計算機プログラミング I (5)

配列の活用

2004 年 11 月 8 日　増原 英彦

1　配列を使わないプログラムから配列を使うプログラムへ

配列を活用した複雑なプログラムをいきなり作るのは簡単でない。ここでは、そのようなプログラムを作るための 1 つの考え方として、

- まず、配列を使わずにプログラムを作ってみる
- そして、作ったプログラムの中にある変数を配列で置き換えてみる

方法を紹介する。

例としてモーフィング (morphing) を行うプログラムを作ることを考える。モーフィングとは、人間の顔を虎の顔に連続的に変化させるアニメーションのように、ある形から別の形へ連続的に変化する図形を表示することである。ここでは、簡単のために円が星形へ 4 段階に変化する場合を考え、アニメーションではなく変化する図形を並べて描くことにする。

1.1　まず配列を使わずにプログラムを作る

プログラムは 4 匹のタートルを使って、各タートルが 1 つの図形を少しずつ並行して描くことにする。両端のタートルは円と星形を描き、間の 2 つのタートルは両端のタートルの位置をもとに、その内分点へ移動し続けることになる。

この程度のものであれば、配列を使わずともプログラムを作ることができる (図 1)。

16 行目からの繰り返しが、各図形を N 回に分けて描く。その内側では、

- (円) m_0 は正 N 角形を描く (17, 18 行目)
- (星) m_3 は 5 つの辺を順に描くが、各辺を $N/5$ 回に分割して描く (20–22 行目)
- (中間) $m_i (1 \leq i \leq 2)$ は、m_0 と m_3 を $i : (3 - i)$ に内分する点に移動する (24–29 行目)

練習 5-1: 配列を使わないモーフィング　図 1 を入力して実行してみよ。星形はどうやって N 回に分けて描かれているのだろうか?

1.2　次にどの変数が配列になるかを考える

プログラム中に沢山の変数が規則的に使われている場合、それは配列によって置き換えることができる可能性が高い。例えば、10–13 行目は `m0` から `m3` までの 4 つの変数を規則的に作っている。このような変数は配列によって置き換えることができる可能性が高い。

また、プログラムの中の「個数」や「回数」を変更することを考えてみて、そのときに変数を追加しなければいけないとすれば、そのような変数は配列によって置き換えることができる可能性が高い。例えば、このプログラムでは、4 匹のタートルによって図形を描いていたが、これを 10 匹のタートルによって描くように変更する場合には、どのように変更すべきだろうか? この場合もやはり `m4`, `m5`, ... `m3` という変数を用意することになるだろう。

以上のような考察から、`m0` から `m3` までを配列として表わすことがよいことが分かる。

```
1  public class MorphWithoutArray {
2      public static void main(String[] args){
3
4          int width = 145;                        // タートルの間隔
5          int steps = 30;                         // 図形の分割数
6
7          TurtleFrame f = new TurtleFrame(600,300);
8
9          //タートル 0 から 3 を等間隔な位置に生成して 画面に表示
10         Turtle m0 = new Turtle(width*0+10,150,0); f.add(m0);
11         Turtle m1 = new Turtle(width*1+10,150,0); f.add(m1);
12         Turtle m2 = new Turtle(width*2+10,150,0); f.add(m2);
13         Turtle m3 = new Turtle(width*3+10,150,0); f.add(m3);
14
15         // steps 回の繰り返しで図形を描く
16         for(int j = 1; j <= steps; j++) {
17             m0.fd(15);                          // m0 に円の一部を
18             m0.rt(360/steps);                   // 描かせる
19
20             m3.fd(24);                          // m3 に星の一部を
21             if (j % (steps/5) == 0)             // 描かせる。steps/5 回に
22                 m3.rt(2*360/5);                 // 一度回転させる
23
24             int x0 = m0.getX(), x3 = m3.getX(); // 両端のタートルの
25             int y0 = m0.getY(), y3 = m3.getY(); // 座標を調べて
26
27             //中間のタートルを内分点に移動させる
28             m1.moveTo((x3-x0)*1/3+x0, (y3-y0)*1/3+y0, 0);
29             m2.moveTo((x3-x0)*2/3+x0, (y3-y0)*2/3+y0, 0);
30         }
31
32     }
33 }
```

図 1: 配列を使わずに Morphing を行うプログラム

練習 5-2: 配列の候補　上では m0 から m3 までを配列とすることに決めたが、m1, m2 だけを配列にする方法も考えらる。そのようにした場合に、「中間の図形」を増やす手間は変わるだろうか?

