Need to check why bubbles are not bursting(burstbubble method)…”Invalid!” error window is popping up after one bubble is burst while playing game …remaining I will work on …Just for your information I kept all the requirements for the game below Highlighted which requirement is not working.

**THANK YOU!!**

**Requirements:**

I. The system shall have two GUIs (JFrames).

* The first GUI shall have two Jbuttons (Start and Restart) and a JSlider. The Start button shall be used to start the game. Once it’s pressed the second GUI displays and the game starts. Once the game ends, the Restart button shall be used to start the game over. The JSlider shall be used for the player to select the game difficulty: Easy (4 bubbles), Medium (5 bubbles), Hard (6 bubbles).
* The second GUI should contain the field where the game will be played.

II. The Playing Field

* The playing field shall be defined as the dimensions of the JPanel.

III. Reposition\_Global

For Round 1

* The program shall prompt the user to define the origin for each bubble by clicking the mouse on the JPanel. If the user defines an origin that will result in a bubble not fully contained within the JPanel’s dimensions, the program shall return an error and prompt the user to make another selection. Round 1 shall start as soon as the number of user selections equals the number of bubbles for the specified game difficulty.

For Rounds 2 – 10

* Positions or repositions bubbles (circles) by choosing random coordinates from the entire playing field. A Java random number function/method shall be used for any operation that requires randomness.

IV. Reposition\_Local

* Repositions bubbles by choosing a random coordinate from each bubble’s local neighborhood. The local neighborhood is defined as a subset of the playing field where the bubble is allowed to roam. For example, if a bubble’s center coordinate is (100, 100) and we define a local neighborhood of 50, then the bubble can take a random hop in the bounding box (neighborhood) – (50,150), (50, 150), (150, 50), (150,150). Bubbles

make hops in their local neighborhoods until they are burst. Bubbles are not allowed to hop outside of the playing field. The local neighborhood expands as the rounds increase. Assume a neighborhood of 50 for Round 1, and increase by 18 for each additional round.

V. Collision Avoidance

* A bubble is not allowed to overlap or occupy the same space as another bubble.

VI. Bubble Burst

* A bubble is burst once a player makes a mouse click inside any bubble (circle). Bubbles disappear once they are burst.

VII. Rounds

* A new round starts when all the bubbles are burst. New bubbles are then respawned using reposition\_global. The round number shall be displayed.

VIII. Game Over

* The game is declared over when the player clicks anywhere on the playing field, the player successfully completes 10 rounds, or the player takes too long to finish a round. A JOption shall appear once the game is over, and it shall display “Game Over.”

IX. Timer

The timer starts at the beginning of each round. As the rounds increase, the time to complete each round decreases. The round 1 timer is 15 seconds, and it decreases by 1 second each additional round. The timer shall be displayed.

# CODE:

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Hashtable;

import java.util.Iterator;

import java.util.List;

import java.util.Random;

class GameController {

private JFrame firstGUI;

private JFrame secondGUI;

private BubblePanel bubblePanel;

public GameController() {

firstGUI = new JFrame("Bubble Burst Game");

firstGUI.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

firstGUI.setSize(300, 200);

JButton startButton = new JButton("Start");

JButton restartButton = new JButton("Restart");

JSlider difficultySlider = new JSlider(JSlider.HORIZONTAL, 4, 6, 4);

difficultySlider.setMajorTickSpacing(1);

difficultySlider.setPaintTicks(true);

Hashtable<Integer, JLabel> labelTable = new Hashtable<>();

labelTable.put(4, new JLabel("Easy"));

labelTable.put(5, new JLabel("Medium"));

labelTable.put(6, new JLabel("Hard"));

difficultySlider.setPaintLabels(true);

difficultySlider.setLabelTable(labelTable);

startButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

int bubblesToCreate = difficultySlider.getValue();

startGame(bubblesToCreate);

}

});

restartButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

restartGame();

}

});

firstGUI.setLayout(new FlowLayout());

firstGUI.add(startButton);

firstGUI.add(restartButton);

firstGUI.add(difficultySlider);

firstGUI.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

firstGUI.setSize(300, 150);

firstGUI.setVisible(true);

}

private void startGame(int bubblesToCreate) {

firstGUI.setVisible(false);

secondGUI = new JFrame("Create Bubbles");

bubblePanel = new BubblePanel(this, bubblesToCreate);

secondGUI.add(bubblePanel);

secondGUI.setSize(800, 600);

secondGUI.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

secondGUI.setVisible(true);

}

private void restartGame() {

if (secondGUI != null) {

secondGUI.setVisible(true);

firstGUI.setVisible(false);

}

}

public void showCongrats() {

JOptionPane.showMessageDialog(secondGUI, "Round " + bubblePanel.getRound() + " completed!");

bubblePanel.incrementRound();

bubblePanel.repositionGlobal(); // Respawn new bubbles for the next round

restartGame();

}

}

class Bubble {

private int x, y, radius;

public Bubble(int x, int y, int radius) {

this.x = x;

this.y = y;

this.radius = radius;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public int getRadius() {

return radius;

}

public void draw(Graphics g) {

g.setColor(Color.BLUE);

g.fillOval(x - radius, y - radius, 2 \* radius, 2 \* radius);

}

public boolean contains(int x, int y) {

int dx = this.x - x;

int dy = this.y - y;

return dx \* dx + dy \* dy <= radius \* radius;

