数理的思考とはどんなものか

北里大学　理学部物理　十河　清

1　数理的思考とは？

…哲学は、眼のまえにたえず開かれているこのもっとも巨大な書 [すなわち宇宙] のなかに、書かれているのです。しかし、まずその言語を理解し、そこに書かれている文字を解読することを学ばないかぎり、理解できません。その書は数学の言語で書かれており、その文字は三角形・円その他の幾何学的図形であって、これらの手段がなければ、人間の力ではそのことばを理解できないのです。…（G. ガリレイ）

「自然の法則」が存在するというのはけっして自然なことではなく、ましてや人間がその自然の法則を発見できるというのは、まったく不自然なことなのである。（E.P. ウィグナー）

数理的思考 ４つの原理

ものごとを数理的に考えるとはどういうことであるか。以下に例を挙げながらその特徴のいくつかを分析してみましょう。

1.「類別（るいべつ、classification）」の原理

世の中には似たものがたくさんあります。異なる２つのものを見るとき、ある性質のみに着目してその他の違いを無視すれば、それらを「同じもの」とみなすことができる、ということがよくあります。このときわれわれは「分類・類別」をやっているのです。

例えば初等幾何学では最初に三角形や円に関する諸性質を学びますが、「なぜ三角形や円ばかりをしつこくやるのだろう？」と疑問に思ったことはありませんか。幾何学は図形の諸性質を調べる学問です。そのうちの多角形（三角形はそのひとつ）は「辺の数」で分類できますが（三角形は辺の数＝３）、ここで重要なことは「一般の多角形は三角形の集

まりに分割できる」という事実です。ですから三角形について詳しく調べておけば、その結果は一般の多角形に応用できるわけです。

これが分類・類別の効用のひとつに他なりません：例えばデカルトの『方法序説』ではその指導原理の２番目に「分割」が挙げられています（１番目は「懐疑＝明らかなこと以外はすべて疑え」でした)。自然科学においては、物質や現象をそれを構成している要素に分割して考える、ということが極めて有効です。例えば物理なら「原子や素粒子」、化学では「反応の素過程」、生物では「遺伝子」などがこれにあたります。系統的な分類ができれば、「記憶すべきことが少なくて済む」という副次的効能も生じます：これを「思惟経済」と呼ぶことがあります。

分類・類別のもうひとつの効用は、類別によって同類とみなされるものに対しては「類比（るいひ、analogy）が使える」ということです。例えば、力学における「バネの振動」の問題と電気回路における「LC 回路」の問題は、記号の違いを別にすれば数学的にはどちらも同じ微分方程式に帰着します。

そして「方程式が同じなら答えも同じ」（R.P. ファインマン）というわけで、どちらも単振動になることがわかります。こうしてどちらか片方を理解していれば、他方もそれとの類推で容易にわかるわけです。これらを理解することは、結局その背後にある「振動の微分方程式」の数理を理解することに他なりません。数理的思考に慣れてくると、振動の方程式を見れば「何かが振動している様子」が目に浮かぶようになります（その「何か」が何であるかは、ひとによって違いますが)。

「これはあれと一緒」とか「あれはこれと似たようなもの」といった捉えかたは、ものごとを理解する第一歩といえます。湯川秀樹先生はお弟子さんに生物学への関心を促して、「生物（いきもの）も物（もの）や」とおっしゃられたといいます。筆者は（覚えることが多くて）高校生物が苦手でしたが、これを聞いて生物学への感じ方が少し変わりました(相変わらず苦手ですが)。

最後に、何はともあれ人類が分類・類別せずにはいられない最大の理由は「それがおもしろいから」に尽きるでしょう。動物園に行くと飽きずに時を過ごせますが、それは種の多様性に対する好奇心が尽きないから、というのに似ています。皆さんもおおいに分類・類別して「違いのわかる人間」になりましょう。

2.「相対性（そうたいせい、relativity)」の原理

ここで相対性というのは、もちろんガリレイ・アインシュタインの相対性原理が念頭にあるのですが、もう少し一般的なことを想定しています。H. ワイルという数学者は（自身も相対論や量子力学の発展に寄与しましたが)、相対性理論の背後にある論理を一般化して、おおよそ次のような概念を導入しました。それは「座標化（ざひょうか、coordinatization)」と「枠独立性（わくどくりつせい、frameindependency)」というものです。