1.3　変数を配列に置き換える

m0 から m3 を配列で置き換えることにする。10–13 行目で

```
Turtle m0 = new Turtle(...
Turtle m1 = new Turtle(...
Turtle m2 = new Turtle(...
Turtle m3 = new Turtle(...
```

4 つの Turtle 型の変数を宣言しているので、これを次のように置き換える。

```
Turtle[ ] hm = new Turtle[4];
hm[0] = new Turtle(...
hm[1] = new Turtle(...
hm[2] = new Turtle(...
hm[3] = new Turtle(...
```

最初の行が、Turtle オブジェクトを 4 つ格納できる配列 hm を宣言している。この配列 hm を使うことで、いままで変数 m0 を使っていたものを hm[0] に置き換えることができる。

最後に、繰り返し行われている命令を for 文などによる繰り返しに置き換える。上の部分であれば、

```
hm[0] = new Turtle(width*0+10,150,0);  f.add(hm[0]);
hm[1] = new Turtle(width*1+10,150,0);  f.add(hm[1]);
hm[2] = new Turtle(width*2+10,150,0);  f.add(hm[2]);
hm[3] = new Turtle(width*3+10,150,0);  f.add(hm[3]);
```

のようになっていたが、これはよく見ると、タートルの番号 (0〜3) を除いて全く同じ形をしているので、

```
for (int i = 0; i < 4; i++) {
    hm[i] = new Turtle(width*i+10,150,0);  f.add(hm[i]);
}
```

のように変更することができる。

練習 5-3: 配列によるモーフィング　図 1 の他の部分も変更して、配列と繰り返しを使うプログラムを完成させよ。

練習 5-4: タートル数の変更　10 匹のタートルでモーフィングを行うように上のプログラムを変更してみよ。配列を使わない場合と比べて、簡単にできるようになっただろうか? 第 2 回資料で示した「キーボードから数値を受け取る機能」を利用して、キーボードから指定された数のタートルによってモーフィングを行うプログラムを作ることはできるだろうか?

練習 5-5: 図形の外挿　プログラムを拡張して、外分点を移動するようなタートルを作り図形を描かせてみよ。

練習 5-6: 4 つの図形のモーフィング　4 つの図形のモーフィングを行うプログラムを作ってみよ。

練習 5-7: 手描きの図のモーフィング　後で説明する手描きの図の説明を参考に、手描きの図を入力し、それをモーフィングさせるプログラムを作ってみよ。

2　記憶力ゲーム

記憶力ゲームとは、何匹かのタートルが回転する順番を覚え、その順にタートルをクリックするゲームである。ここでは 4 匹のタートルが 5 度回転するとする。

プログラムとしては、「4 匹のタートル」のための配列と、「問題」つまり、i 回目に回転するタートルの番号のための配列の 2 つが必要となる。前者は `Turtle` 型の値をしまう配列であり、後者は `int` 型の値をしまう配列であることに注意。

また、このゲームでは、マウスでクリックされた場所を調べる必要がある。ここでは配布プログラムに含まれている `ClickableTurtleFrame` というクラスを、`TurtleFrame` のかわりに用いる。(ClickableTurtleFrame の使い方は、配布プログラムの ClickableTFDemo.java が参考になる。)

