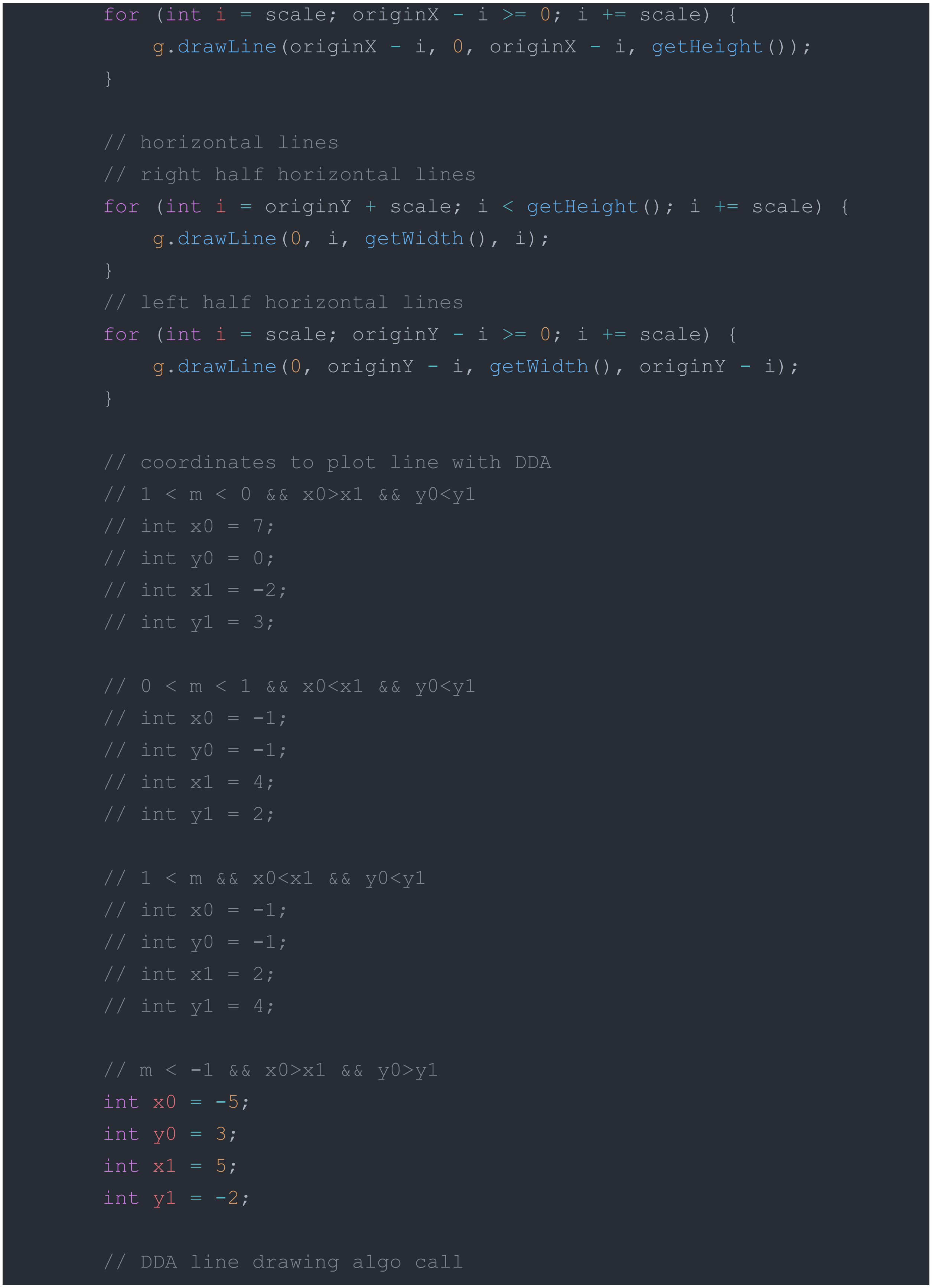
Assignment 2

By Neeraj Pratap Hazarika
2020csb040

Part- I: 1 week
1. Draw straight line using the following line drawing methods keeping the same grid structure in order
to view resolution for each case.
i) DDA ii) Bresenham's iii) Midpoint

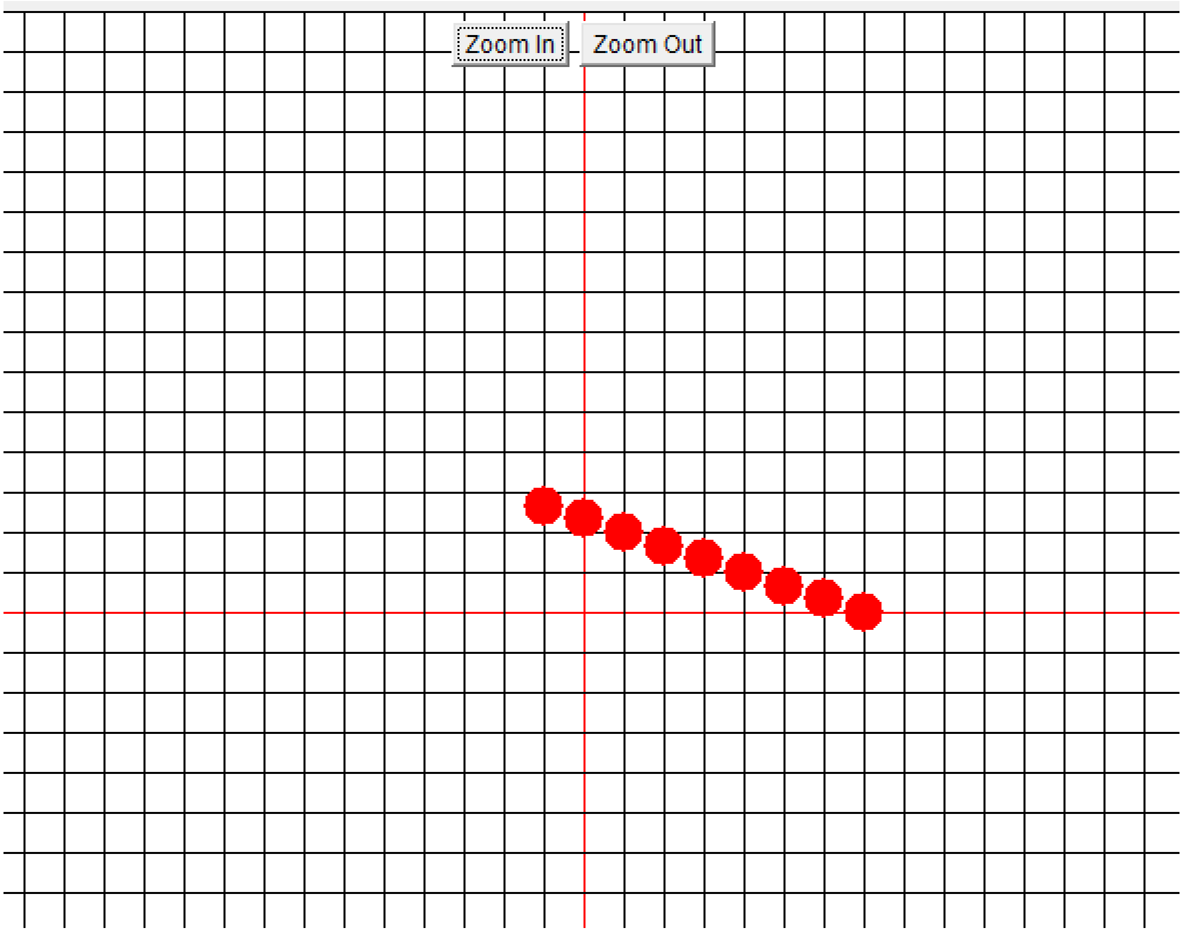
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

public class test extends Applet implements ActionListener {
    // scale (distance between two points in coordinate plane
    int scale = 20;

    public void init() {
        setBackground(Color.white);
        Button zoom_in = new Button("Zoom In");
        Button zoom_out = new Button("Zoom Out");
        add(zoom_in);
        add(zoom_out);
        zoom_in.addActionListener(this);
        zoom_out.addActionListener(this);
    }

    public void plotpoint(Graphics g, int x, int y, Color C) {
        int plotpoint_x = x - scale / 2; // shifting x coordinate to
appletcoordinate
        int plotpoint_y = y - scale / 2; // shifting y coordinate to
appletcoordinate
        g.setColor(C);
        g.fillOval(plotpoint_x, plotpoint_y, scale, scale);
    }

    public int round(float coordinate, int axis) {

        if (axis == 0) {
            int prev = (int) (coordinate / scale) * scale;
            int next = prev + scale;

            if (coordinate - prev < next - coordinate)
                return prev;
            else
                return next;
        } else {
            int prev = (int) (coordinate / scale) * scale;
            int next = prev + scale;

            if (coordinate - prev < next - coordinate)
                return prev;
            else
                return next;
        }
    }