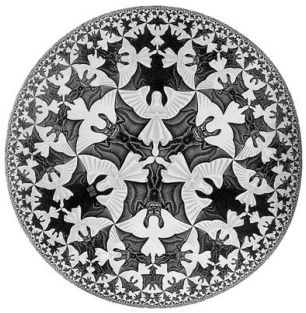
相対論によれば、運動を記述するには位置を指定する「座標」の導入が必要ですが、一方で運動法則はその際の座標の取り方＝「枠」（直角座標・極座標や運動座標など）とは無関係に成り立つ性質のことに他なりません：運動する粒子は自分がどんな座標系にいるかなど知ったことではない。そんなわけで、いわば座標は「屋根に登るための梯子」のようなもので、考える際の手段・方便にすぎないのです：屋根に上がるにはそれが必要だが、上がってしまえば不要になる。そして屋根に登ればすばらしい眺望が開ける、それこそが本質だというのです。ワイルのこの考えは彼の「プラトン主義的数学観」によるものですが、禅の哲学にも通じるところがあります（屋根の比喩は禅の公案から借用しました)。

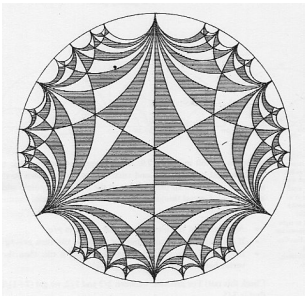
ともあれこのような考え方は、相対論にとどまらず物理や数学のいろいろな理論のなかに、手を変え品を換えて現れていることがわかります。例を挙げれば、(１) 特殊および一般相対性理論、(２) 量子力学の変換理論＝ハイゼンベルグ描像とシュレーディンガー描像の等価性、(３) 局所ゲージ不変性の原理＝素粒子の標準理論から重力まで、(４) ユークリッド・非ユークリッド・射影の各幾何学を統合する F. クラインの構想、などなど枚挙に暇がありません。

われわれは、ともすれば座標化の手続きの煩雑さ（極座標への変換はその典型！）にまぎれて、それには依らない本質（この場合は回転不変性）を見失いがちですが、そういうことのないようにしたいものです。

3.「双対性（そうついせい、duality)」の原理

これはあまり聞き馴染みのない言葉だと思いますが、これも物理・数学に頻繁に登場する普遍的概念のひとつです。２つの異なるものが、互いに「組み」になって関係しているという事態が生じることがよくあります。少々回りくどいのですが、ひとつの例を挙げて示しましょう。下左の図は 1960 年に M.C. エッシャーが描いた「天国と地獄」という、円を相似な図形で埋め尽くした大変おもしろい絵です。このように「部分が全体と相似になっている」現象を一般に「自己相似性（じこそうじせい、self similarity)」といい、別名「フラクタル」と呼んでいます（B. マンデルブロの命名)。





一方で右の図は 1926 年刊行の F. クラインによる『19 世紀数学の発展』にある図ですが、両者をよく較べれば本質的には同じ図といってよいことがわかるでしょう：もちろん数学者ならぬ芸術家エッシャーは右図の存在は知りませんでした！クラインの図は、19 世紀に数学者 F. ガウスが発見した「モジュラー関数」というものに関係して現れます。モジュラー関数 $f(z)$ は、複素数の z に対して $f(-/z) = f(z)$ という関数等式を満たすのですが、これは引数がゼロの近所と無限大の近所で、関数の振る舞いが互いに同じであることを意味します。このように「$z \Leftrightarrow -1/z$ の組み」に関してモジュラー関数は「双対（そうつい、dual)」になっているのです。この「両極端が互いに等しい」という現象は、今の場合「共形不変性（きょうけいふへんせい、conformal invariance)」と呼ばれる性質の結果として生じています。

ところで、自己相似性の特徴のひとつは「基準単位が存在しない」ということです。どこまでいってもきりがないので「単位＝素」がないのです。逆にこういう場合には、不思議なことに「自己相似性」従って「モジュラー関数」がよく現れます。例えば、２次相転

移の臨界点であるとか、素粒子の弦理論などにおいては、「基準となる長さが存在しない」ことから共形不変性が成立し、その結果モジュラー関数が登場します。これらは最近の数理物理における成果のひとつですが、エッシャーの絵が遠くこんなところまで関係しているとは、想像を超えた不思議ではないでしょうか。