プログラムの概略は下のようになる。

```
//クリック可能な画面を作る
ClickableTurtleFrame f = new ClickableTurtleFrame();

大きさ 4 の Turtle 型の配列 hm を宣言し、
タートルを間隔 100 で生成して hm[0]〜hm[3] にしまう

大きさ 5 の int 型の配列 problem を宣言する

// 問題を表示する
for (int i = 0; i < 5; i++) {
   int j = (int)(Math.random()*5); // 問題 (0 以上 4 以下の乱数) を作る
   配列 problem の i 番目に問題 (つまり j) を覚える
   j 番目のタートルを 360 度回転させる
}

// 答え合わせをする
for (int i = 0; i < 5; i++) {
   int x = f.getClickedX(); //画面をクリックしてもらいその X 座標と
   int y = f.getClickedY(); //Y 座標を調べる (実は Y 座標はいらない)
   int answer = x からタートルの番号を逆算;
   if (answer が正しい答えか) {
     answer 番目のタートルを 360 度回転させる;//正解
   } else {
     answer 番目のタートルを 200 歩後退させる; //不正解
   }
}
```

練習 5-8: 記憶力ゲーム　記憶力ゲームを完成させよ。また、タートルの数や、問題の長さを簡単に変更できるようにしてみよ。

練習 5-9: 配列の効用　記憶力ゲームを配列を使わずに作ることは可能だろうか? 可能だとすればどの部分が最も面倒になるだろうか?

3　滑らかな曲線

マウスでクリックした場所を覚えておき、それらの間を滑らかにつないだ絵を描くことでお絵かきをするプログラムを考える。

そのような曲線を描く方法には様々なものがあるが、その 1 つに Béizer 曲線というものがある。これは、3 点 p_0, p_1, p_2 をつなぐ曲線を

$$p(t) = (1-t)^2 p_0 + 2(1-t)t p_1 + t^2 p_2 \qquad (0 \le t \le 1)$$

という方程式で与えるものである。($p_0, p_1, p_2, p(t)$ は全てベクトルである。) これをもとに N 本の直線で近似した曲線を描くには、

```
タートルを p0 へ移動
for (i を 0 から N まで変化) {
    double t = ((double)i)/N;     //i を実数に変換して N で割る
    t, p0, p1, p2 をもとに p(t) の座標を計算
    タートルをそこへ移動
}
```

のようにすればよい。

さらに長い曲線を描くには、

- N 個の座標を入力する ($p_0, p_1, \ldots, p_{N-1}$ とする)
- j を 0 から $N-3$ まで変化させて
  p_j, p_{j+1}, p_{j+2} をつなぐ曲線を上の方法で描く

のようにすればよい。

問題は、N 個の座標を入力する方法であるが、これは

```
for (i を 0 から N まで変化) {
   int x = f.getClickedX();
   int y = f.getClickedY();
   Turtle を (x,y) に作り、i 番目として覚える
}
```

ようにすればできる。

練習 5-10: 滑らかな曲線　プログラムを完成させて、滑らかな曲線を描いてみよ。

練習 5-11: 可変個の点による図形　決められた数の点でなく、画面の左上端をクリックすると点の入力を終了させるようにすると、好きな数の点から成る図形を描くことができる。そのようにプログラムを変更してみよ。

4　手描きの図形によるデザイン

マウスでクリックした場所を覚えておき、それを縮小した図形を繰り返し描くと紋章のような図案ができる。

このようなプログラムは、最初に基本的には N 個の点を覚えておき、次に覚えておいた通りに移動させることを繰り返す。その際、点の位置そのものでなく、角度と距離を覚えると縮小・回転した図形を簡単に描くことができる。

図形を覚える部分は、次のようになる:

```
Turtle m = タートルを作る;
for (i を 0 から N まで変化) {
    int x = f.getClickedX();
    int y = f.getClickedY();
    m の現在位置から (x,y) への角度と距離を計算
    i 番目の角度と距離を覚える
    m を回転して移動
}
```

図案を描く部分は次のようになる:

```
for (正 M 角形を描く繰り返し) {
    m を正 M 角形の j 番目の頂点に移動
    for (i を 0 から N まで変化) {
       m を i 番目の角度だけ回転
       m を i 番目の距離の半分だけ移動
    }
}
```

