```
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Random;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BubbleBurst {

   private Random rand = new Random();

   private  final int size = 50;
   private  final int ROUNDS = 10;
   private  final int nbor = 18;
   private  final int RADIUS = 30;


   public static void main(String[] args) {
      new BubbleBurst();
   }

   public BubbleBurst() {
      frame1 = new Frame1();
      frame1.setVisible(true);
      frame2 = new Frame2(frame1);
   }

   private Frame1 frame1;
   private Frame2 frame2;

   private class Frame1 extends JFrame {


      public Frame1() {
```

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setTitle("Bubble Burst Game");
setLayout(new FlowLayout(FlowLayout.CENTER));
setPreferredSize(new Dimension(250, 150));



JPanel sPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

JSlider difficultyBar = new JSlider(JSlider.HORIZONTAL, 4, 6, 4);
difficultyBar.setPaintLabels(true);
difficultyBar.setPaintTicks(true);

Hashtable<Integer, JLabel> lTable = new Hashtable<>();
JPanel Panel = new JPanel(new FlowLayout(FlowLayout.CENTER));
lTable.put(4, new JLabel("Easy 4"));
JButton Start = new JButton("Start");
lTable.put(5, new JLabel("Medium 5"));
Panel.add(Start);
lTable.put(6, new JLabel("Hard 6"));
JButton Restart = new JButton("Restart");
sPanel.add(difficultyBar);
difficultyBar.setLabelTable(lTable);
Panel.add(Restart);

add(Panel);
add(sPanel);



Start.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        frame2.StartGame(difficultyBar.getValue());
    }
```

```
        });

        Restart.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                frame2.RestartGame(difficultyBar.getValue());
            }
        });

        pack();
        setLocationRelativeTo(null);
    }
}

private class Frame2 extends JFrame {
    private int cRound = 1;
    private int stage;
    private GP gp;
    private ArrayList<Bubble> bubbles = new ArrayList<>();
    private Timer bubbleTimer;
    private final int delay = 2000;
    private boolean Check = true;


    public Frame2(Frame1 frame1) {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(false);
        setTitle("Play Ground");
        gp = new GP();
        add(gp);
        pack();
        setLocationRelativeTo(null);

    }
```

```
public void StartGame(int stage) {
    frame1.setVisible(false);
    setVisible(true);
    this.stage = stage;
    gp.StartNewRound();
}

public void RestartGame(int stage) {
    frame1.setVisible(false);
    setVisible(true);
    this.stage = stage;
    cRound = 1;
    gp.RestartGame();
}

private class GP extends JPanel {


    public GP() {
        setBackground(Color.WHITE);
        setPreferredSize(new Dimension(400, 400));
        bubbleTimer = new Timer(delay, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                update();
            }
        });

        bubbleTimer.start();

        setLayout(new BorderLayout());

        addMouseListener(new BubbleMouseListener());
```

```
}

class BubbleMouseListener extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (!Check) {
            boolean noOverlap = true;
            for (Bubble bubble : bubbles) {
                if (bubble.has(e.getPoint()) || bubble.overlap(e.getPoint())) {
                    noOverlap = false;
                    break;
                }
            }

            if (noOverlap) {
                gameOver("Game over, You lost.");
            } else {
                burstBubble(e.getPoint());
            }


        } else {
            Point point = e.getPoint();
            boolean noOverlap = true;
            for (Bubble b : bubbles) {
                if (b.has(point) || b.overlap(point)) {
                    noOverlap = false;
                    break;
                }
            }

            if (inside(point) && noOverlap) {
                put(point);
            } else {
```

```
                    JOptionPane.showMessageDialog(GP.this, "Bubble overlap detected.");
                }

            }
        }
    }

    private void gameOver(String message) {
        JOptionPane.showMessageDialog(this, message);
        frame1.setVisible(true);
        Frame2.this.setVisible(false);
        resetGame();
    }

    private void resetGame() {
        cRound = 1;
        bubbles.clear();
        Check = true;
        repaint();
    }

    private boolean inside(Point point) {

        return  (RADIUS / 2) <= point.x && (RADIUS / 2) <= point.y && getWidth() - (RADIUS / 2) >= point.x && getHeight() - (RADIUS / 2) >= point.y;
    }

    public void StartNewRound() {
        Check = true;
        bubbles.forEach(bubble -> {
            bubble.burst=false;
            bubble.nborSize=size + nbor * (cRound - 1);
        });

        repaint();
```

```
}

private void update() {
    if (!Check) {
        bubbles.forEach(bubble -> bubble.localReposition(getWidth(), getHeight()));
        repaint();
    }
}

public void RestartGame() {
    cRound = 1;
    bubbles.forEach(bubble -> bubble.nborSize = size);
    StartNewRound();
}

private void put(Point point) {
    if ( stage > bubbles.size() ) {
        int inborSize = nbor * (cRound - 1)+size ;
        Bubble newBubble = new Bubble(point, randColor(), inborSize);

        boolean overlaps = bubbles.stream().anyMatch(bubble -> newBubble.overlap(bubble));

        if (overlaps) {
            JOptionPane.showMessageDialog(this, "Bubble overlap detected.");
        }
        else {
            bubbles.add(newBubble);
            if (bubbles.size() == stage) {
                JOptionPane.showMessageDialog(GP.this, "Game begins!");
                Check = false;
            }
            repaint();
        }
```

