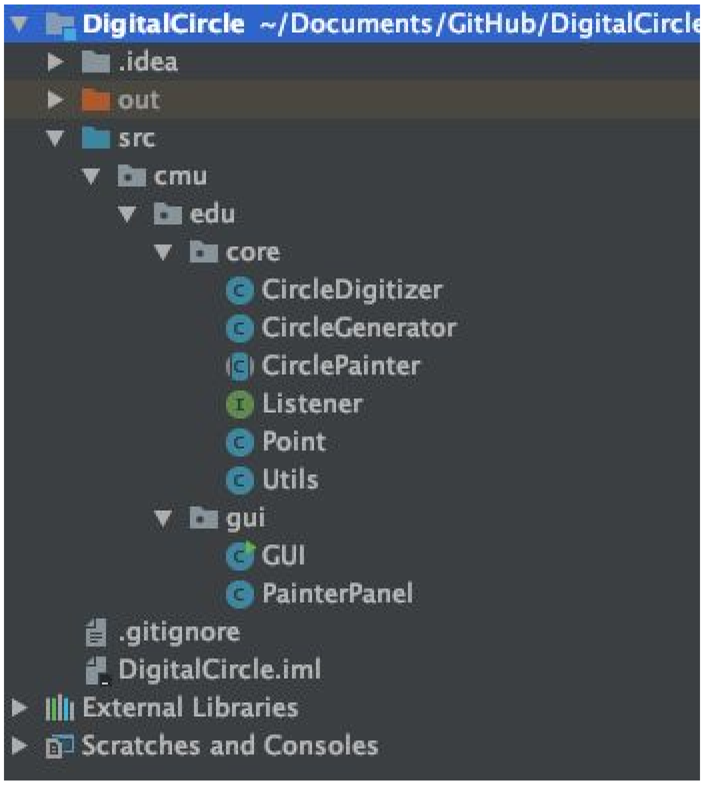
Documentation For Circle Painter

Chengyongping Lu

● Overview

This system (Circle Painter) mainly realized two functions - generating circles and highlighting nearest points according to the movement of the mouse, creating a best fitting circle based on the points the user selected.
The system is developed in Java. I mainly used Java Swing to build the application UI and used the observer design pattern to make the code more organized and maintainable.

● Code structure



```
DigitalCircle  ~/Documents/GitHub/DigitalCircle
  .idea
  out
  src
    cmu
      edu
        core
          CircleDigitizer
          CircleGenerator
          CirclePainter
          Listener
          Point
          Utils
        gui
          GUI
          PainterPanel
  .gitignore
  DigitalCircle.iml
External Libraries
Scratches and Consoles
```

The code consists of two packages - core and gui.
The core package contains the following interfaces or classes:

CircleDigitizer: Mainly implement the Part I function - drawing three circles and highlighting nearest points based on the mouse movement.

CircleGenerator: Mainly implement the Part II function - creating a best-fitting circle according to the toggled on points on the panel.

CirclePainter: An abstract parent class for CircleDigitizer and CircleGenerator, defining several common functions.

Listener: Defining the UI operations like drawing circles and toggling points.

Point: Class for points on the panel.

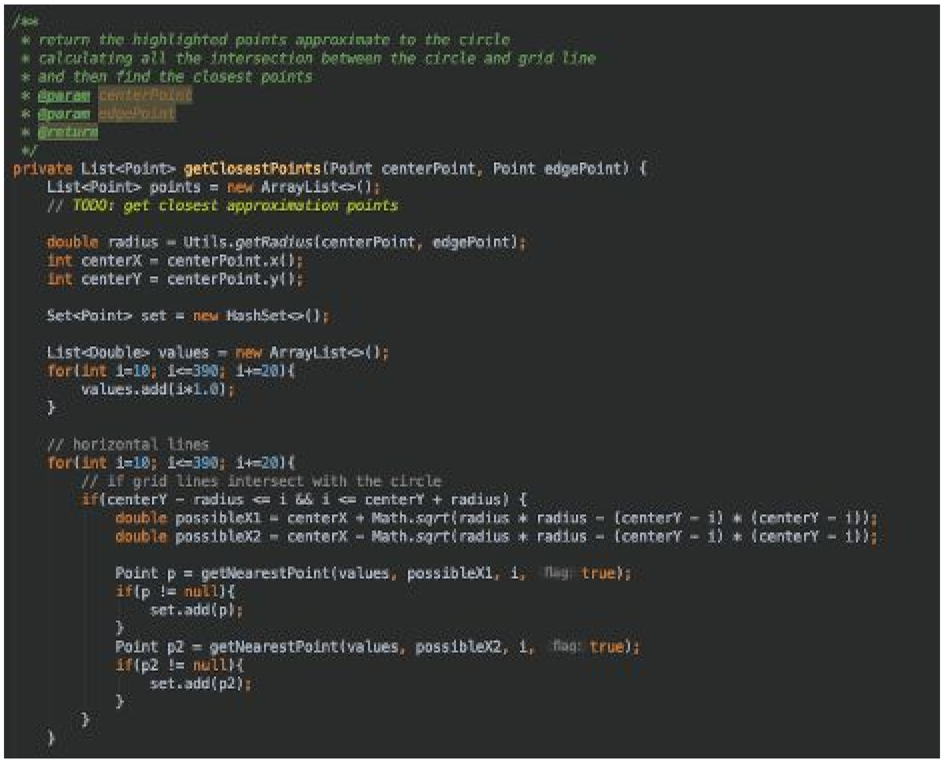
Utils: Basic helper functions for calculating distances or circle radius.