    // DDA line drawing algorithm implementation
    /*
     * Calculate dx and dy, based on who difference is larger, that
coordinate will
     * increment by 1 each time while other will increase by slope
     */
    public void DDALine(Graphics g, int originX, int originY, int x0, int
y0, int x1, int y1) {

        // calculate dx and dy
        int dx = x1 - x0;
        int dy = y1 - y0;

        int step;

        // if dx > dy we will take step as dx
        // else we will take step as dy to draw the complete line
        if (Math.abs(dx) > Math.abs(dy))
            step = Math.abs(dx);
        else
            step = Math.abs(dy);
```

```java
        // calculate x-increment and y-increment for each step
        float x_incr = (float) dx / step;
        float y_incr = (float) dy / step;

        // take the initial points as x and y
        float x = originX + x0 * scale;
        float y = originY - y0 * scale;

        for (int i = 0; i < step; i++) {
            plotpoint(g, round(x, 0), round(y, 1), Color.red);
            x += x_incr * scale;
            y -= y_incr * scale;
        }
    }

    /*
     * Debugging
     * scale = 20, x_incr = 1, y_incr = 3/2 = 1.5
     * x = 0, y = 0
     * x = 1*20 = 20, y = 1.5*20 = 30
     */

    public void paint(Graphics g) {
        // shift the origin and put the coordinates in new variables
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;

        // drawing coordinates lines
        g.setColor(Color.red);
        g.drawLine(originX, 0, originX, getHeight());
        g.drawLine(0, originY, getWidth(), originY);

        // drawing Grid
        // vertical lines
        g.setColor(Color.black);
        // right half vertical lines
        for (int i = originX + scale; i < getWidth(); i += scale) {
            g.drawLine(i, 0, i, getHeight());
        }
        // left half vertical lines
```

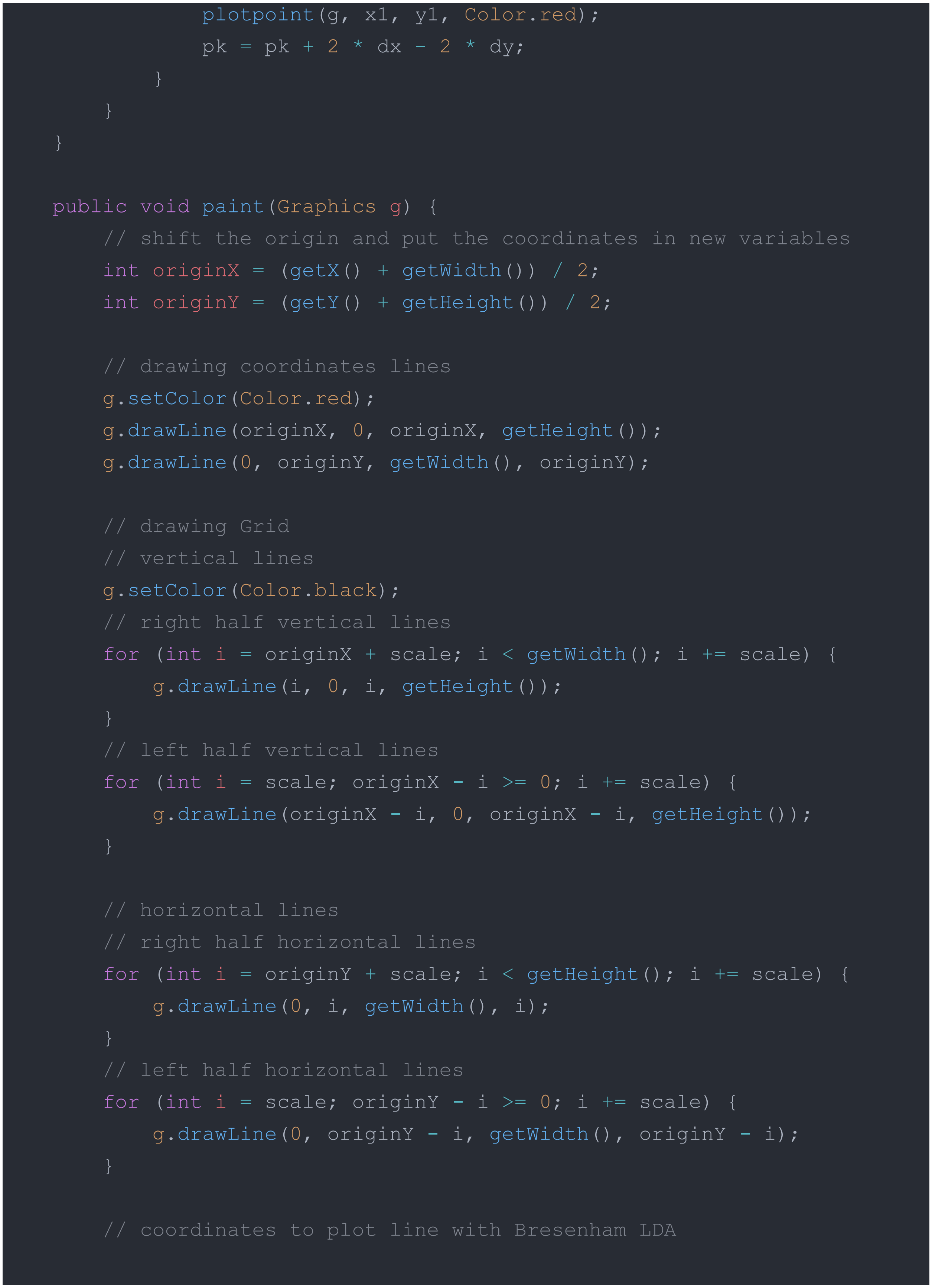


```
        for (int i = scale; originX - i >= 0; i += scale) {
            g.drawLine(originX - i, 0, originX - i, getHeight());
        }

        // horizontal lines
        // right half horizontal lines
        for (int i = originY + scale; i < getHeight(); i += scale) {
            g.drawLine(0, i, getWidth(), i);
        }
        // left half horizontal lines
        for (int i = scale; originY - i >= 0; i += scale) {
            g.drawLine(0, originY - i, getWidth(), originY - i);
        }

        // coordinates to plot line with DDA
        // 1 < m < 0 && x0>x1 && y0<y1
        // int x0 = 7;
        // int y0 = 0;
        // int x1 = -2;
        // int y1 = 3;

        // 0 < m < 1 && x0<x1 && y0<y1
        // int x0 = -1;
        // int y0 = -1;
        // int x1 = 4;
        // int y1 = 2;

        // 1 < m && x0<x1 && y0<y1
        // int x0 = -1;
        // int y0 = -1;
        // int x1 = 2;
        // int y1 = 4;

        // m < -1 && x0>x1 && y0>y1
        int x0 = -5;
        int y0 = 3;
        int x1 = 5;
        int y1 = -2;

        // DDA line drawing algo call
        DDALine(g, originX, originY, x0, y0, x1, y1);
    }

    public void actionPerformed(ActionEvent e) {
        String st = e.getActionCommand();
        if (st.equals("Zoom In"))
            scale += 4;
        else
            scale -= 4;
        repaint();
    }
}
```

Output:



```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

public class test extends Applet implements ActionListener {
    // scale (distance between two points in coordinate plane
    int scale = 20;

    public void init() {
        setBackground(Color.white);
        Button zoom_in = new Button("Zoom In");
        Button zoom_out = new Button("Zoom Out");
        add(zoom_in);
        add(zoom_out);
        zoom_in.addActionListener(this);
        zoom_out.addActionListener(this);
    }

    public void plotpoint(Graphics g, int x, int y, Color C) {
        int plotpoint_x = x - scale / 2; // shifting x coordinate to
appletcoordinate
        int plotpoint_y = y - scale / 2; // shifting y coordinate to
appletcoordinate
        g.setColor(C);
        g.fillOval(plotpoint_x, plotpoint_y, scale, scale);
    }

    // Brehensam line drawing algorithm implementation
    /*
     * - This algorithm is an extension to mid point algorithm for using
integer
     * arithmetic only
     * - acc. to it, D (Difference b/w f(x0,y0+1/2) and f(x0,y0)) = dy -
1/2 dx =
     * 2dy - dx (since only sign matters for considering case)
     * - if D is +ve => choose next point as (x0+1,y0+1) => change in D
will be => D
     * + 2dy - 2dx
     * - otherwise => choose next point as (x0+1,y0) => change in D will
be => D +
     * 2dy
     * - for it to work with negative slop we just need to switch the x0,
y0 with
     * x1,y1
     * - for it to work with slope > 1 we need to switch x and y
coordinates
     */
    public void bresenhamlow(Graphics g, int originX, int originY, int dy,
int dx, int x1, int y1, int x2, int y2) {