図形を覚えるところで「m の現在位置・方向から (x, y) への角度と距離を計算」するとあるが、これは少し考察が必要である。いま、m の現在の位置を (x_0, y_0) とすると、(x_0, y_0) から (x, y) への距離 l は

$$l = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

である。一方、(x_0, y_0) から (x, y) 方向の角度 θ は (x 軸方向を 0 とすると)

$$\frac{y - y_0}{x - x_0} = \tan\theta$$

であるから、tan の逆関数 arctan によって

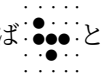
$$\arctan\frac{y - y_0}{x - x_0} = \theta$$

と求められる[1]。これによって求めた角度と、m の現在の向き (Y 軸方向が 0 で、角度であることに注意) との差をとれば、m から見た向きが求められる。

[1]ただしこれは $x - x_0$ が正のときであり、零のときや負のときは特別扱いが必要であることに注意せよ。因みに、Java 言語には `double Math.atan2(double,double)` という便利なクラスメソッドがある。`Math.atan2(y,x)` は原点から (x, y) への角を計算してくれる便利なメソッドである。また、Java 言語の三角関数での角はラジアンであることに気を付けよ。

5　細胞状自動機械 (cellular automata)

Conway のライフゲームとは、格子状の区画に置かれた細胞の配置から、一定の規則に従って細胞を増殖・死滅させてその変化を見るものである。単純な規則とごく少数の細胞の初期配置から出発して、非常に複雑な変化をすることが知られている。

時刻 0 の細胞の配置は与め決められているものとする。例えば  という配置 (黒丸が細胞の存在を表わす) は有名である。

時刻 t ($t > 0$) の細胞の配置は、時刻 $(t-1)$ の配置で決まる。時刻 t において区画 (x, y) に細胞があるかないかは、時刻 $(t-1)$ における同じ場所の細胞の有無と、時刻 $(t-1)$ におけるその周囲 8 区画の細胞の個数で決めまる:

現在そこに細胞があるか?	有る	有る	無い	無い
現在周囲に何個細胞があるか?	2 または 3	0,1 または 4 以上	3	3 以外
次の時刻にそこはどうなるか?	生き延びる	死滅する	生まれる	生まれない

この規則に従うと、上の例は時刻 1 で  、時刻 2 で  のようになる (★が新しく生まれた細胞を、○が死滅した細胞を表わす)。

さて、これをプログラムにするのは大変なので、少し簡単にした問題を考える。時刻 t においては、$|x-y| = t$ である区画にだけ細胞が存在するような世界を考える。(つまり細胞の生死は周囲の状況に依らない。) これをプログラムにしたものの概略は図 2 のようになる。

このプログラムでは (x, y) に細胞があるかどうかを、二次元配列 `world` の 要素 `world[x][y]` に `Turtle` オブジェクトがしまわれているかどうかで表わしている。そのために、6 行目で `Turtle` 型の配列 `world` を作っている。(Java 言語でオブジェクト型の配列を作ると、最初は全ての要素が「空」(`null`) になっるいることに注意。)

このようにしておくとゲーム全体の進行は、各時刻ごとに、`world` にしまわれているタートルの有無をもとにして、次の時刻のタートルの生死を決め、それに従って画面の表示を変えてゆけばよい。ここで「次の時刻の様子」と「現在の時刻の様子」を区別するため、次の時刻の様子は `nextWorld` という別の配列に覚えることにする (9 行目)。

ある区画におけるタートルの生死を決めるときには、次の 4 つの可能性[2]がある。

現在はタートルが存在し、次の時刻では無くなる場合: 現在タートルが存在するかどうかは配列 `world` の `x`, `y` 要素が「空 (`null`)」かどうかで調べられる。13 行目はそれに加えてその区画が $|x-y| = t$ を満たしていないことを調べている。