４.「算法（アルゴリズム、algorithm)」の重要性

アルゴリズム（算法、さんぽう）とは近世アラビアの数学者アル・フワリズミの名前に由来しています。彼が１次・２次方程式の解法（代入、約分など）について著述したことから、一般に問題を有限回の手続きで解く方法＝算法のことをアルゴリズムというのです。ただし、ここで「有限回」というのがちょっとしたミソではあります。

少し脱線しますと、連立方程式 $x+y=5,\ 3x+y=11$ を解くのに、第１式を $x=5-y$、第２式を $y=11-x$ とやり「x を求めるには y が、y を求めるには x が要る」といって頭をかかえた、という笑えない話がありますが、「代入」を忘れたのでは算法になりません。

さて問題に応じていろいろな解法があり得ますが、良いアルゴリズムとは「できるだけ短い手順で、できるだけ正確な解に到達する」方法のことに他なりません。欲をいえば、それが「簡単な手続き」であれば、なおさら好都合ではあります。

計算機（コンピュータ）を利用したアルゴリズムの場合、とくに有用なのはそれが「繰り返し＝再帰（さいき、recursion)」から成っている場合で、そういうことなら計算機は非常に得意なのです。再帰の例としてよく挙げられるのは、階乗の計算や数学的帰納法などですが、他にもおもしろい例が山ほどあります。

「３山くずし」をやってみよう

計算機アルゴリズムの例として、あるゲームを紹介しましょう。これは「３山くずし」といって、じつは「先手必勝」であることが知られているゲームです。３つの山にそれぞれ３、５、７個のカードが置かれていて、これを交互に取り合います。ルールは

(１) １回につき、ひとつの山から１個以上（何個でも）取る。

(２) 最後のカードを取ったほうが負け。

というものです。付録２に Java のソースを載せたので、入力＆実行してみてください。あなたが先手でコンピュータに勝てるでしょうか。

(注) 最後のルール (２) は通常のものとは逆なので、これは「逆形の３山くずし」と呼ばれています（一松信『石取りゲームの数理』)。

プログラム中でどういうアルゴリズムが使われているか（＝必勝法）を知りたい場合は、是非ソースを解読してください。「再帰」が有効に使われていることがわかります。また、計算機に勝つには「２進法の計算」に慣れる必要があることもわかるでしょう。簡単な解説は付録１にあります。

以上は既知のアルゴリズムの一例にすぎませんが、他にも「良いアルゴリズム」が求められている問題は数多くあります。中には有名な難問というのもありますが、実用的な観点からの要求もたくさん残っています。例えば「バイオ・インフォマティクス」の課題は、「DNA 塩基配列パターンからタンパク質への写像、およびその逆写像をいかに効率よく

求めるか」という問題といえます。これは情報理論において、正規表現あるいはオートマトン理論として知られているものの特別な場合に他なりません。もちろん、このように問題を分類したからといって直ちに良い解答が得られるわけではありませんが、背後にある「数理」を知ることは、役に立つことが多いものです。

２つのまったく異なる問題に、同じアルゴリズムが使われるというケースもよくあります。例えば、RSA（Rivest-Shamir-Adleman）という公開鍵暗号には整数論の「オイラーの定理」が有効に使われています。また、いろいろなソリトン方程式が、じつは 19 世紀の数学者に知られていたという事実が続々発見されています。この場合には（忘れられていて）直接の役には立たなかったわけですが、その時代の状況を反省することによって「別の観点が得られる」という効能があり、こうして「ベックルンド変換」というものに新しい光が当てられています。

原理は使わないと身につかない

以上「数理的思考法」の特徴と考えられるものを４つほど、思いつくままに挙げてみました。他にも書き残した話題がいくつかあるのですが、冗長になってもいけませんからここまでにしておきましょう。わかりにくいところが多いのではないかと心配しますが、具体的な例をいろいろ考えて、実際に手を使って試してみないことには、こういう問題は本当にはわからないとしたものなのです。皆さんには、想像をたくましくして、自分でいろいろな例を考えて見ることを期待しています。

2　付録：「３山くずし」について

１.「３山くずし」の必勝法

「必勝法」を理解するには、２進数の数学を学ぶ必要があります。

まず記法をつぎのように定めます。すなわち、３山の状態を 10 進数 $a,\ b,\ c$ を用いて $(a,\ b,\ c)$ とあらわします。たとえば、初期状態は $a = 3,\ b = 5,\ c = 7$ ですから $(3,5,7)$ とあらわされます。