        // dy is always positive to make it compatible with algorithm (pk
value)
        if (dy < 0) {
            dy = Math.abs(dy);
        }

        // pk is initial decision making parameter
        int pk = 2 * dy - dx;

        x1 = originX + x1 * scale;
        y1 = originY - y1 * scale;
        x2 = originX + x2 * scale;
        y2 = originY - y2 * scale;

        plotpoint(g, x1, y1, Color.red);

        for (int i = 0; i < Math.abs(dx); i++) {

            x1 += scale;

            if (pk < 0) {
                plotpoint(g, x1, y1, Color.red);
                pk = pk + 2 * dy;
            } else {
                if (y1 < y2) {
                    y1 += scale;
                } else {
                    y1 -= scale;
                }
                plotpoint(g, x1, y1, Color.red);
                pk = pk + 2 * dy - 2 * dx;
            }
        }
    }

    public void bresenhamhigh(Graphics g, int originX, int originY, int
dy, int dx, int x1, int y1, int x2, int y2) {

        // dy is always positive to make it compatible with algorithm (pk
value)
        if (dx < 0) {
            dx = Math.abs(dx);
        }

        // pk is initial decision making parameter
        int pk = 2 * dx - dy;

        x1 = originX + x1 * scale;
        y1 = originY - y1 * scale;
        x2 = originX + x2 * scale;
        y2 = originY - y2 * scale;

        plotpoint(g, x1, y1, Color.red);

        for (int i = 0; i < Math.abs(dy); i++) {

            y1 -= scale; // subtracting scale to increase plotpoint along
applet coordinate

            if (pk < 0) {
                plotpoint(g, x1, y1, Color.red);
                pk = pk + 2 * dx;
            } else {
                if (x1 < x2) {
                    x1 += scale;
                } else {
                    x1 -= scale;
                }
```

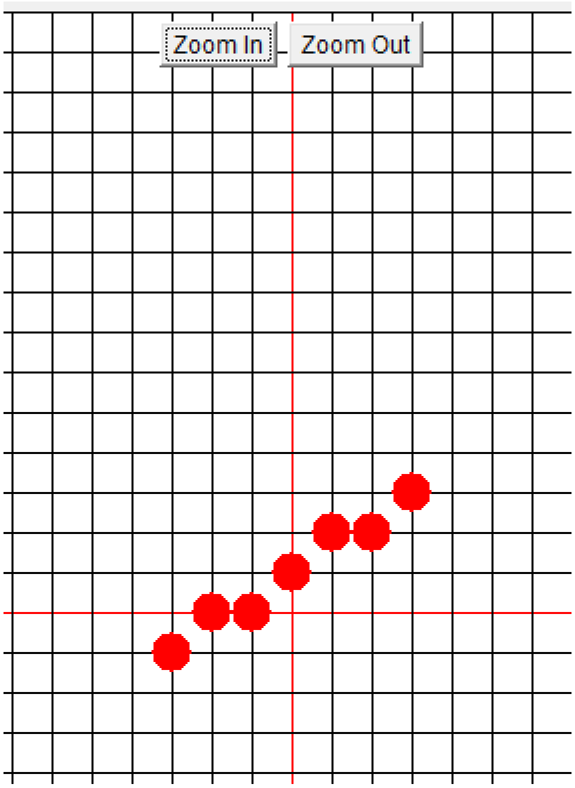


```
            plotpoint(g, x1, y1, Color.red);
            pk = pk + 2 * dx - 2 * dy;
        }
    }
}

public void paint(Graphics g) {
    // shift the origin and put the coordinates in new variables
    int originX = (getX() + getWidth()) / 2;
    int originY = (getY() + getHeight()) / 2;

    // drawing coordinates lines
    g.setColor(Color.red);
    g.drawLine(originX, 0, originX, getHeight());
    g.drawLine(0, originY, getWidth(), originY);

    // drawing Grid
    // vertical lines
    g.setColor(Color.black);
    // right half vertical lines
    for (int i = originX + scale; i < getWidth(); i += scale) {
        g.drawLine(i, 0, i, getHeight());
    }
    // left half vertical lines
    for (int i = scale; originX - i >= 0; i += scale) {
        g.drawLine(originX - i, 0, originX - i, getHeight());
    }

    // horizontal lines
    // right half horizontal lines
    for (int i = originY + scale; i < getHeight(); i += scale) {
        g.drawLine(0, i, getWidth(), i);
    }
    // left half horizontal lines
    for (int i = scale; originY - i >= 0; i += scale) {
        g.drawLine(0, originY - i, getWidth(), originY - i);
    }

    // coordinates to plot line with Bresenham LDA
```

```
        // m=1 && x0<x1 && y0<y1
        // int x0 = 1;
        // int y0 = 1;
        // int x1 = 3;
        // int y1 = 3;

        // m=1 && x0>x1 && y0>y1
        // int x0 = 3;
        // int y0 = 3;
        // int x1 = 1;
        // int y1 = 1;

        // m<1 && x0<x1 && y0>y1
        // int x0 = -3;
        // int y0 = 3;
        // int x1 = 3;
        // int y1 = -1;

        // m<1 && x0>x1 && y0>y1
        // int x0 = 3;
        // int y0 = 3;
        // int x1 = -3;
        // int y1 = -1;

        // m>1 && x0<x1 && y0>y1
        // int x0 = -1;
        // int y0 = -1;
        // int x1 = 2;
        // int y1 = 4;

        // m>1 && x0>x1 && y0>y1
        int x0 = 3;
        int y0 = 4;
        int x1 = -1;
        int y1 = -1;

        /*
         * 4 cases :
         * |m|>1 => y+=1, x+=0 or 1/2 (in case of y0>y1 switch coordinates x0,x1 &
         * y0,y1)
         * |m|<1 => x+=1, y+=0 or 1/2 (in case of x0>x1 switch coordinates
x0,x1 &
         * y0,y1)
         * we will only make the algo for |m|<1, it can also work for
|m|>1 by just
         * switch x and y axis
         */

        // bresenham line drawing algo call
        if (Math.abs(x1 - x0) >= Math.abs(y1 - y0)) {
            if (x0 <= x1)
                bresenhamlow(g, originX, originY, y1 - y0, x1 - x0, x0,
y0, x1, y1);
            else
                bresenhamlow(g, originX, originY, y0 - y1, x0 - x1, x1,
y1, x0, y0); // switching coordinates and

// switching

// slope coordinates
        } else {
            // originX and originY variables arent switched because only
theoritically we
            // switched coordinates for algorithms to plot points on right
position on
            // applet coordinate we need to use the originX for y
coordinates and originX
            // for x coordinates
            if (y0 <= y1)
                bresenhamhigh(g, originX, originY, y1 - y0, x1 - x0, x0,
y0, x1, y1);
            else
                bresenhamhigh(g, originX, originY, y0 - y1, x0 - x1, x1,
y1, x0, y0); // switching coordinates and

