ユーザインタフェース＃4 — インタラクションの新しい工夫

久野　靖*

2012.10.30

1 Reading

1.1 次回の紹介論文

- 岡野さん — Joey Scarr, Andy Cockburn, Carl Gutwin, Andrea Bunt, Improving Command Selection with CommandMaps, CHI 2012, pp. 257-266, 2012.
- 木谷さん — Anne Marie Piper, NadirWeibel, James D. Hollan, TAP & PLAY: An End-User Toolkit for Authoring Interactive Pen and Paper Language Activities, CHI 2012, pp.149-158, 2012.
- 海老原さん — Chad C. Tossell, Philip Kortum, Ahmad Rahmati, Clayton Shepard, Lin Zhong, Characterizing Web Use on Smartphones, CHI 2012, pp. 2769-2778, 2012.
- 矢田さん — Haoqi Zhang, Edith Law, Robert C. Miller, Krzysztof Z. Gajos, David C. Parkes, Eric Horvitz, Human Computation Tasks with Global Constraints, CHI 2012, pp. 217-226, 2012. または WesleyWillett, Jeffrey Heer, Maneesh Agrawala, Strategies for Crowdsourcing Social Data Analysis, CHI 2012, pp. 227-236, 2012.
- 小和瀬さん — Lester Holtzblatt, Jill L. Drury, Daniel Weiss, Laurie E. Damianos, Donna L. Cuomo, Evaluation of the Uses and Benefits of a Social Business Platform, CHI 2012, pp. 721-736.

論文紹介に入らない人は必ず紙ないしそれ相当 (PDF) のレポートを提出してください。レポートの提出期限は「11/20(火) 一杯」とさせて頂きます。

1.2 ヒューメイン・インタフェース 第3章「意味, モード, モノトニー, そして神話」

この本は非常に面白い読みものですが、モードについてとくに面白い話題を提供してくれています。

- チェックボタン、トグルボタンはよくない — ラジオボタンの方が常によい
- モードの定義 — 「1 つのジェスチャに対する反応が違えばモードがある」
- モードに「注意の所在」が無いと間違えるが、注意を無理矢理引き付けるのもよくない
- カスタマイズもモードなのでよくない — 「カスタマイズ機能の学習と操作に費される時間は、実際の作業のうちでもっとも無駄な作業と言える」(そこまで言わなくても…)
- 可変ボタンはよくない/ボタンの数が少ないのがよいとは言えない
- 擬似モード (何かを押している間だけ働く) は OK
- 可変メニューはよくない (最後の 1 つを先頭にコピーとかはよい)

*経営システム科学専攻

- 「擬似モードにない場合はコンテンツの生成」
- 名詞=動詞と動詞=名詞
- アフォーダンス/BART の券売機
- モノトニー「動作 a を起動する方法は g しかない」←メニュー+ショートカットはダメということに
- 初心者と熟練者を区別するのはよくない

1.3 ヒューメイン・インタフェース 第 4 章「定量化」

この章は KLM をかなり丁寧に説明しているので、前回の復習 (というか前回はほんとに駆け足でしか説明できませんでした) に良かったと思います。例題も摂氏華氏変換ですし。また、Fitzz の法則 (これも前回やりました) と Hick の法則 (これはやってません) も出て来ました。

2 ユーザインタフェースとフレームワーク

2.1 フレームワークと MVC

フレームワーク (framework) とは、そのままでは「枠組み」という意味の英語ですが、ソフトウェアの文脈でいうと「プログラムの構造を一般化したもの」という意味合いがあります。たとえば、自動車の設計は車種ごとに様々ですが、「車体に車輪がついていて、エンジン→クラッチ→変則器→車軸の順に動力を伝達して駆動」というフレームワークは一緒です。これと同様に、ユーザインタフェースを持つソフトウェアや、単体の GUI 部品についても、「何でも勝手に設計してね」というよりも「このようなフレームワークに沿って作ります」という形にした方が設計も理解しやすく、動作にも統一性が持たせられるわけです。

フレームワークの代表的なものに、Alto の Smalltalk-80 システムで最初に作り出された「MVC(Model-View-Controller) フレームワーク」があります。これは、ユーザインタフェースを持つソフトウェアを次の 3 つの要素に分解して扱うものです。

- Model — インタフェースによって操作されているものの「本質部分」。たとえば 3 次元グラフィクスなら「空間内の物体の形状や色や光源の配置や明るさ」など。
- View — モデルを画面に表示している部分。1 つのモデルに対してビューが複数あってもよいし (例: 立面図、平面図、…)、ビューごとに表示のさせ方が違ってもよい。
- Controller — スライダなど、モデルやビューの見え方に変更を及ぼすための機能を持つ部分。