準備：　２進法表示と排他的論理和

10 進数と２進数の相互変換:

任意の正または 0 の 10 進整数 x をつぎのように２進数表示して、各係数を求める（ただし x が 7 以下のとき）

$$x = x_0 + x_1 * 2 + x_2 * 4$$

ここで $x_0,\ x_1,\ x_2$ は 0 または 1 である。したがって、初期状態では、$a = 3$ $(a_0 = 1,\ a_1 = 1,\ a_2 = 0)$, $b = 5$ $(b_0 = 1,\ b_1 = 0,\ b_2 = 1)$, $c = 7$ $(c_0 = 1,\ c_1 = 1,\ c_2 = 1)$ となります。

排他的論理和:

つぎに、係数の各項ごとに「排他的論理和」とよばれる加法 $+$ を定義する。

ルール：　0+0=0,　1+0=1,　0+1=1,　1+1=0

３つ以上の和については、加法の結合則：$(x+y)+z=x+(y+z)$ が成り立つものとします。たとえば初期状態では

$3+5+7=(1+1\cdot 2+0\cdot 4)+(1+0\cdot 2+1\cdot 4)+(1+1\cdot 2+1\cdot 4)=1+0\cdot 2+0\cdot 4=1$

となっています。次項で説明するように、現在の状態に対して毎回この足し算を実行して、最善手を決めるのです。

必勝法

良形と悪形

状態 $(a,\ b,\ c)$ が「良形」であるとは、つぎの場合をいいます。

(1)　$a,\ b,\ c$ が全て 1 か 0 のとき：　$a+b+c=1$ (2)　それ以外の場合：$a+b+c=0$

したがって、初期状態は 3+5+7=1 でしたから「悪形」となっています。

(1) の場合、良形は (1,0,0), (0,1,0), (0,0,1), (1,1,1) の４通りに限られます。

必勝法

３山くずしのルール「ある 1 つの山から 1 つ以上を取る」の場合

「次の手によって、悪形は良形にも悪形にもなるが、良形は必ず悪形になる」

という法則が成り立ちます（証明を考えてみてください！）。したがって、

初期状態が悪形のときは「先手必勝」

なのです。すなわち、先手側は「状態を良形にして相手に渡す」というのが必勝法となります。

具体的には $a,\ b,\ c$ のうちで最大のものに対して、たとえば今それが c とすれば、上記 (1)(2) の場合に応じて、(1) の場合は「$a+b+c'=1$」(2) の場合は「$a+b+c'=0$」となるような c' に変えてやればよいのです。

最善手

初期状態が (3,5,7) のような悪形の場合は「先手必勝」ですから、後手側としては先手の悪手に期待するしかありません。

すなわち先手の誤り「悪形→悪形」を捉えて、攻守の逆転を図るのです。そのためには「マギレ」を生じるような手、心理学的には「できるだけ少ない数の石を取る」というのが最善手となるでしょう。