// switching

// slope coordinates
        }
```

```
    }

    public void actionPerformed(ActionEvent e) {
        String st = e.getActionCommand();
        if (st.equals("Zoom In"))
            scale += 4;
        else
            scale -= 4;
        repaint();
    }
}
```

Output :

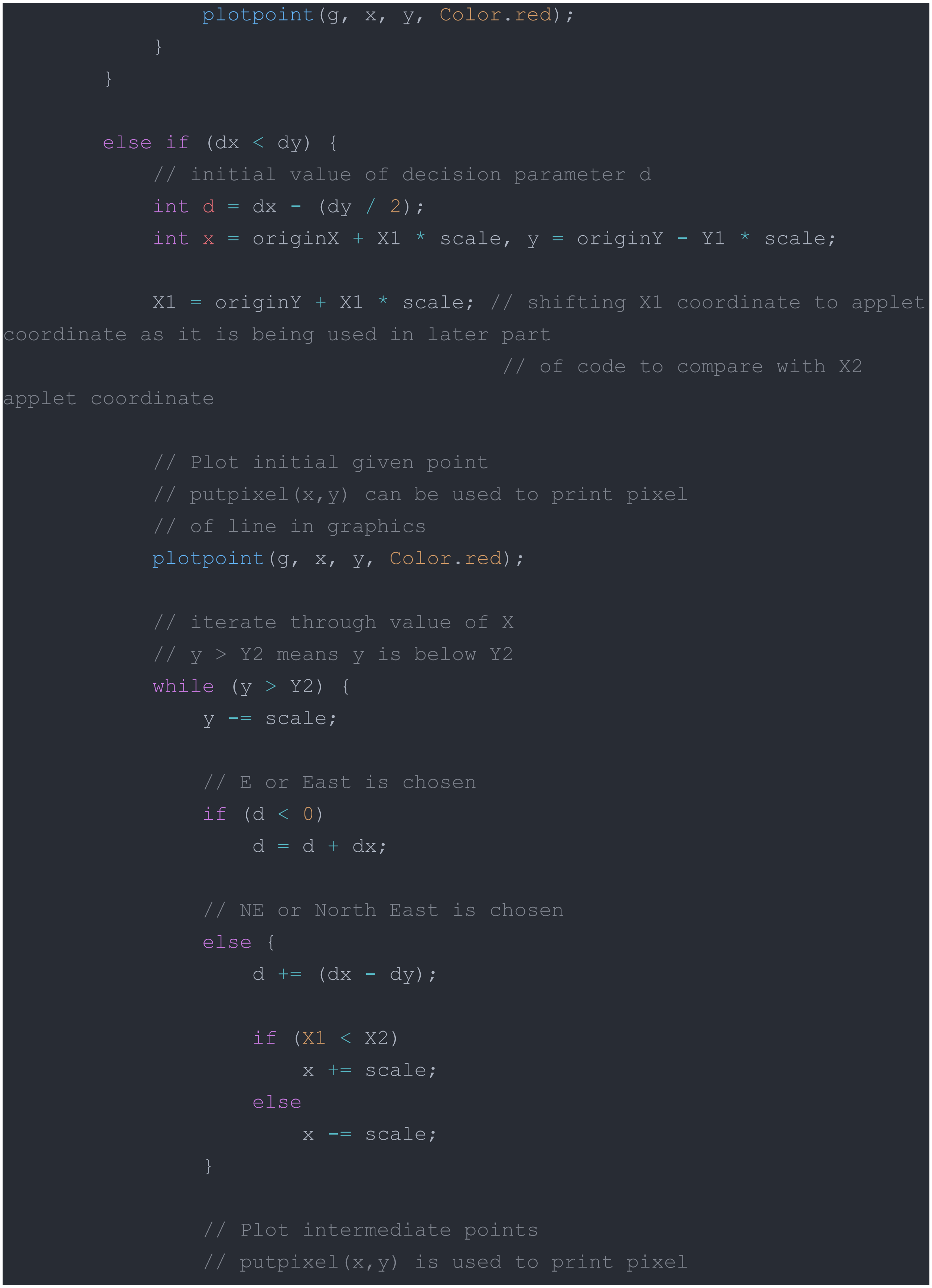


iii)

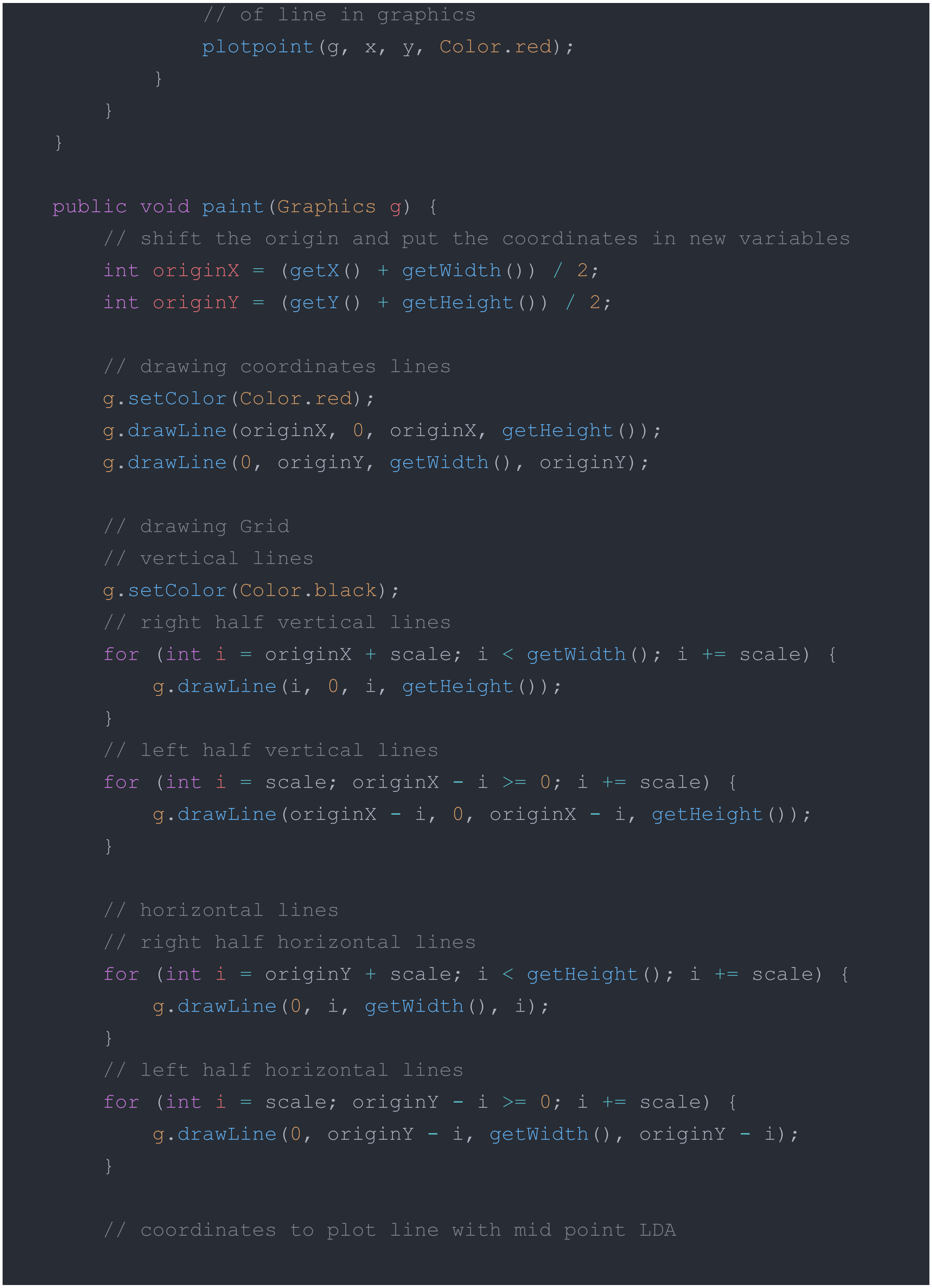
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;
```

```
public class test extends Applet implements ActionListener {
    // scale (distance between two points in coordinate plane
    int scale = 20;

    public void init() {
        setBackground(Color.white);
        Button zoom_in = new Button("Zoom In");
        Button zoom_out = new Button("Zoom Out");
        add(zoom_in);
        add(zoom_out);
        zoom_in.addActionListener(this);
        zoom_out.addActionListener(this);
    }

    public void plotpoint(Graphics g, int x, int y, Color C) {
        int plotpoint_x = x - scale / 2; // shifting x coordinate to
appletcoordinate
        int plotpoint_y = y - scale / 2; // shifting y coordinate to
appletcoordinate
        g.setColor(C);
        g.fillOval(plotpoint_x, plotpoint_y, scale, scale);
    }

    void midPoint(Graphics g, int originX, int originY, int X1, int Y1,
int X2, int Y2) {
        // calculate dx & dy

        int dx = X2 - X1;
        int dy = Y2 - Y1;

        // shifting coordinates to applet coordinate
        X2 = originX + X2 * scale;
        Y2 = originY - Y2 * scale;

        if (dy <= dx) {

            // keeping dy always positive for algorithm to be compatible
to work with Y1>Y2
            if (dy < 0)
```