The gui package contains the following classes:

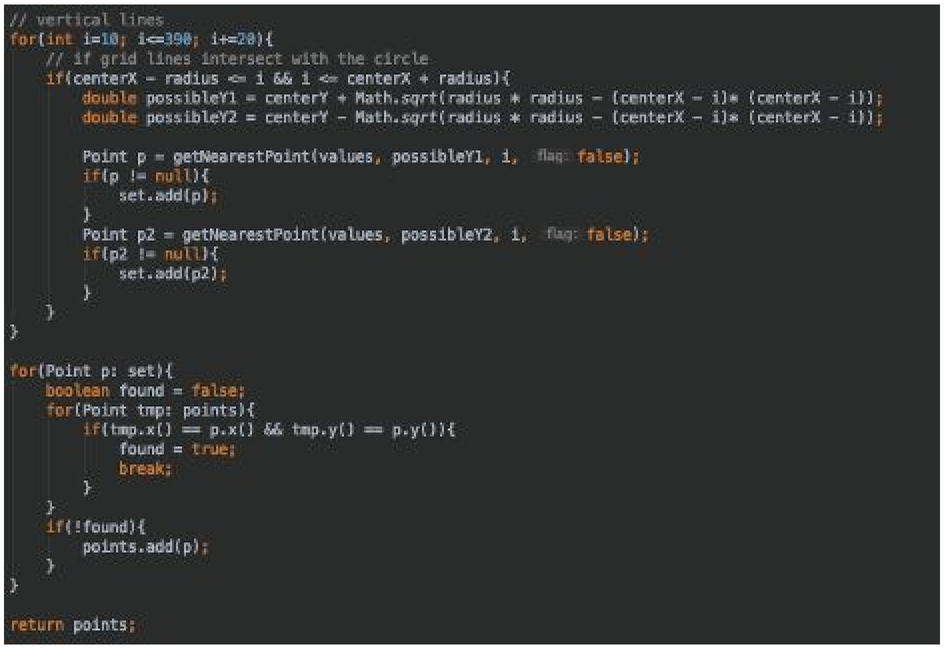
GUI: It contains the basic user interface including the welcome page and navigation buttons.

PainterPanel: Main class for dealing with user's operations and interaction with the backend.