MVC の 3 要素はそれぞれ互いに依存関係を持ちます。たとえば、コントローラの操作によってモデルが変更を受け、モデルが変化すると、それを表示しているビューの内容が変化する必要があります。このような MVC のフレームワークに基づいて構成することで、さまざまなユーザインタフェースがうまく作れることが分かった、というのが Smalltalk-80 の貢献です。

さて、一般的なインタフェースは MVC もいいのですが、「GUI 部品」のレベルで考えると規模が小さすぎて 3 つには分けにくいです。そこで、Swing では V と C を合わせたものを delegate と呼び、これとモデルとが合わさったものが GUI 部品、という考え方を取っています。モデルと delegate の間のインタフェースは部品ごとに決まっています。そして、とくに複雑な機能を持つ部品では、プログラマが複雑なモデルを提供し、それをもとに GUI 部品が表示と操作を受け持つ、ということができます。具体的には `JList`、`JTable`、`JTree`、`JTextPane`、`JEditorPane` などがそうです。

ただし、これらの部品それぞれについて、いちいちモデルから書かないと使えないというのでは不便なので、「デフォルトの」モデルも用意されています。たとえば、`JTable` のためのデフォルトのモデル `DefaultTableModel` は、縦横のセルに値を保持するようなものです (確かにそれが一般的ではあります)。しかし、自前でモデルを用意することで、これとは違ったものが色々実現できます。

2.2 例題: JTable

上で説明したように、JTable は (ちょうど Excel の画面のように) 縦横にます目が並んだ形の GUI 部品です。そして、その中に表示されるものは TableModel というインタフェースに従うモデルオブジェクトによって定まります。

ここでは、JTable 向けにさまざまなモデルを作るときの土台として使える AbstractTableModel オブジェクトを継承して、簡単なモデルを作ってみます。作成するクラスで必ず定義しなければならないメソッドは次のものです。

- int getRowCount()、int getColumnCount() — 表の行数、列数を返す。
- boolean isCellEditable(int r, int c) — r 行 c 列が編集可能かどうかの真偽値を返す。
- Object getValueAt(int r, int c) — r 行 c 列に表示すべき内容を返す。
- setValueAt(Object o, int r, int c) — r 行 c 列に設定すべき内容を渡される。

ここで「内容」としては Object をやりとりしますが、実質としては文字列がやりとりされると思っておけばいいでしょう。

では、「等差数列を表示する」という機能を持ったモデルを使った例を見てみましょう。このプログラムでは、BorderLayout を使って中央に JTable をスクロール機能つきではめるだけで、あとの処理はすべてモデルの中で行っています。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;

public class Sample35 extends JPanel {
  public Sample35() {
    setLayout(new BorderLayout());
    add(new JScrollPane(new JTable(new MyTableModel())));
  }
  public static void main(String[] args) {
    JFrame app = new JFrame();
    app.add(new Sample35());
    app.setPreferredSize(new Dimension(300, 400));
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
  }

  class MyTableModel extends AbstractTableModel {
    double init = 1.0, step = 0.1;
    public int getRowCount() { return 25; }
    public int getColumnCount() { return 3; }
    public boolean isCellEditable(int r, int c) {
      if(r == 1 && c == 1) return true;
      return false;
    }
    public Object getValueAt(int r, int c) {
      if(c == 0) {
```