```
    }
}

private void burstBubble(Point clickPoint) {
    for (int i = 0; i < bubbles.size(); i++) {
        if (bubbles.get(i).has(clickPoint)) {
            bubbles.remove(i);
            i--;
        }
    }
    if (bubbles.isEmpty()) {
        if (cRound >= ROUNDS) {
            JOptionPane.showMessageDialog(GP.this, "YOU WON");
            frame1.setVisible(true);
            Frame2.this.setVisible(false);
        } else {
            cRound++;
            innborSize();
            globalReposition();
        }
    }
    repaint();
}

private Color randColor() {
    int red = (int) (Math.random() * 256);
    int green = (int) (Math.random() * 256);
    int blue = (int) (Math.random() * 256);
    return new Color(red, green, blue);
}


private void globalReposition() {
    bubbles.clear();
```

```
        while ( stage > bubbles.size() ) {
                Point point = new Point(RADIUS + rand.nextInt(getWidth() - 2 * RADIUS), RADIUS +
rand.nextInt(getHeight() - 2 * RADIUS));
            boolean overlaps = false;
            for (Bubble b : bubbles) {
               if (b.overlap(point)) {
                  overlaps = true;
                  break;
               }
            }

            if (!overlaps) {
               addNewBubble(point, randColor(), size + nbor * (cRound - 1));
            }
        }
        Check = false;
        repaint();
    }

    private void addNewBubble(Point point, Color color, int size) {
        Bubble newBubble = new Bubble(point, color, size);
        bubbles.add(newBubble);
    }



    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        for (int i = 0; i < bubbles.size(); i++) {
            Bubble bubble = bubbles.get(i);

            g.setColor(bubble.colour);

            int boxX=bubble.o.x - RADIUS / 2;
```

```
                int boxY=bubble.o.y - RADIUS / 2;

                g.fillOval(boxX,boxY, RADIUS, RADIUS);

                if (!bubble.burst) {
                    g.setColor(Color.BLACK);
                    g.drawRect(bubble.nborCenter.x - bubble.nborSize / 2, bubble.nborCenter.y - bubble.nborSize / 2,
bubble.nborSize, bubble.nborSize);
                }
            }

            g.drawString("Round: " + cRound, 175, 20);
            g.setColor(Color.BLACK);
        }

        private void innborSize() {
            for (int i = 0; i < bubbles.size(); i++) {
                Bubble bubble = bubbles.get(i);
                bubble.nborSize += nbor * 2;
            }

        }
    }
}

    private class Bubble {
        int nborSize;
        Color colour;
        Point o;
        boolean burst = false;
        Point nborCenter;

        public Bubble(Point o, Color c, int s) {
            this.colour = c;
            this.nborSize = s;
```

```
        this.nborCenter = new Point(o.x, o.y);
        this.o = new Point(o.x, o.y);
    }


    public boolean has(Point p) {
        return Math.hypot(Math.abs(o.y - p.y), Math.abs(o.x - p.x)) <= RADIUS / 2;
    }


    public boolean overlap(Point point) {
        return Math.hypot(this.nborCenter.y - point.y , this.nborCenter.x - point.x) < RADIUS / 2;
    }

    public boolean overlap(Bubble other) {

            return  Math.hypot(this.nborCenter.x  -  other.nborCenter.x  ,  this.nborCenter.y  -  other.nborCenter.y) <
(other.nborSize / 2) + (this.nborSize / 2);
    }

    public void localReposition (int pW, int pH) {
        int z =RADIUS;
        int y = Math.max(0,nborCenter.y - nborSize / 2);
        int Y = Math.min(pH,nborCenter.y + nborSize / 2);
        int x = Math.max(0,nborCenter.x - nborSize / 2);
        int X = Math.min(pW,nborCenter.x + nborSize / 2);

        y = y + z / 2;
        Y = Y - z / 2;
        x = x + z / 2;
        X = X - z / 2;


        if ( x < X &&  y < Y) {
            int randX = x + (int) (Math.random() * (X - x));
```

```
                int randY = y + (int) (Math.random() * (Y - y));
                o = new Point(randX, randY) ;
            }
        }
    }
}
```