# 計算機科学第一最終レポート

高野成章

2016 年 2 月 15 日提出

## 1 はじめに

今回の課題はお絵かきプログラムの実装である。大きく分けてバグの発見・修正および、新たな機能の追加を行った。複数の形が変えるようにオブジェクトを作成します。また、その形に対して変換を及ぼす。得に色、削除、拡大が課題でありました。ここらは mandelbrotset お同様に分離して考えることができると思います。指定された基本機能に加え、折れ線、打点、クリア（reset) を追加しました。キーボードによる入力で制御したいという試みで CuiControl を設けましたが、実装には至らなかったので、実際のコードのほうではコメントアウトの状態にしています。

## 2 図形オブジェクト

### 2.1 Ellipse

rectangle を参考にしました。ellipse の場合は中心の座標、短径、長径を指定します。描画するとき、drag することで中心および径が変わります。中心は min を指定しているので、右下方向に drag したときは最初にクリックした点、左上方向の場合は MouseEvent が示す点になります。

```
object EllipseControl {

  var e  = new Ellipse {}
  var p0 = new Point2D(0,0)

  def onPress(ev: MouseEvent) {
   p0 = new Point2D(ev.x, ev.y)
   e = new Ellipse{
    centerX = p0.x ; centerY = p0.y
    stroke = strokeColor; fill = Color.Transparent
   }
```

```
   drawingPane.children += e
   shapes += TDEllipse(e)
    }

  def onDrag(ev: MouseEvent) {
    e.centerX = min(p0.x, ev.x);  e.centerY = min(p0.y, ev.y)
    e.radiusX = abs(p0.x-ev.x);   e.radiusY = abs(p0.y-ev.y)
  }

  def onRelease(ev: MouseEvent) {
    e.fill = fillColor
  }
}
```

## 2.2 Line

線分の実装は始点 (startX,startY) と終点 (endX.endY) を指定することで求めています。onDrag で終点をマウスの位置に移動しています。

```
object LineControl {
 var l = new Line {}
 var p0 = new Point2D(0,0)

 def onPress(ev:MouseEvent){
 p0 = new Point2D(ev.x, ev.y)

  l = new Line \{
  startX = p0.x ; startY = p0.y
  endX = p0.x ; endY = p0.y
  stroke = strokeColor; fill = Color.Transparent
  }
  drawingPane.children += l
  shapes += TDLine(l)
  }

def onDrag(ev: MouseEvent){
```

```
 l.endX = ev.x ; l.endY = ev.y
 }

def onRelease(ev: MouseEvent){
 l.fill = fillColor
 }
}
```

なお、線分の選択の実装のために、subline も定義しました。subline は line を書くと同時に太い透明な線を後ろに書くことで、実際に選択しやすくなるというものです。subline は変数を s(透明な線分) および l(実際に引いている線分) を引数にとり、始点、終点を一致させることになっている。

```
def subline(s:Line, l:Line){
   s.startX = l.startX() /*色を指定していないので透明になる*/
   s.startX = l.startY()
   s.endX = l.endX()
   s.endY = l.endY()
   s.strokeWidth = 20
}
```

## 2.3 Polyline

Polyline(折れ線) も実装しました。

```
object PolylineControl {
  var p = new Polyline{strokeWidth = 2 }
  var p0 = new Point2D(0,0)
  var touch = 0

  def onPress(ev: MouseEvent){
   if (touch == 0){
    drawingPane.children += p
    shapes += TDPolyline(p)
    touch = 1
    }
```

```
   if (ev.clickCount != 1) {
    p = new Polyline{strokeWidth = 2}
    touch = 0
    }

    else {
    p0 = new Point2D(ev.x,ev.y)
    p.points ++= List(p0.x,p0.y)
    p.stroke = strokeColor
    }
 }
  def onRelease(ev: MouseEvent){
    p.stroke = strokeColor
  }
 }
```

touch=0 は制御点を打ったかどうかを調べるものです。touch = 0 の状態でクリックした場所が始点となります。clickcount が 1 でない場合、つまりダブルクリックをした場合などは、check を 0 に戻すことで、新たな始点の設定が可能になります。check が 1 の場合は polyline のリストにクリックした点を格納し、点と点をつなぎます。ここで問題点は polyline の選択方法が実装できなかったため、onDrag を定義することや色を変えることもできません。