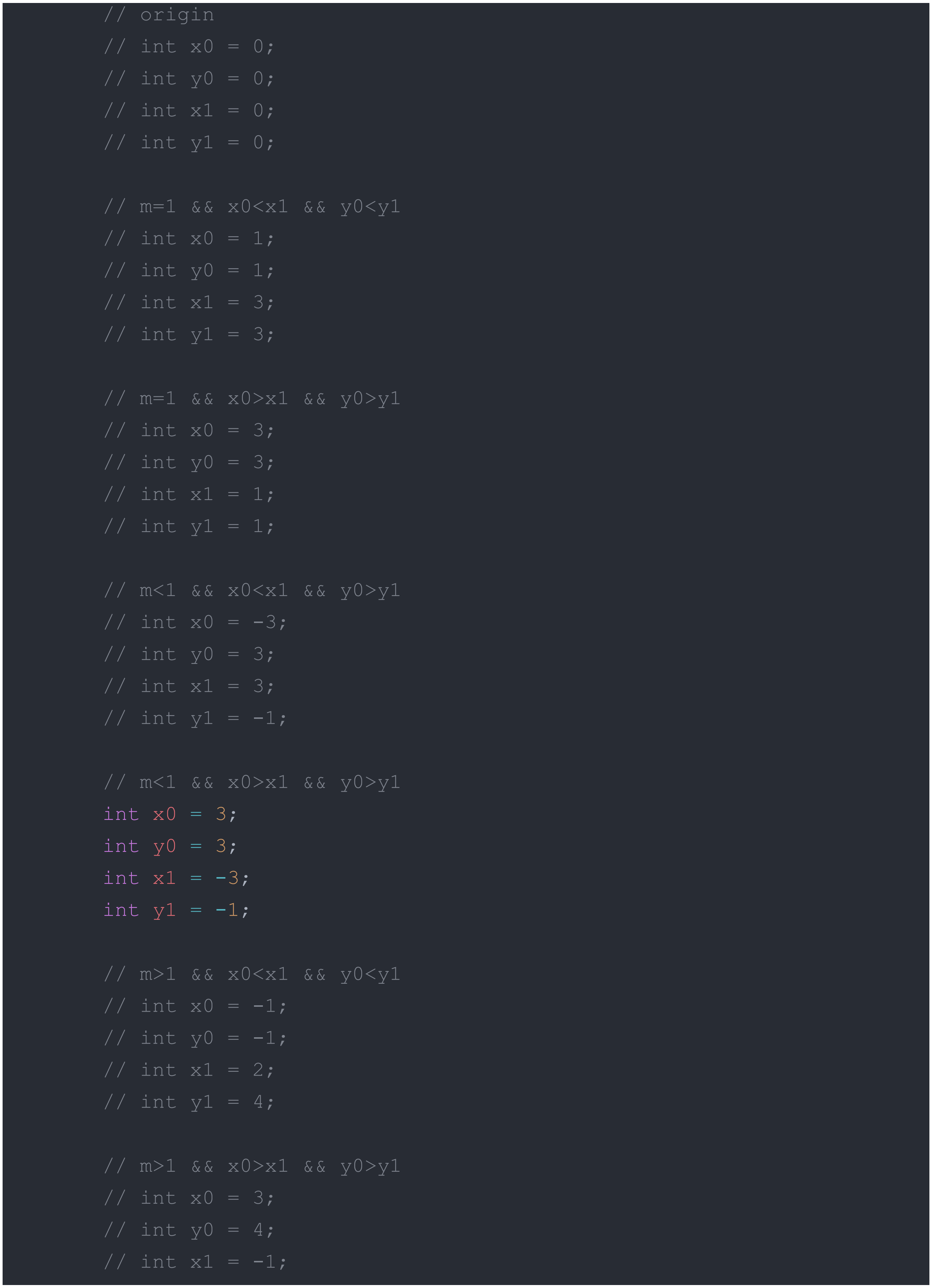
```
                dy = -dy;

            // initial value of decision parameter d
            int d = dy - (dx / 2);
            int x = originX + X1 * scale, y = originY - Y1 * scale;

            Y1 = originY - Y1 * scale; // shifting Y1 coordinate to applet
coordinate as it is being used in later part
                                        // of code to compare with Y2
applet coordinate

            // Plot initial given point
            // putpixel(x,y) can be used to print pixel
            // of line in graphics
            plotpoint(g, x, y, Color.red);

            // iterate through value of X
            while (x < X2) {
                x += scale;

                // E or East is chosen
                if (d < 0)
                    d = d + dy;

                // NE or North East is chosen
                else {
                    d += (dy - dx);

                    if (Y1 > Y2) // Y1 & Y2 have already been shifted to
applet coordinate, so Y1>Y2 means Y1 is
                                    // below Y2
                        y -= scale; // decrease y with scale make y go
down in applet coordinate
                    else
                        y += scale;
                }

                // Plot intermediate points
                // putpixel(x,y) is used to print pixel
                // of line in graphics
```



```
                plotpoint(g, x, y, Color.red);
            }
        }

        else if (dx < dy) {
            // initial value of decision parameter d
            int d = dx - (dy / 2);
            int x = originX + X1 * scale, y = originY - Y1 * scale;

            X1 = originY + X1 * scale; // shifting X1 coordinate to applet
coordinate as it is being used in later part
                                        // of code to compare with X2
applet coordinate

            // Plot initial given point
            // putpixel(x,y) can be used to print pixel
            // of line in graphics
            plotpoint(g, x, y, Color.red);

            // iterate through value of X
            // y > Y2 means y is below Y2
            while (y > Y2) {
                y -= scale;

                // E or East is chosen
                if (d < 0)
                    d = d + dx;

                // NE or North East is chosen
                else {
                    d += (dx - dy);

                    if (X1 < X2)
                        x += scale;
                    else
                        x -= scale;
                }

                // Plot intermediate points
                // putpixel(x,y) is used to print pixel
```



```
            // of line in graphics
            plotpoint(g, x, y, Color.red);
        }
    }
}

public void paint(Graphics g) {
    // shift the origin and put the coordinates in new variables
    int originX = (getX() + getWidth()) / 2;
    int originY = (getY() + getHeight()) / 2;

    // drawing coordinates lines
    g.setColor(Color.red);
    g.drawLine(originX, 0, originX, getHeight());
    g.drawLine(0, originY, getWidth(), originY);

    // drawing Grid
    // vertical lines
    g.setColor(Color.black);
    // right half vertical lines
    for (int i = originX + scale; i < getWidth(); i += scale) {
        g.drawLine(i, 0, i, getHeight());
    }
    // left half vertical lines
    for (int i = scale; originX - i >= 0; i += scale) {
        g.drawLine(originX - i, 0, originX - i, getHeight());
    }

    // horizontal lines
    // right half horizontal lines
    for (int i = originY + scale; i < getHeight(); i += scale) {
        g.drawLine(0, i, getWidth(), i);
    }
    // left half horizontal lines
    for (int i = scale; originY - i >= 0; i += scale) {
        g.drawLine(0, originY - i, getWidth(), originY - i);
    }

    // coordinates to plot line with mid point LDA
```

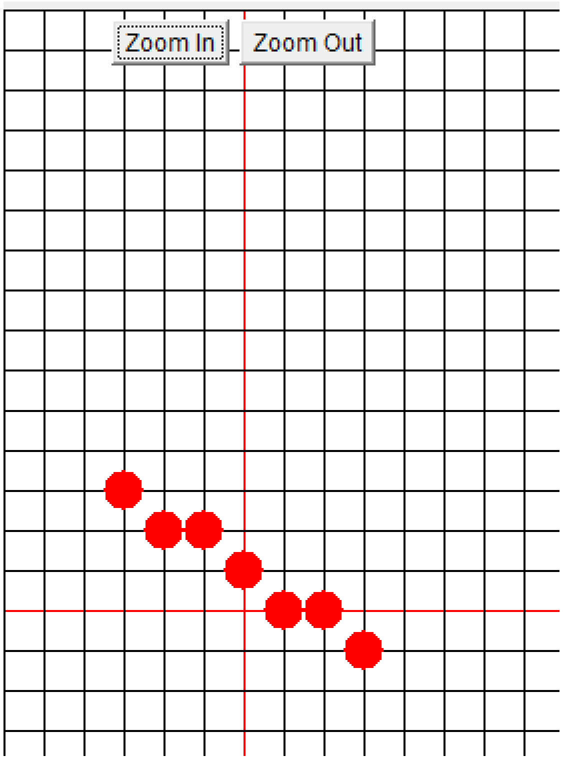


```
    // origin
    // int x0 = 0;
    // int y0 = 0;
    // int x1 = 0;
    // int y1 = 0;

    // m=1 && x0<x1 && y0<y1
    // int x0 = 1;
    // int y0 = 1;
    // int x1 = 3;
    // int y1 = 3;

    // m=1 && x0>x1 && y0>y1
    // int x0 = 3;
    // int y0 = 3;
    // int x1 = 1;
    // int y1 = 1;

    // m<1 && x0<x1 && y0>y1
    // int x0 = -3;
    // int y0 = 3;
    // int x1 = 3;
    // int y1 = -1;

    // m<1 && x0>x1 && y0>y1
    int x0 = 3;
    int y0 = 3;
    int x1 = -3;
    int y1 = -1;

    // m>1 && x0<x1 && y0<y1
    // int x0 = -1;
    // int y0 = -1;
    // int x1 = 2;
    // int y1 = 4;

    // m>1 && x0>x1 && y0>y1
    // int x0 = 3;
    // int y0 = 4;
    // int x1 = -1;
```

```
        // int y1 = -1;

        /*
         * 4 cases :
         * |m|>1 => y+=1, x+=0 or 1/2 (in case of y0>y1 switch coordinates
x0,x1 &
         * y0,y1)
         * |m|<1 => x+=1, y+=0 or 1/2 (in case of x0>x1 switch coordinates
x0,x1 &
         * y0,y1)
         * we will only make the algo for |m|<1, it can also work for
|m|>1 by just
         * switch x and y axis
         */

        // midpoint line drawing algo call
        if (Math.abs(x1 - x0) >= Math.abs(y1 - y0)) {
            if (x0 <= x1)
                midPoint(g, originX, originY, x0, y0, x1, y1);
            else
                midPoint(g, originX, originY, x1, y1, x0, y0); //
switching coordinates and
                                                                 //
switching
                                                                 // slope
coordinates
        } else {
            // originX and originY variables arent switched because only
theoritically we
            // switched coordinates for algorithms to plot points on right
position on
            // applet coordinate we need to use the originX for y
coordinates and originX
            // for x coordinates
            if (y0 <= y1)
                midPoint(g, originX, originY, x0, y0, x1, y1);
            else
                midPoint(g, originX, originY, x1, y1, x0, y0); //
switching coordinates and
```

```
                                                                //
switching
                                                                // slope
coordinates
        }
    }

    public void actionPerformed(ActionEvent e) {
        String st = e.getActionCommand();
        if (st.equals("Zoom In"))
            scale += 4;
        else
            scale -= 4;
        repaint();
    }
}
```

