Cover Page

*I, Xiaoxi Zheng*\_\_\_ affirm that the work submitted is my own and that the Honor  
Code was neither bent nor broken.

The easiest part of this HW is the setup of this project, since the structure of code  
was quite straight forward. The more difficult parts of the HW is buried  
within working around with JPanel and the way it displays things, and the exact way setline() method is  
use so not just the last stem is displayed. I also spend some time creating a color array object and an array object to store the float value of basic strokes.   
I believe the objective of this assignment was for us  
to understand and implement setting lines using BufferedImage’s Graphic2D “tool” set. While given us  
some general practice for methods interacting together.

# Codes:

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.\*;

import java.io.\*;

import javax.imageio.\*;

import javax.swing.\*;

import javax.swing.JOptionPane;

import java.util.Random;

import java.lang.Math.\*;

import java.lang.Math;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.\*;

import java.awt.geom.Line2D;

public class hw05a{

private static final int WIDTH = 401;

private static final int HEIGHT = 401;

public static void main( String[] args){

JFrame frame = new ImageFrame(WIDTH, HEIGHT);

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

frame.setVisible(true);

}

}

//####################################################################

class ImageFrame extends JFrame{

///private final JFileChooser chooser;

private BufferedImage image = null;

private Random rand = new Random();

private double randX;

private int size;

private double x;

private double y;

private int stems;

private int stepsPerStem;

private int steps;

private double alpha;

private double beta;

private double theta;

private double deltaTheta;

private int rho;

private int deltaRho;

private int direction;

private double tao;

private int endColor;

private int startColor;

private Color [] colorArray;

private float [] basicStrokeWeight;

//=========================

public ImageFrame(int width, int height){

//setup the frame's attributes

this.setTitle("CAP 3027 2015 - HW05a -XiaoxiZheng");

this.setSize( width, height );

//initialize start and end colors as black and green

startColor = 0xf4a460;

endColor = 0x32cd32;

addMenu();////add a menu to the frame

//setup the file chooser dialog

//chooser = new JFileChooser();

//chooser.setCurrentDirectory(new File("."));

}

private void addMenu(){

//setup the frame's munu bar

//File menu

JMenu fileMenu = new JMenu("File");

//===Crystal on Toroid plane

JMenuItem directedRW = new JMenuItem("Directed random walk plant");

directedRW.addActionListener( new ActionListener(){

public void actionPerformed( ActionEvent event){

CreateBufferedImageT();

}

} );

fileMenu.add(directedRW);

JMenuItem setColor = new JMenuItem("Set starting and end color");

setColor.addActionListener( new ActionListener(){

public void actionPerformed( ActionEvent event){

promptForSettingStartColor();

promptForSettingEndColor();

}

} );

fileMenu.add(setColor);

//Exit

JMenuItem exitItem = new JMenuItem("Exit");

exitItem.addActionListener( new ActionListener(){

public void actionPerformed(ActionEvent event){

System.exit( 0 );

}

} );

fileMenu.add( exitItem);

//attach menu to a menu bar

JMenuBar menuBar = new JMenuBar();

menuBar.add( fileMenu);

this.setJMenuBar( menuBar);

}

private void CreateBufferedImageT(){

size = promptForSize();

stems = promptForStems();

stepsPerStem = promptForStepsPerStem();

alpha = promtForTransmiteProb();

deltaTheta = promptForMaxRotation();

deltaRho = promptForMaxGrowthSegment();

BufferedImage image = new BufferedImage(size,size,BufferedImage.TYPE\_INT\_ARGB);

RenderingHints hint =

new RenderingHints( RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

Graphics2D target = (Graphics2D)image.createGraphics();

target.setRenderingHints( hint );

setBG\_black(size,image);

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

Color[] colorArray = new Color[stepsPerStem];

float [] basicStrokeWeight = new float[stepsPerStem];

colorArray = interpolateColor(stepsPerStem,startColor,endColor);

basicStrokeWeight = interpolateWeight(stepsPerStem,6.0,0.5);

//initial

target.setColor(colorArray[0]);

BasicStroke stroke = new BasicStroke(6);

target.setStroke( stroke );

theta = Math.PI/2;

rho = 1;

beta = 1-alpha;

x = size/2;

y = size/2;

double rand1 = rand.nextDouble();

if (rand1>=0.5){

direction = -1;

} else {

direction = 1;

}

double[] coord = new double[2];

coord = toCartesian(rho,theta\*direction);

//Line2D line2d = new Line2D.Double(x,y,x,y-coord[1]);

Line2D line2d = new Line2D.Double(x, y, x, y+rho);