なお、Toggle のほうは次のように設定しました。

```
  new ToggleButton{
      id = "Polyline"
      graphic = new Polyline{
        stroke = Color.Black
         points ++= List(0,0,11,29,22,0,33,29)
      }
      toggleGroup = shapeGroup
    }
```

List の値は (x0,y0,x1,y1,...) のように格納されています。ここに指定した値は、選択ツールボックスの大きさが均一の大きさになるようにしています。(選択ツールボックスのデザインが折れ線であり、その形を指定している)

## 2.4 Dots

打点の実装を行いました。打点といっても実際は楕円を drag するとともに書いていくというものです。

```
object DotsControl {
  var e = new Ellipse{}

  def onPress(ev:MouseEvent){
    e = new Ellipse{
    fill=strokeColor;stroke=strokeColor;e.radiusX=width
    e.radiusY=width;e.centerX=ev.x;e.centerY=ev.y
    }
    drawingPane.children += e
  }
  def onDrag(ev:MouseEvent){
    e = new Ellipse{
      fill=strokeColor;stroke=strokeColor;e.radiusX=width;e.radiusY=width
                      e.centerX=ev.x;e.centerY=ev.y
    }
    drawingPane.children += e
  }
}
```

# 3 コントロール部分

## 3.1 SelectControl

### 3.1.1 選択の判定

線分の判定としては、subline を contains を用いて行いました。ただし、この実装の問題点は実線 l を中線として subline s ができるのではなく、subline の一方の縁に l がくるようになっているというところです。すなわち、選択できる領域は増えますが、l の上側あるいは下側についてのみ増えていることになります。選択を行うことで影がつきますが、これを最新の選択のみにつける方法として、選択した瞬間にすべてのケース (選択した場所が図形でないケースも) で影を null にし、そのあと選択したものだけに影をつけるようにした。また、前にあるものから選択できるように、shape を含む buffer に reverse をかましました。

### 3.1.2 onDrag

選択した図形を移動させるメソッドです。長方形の移動に関して、(x1,y1) は最初の位置、(x0,y0) は図形をクリックした座標を示します。ベクトルで考えると、(r.x,r.y) (x1,x0) = (ev.x, ev.y) (x0,y0) が成り立つので、(r.x, r.y) が求まります。線に関しては選択すると同時に p0 を線分の中点にし、それを以下のようにして動かしています。楕円は中心を動かしています。

```
def onDrag(ev: MouseEvent){
val x0 = p0.x ; val y0 = p0.y

selection match {
  case TDRectangle(r) => r.x =x1 + ev.x - x0 ; r.y = y1 + ev.y - y0
  case TDLine(l) => l.startX() = ev.x-p0.x ; l.startY() = ev.y -p0.y
                    l.endX = ev.x + p0.x; l.endY = ev.y + p0.y
  case TDEllipse(e) => e.centerX = ev.x; e.centerY = ev.y
  case TDNoShape() =>
 }
}
```

### 3.1.3 keyevent の実装

keyevent の実装。選択ボタン (Toggle) をクリック、図形をクリック、そのあと delete ボタンを押すことで選択した図形を消せるようにしました。また、CTRL+r ですべての図形を消去できるように設定ました。

```
def remove(s:TDShape):Unit = {
  s match {
    case TDRectangle(r) => drawingPane.children -= r
    case TDLine(l) => drawingPane.children -= l
    case TDEllipse(e) => drawingPane.children -=e
    case TDPolyline(p) => drawingPane.children -=p
  }
  shapes -= s
}

def reset = {
  drawingPane.children = Nil
  shapes = Buffer()
```

```
}
```