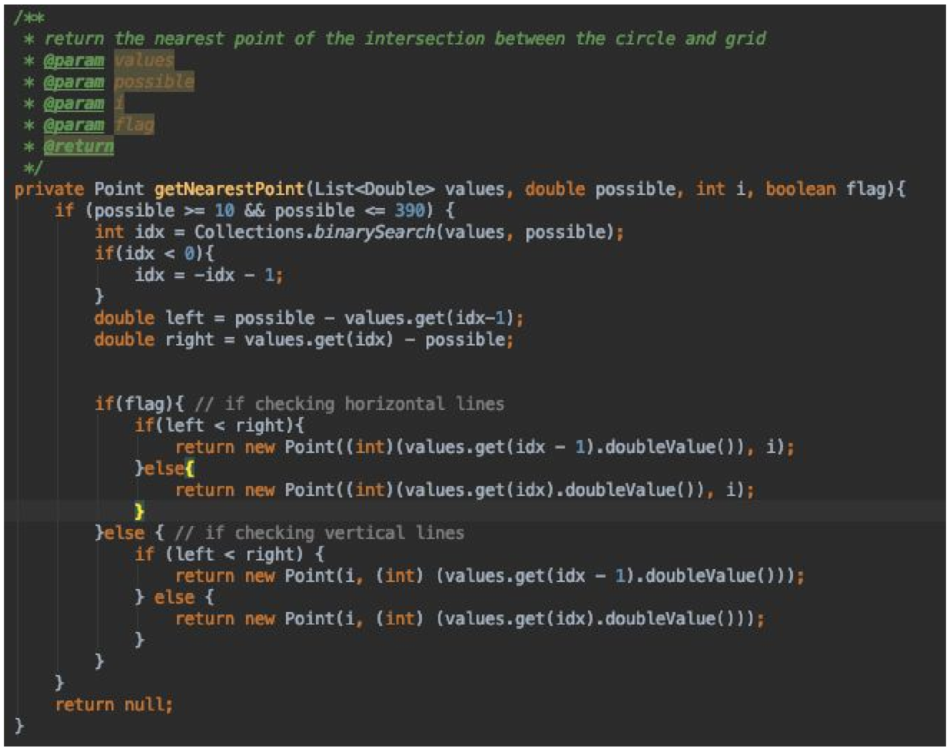
- Algorithms



```java
/**
 * return the highlighted points approximate to the circle
 * calculating all the intersection between the circle and grid line
 * and then find the closest points
 * @param centerPoint
 * @param edgePoint
 * @return
 */
private List<Point> getClosestPoints(Point centerPoint, Point edgePoint) {
    List<Point> points = new ArrayList<>();
    // TODO: get closest approximation points

    double radius = Utils.getRadius(centerPoint, edgePoint);
    int centerX = centerPoint.x();
    int centerY = centerPoint.y();

    Set<Point> set = new HashSet<>();

    List<Double> values = new ArrayList<>();
    for(int i=10; i<=390; i+=20){
        values.add(i*1.0);
    }

    // horizontal lines
    for(int i=10; i<=390; i+=20){
        // if grid lines intersect with the circle
        if(centerY - radius <= i && i <= centerY + radius) {
            double possibleX1 = centerX + Math.sqrt(radius * radius - (centerY - i) * (centerY - i));
            double possibleX2 = centerX - Math.sqrt(radius * radius - (centerY - i) * (centerY - i));

            Point p = getNearestPoint(values, possibleX1, i,  flag: true);
            if(p != null){
                set.add(p);
            }
            Point p2 = getNearestPoint(values, possibleX2, i,  flag: true);
            if(p2 != null){
                set.add(p2);
            }
        }
    }
```



```java
// vertical lines
for(int i=10; i<=390; i+=20){
    // if grid lines intersect with the circle
    if(centerX - radius <= i && i <= centerX + radius){
        double possibleY1 = centerY + Math.sqrt(radius * radius - (centerX - i)* (centerX - i));
        double possibleY2 = centerY - Math.sqrt(radius * radius - (centerX - i)* (centerX - i));

        Point p = getNearestPoint(values, possibleY1, i,  flag: false);
        if(p != null){
            set.add(p);
        }
        Point p2 = getNearestPoint(values, possibleY2, i,  flag: false);
        if(p2 != null){
            set.add(p2);
        }
    }
}

for(Point p: set){
    boolean found = false;
    for(Point tmp: points){
        if(tmp.x() == p.x() && tmp.y() == p.y()){
            found = true;
            break;
        }
    }
    if(!found){
        points.add(p);
    }
}

return points;
```