//System.out.println(x);

target.draw(line2d);

x = line2d.getX2();

y = line2d.getY2();

for (int j=0;j<stepsPerStem;j++) {

System.out.println("color at the step: " + colorArray[j]);

System.out.println("stroke at the step: " + basicStrokeWeight[j]);

BasicStroke stroke\_ = new BasicStroke( basicStrokeWeight[j]);

target.setStroke( stroke\_ );

target.setColor(colorArray[j]);

if (direction == -1) {

tao = alpha;

} else {

tao = beta;

}

randX = rand.nextDouble();

if (randX>tao) {

direction = 1;

} else {

direction = -1;

}

//compute offset

double randT = rand.nextDouble();

rho=rho+deltaRho;

theta = (deltaTheta\*randT\*direction)+theta;

double[] newCoord = new double[2];

newCoord = toCartesian(rho, theta);

line2d.setLine(x, y, x+newCoord[0], y-newCoord[1]);

target.draw(line2d);

x = line2d.getX2();

y= line2d.getY2();

}

displayFile(image);

}

//displayFile(image);

}

private Color[] interpolateColor(int stepsPerStem\_,int startColor,int endColor){

Color [] colorArray\_ = new Color [stepsPerStem\_];

double[] colorInfoStart = extraction(startColor);

double[] colorInfoEnd = extraction(endColor);

double deltaRGen = (colorInfoEnd[1] - colorInfoStart[1])/stepsPerStem\_; //[1]--channel for red

double deltaGGen = (colorInfoEnd[2] - colorInfoStart[2])/stepsPerStem\_; //[2]--channel for green

double deltaBGen = (colorInfoEnd[3] - colorInfoStart[3])/stepsPerStem\_; //[3]--channel for blue

double redGeneral = colorInfoStart[1]; //start paiting @ left

double greenGeneral = colorInfoStart[2]; //start paiting @ left

double blueGeneral = colorInfoStart[3]; //starting paiting @ left

for(int x = 0; x<stepsPerStem\_ ;x++){

redGeneral = redGeneral + deltaRGen; //bc red starts from the left

greenGeneral = greenGeneral + deltaGGen;

blueGeneral = blueGeneral + deltaBGen;

//clamping

if(redGeneral>255){

redGeneral = 255;

}

if(redGeneral<0){

redGeneral = 0;

}

if(greenGeneral>255){

greenGeneral = 255;

}

if(greenGeneral<0){

greenGeneral = 0;

}

if(blueGeneral>255){

blueGeneral = 255;

}

if(blueGeneral<0){

blueGeneral = 0;

}

colorArray\_[x] = new Color((int)redGeneral,(int)greenGeneral,(int)blueGeneral);

}

return colorArray\_;

}

private float [] interpolateWeight(int stepsPerStem\_,double startWeight\_, double endWeight\_){

float [] basicStrokeWeight\_ = new float [stepsPerStem\_];

float strokeW = (float)startWeight\_;

float deltaStrokeW = (float)(startWeight\_ - endWeight\_)/stepsPerStem\_ ;

for(int i = 0; i<stepsPerStem\_ ;i++){

strokeW = strokeW - deltaStrokeW; // bc start stroke is actually bigger than end stroke

basicStrokeWeight\_[i] = strokeW;

}

return basicStrokeWeight\_ ;

}

private static double[] extraction(int ARGB\_){

double[] extractionArray;

extractionArray = new double[4];

//extractionArray -- extraction[0] = alpha values;

//extraction[1] = red values; & etc with ARGB

extractionArray[0] = ARGB\_>>>24;

extractionArray[1] = (ARGB\_<<8) >>> 24;

extractionArray[2] = (ARGB\_<<16)>>>24;

extractionArray[3] = (ARGB\_<<24)>>>24;

return (extractionArray);

}

private double[] toCartesian(double rho\_, double theta\_){

double [] tempA = new double[2];

double x = Math.cos(theta\_)\*rho;

double y = Math.sin(theta\_)\*rho;

tempA[0] = x;

tempA[1] = y;

return tempA;

}

private int promptForSize(){ //helper method to bufferedIMage methods

//try catch statement for non int inputs.

String input = JOptionPane.showInputDialog("Please enter the size of your canvas");

if(valideInput(input)){

int size = Integer.parseInt(input);

return size;

}

else{

return promptForSize(); //if input was invalide, prompt for size again.

}

}

private void promptForSettingStartColor(){

String input = JOptionPane.showInputDialog("Please enter start color for the stem");

try{

int color\_ = (int)Long.parseLong(input.substring(2,input.length()),16 );

startColor = color\_;

}

catch(Exception e){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

}

}

private void promptForSettingEndColor(){ //helper method to bufferedIMage methods

String input = JOptionPane.showInputDialog("Please enter end color for the stem");

try{

int color\_ = (int)Long.parseLong(input.substring(2,input.length()),16 );

endColor = color\_;

}

catch(Exception e){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

}

}

private int promptForStems(){ //helper method to bufferedIMage methods

//try catch statement for non int inputs.