## 3.2 SizeControl

図形の拡大縮小のところです。本当は選択の部分 (SelectControl) に含めたかったのですが、クリックした店の保存・固定および、onDrag の部分を使い分ける必要があるので分離しました。2つ目の selection match で x1,y1(x2,y2) に形の長さ・大きさを保存し、onDrag の selection match で長さ、太さを MouseEvent の座標の変化に応じて拡大縮小を行います。例えば、長方形の場合-p0.x なる項がありますが、ここは SelectControl の onDrag と同様に、図形をクリックした地点を中心とり、そこから MouseEvent がどれくらい動いたかを求める式です。

```
object SizeControl{
  var selection: TDShape = TDNoShape()
  var p0 = new Point2D(0,0)

  var x1:Double = 0 ; var y1:Double = 0
  var x2:Double = 0 ; var y2:Double = 0

  def onPress(ev:MouseEvent){
    val x = ev.x; val y = ev.y
    val oShape = shapes.reverse.find((shape: TDShape)=>
        shape match {
          case TDRectangle(r) => r.contains(x,y)
          case TDEllipse(e) => e.contains(x,y)
          case TDLine(l) => l.contains(x,y)
          case _ => false
    })

    selection = oShape match{
      case Some(shape) => shape
      case _ => TDNoShape()
    }

    selection match {
   case TDRectangle(r) => x1 = r.width() ; y1 = r.height()
   case TDEllipse(e) => x1 = e.radiusX(); x2 = e.radiusY()
   case TDLine(l) => x1 = l.startX(); y1 = l.startY()
```

```
                    x2 = l.endX(); y2 = l.endY()
     case _ =>
    }
    p0 = new Point2D(ev.x, ev.y)
  }

  def onDrag(ev:MouseEvent){
    val x = ev.x ; val y = ev.y
    selection match {
      case TDRectangle(r) => r.width = abs(x1 + ev.x-p0.x)
                             r.height = abs(y1+ev.y-p0.y)
      case TDEllipse(e) => e.radiusX = abs(x1+ev.x-p0.x)
                           e.radiusY = abs(y1+ev.y-p0.y)
      case TDLine(l) => l.startX = x1+ev.x-p0.x; l.startY= y1+ev.y-p0.y
                        l.endX=x2+ev.x-p0.x;l.endY=y2+ev.y-p0.y
    }
  }
}
```

## 3.3 CuiControl

キーボードだけで操作を可能にする (すなわち Toggle を推す必要がない) オブジェクトを設けましたが、どういうわけか機能しませんでした。

```
object CuiControl{
  def onKeyPress(ev: KeyEvent){
    if (ev.isControlDown){
      ev.code match{
        case KeyCode.R => reset
        case KeyCode.ESCAPE => System.exit(0)
        case _ =>
        }
      }
    }
  }
```

## 4 Colorpicker およびその他 Toggle について

縁の色および、塗りつぶしに関しては、Colorpicker で選択した色を val c に保存し、選択した図形によって strokeColor、fillColor を行う match case 文を書きました。

```
val colorTools = Seq(
  new ColorPicker(strokeColor) {
    onAction = { e: ActionEvent => strokeColor = value()
    val c = value()
    strokeColor = Color.hsb(c.hue, c.saturation, c.brightness, 0.5)
    SelectControl.selection match {
        case TDRectangle(r) => r.stroke = strokeColor
        case TDEllipse(e) => e.stroke = strokeColor
        case TDLine(l) => l.stroke = strokeColor
        case TDPolyline(p) => p.stroke = strokeColor
        case _ => ()
        }
    }
  },
  new ColorPicker(fillColor) {
    onAction = { e: ActionEvent =>
      val c = value()
      fillColor = Color.hsb(c.hue, c.saturation, c.brightness, 0.5)
      SelectControl.selection match {
        case TDRectangle(r) => r.fill = fillColor
        case TDEllipse(e) => e.fill = fillColor
        case TDLine(l) => l.fill = fillColor
        case _ => ()
      }
    }
  })
```

## 5 その他 (時間の制約上) 修正できなかった点

- 図形が drawingPane を超えるサイズになってしまう。選択ツールにオーバーラップすると終了するしかない。

- polyline,Dots を書いてからは、選択機能が使えなくなる。それらを選択せざるを得ないのだが、選択のメソッドがオブジェクトに定義されていないのが問題点
- trait を用いて SelectControl,SizeControl や EllipseControl,RectangleControl を共通化すること

# 6 おわりに

実装するにあたって、他の人に助言をいただいたり、scalafx のサンプルコードなども参考にしました。その上でコードを理解し、使いこなせるような訓練ができたと思います。一からオブジェクトをデザインするにはもっとたくさんの経験が必要だと感じます。まずは読解力を高め、新たに創造する技術は今後の課題します。

前期はカリー化などをはじめ、基本的なメソッドを一から作る練習をしましたが、後期では既存のライブラリを使いこなす能力が必要でしたが、開発には両方の能力が備わっていることが望ましいのだと感じます。scalafx のドキュメントですが、英語の問題ではなく、例の説明が少ないため使いこなすことができませんでした。scalafx に関するテストについてですが、GUI プログラミング寄りに感じたので、ユーザーが報告する（すなわち、実際に動かして試してみる）というのが効率的だと感じました（少なくとも短期間の間での実装で)。