```
        return new Integer(r);
      } else if(c == 1) {
        return String.format("%.5f", 1.0 + r*step);
      } else {
        return "?";
      }
    }
    public void setValueAt(Object o, int r, int c) {
      if(r == 1 && c == 1) {
        step = Double.parseDouble(o.toString()) - 1.0;
      }
      fireTableDataChanged();
    }
  }
}
```

モデルクラスは上述のように AbstractTableModel を土台としていて、その概要は次の通りです。

- 初期値では初校は 1、階差は 0.1。
- 表の行数は 25、カラム数は 3。
- 1 行 1 列のみ編集可能。
- 0 カラムには行番号、1 カラムには等差数列、2 カラムには「?」を表示。
- 1 行 1 列に値がセットされたら、実数値に変換して初校を引くことで新しい階差を設定。

なお、fireTableDataChanged() は「内容が変わったから表示し直して」という意味のメソッドです。

演習 1 例題をそのまま動かしなさい。動いたら、次のような変更を行ってみなさい。

a. カラム 2 にカラム 1 の 2 倍の値が表示されるようにする。
b. さらに、カラム 4 まで用意して 3 倍、4 倍も表示する。
c. カラム 1 行 0 も編集可能にして、初項を入力できるようにする。
d. カラム 1 の他の行も編集可能にして、それに応じて階差を計算する。

3 インタラクションの新しい工夫

3.1 従来の部品だと…

ここまに出て来た GUI 部品では、操作時のさまざまな動作は予め部品側で作成されていて、プログラムを作るといってもそれを使うだけである。しかしそれでは、インタラクション上の新しい工夫は起こらない。今日ではスマホやタブレットなどの小さい画面の機器が普及してきて、それらの限られた画面で使いやすいインタラクションを行うため、さまざまな工夫がなされるようになってきている。

ここではその片鱗を体験してもらうため、「細長いリストの上をナビゲーションする」という汎用的なモデルを題材として考えます。リストは縦の長さが 100 万ピクセル (窓の高さの 2500 倍) という巨大な大きさで、中身としては連番の数字が描かれています。この中で、指定した番号にすばやく移動する効率を考えよう、ということです。

まず標準的な GUI 部費である JScrollPane を使った版を示します。

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

public class Sample40 extends JPanel {
  JScrollPane scr = new JScrollPane(new MyPanel());

  public Sample40() {
    setLayout(new BorderLayout()); add(scr);
    scr.getViewport().setViewPosition(new Point(0, 500000));
  }
  public static void main(String[] args) {
    JFrame app = new JFrame();
    app.add(new Sample40());
    app.setPreferredSize(new Dimension(300, 400));
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
  }
  static class MyPanel extends JPanel {
    Font fn = new Font("Helvetica", Font.PLAIN, 24);
    public MyPanel() {
      setOpaque(false);
      setDoubleBuffered(false);
      setPreferredSize(new Dimension(280, 1000000));
    }
    public void paintComponent(Graphics g) {
      g.setColor(new Color(120,120,120)); g.setFont(fn);
      Rectangle r = g.getClipBounds();
      int ymin = trunc(r.y, 50), ymax = ceil(r.y+r.height, 50);
      for(int y = ymin; y <= ymax; y += 50) {
        g.drawString(String.format("%d", y/50), 20, y+20);
      }
    }
  }
  private static int trunc(int i, int m) { return i - i % m; }
  private static int ceil(int i, int m) { return i + m - i % m; }
}
```

MyPanel から読みます。このクラスは Jpanel の一種ですが、ただしメモリの必要なダブルバッファリング描画を OFF にした上で、自分のサイズを 280 × 1000000 に設定しています。また、描画のときには 50 ピクセル間隔で並んだ番号を表示します。ちなみに、Graphics の getClipBounds() は「現在見える範囲」を教えてくれるので、その下限から上限の範囲でだけ番号を表示します。このとき、50 ピクセル単位で切捨て/切り上げをするために、補助手続き trunc() と ceil() を定義しています。次に main ですが、これはこれまでやったのと全く同様です。最後に本体ですが、まず内容をスクロールバーつきにさせてくれる部品である JScrpllPane を用意して、MyPanel をはめます。その後コンストラクタでは、部品を中央にはめるようにして、スクロール位置を半分の 500000 に設定します。これで、0〜19999 の数値が並んだだけの窓ができます。

演習 2 このプログラムで「指定した番号が窓内に見えるまでスクロール」するのにどれくらい時間が掛かると思うか予想しなさい。次に、実際に測ってみなさい。また、この方式の「利点」「弱点」をできるだけ多く挙げてみなさい。

演習 3 このプログラムの代替案、つまり標準のスクロールバー「以外の」方法でどのようなスクロール方法があるか考えなさい。とくに、上記のような「巨大な」リストをスクロールして目的位置を表示させる上で「有効な」方法が考えられるとなおよいです。

3.2 普通のドラグと普通でないドラグ

では次に、タブレット PC などのように「画面をポインタでドラグ」する方式でスクロールさせてみましょう。

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

public class Sample41 extends JPanel {
  JViewport view = new JViewport();
  MyPanel panel = new MyPanel();
  int py = 500000, basey, mousex, mousey;

  public Sample41() {
    view.setView(new MyPanel());
    view.setViewPosition(new Point(0, py));
    setLayout(new BorderLayout()); add(view);
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) {
        mousex = evt.getX(); mousey = evt.getY(); basey = py;
      }
    });
    addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent evt) {
        int dy = mousey - evt.getY();
        int acc = 1;
        py = basey + acc*dy;
        view.setViewPosition(new Point(0, py));
      }
    });
  }
  public static void main(String[] args) {
    JFrame app = new JFrame();
    app.add(new Sample41());
    app.setPreferredSize(new Dimension(300, 400));
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
  }
```