この場合は、タートルを画面から消す必要がある。表示されているタートルは `world` の (x,y) 要素であるので、14 行目のようすればよい。

現在はタートルが存在しておらず、次の時刻に生まれる場合: この場合は新たにタートルを作り、画面に表示する。そして、将来このタートルが死滅したときに消去できるように、`nextWorld` の (x,y) 要素にこのタートルを格納しておく。

現在はタートルが存在し、次の時刻も存在する場合: この場合は画面を変更する必要はないが、「次の世界」にこのタートルが引き継がれるように、`nextWorld` の (x,y) 要素に `world` の (x,y) 要素であったタートルをコピーしておく。

現在はタートルが存在しておらず、次の時刻も存在しない場合: この場合は何もしなくてよい。

このようにして「次の世界」の様子が配列 `nextWorld` の中に求められたら、次の時刻に進むために「現在の世界」(つまり `world`) を `nextWorld` の中身で置き換える必要がある (27 行目)。

[2] この簡単なライフゲームでは 4 つの場合全てが有り得るわけではないが、説明のために全ての場合を挙げている。

```
1  int size = 40;                                          // 「世界」size*size の大きさとする
2  int distance = 10;                                      // タートルを置く間隔
3  TurtleFrame f = new TurtleFrame(size*distance,size*distance);
4  f.mesh =true;                                           // 網目を表示
5
6  「今の世界」の様子をしまう配列 world を作る
7
8  for (int t=0; t<size; t++) {                            // 時刻 t を 0..size まで変化
9    「次の世界」の様子を表す配列 nextWorld を作る
10   for (int x = 0; x < size; x++)                        // 「世界」の各区画に
11     for (int y = 0; y < size; y++) {                    // ついて繰り返す
12       //今の世界の区画 (x,y) に細胞があり、次時刻で死滅する場合
13       if (world[x][y]!=null && |x − y| ≠ t)
14         f.remove(world[x][y]);                          //画面からタートルを消す
15       //今の世界の区画 (x,y) に細胞があり、次時刻に生き延びる場合
16       if (world[x][y]!=null && |x − y| = t)
17         次の世界の区画 (x,y) に world[x][y] をしまう
18       //現在 (x,y) に細胞がなく、次時刻に生まれる場合
19       if (world[x][y]==null && |x − y| = t) {
20         Turtle m = new Turtle(x*distance,y*distance,0);  //(x,y) にタートルを作り
21         f.add(m);                                       //画面に追加して
22         次の世界の区画 (x,y) にしまう
23       }
24     }
25   f.repaint();                                          //追加したタートルを表示するおまじない
26
27   world を nextWorld で置き換える
28
29   try { Thread.sleep(1000); }                           //時間かせぎの
30   catch (InterruptedException e) {}                     //ためのおまじない
31 }
```

図 2: 簡単なライフゲームのプログラム

練習 5-12: 簡単ライフゲーム　図 2 を完成させよ。どのような模様が表示されるか?

練習 5-13: 本格ライフゲーム　図 2 をもとに Conway のライフゲームを完成させよ。そのためには、次のような変更が必要であろう:

- 6 行目の後に、時刻 0 でのタートルの配置を作る
- 13・16・19 行目などの条件のところを変更し、周囲 8 つの区画に存在しているタートルの個数によって生死を決める

練習 5-14: 今の世界と次の世界　プログラムでは今の世界を表わす world とは別に、次の世界を表わす nextWorld を用意して (9 行目) タートルの配置を決め、最後に world と入れ替えている (27 行目)。nextWorld を用意せずに world だけで処理しては何がいけないのだろうか?

練習 5-15: 初期配置　時刻 0 におけるタートル (細胞) の配置によってその後の様子は大きく変わってゆくことが知られている。ClickableTurtleFrame を使って、マウスによって時刻 0 の配置を指定できるようせよ。

練習 5-16: 初期配置　ランダムな初期配置を作るようなプログラムにしてみよ。人口 (?) 密度とその後の変化には関係があるだろうか?

練習 5-17: ルールの変更　周囲の細胞の数によって生死を決めるルールを変更するとどうなるか?