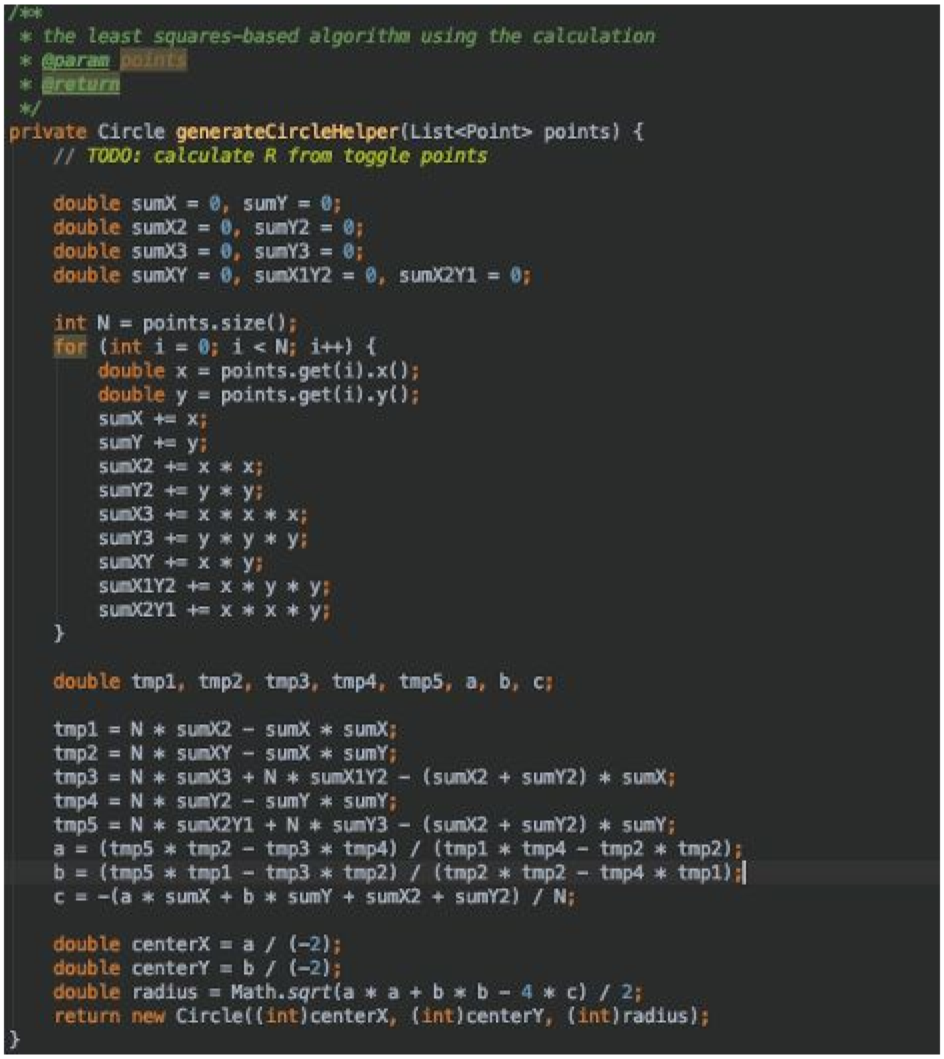


```java
/**
 * return the nearest point of the intersection between the circle and grid
 * @param values
 * @param possible
 * @param i
 * @param flag
 * @return
 */
private Point getNearestPoint(List<Double> values, double possible, int i, boolean flag){
    if (possible >= 10 && possible <= 390) {
        int idx = Collections.binarySearch(values, possible);
        if(idx < 0){
            idx = -idx - 1;
        }
        double left = possible - values.get(idx-1);
        double right = values.get(idx) - possible;


        if(flag){ // if checking horizontal lines
            if(left < right){
                return new Point((int)(values.get(idx - 1).doubleValue()), i);
            }else{
                return new Point((int)(values.get(idx).doubleValue()), i);
            }
        }else { // if checking vertical lines
            if (left < right) {
                return new Point(i, (int) (values.get(idx - 1).doubleValue()));
            } else {
                return new Point(i, (int) (values.get(idx).doubleValue()));
            }
        }
    }
    return null;
}
```

Above is the algorithm to find the nearest points according to the mouse movement. This function contains two parameters - centerPoint and edgePoint. CenterPoint is the circle center and the edgePoint is the point where the user releases their mouse. The basic idea for this function is using all possible horizontal and vertical lines to intersect with the circle. After we calculate all possible intersections we will try to calculate the distances between this intersection and the left point or right point. Then we choose a closer point

and add the point into the set. After the duplication checking, we add the point into the return point list.
The algorithms contains the following steps:

1. Generate a list containing [10, 30, 50, 70, …, 390], which are all possible values for the grid lines.
2. Check each horizontal line to see whether they intersect with the circle. Get two possible intersections.
3. For each intersection, find distances between this intersection and the closest left & right point. (Or closest upper or downer point). Choose a closer one and add this point to the set.
4. For all the vertical lines, do the same.
5. Do the duplicate checking and add all points into the list and return.