```
  static class MyPanel extends JPanel {
    Font fn = new Font("Helvetica", Font.PLAIN, 24);
    public MyPanel() {
      setOpaque(false);
      setDoubleBuffered(false);
      setPreferredSize(new Dimension(280, 1000000));
    }
    public void paintComponent(Graphics g) {
      g.setColor(new Color(120,120,120)); g.setFont(fn);
      Rectangle r = g.getClipBounds();
      int ymin = trunc(r.y, 50), ymax = ceil(r.y+r.height, 50);
      for(int y = ymin; y <= ymax; y += 50) {
        g.drawString(String.format("%d", y/50), 20, y+20);
      }
    }
  }
  private static int trunc(int i, int m) { return i - i % m; }
  private static int ceil(int i, int m) { return i + m - i % m; }
}
```

先の例題と違うのは本体部分だけです。まず、JScrollPane の代わりに内容の位置を制御するだけの部品 JViewport を使い、MyPanel はその中にはめます。そして、現在のスクロール位置 (Y のみ)、ドラグ開始したときのマウス位置、およびドラグ開始したときのスクロール位置 (Y のみ) を変数に覚えるようにします。そして、ドラグ中は Y 位置を「開始時の Y 位置+Y 方向の変移」に常に設定することで、ドラグとともに変移が変化する。

しかしこのまま使ってみるとすぐ分かるが、「1:1 の比率で」ドラグしていたら永遠に目的の場所に到達しない。そこでたとえば「倍率」(変数 acc で設定できるようになっている) を大きくしたらいいとか、倍率を一定ではなくさまざまな工夫で大きくしたらいいとか、考えるわけである。

演習 4 この例題をそのまま動かしなさい。動いたら次に、倍率を「さまざまな工夫で」変更することで、スムーズに「目的の番号まで」移動できるようにしてみなさい。

3.3 「投げる」インタフェース

現在のタブレット製品では、内容をドラグして手を離すと、「そのまましばらく画面が動き続けて止まる」ようになっている。それだとあまり遠くまでは行けないが、たとえば「ずっと動き続けていて、再度マウスボタンを押し下げるとブレーキが掛かる」ようにすれば、高速な移動に役立つかも知れない。また、「動いている途中にシフトキーを押し下げると加速する」などの機能も追加してみた。

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.Timer;

public class Sample42 extends JPanel {
  JViewport view = new JViewport();
  MyPanel panel = new MyPanel();
  boolean down = false, shift = false;
```

```
int py = 500000, basey, mousex, mousey;
double time = 0.0, vy = 0.0;
long baset = System.currentTimeMillis();

public Sample42() {
  view.setView(new MyPanel());
  view.setViewPosition(new Point(0, py));
  setLayout(new BorderLayout()); add(view);
  addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent evt) { shift = evt.isShiftDown(); }
    public void keyReleased(KeyEvent evt) { shift = evt.isShiftDown(); }
  });
  addMouseListener(new MouseAdapter() {
    public void mousePressed(MouseEvent evt) {
      requestFocus(); down = true; time = now();
      mousex = evt.getX(); mousey = evt.getY(); basey = py;
    }
    public void mouseReleased(MouseEvent evt) {
      down = false; time = now();
    }
  });
  addMouseMotionListener(new MouseMotionAdapter() {
    public void mouseDragged(MouseEvent evt) {
      int dy = mousey - evt.getY();
      int acc = 1 + Math.abs(dy) / 5;
      double dt = now()-time; if(dt <= 0.0) { return; }
      double vy1 = dy/dt * acc;
      vy = 0.7*vy + 0.3*vy1;
      py = basey + acc*dy;
      view.setViewPosition(new Point(0, py));
    }
  });
  new Timer(20, new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      if(down) { vy *= 0.9; }
      if(down || Math.abs(vy) < 80.0) { return; }
      double dt = now() - time;
      if(shift) { py += 4*vy*dt; }
      py += vy*dt; time += dt;
      view.setViewPosition(new Point(0, py));
    }
  }).start();
}
public double now() { return 0.001*(System.currentTimeMillis()-baset); }
public static void main(String[] args) {
  JFrame app = new JFrame();
  app.add(new Sample42());
  app.setPreferredSize(new Dimension(300, 400));
```

```
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
  }
  static class MyPanel extends JPanel {
    Font fn = new Font("Helvetica", Font.PLAIN, 24);
    public MyPanel() {
      setOpaque(false);
      setDoubleBuffered(false);
      setPreferredSize(new Dimension(280, 1000000));
    }
    public void paintComponent(Graphics g) {
      g.setColor(new Color(120,120,120)); g.setFont(fn);
      Rectangle r = g.getClipBounds();
      int ymin = trunc(r.y, 50), ymax = ceil(r.y+r.height, 50);
      for(int y = ymin; y <= ymax; y += 50) {
        g.drawString(String.format("%d", y/50), 20, y+20);
      }
    }
  }
  private static int trunc(int i, int m) { return i - i % m; }
  private static int ceil(int i, int m) { return i + m - i % m; }
}
```