ごらんのように、上記の必勝法は任意の初期状態に対しても、また山の数が３つ以上でもそのまま使えます。友達や家族を相手に「N 山くずし」を楽しんでみてください。

２．Java のソースプログラム

```
// 対戦型３山くずし（ニム）
// 2004/08/11 作成
/*
<applet code="nim.class" width="780" height="500">
</applet>
*/
import java.applet.Applet;
```

```
import java.awt.*;
import java.awt.event.*;

public class nim extends Applet implements ItemListener, ActionListener{
  //
  protected final int A=0, B=1, C=2;
  protected final int YES=0, NO=1;
  protected final int YOU=0, CMP=1;
  static int yama, num;//yama=(A, B, C); num=取る石の数
  static int[] residue=new int[3];//残る石の数（＝現在の状態）
  static int flag, turn;//フラッグ、手番

  int width, height;
  Image  img, domino;
  Graphics g;
  Choice ch1=new Choice();//山
  Choice ch2=new Choice();//個数
  Button ok, next, retry;
  Label row1, row2, row3, row4, row5;
  Label lbA, lbB, lbC;
  Label lb1, lb2, lb3;
  TextField dialog;
  String empty, good, warn, win, lose;//メッセージ

  public void init(){
  //全画面
    Dimension dim=getSize();
    width=dim.width; height=dim.height;
    g=getGraphics();
    setLayout(null);
  //説明文
    g.setColor(Color.black);
    row1=new Label("これは「３山くずし」を、コンピュータとの対戦形式で実行する
プログラムです。");
    row1.setBounds(50,50,700,20);
    add(row1);
    row2=new Label("ゲームのルールは次の２つ：");
    row2.setBounds(50,70,700,20);
    add(row2);
    row3=new Label("（１）交互に３つの山Ａ、Ｂ、Ｃのひとつから、１つ以上のカー
ドを取る（全部もＯＫ）。");
```

```
    row3.setBounds(70,90,700,20);
    add(row3);
    row4=new Label("（2）最後のカードを取ったほうが「負け」となる。");
    row4.setBounds(70,110,700,20);
    add(row4);
    row5=new Label("このゲームは先手必勝法のあることが知られています。先手でコ
ンピュータに勝てるか試してください");
    row5.setBounds(50,130,700,20);
    add(row5);
  //盤面
    domino=getImage(getCodeBase(),"domino.jpg");
    lbA=new Label("A"); lbB=new Label("B"); lbC=new Label("C");
    g.setColor(Color.black);
    lbA.setBounds(150,190,20,20);
    add(lbA);
    lbB.setBounds(150,270,20,20);
    add(lbB);
    lbC.setBounds(150,350,20,20);
    add(lbC);
  //操作盤
    ch1.addItem("A"); ch1.addItem("B"); ch1.addItem("C");
    ch1.select("A"); yama=A;
    ch1.setBounds(120,420,40,20); add(ch1);
    ch1.addItemListener(this);
    ch2.addItem("1"); ch2.addItem("2"); ch2.addItem("3");
    ch2.addItem("4"); ch2.addItem("5"); ch2.addItem("6");
    ch2.addItem("7");
    ch2.select("1"); num=1;
    ch2.setBounds(210,420,40,20); add(ch2);
    ch2.addItemListener(this);
    lb1=new Label("あなたは");
    lb1.setBounds(50,420,60,20);
    add(lb1);
    lb2=new Label("から");
    lb2.setBounds(170,420,30,20);
    add(lb2);
    lb3=new Label("個を取ります：←　選んで、押してください　→");
    lb3.setBounds(260,420,270,20);
    add(lb3);
    ok=new Button("決定"); ok.setBounds(540,420,60,20);
    ok.addActionListener(this); add(ok);
```

```
  next=new Button("押すと計算機が応手します"); next.setBounds(50,450,200,20);
  next.addActionListener(this); add(next);
//ダイアログ
  dialog=new TextField();
  dialog.setBounds(610,420,120,20); add(dialog);
  empty=new String("");
  warn=new String("反則です！");
  good=new String("良い手でした！");
  win=new String("あなたの勝ち！");
  lose=new String("あなたの負け！");

  retry=new Button("再挑戦");
  retry.setBounds(540,450,60,20);
  retry.addActionListener(this);
  add(retry);

  InitState();
}

public void InitState(){
  residue[0]=3; residue[1]=5; residue[2]=7;
  ch1.select("A"); yama=A;
  ch2.select("1"); num=1;
  Show(empty); turn=YOU;
  repaint();
}

public void itemStateChanged(ItemEvent ie){
  if(ie.getSource()==ch1){
    String st=ch1.getSelectedItem();
    if(st.equals("A")){ yama=A;
      }else if(st.equals("B")){ yama=B;
      }else if(st.equals("C")){ yama=C;
    }
  }
  if(ie.getSource()==ch2){
    String strg=ch2.getSelectedItem();
      num=Integer.parseInt(strg);
  }
}
```

```
public void actionPerformed(ActionEvent ae){
  if(ae.getSource()==ok){
    flag=IsLegal(num);
    if((flag==YES)&&(turn==YOU)){
