**Project 4 – Dot Illusion**

**(10 points)**

**Name:** Ozaner Hansha

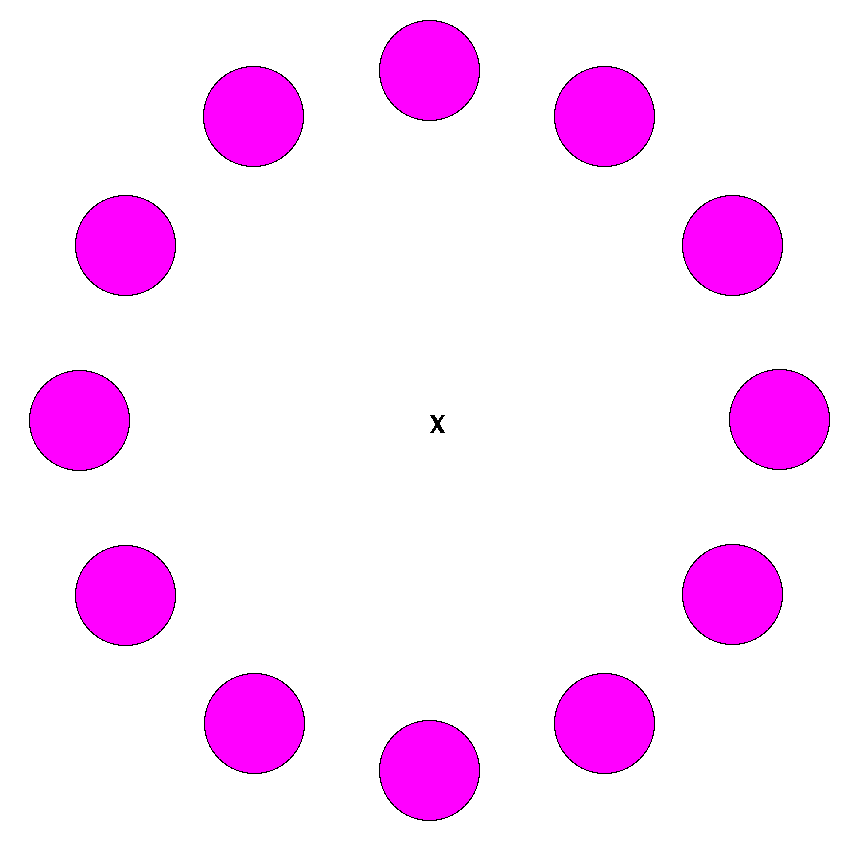
**Due Date: Monday, November 23, 2015**

**Description:**

We have been looking at some techniques for animation. One of those techniques involves the manipulation of transparency/visibility. In this project, you will be recreating a famous psychological illusion. (Thanks to Caroline Viscovich for making me aware of this.)

It turns out that if you stare at the center of a ring of rotating fuchsia (or magenta) dots, your eye will begin seeing (lime) green colors instead. “From a physiology perspective, when the human eye is exhausted by a dominant color (especially pure/saturated hues), it naturally reveals the complementary color.” (<http://www.threedeepmarketing.com/blog/color-conversion-science-clockwork-model/>) Marketers and graphic designers make direct use of human color perception in their design techniques.

Here is the pattern of interest for this project:



Carry out the following steps:

1. Download the Eclipse project for this project – APCS2015Proj04DotIllusion.
2. Make sure that the ACMLibrary project is on your build path.
3. In the class DotIllusion, carry out the directions in the commented code.
4. Here are some things that your code must do:
5. Design it to lay out any number of dots in a circle. The number of dots should be specified in a class constant.
6. The size of the circle and the size of the dots should be based on a percentage of the window size. This is generally consistent with the principle that we do not want “magic” numbers hardwired into the body of our code; instead they should be represented in easily adjustable constants.
7. The run() method must manipulate the visibility of the objects to give the impression of rotation among the objects. Consider using the pause() method inherited from the GraphicsProgram class. The program can simply run until terminated.
8. Try resizing the window while your program is running. What happens? Study the code that I provided in the catchResizeEvents() method and see what you can make of it.
9. Submission:
10. Fill out your name at the top of this page.
11. Copy and paste your Java source code listing to the end of this document. Be careful when copying and pasting to preserve the font information from Eclipse. It makes reading your code much easier.
12. Append a screen shot of your graphics window after your code at the end of this document.
13. Save this document as “Project 4 – Dot Illusion (*Lastname Firstname*).doc”  
    Upload your Word document to the Hand-In folder that you created and shared with me. Do not submit a Google document. (You can drag-and-drop the document into the folder from a file browser.)

**Final Code for DotIllusion:**

package unit4;  
  
import java.awt.Color;  
import java.awt.Font;  
import java.awt.event.ComponentAdapter;  
import java.awt.event.ComponentEvent;  
  
import acm.graphics.GLabel;  
import acm.graphics.GObject;  
import acm.graphics.GOval;  
import acm.graphics.GPoint;  
import acm.graphics.GScalable;  
import acm.program.GraphicsProgram;  
  
/\*\*  
 \* Optical illusion from cognitive psychology.  
 \* The idea and original code by Caroline Viscovich.<br>  
 \*   
 \* AP Computer Science - Pd. 7<br>  
 \* November 23rd, 2015<br>  
 \* Dr. Jones<br>  
 \* @author Ozaner Hansha  
 \*/  
@SuppressWarnings("serial")  
public class DotIllusion extends GraphicsProgram   
{   
 /\*\*  
 \* The Width of the window.  
 \*/  
 private static final int DEFAULT\_WIDTH = 800;  
   