演習 5 この例題をそのまま動かしなさい。動いたら次に、動き方を「さまざまな工夫で」変更することで、よりスムーズに「目的の番号まで」移動できるようにしてみなさい。

4 「アイコン投げ」

4.1 アイコン投げの由来とデモ

上の例題では「画面の中」を投げていましたが、もっと自然なのは画面上にある「アイコン」を投げることであるはずです (大きさがちょうどよいし、ドラグしてそのまま離すというのはありがちに思えます)。

もともと、WIMP ユーザインタフェースでさまざまな操作をするときには、メニューでもボタンでも「この場所までポインタを移動する」という操作が伴いますが、この操作の時間は前回やったように Fitzz の法則で制約されています。しかしこの制約は、もしかしたら「動かしたまま離す」という操作では不要なのではないか、というのが私たちのアイデアでした。

あともう 1 つ、ユーザインタフェースの操作は「X を選ぶ」という形で「N 個の選択肢から 1 つを選ぶ」のが普通ですが、ドラグ&ドロップであれば「N 個のものからドラグして、M 個のうちどれかへドロップする」という形で選択できるので、組み合わせの数が一気に多くなります。ですから、ドラグ&ドロップを中心にしたインタフェースであれば多数の命令が少ない操作で行える可能性がある、と考えました。

そして、ドラグ&ドロップも普通に操作していれば Fitzz の法則に制約されますが、「ドラグ対象のアイコンをドロップ対象のアイコンに向けて動かした状態で離す」操作、つまり「投げる」操作を使えばこれが高速にできるのでは、というのがアイデアだったわけです。

百聞は一見にしかず、とりあえず、以前に作ったデモプログラムを動かしてみてください。

```
/u1/kuno/work/x11/throw/demo/demo1
```

/u1/kuno/work/x11/throw/demo/demo2

4.2 例題: ドラグできるアイコン

ここからは、「アイコン投げ」のような動作がどのようにして作れるか、順次例題を追って見ていただきます。最初は、ただドラグできるアイコンが画面に現れるだけです。このプログラムでは DrawSet というオブジェクトがアイコンの集まり (画面上に現れるもの) を管理するので、本体部分からはマウスボタンの押し/離し/ドラグをそのオブジェクトに転送するだけです。DrawSet の側は、DrawObj というインタフェースに従うもの (「画面に表示できるもの」) の集まりを管理するという形で一般的に作られています。そして DraggableIcon がこのインタフェースを実装するオブジェクト (ドラグできるアイコン) を定義します。

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;

public class Sample43 extends JPanel {
  DrawSet set = new DrawSet();
  public Sample43() {
    setOpaque(false);
    set.addObj(new DraggableIcon("A", 100, 100));
    set.addObj(new DraggableIcon("B", 120, 120));
    set.addObj(new DraggableIcon("C", 140, 140));
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) {
        set.mousePressed(evt.getX(), evt.getY()); repaint();
      }
      public void mouseReleased(MouseEvent evt) {
        set.mouseReleased(evt.getX(), evt.getY()); repaint();
      }
    });
    addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent evt) {
        set.mouseDragged(evt.getX(), evt.getY()); repaint();
      }
    });
  }
  public void paintComponent(Graphics g) { set.draw(g); }
  public static void main(String[] args) {
    JFrame app = new JFrame();
    app.add(new Sample43());
    app.setPreferredSize(new Dimension(400, 400));
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
  }
```