}

public void repositionWithinNeighborhood(Rectangle neighborhood) {

Random rand = new Random();

int localX = rand.nextInt(2 \* neighborhood.width) - neighborhood.width;

int localY = rand.nextInt(2 \* neighborhood.height) - neighborhood.height;

this.x = Math.max(neighborhood.x + radius, Math.min(neighborhood.x + neighborhood.width - radius, this.x + localX));

this.y = Math.max(neighborhood.y + radius, Math.min(neighborhood.y + neighborhood.height - radius, this.y + localY));

}

}

class BubblePanel extends JPanel {

private List<Bubble> bubbles = new ArrayList<>();

private int bubblesToCreate;

private Rectangle neighborhood = null;

private JButton startBurstingButton;

private GameController gameController;

private int round = 1;

private int initialNeighborhoodSize = 50;

private int neighborhoodSizeIncrement = 18;

private Random rand = new Random();

public BubblePanel(GameController gameController, int bubblesToCreate) {

this.gameController = gameController;

this.bubblesToCreate = bubblesToCreate;

addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

if (bubbles.size() < bubblesToCreate) {

createBubble(e.getX(), e.getY());

} else {

burstBubbles(e.getX(), e.getY());

}

}

});

startBurstingButton = new JButton("Start Game");

startBurstingButton.setVisible(false);

startBurstingButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

startBursting();

}

});

add(startBurstingButton);

Timer timer = new Timer(3000, new ActionListener() {

public void actionPerformed(ActionEvent e) {

updateNeighborhoodSize();

repositionBubbles();

repaint();

}

});

timer.start();

}

private void createBubble(int x, int y) {

if (bubbles.size() < bubblesToCreate) {

if (neighborhood == null) {

neighborhood = new Rectangle(0, 0, getWidth(), getHeight());

}

Bubble newBubble = new Bubble(x, y, 20);

boolean collision = false;

for (Bubble existingBubble : bubbles) {

if (checkCollision(newBubble, existingBubble)) {

collision = true;

break;

}

}

if (neighborhood.contains(x, y) && !collision) {

bubbles.add(newBubble);

if (bubbles.size() == bubblesToCreate) {

startBurstingButton.setVisible(true);

}

} else {

JOptionPane.showMessageDialog(this, "Invalid! Click inside the neighborhood rectangle without overlapping existing bubbles.");

}

repaint();

}

}

private double calculateDistance(Bubble bubble1, Bubble bubble2) {

int dx = bubble1.getX() - bubble2.getX();

int dy = bubble1.getY() - bubble2.getY();

return Math.sqrt(dx \* dx + dy \* dy);

}

private void updateNeighborhoodSize() {

int currentNeighborhoodSize = initialNeighborhoodSize + (round - 1) \* neighborhoodSizeIncrement;

if (neighborhood == null) {

int panelWidth = getWidth();

int panelHeight = getHeight();

int rectWidth = panelWidth - 2 \* currentNeighborhoodSize;

int rectHeight = panelHeight - 2 \* currentNeighborhoodSize;

neighborhood = new Rectangle(currentNeighborhoodSize, currentNeighborhoodSize, rectWidth, rectHeight);

} else {

neighborhood.setSize(getWidth() - 2 \* currentNeighborhoodSize, getHeight() - 2 \* currentNeighborhoodSize);

}

}

private boolean checkCollision(Bubble bubble1, Bubble bubble2) {

double distance = calculateDistance(bubble1, bubble2);

return distance < bubble1.getRadius() + bubble2.getRadius();

}

private void burstBubbles(int x, int y) {

Iterator<Bubble> iterator = bubbles.iterator();

boolean bubbleBurst = false;

while (iterator.hasNext()) {

Bubble bubble = iterator.next();

if (bubble.contains(x, y)) {

iterator.remove();

bubbleBurst = true;

}

}

if (bubbleBurst) {

if (bubbles.isEmpty()) {

gameController.showCongrats();

}

} else {

JOptionPane.showMessageDialog(this, "Invalid! Click inside the neighborhood rectangle without overlapping existing bubbles.");

}

repaint();

}

public void repositionBubbles() {

for (Bubble bubble : bubbles) {

boolean collision;

do {

collision = false;

bubble.repositionWithinNeighborhood(neighborhood);

// Check for collisions with other bubbles in the BubblePanel class

for (Bubble otherBubble : bubbles) {

if (bubble != otherBubble && checkCollision(bubble, otherBubble)) {

collision = true;

break;

}

}

} while (collision);

}

}

private void startBursting() {

startBurstingButton.setVisible(false);

}

public int getRound() {

return round;

}

public void incrementRound() {

round++;

}

// Add a method to respawn bubbles globally

public void repositionGlobal() {

bubbles.clear(); // Clear existing bubbles

for (int i = 0; i < bubblesToCreate; i++) {

int x = neighborhood.x + rand.nextInt(neighborhood.width);

int y = neighborhood.y + rand.nextInt(neighborhood.height);

Bubble newBubble = new Bubble(x, y, 20);

boolean collision;

do {

collision = false;

newBubble.repositionWithinNeighborhood(neighborhood);

// Check for collisions with other bubbles

for (Bubble otherBubble : bubbles) {

if (checkCollision(newBubble, otherBubble)) {

collision = true;

break;

}

}

} while (collision);

bubbles.add(newBubble);

}

}

protected void paintComponent(Graphics g) {

super.paintComponent(g);

for (Bubble bubble : bubbles) {

bubble.draw(g);

}

if (neighborhood != null) {

g.setColor(Color.RED);

g.drawRect(neighborhood.x, neighborhood.y, neighborhood.width, neighborhood.height);

}

g.setColor(Color.BLACK);

g.drawString("Round: " + round, 10, 20);

}

}

public class BubbleBurstGame {

private GameController gameController;

public BubbleBurstGame() {

gameController = new GameController();

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new BubbleBurstGame());

}

}