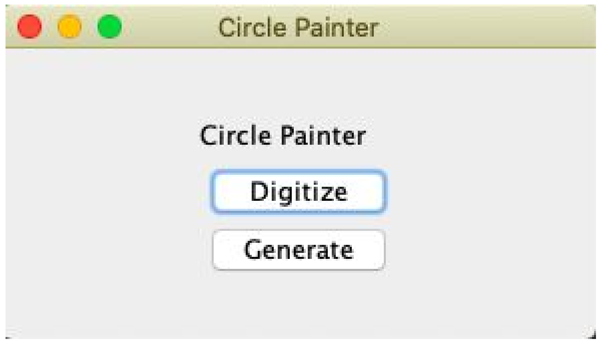
```java
/**
 * the least squares-based algorithm using the calculation
 * @param points
 * @return
 */
private Circle generateCircleHelper(List<Point> points) {
    // TODO: calculate R from toggle points

    double sumX = 0, sumY = 0;
    double sumX2 = 0, sumY2 = 0;
    double sumX3 = 0, sumY3 = 0;
    double sumXY = 0, sumX1Y2 = 0, sumX2Y1 = 0;

    int N = points.size();
    for (int i = 0; i < N; i++) {
        double x = points.get(i).x();
        double y = points.get(i).y();
        sumX += x;
        sumY += y;
        sumX2 += x * x;
        sumY2 += y * y;
        sumX3 += x * x * x;
        sumY3 += y * y * y;
        sumXY += x * y;
        sumX1Y2 += x * y * y;
        sumX2Y1 += x * x * y;
    }

    double tmp1, tmp2, tmp3, tmp4, tmp5, a, b, c;

    tmp1 = N * sumX2 - sumX * sumX;
    tmp2 = N * sumXY - sumX * sumY;
    tmp3 = N * sumX3 + N * sumX1Y2 - (sumX2 + sumY2) * sumX;
    tmp4 = N * sumY2 - sumY * sumY;
    tmp5 = N * sumX2Y1 + N * sumY3 - (sumX2 + sumY2) * sumY;
    a = (tmp5 * tmp2 - tmp3 * tmp4) / (tmp1 * tmp4 - tmp2 * tmp2);
    b = (tmp5 * tmp1 - tmp3 * tmp2) / (tmp2 * tmp2 - tmp4 * tmp1);
    c = -(a * sumX + b * sumY + sumX2 + sumY2) / N;

    double centerX = a / (-2);
    double centerY = b / (-2);
    double radius = Math.sqrt(a * a + b * b - 4 * c) / 2;
    return new Circle((int)centerX, (int)centerY, (int)radius);
}
```

Above is the algorithm to find the best fitting circle based on the toggled-on points. This function contains one parameter - points, which means the list of all toggled on points. The function returns a Circle. The basic idea is that we can denote the circle function as: (x - x1)^2 + (y - y1)^2 = R^2. Thus, all points on the circle satisfy this function. Therefore, we can also conduct the equation: Σ((xi - x)^2 + (yi - y)^2 - R^2)xi = 0, Σ((xi - x)^2 + (yi - y)^2 - R^2)yi=0. Then we can get the equation: Σ(xi^3 - 2*xi^2*x + xi * yi^2 - 2*xi * yi * y)=0 and Σ(xi^2*yi - 2*xi*yi*x + yi^3 -2*yi^2*y)=0. Therefore, we can try to get the value of x, y, R by simple calculation.