```
  interface DrawObj {
    public void draw(Graphics g);
    public boolean hit(double x, double y);
    public void moveTo(double x, double y);
    public void setHighlight(boolean b);
  }
  class DrawSet {
    DrawObj hit = null;
    ArrayList<DrawObj> a = new ArrayList<DrawObj>();
    public void addObj(DrawObj o) { a.add(o); }
    public void draw(Graphics g) { for(DrawObj o: a) { o.draw(g); } }
    public void mousePressed(double x, double y) {
      hit = null;
      for(int i = a.size()-1; i >= 0; --i) {
        if(a.get(i).hit(x, y)) { hit = a.get(i); break; }
      }
      if(hit != null) hit.setHighlight(true);
    }
    public void mouseReleased(double x, double y) {
      if(hit != null) { hit.setHighlight(false); hit = null; }
    }
    public void mouseDragged(double x, double y) {
      if(hit != null) { hit.moveTo(x, y); }
    }
  }
  class DraggableIcon implements DrawObj {
    boolean highlight = false;
    Font fn = new Font("Helvetica", Font.BOLD, 24);
    String label;
    double xpos, ypos, rad = 25.0;
    Color c1 = Color.yellow, c2 = Color.pink;
    public DraggableIcon(String s, double x, double y) {
      label = s; xpos = x; ypos = y;
    }
    public void draw(Graphics g) {
      int r2 = (int)(rad*2);
      g.setColor(highlight ? c2 : c1);
      g.fillOval((int)(xpos-rad), (int)(ypos-rad), r2, r2);
      g.setColor(Color.black); g.setFont(fn);
      g.drawString(label, (int)(xpos-12), (int)(ypos+8));
    }
    public boolean hit(double x, double y) {
      return (xpos-x)*(xpos-x)+(ypos-y)*(ypos-y) < rad*rad;
    }
    public void moveTo(double x, double y) { xpos = x; ypos = y; }
    public void setHighlight(boolean b) { highlight = b; }
  }
}
```

4.3 例題: アイコンのドラグ&ドロップ

次の例題では、アイコンの種類として「ドロップ先」のアイコンである TargetIcon を増やしています。そして、DrawSet のドラグ中の動作として「ドロップ先アイコンにヒットしたらその色を変える」「ヒットした状態でマウスボタンを離したらドロップさせる」という処理を追加しています。

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;

public class Sample44 extends JPanel {
  DrawSet set = new DrawSet();
  public Sample44() {
    setOpaque(false);
    set.addObj(new TargetIcon("A", 300, 100));
    set.addObj(new TargetIcon("B", 300, 200));
    set.addObj(new TargetIcon("C", 300, 300));
    set.addObj(new DraggableIcon("A", 60, 50));
    set.addObj(new DraggableIcon("C", 60, 100));
    set.addObj(new DraggableIcon("B", 60, 150));
    set.addObj(new DraggableIcon("B", 60, 200));
    set.addObj(new DraggableIcon("A", 60, 250));
    set.addObj(new DraggableIcon("C", 60, 300));
    set.addObj(new DraggableIcon("A", 60, 350));
    set.addObj(new DraggableIcon("B", 60, 400));
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) {
        set.mousePressed(evt.getX(), evt.getY()); repaint();
      }
      public void mouseReleased(MouseEvent evt) {
        set.mouseReleased(evt.getX(), evt.getY()); repaint();
      }
    });
    addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent evt) {
        set.mouseDragged(evt.getX(), evt.getY()); repaint();
      }
    });
  }
  public void paintComponent(Graphics g) { set.draw(g); }
  public static void main(String[] args) {
    JFrame app = new JFrame();
    app.add(new Sample44());
    app.setPreferredSize(new Dimension(400, 400));
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
```

```
}
interface DrawObj {
  public void draw(Graphics g);
  public boolean hit(double x, double y);
  public void moveTo(double x, double y);
  public void setHighlight(boolean b);
}
class DrawSet {
  DrawObj hit = null;
  TargetIcon target = null;
  ArrayList<DrawObj> a = new ArrayList<DrawObj>();
  public void addObj(DrawObj o) { a.add(o); }
  public void draw(Graphics g) { for(DrawObj o: a) { o.draw(g); } }
  public void mousePressed(double x, double y) {
    hit = null;
    for(int i = a.size()-1; i >= 0; --i) {
      if(a.get(i).hit(x, y)) { hit = a.get(i); break; }
    }
    if(hit != null) hit.setHighlight(true);
  }
  public void mouseReleased(double x, double y) {
    if(hit == null) return;
    if(target != null) {
      target.drop(hit); target.setHighlight(false); toBack(hit);
    }
    hit.setHighlight(false); hit = null; target = null;
  }
  public void mouseDragged(double x, double y) {
    if(hit == null) return;
    hit.moveTo(x, y);
    TargetIcon t = null;
    for(int i = a.size()-1; i >= 0; --i)
      if(a.get(i) instanceof TargetIcon &&
         a.get(i) != hit &&
         a.get(i).hit(x, y)) { t = (TargetIcon)a.get(i); break; }
    if(target != null) { target.setHighlight(false); target = null; }
    if(t != null) { t.setHighlight(true); target = t; }
  }
  private void toBack(DrawObj o) { a.remove(o); a.add(0, o); }
  private void toFront(DrawObj o) { a.remove(o); a.add(o); }
}
class DraggableIcon implements DrawObj {
  boolean highlight = false;
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  String label;
  double xpos, ypos, rad = 25.0;
  Color c1 = Color.yellow, c2 = Color.pink;
  public DraggableIcon(String s, double x, double y) {
```

```
      label = s; xpos = x; ypos = y;
    }
    public void draw(Graphics g) {
      int r2 = (int)(rad*2);
      g.setColor(highlight ? c2 : c1);
      g.fillOval((int)(xpos-rad), (int)(ypos-rad), r2, r2);
      g.setColor(Color.black); g.setFont(fn);
      g.drawString(label, (int)(xpos-12), (int)(ypos+8));
    }
    public boolean hit(double x, double y) {
      return (xpos-x)*(xpos-x)+(ypos-y)*(ypos-y) < rad*rad;
    }
    public void moveTo(double x, double y) { xpos = x; ypos = y; }
    public void setHighlight(boolean b) { highlight = b; }
  }
  class TargetIcon extends DraggableIcon {
    int dropcount = 0;
    public TargetIcon(String s, double x, double y) {
      super(s, x, y); c1 = Color.blue; c2 = Color.green;
    }
    public void drop(DrawObj o) {
      ++dropcount;
      o.moveTo(xpos+5*dropcount, ypos+5*dropcount);
    }
  }
}
```