      residue[yama]-=num;
      turn=CMP; Show(empty);
      repaint();//描画
    }else{
        Show(warn);//反則
        return;
    }
  }
  if(ae.getSource()==next){
    if(turn!=CMP){
      Show(warn);//反則
      return;
    }else{
      ThinkMove();//応手を考えて
      turn=YOU;
      repaint();//再描画
    }
  }
  if(ae.getSource()==retry){
    InitState();//初期状態に戻し
    repaint();//再描画
  }
}

public int IsLegal(int NUMBER){
  if(residue[yama]<NUMBER){
    return NO;
  }else{
    return YES;
  }
}

public void Show(String message){
  dialog.setText(message);
}

public void ThinkMove(){
```

```
  int m, l, n, ran1, ran2;
  int a, b, c, a0, a1, a2, b0, b1, b2, c0, c1, c2;
  int dummy, dummy1, dummy2, dummy3, sum;

  a=residue[A]; b=residue[B]; c=residue[C];
  m=max(a,b,c); n=min(a,b,c); l=3-(m+n);//res[n]<res[l]<res[m]

  if((residue[n]==0)&&(residue[l]==0)){
    if(residue[m]>1){
      residue[m]=1; Show(lose); return;
    }else if(residue[m]==1){
      Show(win); return;
    }else{
      residue[m]=1; Show(lose); return;
    }
}else if((residue[n]==0)&&(residue[l]==1)){
    residue[m]=0; Show(lose); return;
  }else if((residue[n]==0)&&(residue[l]>1)){
    if(residue[l]==residue[m]){
      Show(win); return;
    }else{
      residue[m]=residue[l]; Show(lose); return;
    }
  }else if((a<=1)&&(b<=1)&&(c<=1)){
    sum=a+b+c;
    if(sum==0){ Show(lose); return;
    }else if(sum==1){ residue[m]=0; Show(win); return;
    }else if(sum==2){ residue[m]=0; Show(lose); return;
    }else if(sum==3){ residue[m]=0; Show(win); return;
    }
  }else{
    dummy=residue[n]%4; a2=(residue[n]-dummy)/4;
    a0=dummy%2; a1=(dummy-a0)/2;
    dummy=residue[l]%4; b2=(residue[l]-dummy)/4;
    b0=dummy%2; b1=(dummy-b0)/2;
    dummy=residue[m]%4; c2=(residue[m]-dummy)/4;
    c0=dummy%2; c1=(dummy-c0)/2;
    dummy1=xor(a2, b2)*4+xor(a1, b1)*2+xor(a0, b0);
    dummy2=xor(a2, c2)*4+xor(a1, c1)*2+xor(a0, c0);
    dummy3=xor(c2, b2)*4+xor(c1, b1)*2+xor(c0, b0);
    if(dummy1<residue[m]){
```

```
      if(residue[l]==residue[n]){
        if(residue[l]==1){
          residue[m]=1; Show(lose); return;
        }else{
          residue[m]=0; Show(lose); return;
        }
      }else{
        residue[m]=dummy1; return;
      }
    }else if(dummy2<residue[l]){
      residue[l]=dummy2; return;
    }else if(dummy3<residue[n]){
      residue[n]=dummy3; return;
    }else{//悪形なので、乱数を使ってそれらしく対応する
      Show(good);
      ran1=(int)(3*Math.random());
      if(ran1>2){
        ran2=1+(int)(residue[m]*Math.random());
        residue[m]-=ran2; return;
      }else if(ran1>1){
        ran2=1+(int)(residue[l]*Math.random());
        residue[l]-=ran2; return;
      }else{
        ran2=1+(int)(residue[n]*Math.random());
        residue[n]-=ran2; return;
      }
    }
  }
}

public int max(int n1, int n2, int n3){
  int dummy;
  if(n1>n2){ dummy=A;
    if(n1>n3){ dummy=A; return dummy;
    }else{ dummy=C; return dummy;
    }
  }else if(n2>n3){ dummy=B; return dummy;
  }else{dummy=C; return dummy;
  }
}
```

```
  public int min(int n1, int n2, int n3){
    int dummy;
    if(n1>n2){ dummy=B;
      if(n2>n3){ dummy=C; return dummy;
      }else{ dummy=B; return dummy;
      }
    }else if(n1>n3){ dummy=C; return dummy;
    }else{dummy=A; return dummy;
    }
  }

  public int xor(int j, int k){
    int dummy=j+k;
    switch(dummy){
      case 0:{
        dummy=0; break;
      }
      case 1:{
        dummy=1; break;
      }
      case 2:{
        dummy=0; break;
      }
    }
    return dummy;
  }

  public void paint(Graphics g){
    g.setColor(Color.white);
    g.fillRect(0, 160, 780, 240);

    for(int i=0; i<3; i++){
      for(int j=0; j<residue[i]; j++){
        g.drawImage(domino, 200+j*30, 190+i*80, 20, 20, this);
      }
    }
  }
}
```