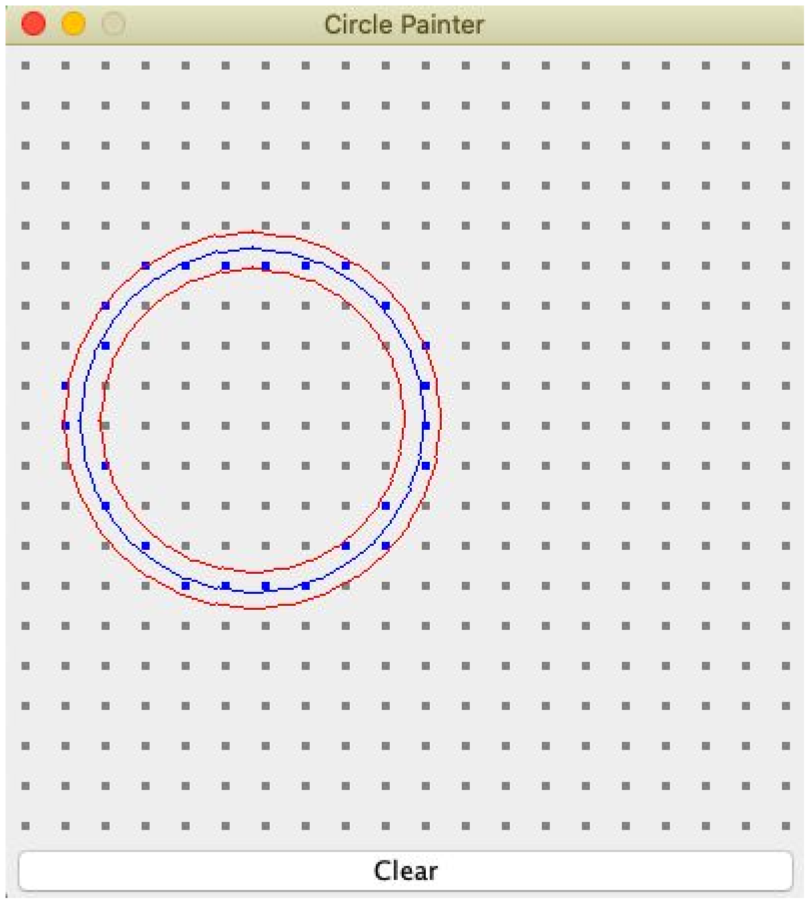
- ## Execution instructions

The project is developed with IntelliJ IDEA. In order to compile and execute it. You can import the project into IntelliJ and run. The main() function is in the GUI.java.

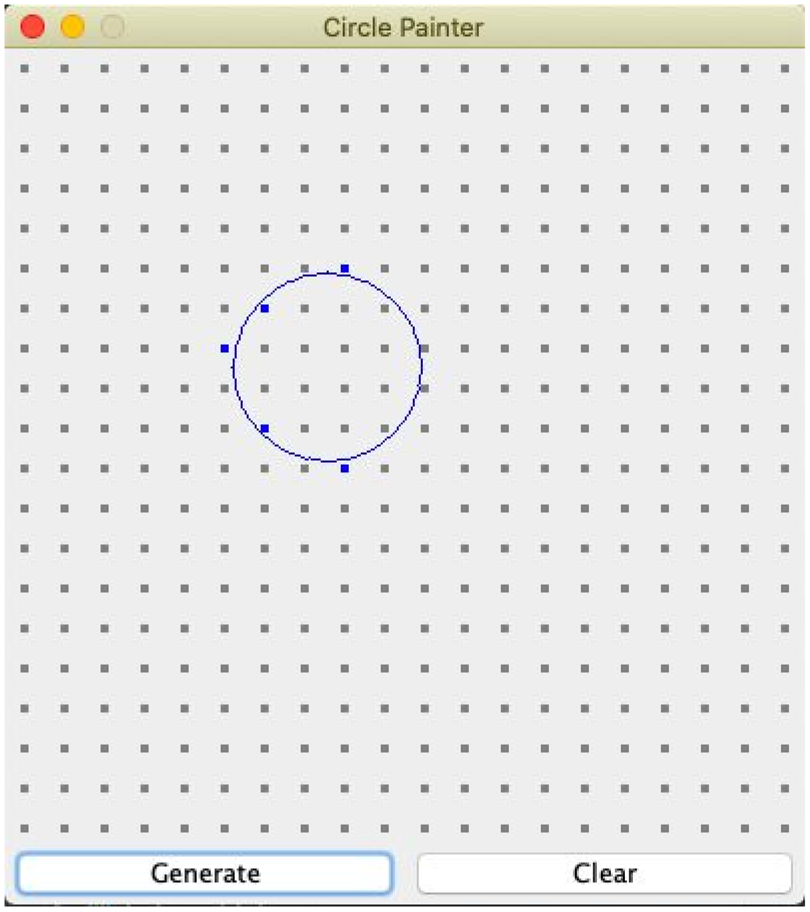
- ## Screenshots & Synopsis



When you execute the project, you will first be required to choose either the “Digitize” or “Generate” button.



If you choose the “Digitize” button, you will then be redirected to a 20*20 panel. All points are painted in gray. You can then click your mouse to choose the circle center and then drag the mouse to draw a circle. The circle is colored blue. The nearest points are also denoted blue. The largest and smallest circles are in color red. You can also click the “Clear” button to clear the panel.



If you click the “Generate” button, you will be redirected to another panel. There you can click on points to toggle them on or off. When a point is toggled on, it is blue. After you toggled on all the points you want, you can click the “Generate” button to generate the best fitting circle. The circle is colored blue. You can also click the “Clear” button to clear the panel.