 /\*\*  
 \* The Height of the window. (Same as width)  
 \*/  
 private static final int DEFAULT\_HEIGHT = DEFAULT\_WIDTH;  
   
 /\*\*  
 \* The Center point of the illusion. Set by the {@link #layoutX()} method.  
 \* @see #layoutX()  
 \*/  
 private static final GPoint CENTER\_POINT = new GPoint(DEFAULT\_WIDTH/2, DEFAULT\_HEIGHT/2);  
   
 /\*\*  
 \* The String displayed in the center of the screen.  
 \*/  
 private static final String CENTER\_STRING = "X";  
   
 /\*\*  
 \* The font of the String displayed in the center.  
 \*/  
 private static final Font CENTER\_FONT = new Font("Ariel", 1, 25);  
   
 /\*\*  
 \* The amount of dots to be displayed equidistant to the center of the window.  
 \*/  
 private static final int AMOUNT\_OF\_DOTS = 12;  
   
 /\*\*  
 \* The array of {@link GOval}s used in the application.  
 \* Defined by the {@link #AMOUNT\_OF\_DOTS}.  
 \* @see #AMOUNT\_OF\_DOTS  
 \*/  
 private static final GOval[] DOTS = new GOval[AMOUNT\_OF\_DOTS];  
   
 /\*\*  
 \* The radius of all the {@link #DOTS}.  
 \*/  
 private static final int DOT\_RADIUS = 30;  
   
 /\*\*  
 \* The color of the dots.  
 \*/  
 private static final Color DOT\_COLOR = Color.MAGENTA;  
   
 /\*\*  
 \* The distance between the center of the window and a given dot.  
 \*/  
 private static final int ILLUSION\_RADIUS = 150;  
   
 /\*\*  
 \* The amount of of time, in milliseconds, that the program pauses between frames.  
 \* @see #pause(double)  
 \*/  
 private static final long PAUSE\_TIME = 50;  
  
 /\*\*  
 \* Temporary variables for window resizing event handling.  
 \* @see #catchResizeEvents()  
 \*/  
 private double wid, ht;  
  
 /\*\*   
 \* Main method to run the dot illusion as an application.  
 \* @param args no args expected  
 \*/  
 public static void main(String[] args)  
 {   
 new DotIllusion().start(args);   
 }   
  
 /\*\*  
 \* Initializes the canvas with an X in the center and a ring of surrounding dots.  
 \*/  
 public void init()  
 {  
 setSize(DEFAULT\_WIDTH, DEFAULT\_HEIGHT);  
 layoutX();  
 createDots();  
 layoutDots();  
 catchResizeEvents();  
 }   
   
 /\*\*  
 \* Lays out the 'X' in the center of the canvas.  
 \*/  
 private void layoutX()  
 {  
 GLabel label = new GLabel(CENTER\_STRING);  
 label.setFont(CENTER\_FONT);  
 add(label, CENTER\_POINT.getX() - label.getWidth()/2,  
 CENTER\_POINT.getY() + label.getHeight()/2 - label.getAscent()/2);  
 }  
   
 /\*\*  
 \* Creates the {@link GOval} objects and places them in {@link #DOTS}.  
 \*/  
 private void createDots()  
 {  
 for(int x = 0; x < AMOUNT\_OF\_DOTS; x++)  
 {  
 GOval dot = new GOval(DOT\_RADIUS\*2,DOT\_RADIUS\*2);  
 dot.setFilled(true);  
 dot.setColor(DOT\_COLOR);  
 DOTS[x] = dot;  
 }  
 }  
   
 /\*\*  
 \* Lays out the {@link #DOTS} in a ring starting at the top, centered on the canvas.  
 \*/  
 private void layoutDots()  
 {  
 for(int x = 0; x < AMOUNT\_OF\_DOTS; x++)  
 {  
 double angle = 2\*Math.PI/AMOUNT\_OF\_DOTS;  
 add(DOTS[x], (CENTER\_POINT.getX() + ILLUSION\_RADIUS \* Math.cos(angle\*x)) - DOT\_RADIUS,  
 (CENTER\_POINT.getY() + ILLUSION\_RADIUS \* Math.sin(angle\*x)) - DOT\_RADIUS);  
 }  
 }  
   
 /\*\*  
 \* Sets up the handler for resize events. This handler catches resize  
 \* events, rescales the (GScalable) objects, and adjusts the locations  
 \* of all GObjects. It does not adjust the font size for GLabels and such.  
 \* @author Mark Jones  
 \*/  
 private void catchResizeEvents()  
 {  
 wid = getWidth();  
 ht = getHeight();  
   
 addComponentListener(new ComponentAdapter()  
 {  
 public void componentResized(ComponentEvent e)  
 {  
 double scaleX = getWidth() / wid, scaleY = getHeight() / ht;  
   
 for (int i = 0; i < getElementCount(); i++)  
 {  
 Object obj = getElement(i);  
 if (obj instanceof GObject)  
 {  
 if (obj instanceof GScalable)  
 ((GScalable) obj).scale(scaleX, scaleY);  
 ((GObject) obj).setLocation(((GObject) obj).getX()\*scaleX, ((GObject) obj).getY()\*scaleY);  
 }  
 }  
 wid = getWidth(); ht = getHeight();  
 }  
 });   
 }  
  
 /\*\*  
 \* Animation technique to make each dot in turn disappear briefly.<br>  
 \* This method is the game loop.  
 \*/  
 public void run()  
 {   
 while(true)  
 {  
 for(GOval c: DOTS)  
 {  
 c.setVisible(false);  
 pause(PAUSE\_TIME);  
 c.setVisible(true);  
 }  
 }  
 }  
}

**Screen Shot**