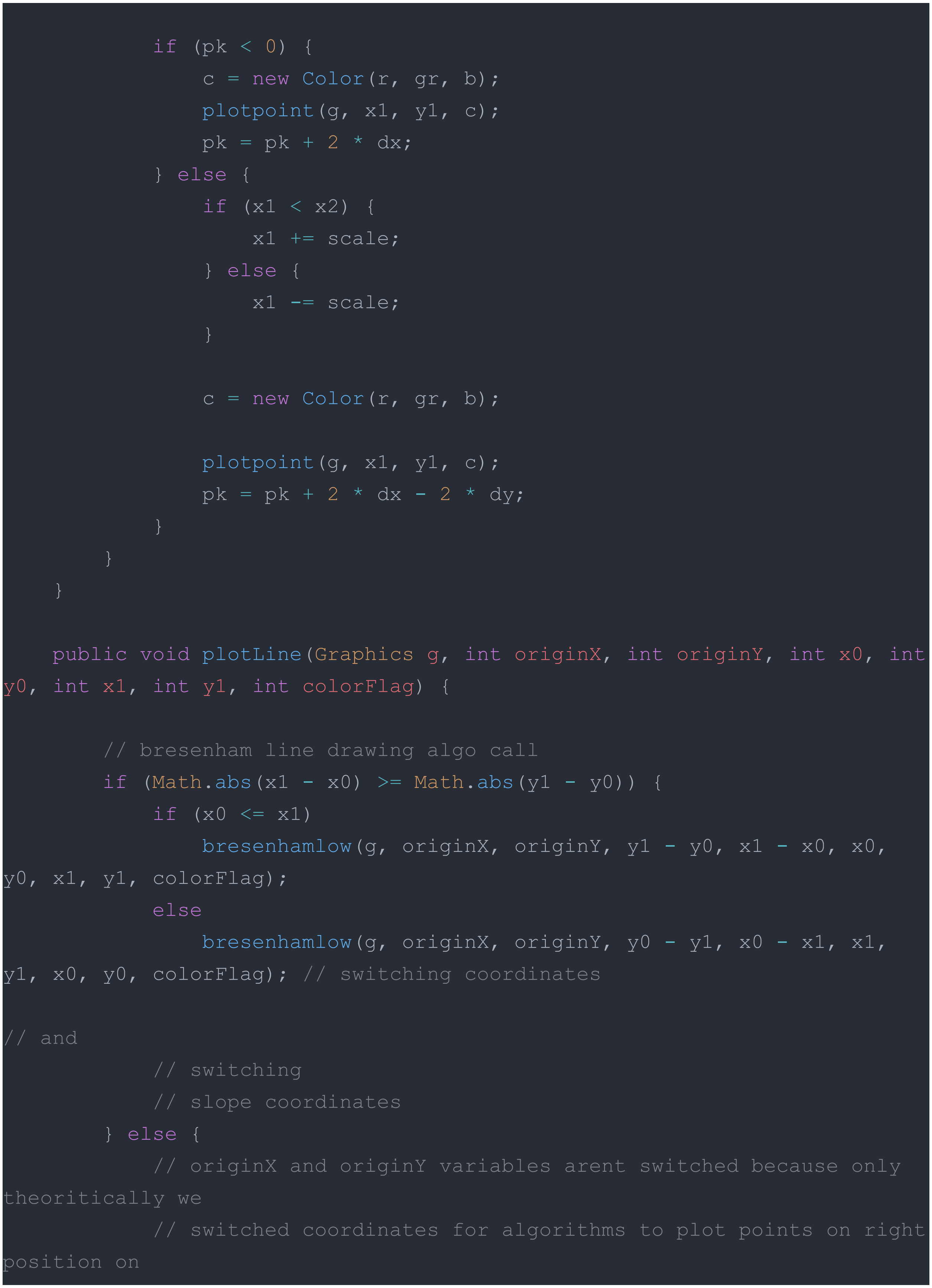
Output :