4.4 例題: アイコンを投げる

最終版ではいよいよ、アイコンが投げられて動くようにします。このため、先のスクロールの場合と同様、タイマーを使用しますが、このプログラムでは DrawSet に対して時間の更新メソッドを繰り返し呼び出す形になっています。時間更新処理の中では、現在動いているアイコンがあればそれをさらに動かし、その結果ターゲットにヒットしたらばドロップさせるという処理を行います。投げられるアイコンそのものは、ドラグできるアイコンを拡張して、XY 方向の速度を持つようにし、時間につれてその速度で動くようにしています。

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;

public class Sample45 extends JPanel {
  DrawSet set = new DrawSet();
  public Sample45() {
    setOpaque(false);
    set.addObj(new TargetIcon("A", 300, 100));
    set.addObj(new TargetIcon("B", 300, 200));
```

```
    set.addObj(new TargetIcon("C", 300, 300));
    set.addObj(new ThrowableIcon("A", 60, 50));
    set.addObj(new ThrowableIcon("C", 60, 100));
    set.addObj(new ThrowableIcon("B", 60, 150));
    set.addObj(new ThrowableIcon("B", 60, 200));
    set.addObj(new ThrowableIcon("A", 60, 250));
    set.addObj(new ThrowableIcon("C", 60, 300));
    set.addObj(new ThrowableIcon("A", 60, 350));
    set.addObj(new ThrowableIcon("B", 60, 400));
    addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent evt) {
        set.mousePressed(evt.getX(), evt.getY()); repaint();
      }
      public void mouseReleased(MouseEvent evt) {
        set.mouseReleased(evt.getX(), evt.getY()); repaint();
      }
    });
    addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseDragged(MouseEvent evt) {
        set.mouseDragged(evt.getX(), evt.getY()); repaint();
      }
    });
    new javax.swing.Timer(20, new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        set.updateTime(System.currentTimeMillis()*0.001); repaint();
      }
    }).start();
  }
  public void paintComponent(Graphics g) { set.draw(g); }
  public static void main(String[] args) {
    JFrame app = new JFrame();
    app.add(new Sample45());
    app.setPreferredSize(new Dimension(400, 400));
    app.pack();
    app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    app.setVisible(true);
  }
  interface DrawObj {
    public void draw(Graphics g);
    public boolean hit(double x, double y);
    public void moveTo(double x, double y);
    public void setHighlight(boolean b);
  }
  class DrawSet {
    DrawObj hit = null;
    TargetIcon target = null;
    ArrayList<DrawObj> a = new ArrayList<DrawObj>();
    public void setTime(double t) {
```

```
  for(int i = 0; i < a.size(); ++i)
    if(a.get(i) instanceof ThrowableIcon)
      ((ThrowableIcon)a.get(i)).setTime(t);
}
public void updateTime(double t) {
  for(int i = 0; i < a.size(); ++i)
    if(a.get(i) instanceof ThrowableIcon) {
      ThrowableIcon o = (ThrowableIcon)a.get(i);
      if(!o.isMoving()) continue;
      o.updateTime(t);
      for(int k = a.size()-1; k >= 0; --k)
        if(a.get(k) instanceof TargetIcon &&
           a.get(k).hit(o.getX(), o.getY())) {
          ((TargetIcon)a.get(k)).drop(o);
          o.setTime(t); toBack(o);
          break;
        }
    }
}
public void addObj(DrawObj o) { a.add(o); }
public void draw(Graphics g) { for(DrawObj o: a) { o.draw(g); } }
public void mousePressed(double x, double y) {
  hit = null;
  for(int i = a.size()-1; i >= 0; --i) {
    if(a.get(i).hit(x, y)) { hit = a.get(i); break; }
  }
  if(hit != null) hit.setHighlight(true);
}
public void mouseReleased(double x, double y) {
  if(hit == null) return;
  if(target != null) {
    target.drop(hit); target.setHighlight(false); toBack(hit);
  }
  hit.setHighlight(false); hit = null; target = null;
}
public void mouseDragged(double x, double y) {
  if(hit == null) return;
  if(hit instanceof ThrowableIcon) {
    ((ThrowableIcon)hit).moveTo(x, y,
        System.currentTimeMillis()*0.001);
  } else {
    hit.moveTo(x, y);
  }
  TargetIcon t = null;
  for(int i = a.size()-1; i >= 0; --i)
    if(a.get(i) instanceof TargetIcon &&
       a.get(i) != hit &&
       a.get(i).hit(x, y)) { t = (TargetIcon)a.get(i); break; }
```

