生物がいたら、次のステップではそのマスは黒になり、その他の場合は白のままである

いろいろと面白いパターンがあるので調べてみよう。

(2) 神経衰弱

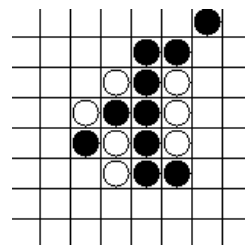
表の中に同じ数字が2個ずつ隠れている。
クリックすると数字が出る。
二枚クリックして、両者が同じなら、その二枚を取れる。違ってたらその二枚は再び隠れる。
全部のカードを早く取ったら勝ち。



(3) オセロ盤

人と対戦できるオセロ盤を作る。次の順に実装していこう。

- (i) 白黒が交互におけるようにする
- (ii) 挟まれた相手の石がひっくり返るようにする
- (iii) なにもひっくり返せないところに石は置けない
- (iv) パスができるようにする（何もひっくり返せないときのため）



(4) 落ち物ゲーム

テトリスやぷよのような落ち物ゲームも、二次元配列を使って作ることができる。

- (i) ブロックを作る処理
 - 1. 作るゲームによるが、テトリスなら4マスでできる図形全て、ぷよなら色違いの二つのマス
- (ii) ブロックを落とす処理
 - 1. 落下中のブロックを下に一マス動かすだけ。
 - 2. 落下中のブロックと、落下後のブロックを区別しておく
- (iii) ブロックを消す処理
 - 1. これもゲームによる。

