CMSC427 fall 2017 Lab 1
Fan Yang

I enabled the user to delete a point by adding a key event in the keyPressed() method. When ‘d’ is pressed, keyPressed() will call the delete() method that I implemented in Polyline class. delete() will then check whether there is a picked point. If a point is picked, that point will be deleted; if no point is picked, the last point added will be deleted.

To save and retrieve polyline information I added two more key events. When ‘s’ is pressed, the program will save the polyline information into a file named “polyline.dat”; when ‘r’ is pressed, the program will retrieve the polyline information from “polyline.dat” and display it on canvas.

For isSimple() method I used the brute force approach to finding line intersections. Every edge is calculated against every other edge of the polyline to see if there exists an intersection, using parametric line equations.

isConvex() is implemented by computing the z-component of the cross product of each consecutive pair of vectors (defined by each triplet of points). Given p[k], p[k+1], p[k+2] each with coordinates x, y:

dx1 = x[k+1]-x[k]
dy1 = y[k+1]-y[k]
dx2 = x[k+2]-x[k+1]
dy2 = y[k+2]-y[k+1]
CrossProduct_z = dx1*dy2 - dy1*dx2

The polygon is convex if the z-components of the cross products are either all positive or all negative. Otherwise the polygon is nonconvex.

isCW() is implemented by calculating signed area of the polygon. If a polygon has vertices (x1,y1), ..., (xn,yn), then the signed area is determined as followed:

$$A = \frac{1}{2}\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + \ldots + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix}\right)$$