Part- II: 1 week

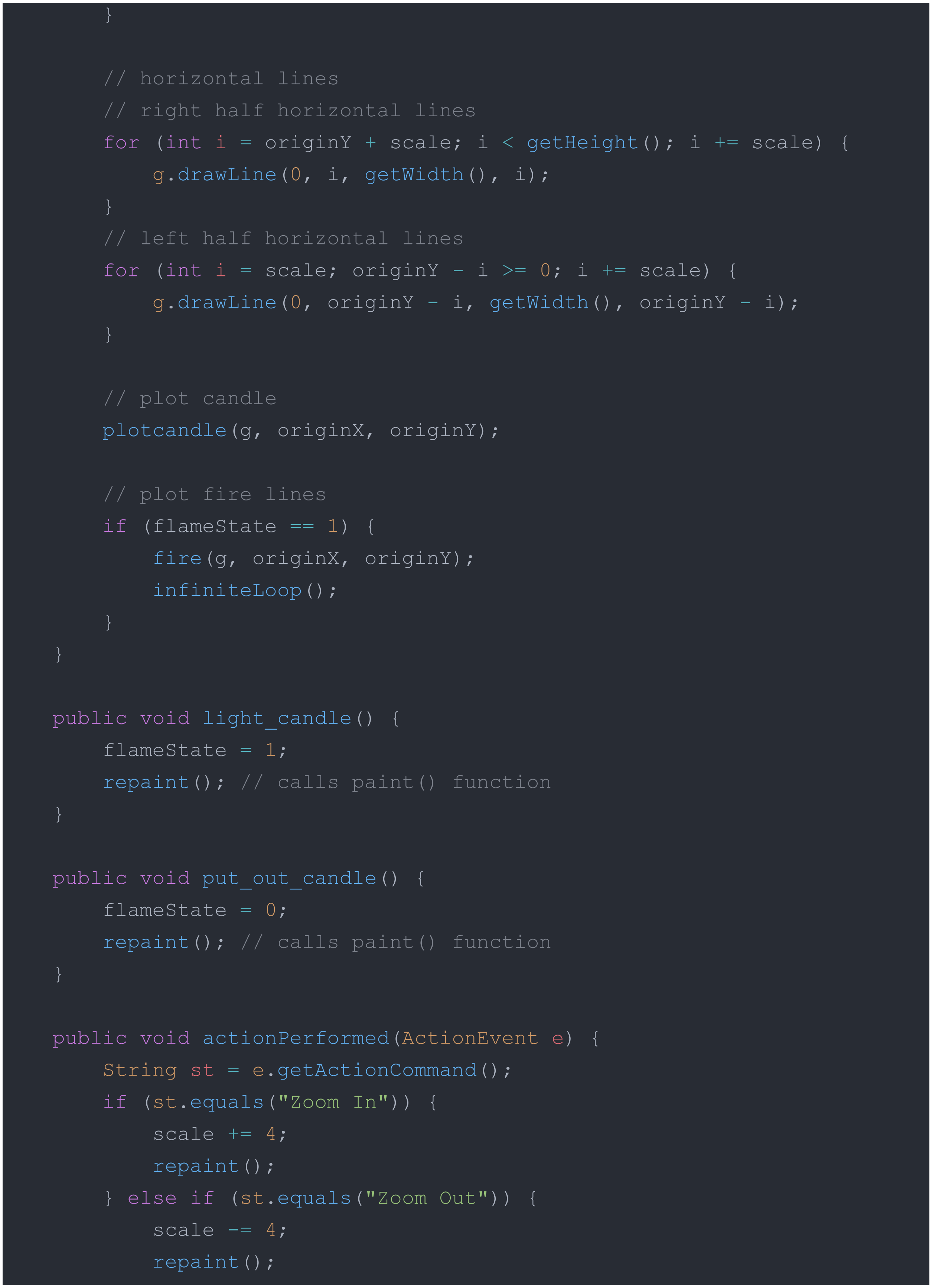
2. (a) Prepare a class ‘Fire’ following instructions below.
i. Fire (Fig. 2) is created by collection of straight lines which are very closed together.
ii. Use any line drawing algorithm that is implemented in Part-I, Assignment 2.
iii. Height of the straight lines change over time by changing endpoints away from the source of fire
iv. Colour of fire may vary as the flame is away from the source.
(b) Hence create a class ‘Candle’ (Fig. 3) having at least two methods light_candle () and put_out_candle ()

```java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

public class test extends Applet implements ActionListener {
    // scale (distance between two points in coordinate plane
    int scale = 20;
    int flameState = 0;

    public void init() {
        setBackground(Color.black);
        Button zoom_in = new Button("Zoom In");
        Button zoom_out = new Button("Zoom Out");
        Button light_candle = new Button("Light Candle");
        Button put_out_candle = new Button("Put Out Candle");
        add(zoom_in);
        add(zoom_out);
        add(light_candle);
        add(put_out_candle);
        zoom_in.addActionListener(this);
        zoom_out.addActionListener(this);
        light_candle.addActionListener(this);
        put_out_candle.addActionListener(this);
    }

    public void plotpoint(Graphics g, int x, int y, Color C) {
        int plotpoint_x = x - scale / 2; // shifting x coordinate to
appletcoordinate
        int plotpoint_y = y - scale / 2; // shifting y coordinate to
appletcoordinate
        g.setColor(C);
        g.fillOval(plotpoint_x, plotpoint_y, scale, scale);
    }

    public void bresenhamlow(Graphics g, int originX, int originY, int dy,
int dx, int x1, int y1, int x2, int y2,
            int colorFlag) {

        // dy is always positive to make it compatible with algorithm (pk
value)
        if (dy < 0) {
            dy = Math.abs(dy);
        }

        // pk is initial decision making parameter
        int pk = 2 * dy - dx;

        int y_color = y1;

        x1 = originX + x1 * scale;
        y1 = originY - y1 * scale;
        x2 = originX + x2 * scale;
        y2 = originY - y2 * scale;

        int r = 255, gr = 255, b = 255;

        Color c = new Color(r, gr, b);

        plotpoint(g, x1, y1, c);

        for (int i = 0; i < Math.abs(dx); i++) {

            x1 += scale;

            if (pk < 0) {
                c = new Color(r, gr, b);
                plotpoint(g, x1, y1, c);
                pk = pk + 2 * dy;
            } else {
                if (y1 < y2) {
                    y1 += scale;
                } else {
                    y1 -= scale;
                }

                // assuming y1<y2 always for fire plot
                y_color += 1;

                if (colorFlag == 1) {
                    if (y_color <= 8) {
                        r = 255;
                        gr = 255;
                        b = 255 - (y_color * 31);
                    } else if (y_color > 8 && y_color <= 16) {
                        r = 255;
                        gr = 255 - ((y_color - 8) * 31);
                        b = 0;
                    } else {
                        r = 255;
                        gr = 0;
                        b = 0;
                    }
                }

                c = new Color(r, gr, b);

                plotpoint(g, x1, y1, c);
                pk = pk + 2 * dy - 2 * dx;
            }
        }
    }

    public void bresenhamhigh(Graphics g, int originX, int originY, int
dy, int dx, int x1, int y1, int x2, int y2,
            int colorFlag) {

        // dy is always positive to make it compatible with algorithm (pk
value)
        if (dx < 0) {
            dx = Math.abs(dx);
        }
        // pk is initial decision making parameter
        int pk = 2 * dx - dy;

        int y_color = y1;

        x1 = originX + x1 * scale;
        y1 = originY - y1 * scale;
        x2 = originX + x2 * scale;
        y2 = originY - y2 * scale;

        int r = 255, gr = 255, b = 255;

        Color c = new Color(r, gr, b);

        plotpoint(g, x1, y1, c);

        for (int i = 0; i < Math.abs(dy); i++) {

            y1 -= scale; // subtracting scale to increase plotpoint along applet coordinate

            // assuming y1<y2 always for fire
            y_color += 1;

            if (colorFlag == 1) {
                if (y_color <= 8) {
                    r = 255;
                    gr = 255;
                    b = 255 - (y_color * 31);
                } else if (y_color > 8 && y_color <= 16) {
                    r = 255;
                    gr = 255 - ((y_color - 8) * 31);
                    b = 0;
                } else {
                    r = 255;
                    gr = 0;
                    b = 0;
                }
            }
```



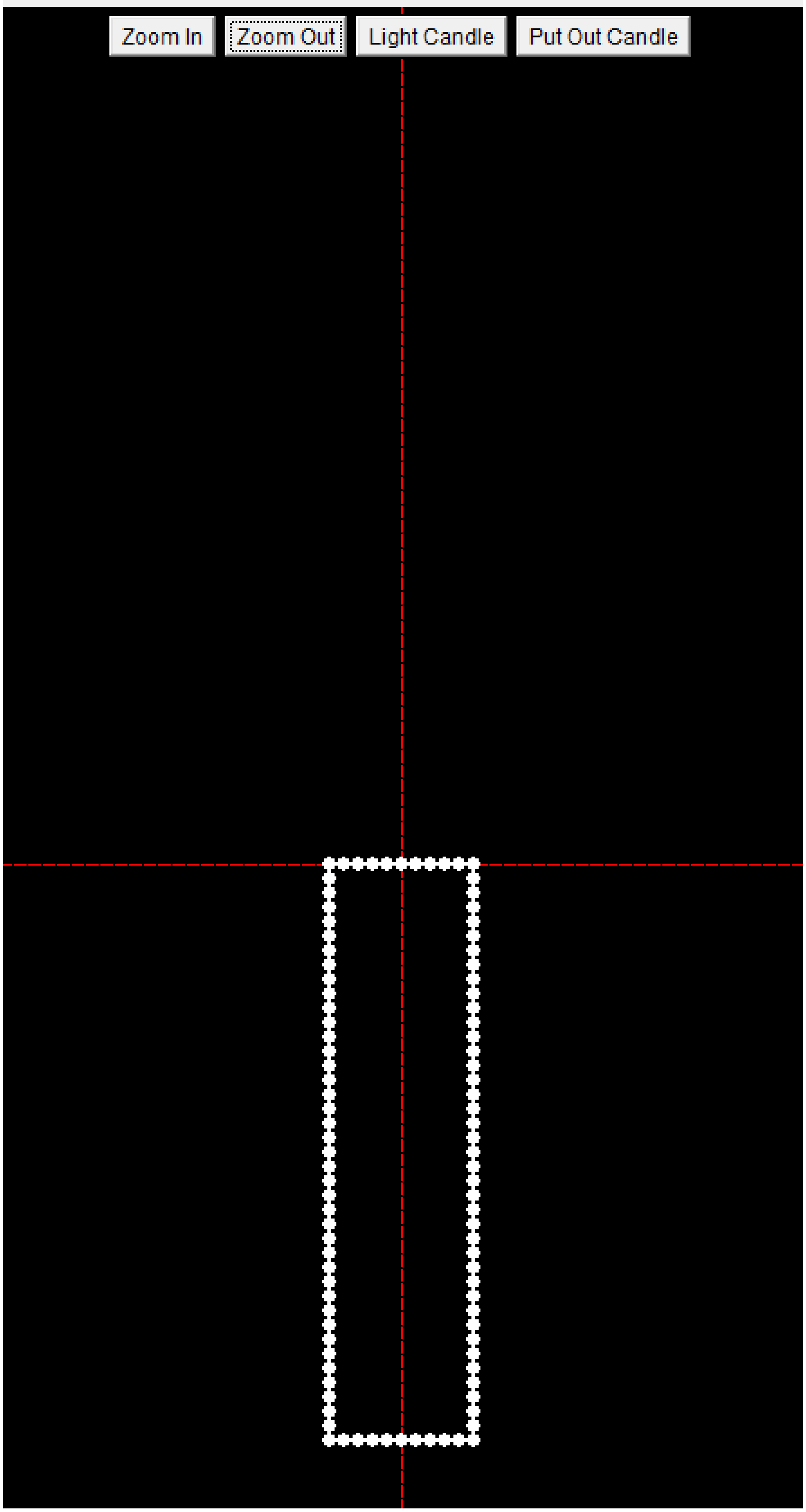
```
            if (pk < 0) {
                c = new Color(r, gr, b);
                plotpoint(g, x1, y1, c);
                pk = pk + 2 * dx;
            } else {
                if (x1 < x2) {
                    x1 += scale;
                } else {
                    x1 -= scale;
                }

                c = new Color(r, gr, b);

                plotpoint(g, x1, y1, c);
                pk = pk + 2 * dx - 2 * dy;
            }
        }
    }

    public void plotLine(Graphics g, int originX, int originY, int x0, int
y0, int x1, int y1, int colorFlag) {

        // bresenham line drawing algo call
        if (Math.abs(x1 - x0) >= Math.abs(y1 - y0)) {
            if (x0 <= x1)
                bresenhamlow(g, originX, originY, y1 - y0, x1 - x0, x0,
y0, x1, y1, colorFlag);
            else
                bresenhamlow(g, originX, originY, y0 - y1, x0 - x1, x1,
y1, x0, y0, colorFlag); // switching coordinates

// and
            // switching
            // slope coordinates
        } else {
            // originX and originY variables arent switched because only
theoritically we
            // switched coordinates for algorithms to plot points on right
position on
```