```
    if(target != null) { target.setHighlight(false); target = null; }
    if(t != null) { t.setHighlight(true); target = t; }
  }
  private void toBack(DrawObj o) { a.remove(o); a.add(0, o); }
  private void toFront(DrawObj o) { a.remove(o); a.add(o); }
}
class DraggableIcon implements DrawObj {
  boolean highlight = false;
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  String label;
  double xpos, ypos, rad = 25.0;
  Color c1 = Color.yellow, c2 = Color.pink;
  public DraggableIcon(String s, double x, double y) {
    label = s; xpos = x; ypos = y;
  }
  public void draw(Graphics g) {
    int r2 = (int)(rad*2);
    g.setColor(highlight ? c2 : c1);
    g.fillOval((int)(xpos-rad), (int)(ypos-rad), r2, r2);
    g.setColor(Color.black); g.setFont(fn);
    g.drawString(label, (int)(xpos-12), (int)(ypos+8));
  }
  public boolean hit(double x, double y) {
    return (xpos-x)*(xpos-x)+(ypos-y)*(ypos-y) < rad*rad;
  }
  public void moveTo(double x, double y) { xpos = x; ypos = y; }
  public double getX() { return xpos; }
  public double getY() { return ypos; }
  public void setHighlight(boolean b) { highlight = b; }
}
class ThrowableIcon extends DraggableIcon {
  double vx = 0.0, vy = 0.0, time = 0.0;
  public ThrowableIcon(String s, double x, double y) { super(s, x, y); }
  public void setTime(double t) { time = t; vx = vy = 0.0; }
  public void moveTo(double x, double y, double t) {
    double dt = t - time; time = t;
    if(dt > 0) {
      double vx1 = (x-xpos)/dt, vy1 = (y-ypos)/dt;
      vx = 0.7*vx + 0.3*vx1; vy = 0.7*vy + 0.3*vy1;
    }
    xpos = x; ypos = y;
  }
  public void updateTime(double t) {
    double dt = t - time; time = t;
    xpos += vx*dt; ypos += vy*dt;
    if(xpos < 20 && vx < 0) vx = vy = 0.0;
    if(xpos > getWidth()-20 && vx > 0) vx = vy = 0.0;
    if(ypos < 20 && vy < 0) vy = vy = 0.0;
```

```
      if(ypos > getHeight()-20 && vy > 0) vx = vy = 0.0;
    }
    public boolean isMoving() { return vx*vx + vy*vy > 0.0; }
  }
  class TargetIcon extends DraggableIcon {
    int dropcount = 0;
    public TargetIcon(String s, double x, double y) {
      super(s, x, y); c1 = Color.blue; c2 = Color.green;
    }
    public void drop(DrawObj o) {
      ++dropcount;
      o.moveTo(xpos+5*dropcount, ypos+5*dropcount);
    }
  }
}
```

演習 6 上の「アイコン投げ」の例題を拡張して、もっと工夫した投げ方ができるようにしてみなさい (例: 衝突半径を増やして投げる時の成功率を大きくする等)。

5 まとめ

本科目「ユーザインタフェース」では、人間とコンピュータのインタフェースを担うソフトウェアの部分について、基本的な考え方や枠組み、および実現技術をテーマとして各回の話題を取り上げてきました。具体的な内容としては次のものがありました。

- ユーザインタフェースの基本概念と基本技術/キーボード入力
- メニュー操作の時間/人間の行動モデル
- 打鍵レベルモデル/GUI 部品と GUI プログラミング
- モデルの分離/新しいインタラクション方式の工夫

各回を通じて一番大切なのは、「アイデアがあったら、それを実現して試してみることで、新たな知見が得られる」ということかと思います。次回論文紹介が残っていますが、とりあえずおつかれ様でした。