String input = JOptionPane.showInputDialog("Please enter the number of stems");

if(valideInput(input)){

int stems\_ = Integer.parseInt(input);

return stems\_;

}

else{

return promptForStems(); //if input was invalide, prompt for size again.

}

}

private int promptForStepsPerStem(){ //helper method to bufferedIMage methods

//try catch statement for non int inputs.

String input = JOptionPane.showInputDialog("Please enter the number steps per stem");

if(valideInput(input)){

int stepsPerStem\_ = Integer.parseInt(input);

return stepsPerStem\_;

}

else{

return promptForStepsPerStem(); //if input was invalide, prompt for size again.

}

}

private double promtForTransmiteProb(){

String input = JOptionPane.showInputDialog("Please enter transmittion probability");

if(valideInputBetween0n1(input)){

double probability\_ = Double.parseDouble(input);

return probability\_;

}

else{

return promtForTransmiteProb(); //if input was invalid, prompt for size again.

}

}

private double promptForMaxRotation(){

String input = JOptionPane.showInputDialog(

"Please enter the maximum Rotation increment");

if(valideInputBetween0n1(input)){

double maxRotate\_ = Double.parseDouble(input);

return maxRotate\_;

}

else{

return promptForMaxRotation(); //if input was invalid, prompt for size again.

}

}

private int promptForMaxGrowthSegment(){

String input = JOptionPane.showInputDialog("Please enter growth segment increment");

if(valideInput(input)){

int growthSeg\_ = Integer.parseInt(input);

return growthSeg\_;

}

else{

return promptForMaxGrowthSegment(); }

}

private boolean valideInput(String input\_){

try{

int num = Integer.parseInt(input\_);

if(num<0){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

catch(NumberFormatException e){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

return false;

}

}

private boolean valideInputBetween0n1(String input\_){

try{

double num = Double.parseDouble(input\_);

if(num<0 || num>1){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

catch(NumberFormatException e){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

return false;

}

}

private void setBG\_black(int size\_, BufferedImage image\_){

for(int x = 0; x<size\_; x++){

for(int y =0; y<size\_; y++){

image\_.setRGB(x,y,0xFF000000);

}

}

}

//just incase if there are any exceptions that wasn't caught in my prompt() method

private void displayFile(BufferedImage image\_){

try{

displayBufferedImage(image\_);

}

catch(Exception exception){

JOptionPane.showMessageDialog(null, "ERRR", "alert", JOptionPane.ERROR\_MESSAGE);

}

}

//display BufferedImage

public void displayBufferedImage( BufferedImage image){

this.setContentPane( new JScrollPane(new JLabel(new ImageIcon(image))));

this.validate();

}

}

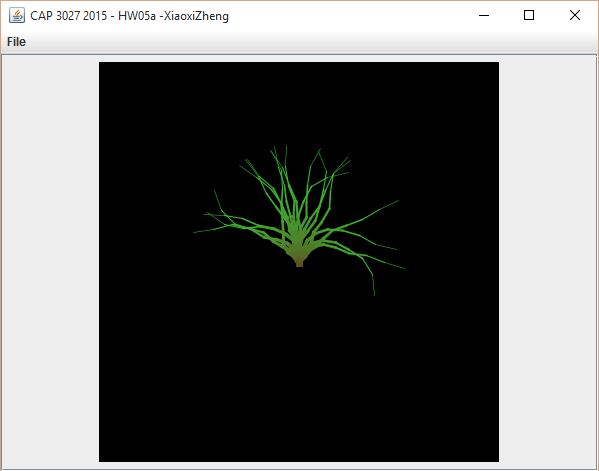
=====================================================================================

Questions  
1. Does the program compile without errors?  
Yes.  
2. Does the program compile without warnings?  
Yes  
3. Does the program run without crashing?  
Yes  
4.Describe how you tested the program.  
I ran several test cases with different user inputs. I gave couple illegal test cases when users input  
negative canvas size, and char/string inputs. I gave big and small test value inputs to make sure  
everything is working. I also gave user inputs that are greater than the size of image to make sure it  
produces the single giant circle in the middle.  
5. Describe the ways in which the program does not meet assignment's specifications.  
None.  
6. Describe all known and suspected bugs.  
There are no known bugs.  
7. Does your program run correctly?

Yes

# Screenshots

Default color: Brown → Green



User Input colors: Red(0xFFFF0000) → Green(0xFF00FF00)