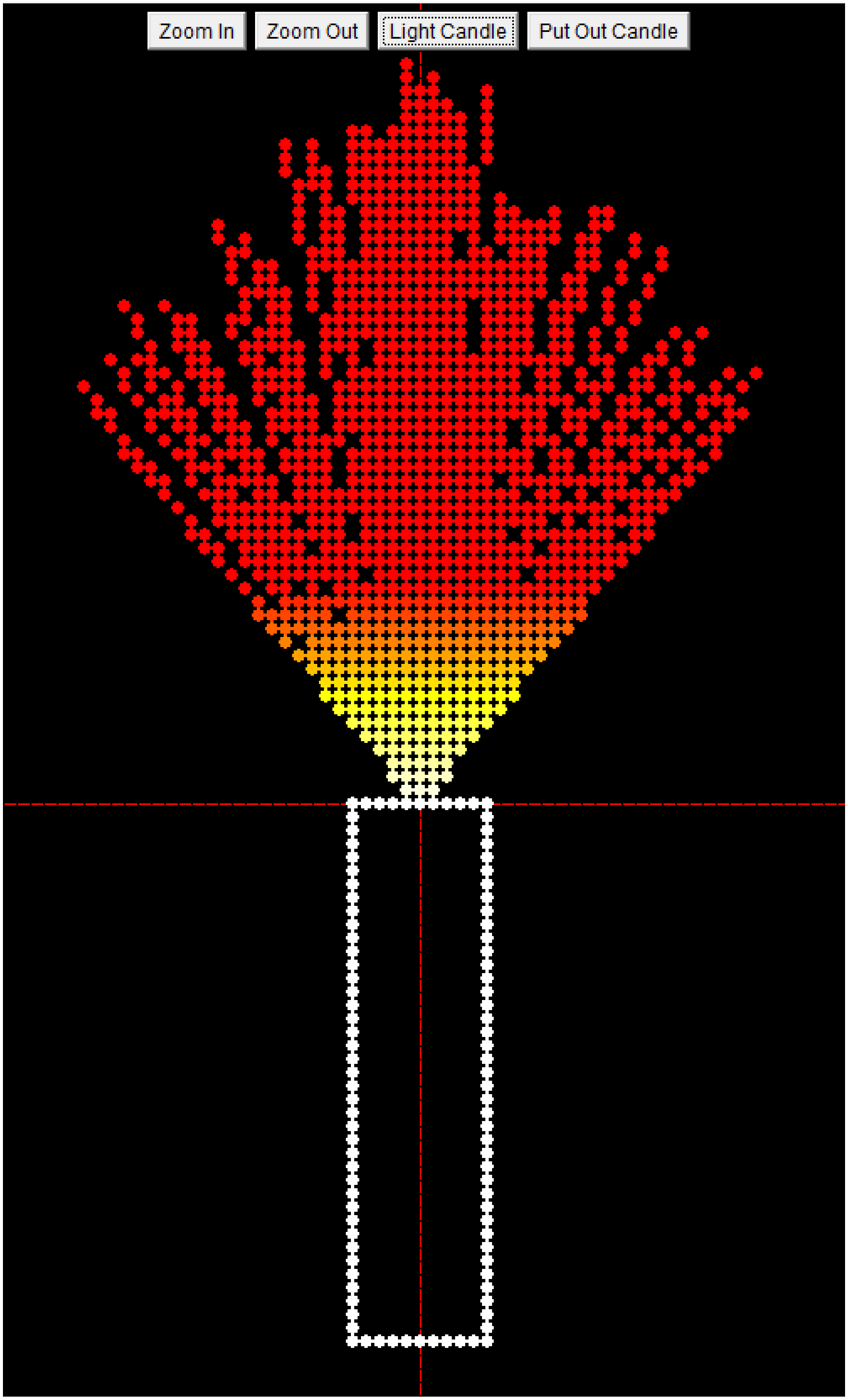
```
            // applet coordinate we need to use the originX for y
coordinates and originX
            // for x coordinates
            if (y0 <= y1)
                bresenhamhigh(g, originX, originY, y1 - y0, x1 - x0, x0,
y0, x1, y1, colorFlag);
            else
                bresenhamhigh(g, originX, originY, y0 - y1, x0 - x1, x1,
y1, x0, y0, colorFlag); // switching

// coordinates and
            // switching
            // slope coordinates
        }
    }

    public void plotRect(Graphics g, int originX, int originY, int rtlxc,
int rtlyc, int rtrxc, int rtryc, int rblxc,
            int rblyc, int rbrxc, int rbryc) {

        plotLine(g, originX, originY, rtlxc, rtlyc, rtrxc, rtryc, 0);
        plotLine(g, originX, originY, rtrxc, rtryc, rbrxc, rbryc, 0);
        plotLine(g, originX, originY, rbrxc, rbryc, rblxc, rblyc, 0);
        plotLine(g, originX, originY, rblxc, rblyc, rtlxc, rtlyc, 0);
    }

    /*
     * Fire algorithm
     * - plot points in 100 directions, -50 to 0, 0 to 50.
     * - based on the value of x, we choose y for plotting the line.
     * - Points close to the source are different in color than points far
away.
     * - color is changed intenally in plotline while plotting the points
     */

    public void fire(Graphics g, int originX, int originY) {

        for (int i = -25; i <= 25; i++) {
            int y = 50;
            if (Math.abs(y) >= Math.abs(i)) {
                y = y - Math.abs(i) + (int) (Math.random() * 10);
                plotLine(g, originX, originY, 0, 0, i, y, 1);
            }
        }

    }

    public void plotcandle(Graphics g, int originX, int originY) {
        plotRect(g, originX, originY, -5, 0, 5, 0, -5, -40, 5, -40);
    }

    public void infiniteLoop() {
        try {
            Thread.sleep(100);
        } catch (Exception e) {

        }
        repaint();
    }

    public void paint(Graphics g) {
        // shift the origin and put the coordinates in new variables
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;

        // drawing coordinates lines
        g.setColor(Color.red);
        g.drawLine(originX, 0, originX, getHeight());
        g.drawLine(0, originY, getWidth(), originY);

        // drawing Grid
        // vertical lines
        g.setColor(Color.black);
        // right half vertical lines
        for (int i = originX + scale; i < getWidth(); i += scale) {
            g.drawLine(i, 0, i, getHeight());
        }
        // left half vertical lines
        for (int i = scale; originX - i >= 0; i += scale) {
            g.drawLine(originX - i, 0, originX - i, getHeight());
```



```
        }

        // horizontal lines
        // right half horizontal lines
        for (int i = originY + scale; i < getHeight(); i += scale) {
            g.drawLine(0, i, getWidth(), i);
        }
        // left half horizontal lines
        for (int i = scale; originY - i >= 0; i += scale) {
            g.drawLine(0, originY - i, getWidth(), originY - i);
        }

        // plot candle
        plotcandle(g, originX, originY);

        // plot fire lines
        if (flameState == 1) {
            fire(g, originX, originY);
            infiniteLoop();
        }
    }

    public void light_candle() {
        flameState = 1;
        repaint(); // calls paint() function
    }

    public void put_out_candle() {
        flameState = 0;
        repaint(); // calls paint() function
    }

    public void actionPerformed(ActionEvent e) {
        String st = e.getActionCommand();
        if (st.equals("Zoom In")) {
            scale += 4;
            repaint();
        } else if (st.equals("Zoom Out")) {
            scale -= 4;
            repaint();
        } else if (st.equals("Light Candle")) {
            light_candle();
        } else {
            put_out_candle();
        }
    }
}
```

Output :


Zoom In
Zoom Out
Light Candle
Put Out Candle


Zoom In
Zoom Out
Light Candle
Put Out Candle