If the area is positive then the polygon is clockwise; if negative then polygon is counterclockwise.

```
// CMSC427
// File: PolylineEditor.pde
// Fan Yang
// Polyline editor main

Polyline polyline;

void setup() {
  size(400,400);
  polyline = new Polyline();
}

void draw() {
  background(255);
  noFill();
  polyline.draw();
```

```
}

void keyPressed() {
  if (key == ‘c’)
     polyline.close();
  else if (key == ‘o’)
     polyline.open();
  // save polyline to "polyline.dat" if ‘s’ is pressed.
  else if (key == ‘s’)
     polyline.save();
  // retrieve polyline information from "polyline.dat" if ‘r’ is pressed.
  else if (key == ‘r’)
    polyline.retrieve();
  // delete a point from the polygon if ‘d’ is pressed.
  else if (key == ‘d’)
    polyline.delete();
  // output the result of isSimple() in the console if ‘1’ is pressed.
  else if (key == ‘1’)
    println("isSimple():  " + polyline.isSimple());
  // output the result of isConvex() in the console if ‘2’ is pressed.
  else if (key == ‘2’)
    println("isConvex():  " + polyline.isConvex());
  // output the result of isCW(), i.e. whether the polygon is clockwise, in the console if ‘3’ is pressed.
  else if (key == ‘3’)
    println("isCW():  " + polyline.isCW());
}

void mousePressed() {
  if (mouseButton == LEFT)
    polyline.add(mouseX,mouseY);
  else if (mouseButton == RIGHT)
    polyline.pick(mouseX,mouseY);
}

void mouseDragged() {
  polyline.pickUpdate(mouseX,mouseY);
}

void mouseReleased() {
  polyline.pickRelease();
}



// CMSC427
// File: Polyline.pde
// Fan Yang
// Polyline class

public class Polyline {
  int pick = -1;
  boolean isClosed = false;
  ArrayList<Point> pline = new ArrayList<Point>();

  // Add point to Polyline
```

```
void add(float mx, float my) {
  pline.add(new Point(mx,my));
}

// delete point from polyline
void delete(){
  if(pline.size() == 0) return;

  // if no point is picked, then remove the last point added.
  if(pick == -1)
    pline.remove(pline.size() - 1);
  // if a point is picked, remove that point.
  else {
    pline.remove(pick);
    pick = -1;
  }
}

// brute force approach to finding if intersection of non-adjacent edges exists.
// Return true if no intersection is found (simple polygon), false otherwise.
boolean isSimple(){
  float epsilon = 0.0000001;
  for(int i = 0; i < pline.size() - 1; i++){
    float x12 = pline.get(i+1).x - pline.get(i).x;
    float y12 = pline.get(i+1).y - pline.get(i).y;
    int endInd = i == 0 ? (pline.size() - 1) : pline.size();
    for(int j = i + 2; j < endInd; j++){
      float x34 = pline.get((j+1)%pline.size()).x - pline.get(j).x;
      float y34 = pline.get((j+1)%pline.size()).y - pline.get(j).y;
      float denominator = x34*y12 - x12*y34;
      if(Double.compare(denominator, 0.0) == 0)
        continue;
      float x13 = pline.get(j).x - pline.get(i).x;
      float y13 = pline.get(j).y - pline.get(i).y;
      float s = (x34*y13 - x13*y34)/denominator;
      float t = (x12*y13 - x13*y12)/denominator;
      if(s > -epsilon && s < 1+epsilon && t > -epsilon && t < 1+epsilon)
        return false;
    }
  }
  return true;
}


boolean isConvex(){
  // if polygon is not simple, exit the program.
  if(!isSimple()){
    println("polygon is not simple!");
    return false;
  }

 /*For each consecutive pair of edges of the polygon (each triplet of points), compute the
    z-component of the cross product of the vectors defined by the edges pointing towards the
    points in increasing order. Take the cross product of these vectors:

       given p[k], p[k+1], p[k+2] each with coordinates x, y:
       dx1 = x[k+1]-x[k]
       dy1 = y[k+1]-y[k]
       dx2 = x[k+2]-x[k+1]
```

```
      dy2 = y[k+2]-y[k+1]
      zcrossproduct = dx1*dy2 - dy1*dx2

    The polygon is convex if the z-components of the cross products are either all positive or
    all negative. Otherwise the polygon is nonconvex.
    If there are N points, calculate N cross products, and use the triplets (p[N-2],p[N-1],p[0])
    and (p[N-1],p[0],p[1]).*/
  float _zcross = 0;
  for(int i = 0; i < pline.size(); i++){
    int j = (i+1)%pline.size(), k = (i+2)%pline.size();
    float dx1 = pline.get(j).x - pline.get(i).x;
    float dy1 = pline.get(j).y - pline.get(i).y;
    float dx2 = pline.get(k).x - pline.get(j).x;
    float dy2 = pline.get(k).y - pline.get(j).y;
    float zcross = dx1*dy2 - dy1*dx2;
    if(i == 0)
      _zcross = zcross;
    else if(_zcross*zcross < 0)
      return false;
  }
  return true;
}

/*The (signed) area of a planar non-self-intersecting polygon with vertices (x1,y1), ...,  (xn,yn) is
  A = 1/2(x1*y2 - x2*y1 + x2*y3 - x3*y2 + ... + x(n-1)*yn – xn*y(n-1) + xn*y1 - x1*yn)
  The area of a convex polygon is negative if the points are arranged in a counterclockwise order, and
  positive if they are in clockwise order (Beyer 1987). */
  boolean isCW(){
    // if polygon is not simple, exit the program
    if(!isSimple()){
      println("polygon is not simple!");
      return false;
    }

    float A = 0;
    for(int i = 0; i < pline.size(); i++)
      A += (pline.get(i).x * pline.get((i+1)%pline.size()).y - pline.get((i+1)%pline.size()).x * pline.get(i).y);
    return A > 0;
  }

  // Save polyline to file "polyline.data".
  // #points on first line, and each point (x,y) on following lines.
  void save(){
    PrintWriter output = createWriter("polyline.dat");
    //output.println(pline.size());
    for(Point p : pline)
      output.println(p.x + "\t" +p.y);
    output.flush();
    output.close();
  }

  // retrieve polyline from file "polyline.dat"
  void retrieve(){
    try{
      pline = new ArrayList<Point>();
      BufferedReader reader = createReader("polyline.dat");
      String line = reader.readLine();
      while(line != null){
        String[] coords = split(line, "\t");
```

```
      int x = int(coords[0]);
      int y = int(coords[1]);
      pline.add(new Point(x, y));
      line = reader.readLine();
    }
    reader.close();
  } catch(IOException e) {
    e.printStackTrace();
  }
}

// Display Polyline
void draw() {

  beginShape();
  for (int i = 0; i < pline.size(); i++) {
      Point p = pline.get(i);
      vertex( p.x, p.y );
  }
  if (isClosed)
      endShape(CLOSE);
  else
      endShape();
  if (pline.size() > 0) {
    Point p = pline.get(0);
    fill(0,0,255,128);
    ellipse(p.x,p.y,15,15);
  }
 fill(255,0,0,128);
 for (int i = 1; i < pline.size(); i++) {
   Point q = pline.get(i);
   ellipse(q.x,q.y,10,10);
    }
 if (pick != -1) {
   fill(0,255,0, 128);
   Point m = pline.get(pick);
   ellipse(m.x,m.y,18,18);
 }
}

// Set to polygon
void close() {
  isClosed = true;
}

// Set to polyline
void open() {
  isClosed = false;
}

// Release the picked point
void pickRelease() {
  pick = -1;
}

// Update pick point location
void pickUpdate(float mx, float my) {
  if(pick != -1) {
    Point p = pline.get(pick);
```

```
    p.x = mouseX;
    p.y = mouseY;
   }
  }

  // Set the picked point
  void pick(float mx, float my) {
   for (int i = 0; i < pline.size(); i++) {
    Point p = pline.get(i);
    if ( dist(p.x,p.y,mouseX,mouseY) < 8)
      pick = i;
   }
  }

  // Internal utility class
  class Point {
   public float x,y;
   public Point( float _x, float _y ) {
     x = _x;
     y = _y;
   }
  }
}
```