LABS #3 – Introduction to Graphics

This assignment is to build and test a basic graphics system based on the code in the book. Later on we will have you write a special program that can only generate graphics for its output.

You will need to create a library file called **`Graphics.py`** that contains the following code from the Computer Science Companion (4TH edition), in this order:

0. **PUT YOUR NAME IN A COMMENT AT THE VERY TOP OF THE FILE!!!**
1. The `Distance` function from page 382 (don't forget to import the math library)
2. All of the graphics code from pages 404-407:
   - The color definitions (`black`, `blue`, etc.)
   - `getRed`, `getGreen`, `getBlue`
   - `makeEmptyPicture`
   - `getWidth`, `getHeight`
   - `PixelIndex`
   - `setPixel`, `getPixel`
   - `HorizontalLine`, `VerticalLine`
   - `WriteString` (for Python 3), `WriteBytes` (for Python 3), `WriteBMP`
3. The `addLine` function from the ***top*** of page 408
4. The circle functions from page 411:
   - `BresenhamCircle`, `BresenhamFilledCircle`

Save this file as **`Graphics.py`** in your Python directory.

This is a lot of code to type in, and it is likely that you will make mistakes. To quickly test if the functions basically work, "run" the file (it won't do anything), then follow the following script by typing in the commands in black. If everything works, you should get the responses from Python as shown in blue:

```
>>> Canvas = makeEmptyPicture(320, 200, gray)
>>> Canvas
{'Width': 320, 'Height': 200, 'Default': [128, 128, 128]}
>>> setPixel(Canvas, 100, 50, green)
>>> Canvas
{'Width': 320, 'Height': 200, 'Default': [128, 128, 128], 5000100: [0, 255, 0]}
>>> getPixel(Canvas, 100, 50)
[0, 255, 0]
>>> setPixel(Canvas, 100, 50, gray)
>>> getPixel(Canvas, 100, 50)
[128, 128, 128]
>>> Canvas
{'Width': 320, 'Height': 200, 'Default': [128, 128, 128]}
>>>
```

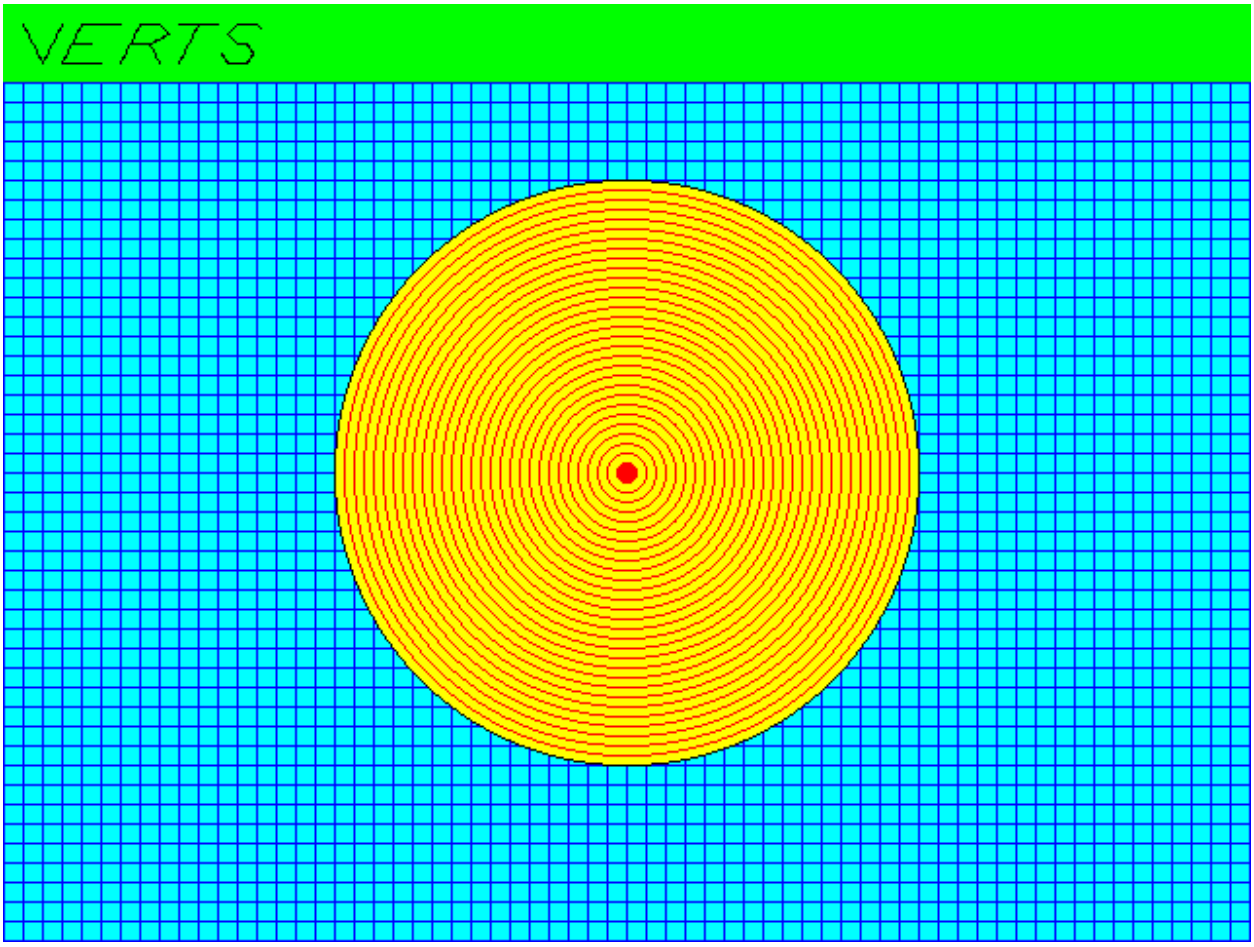
This is a very rudimentary test of the routines, and does not exercise all of the functions, but it will give you some confidence that the underlying pixel model works correctly. The next step is to more rigorously test all the functions in the `Graphics.py` file.

Download the `TestGraphics.py` program from the class site. Store that file in the ***same folder*** as your `Graphics.py` file. If you cannot download the file, here is the source code for that file:

```
#-------------------------------------------------------------
# TEST PROGRAM FOR GRAPHICS LIBRARY
#
# Copyright (C) December 1, 2019 -- Dr. William T. Verts
#-------------------------------------------------------------

from Graphics import *

#-------------------------------------------------------------
# Build the canvas and define variables for later use.
#-------------------------------------------------------------

Canvas = makeEmptyPicture(641, 481, cyan)
XMid = getWidth(Canvas)  // 2
YMid = getHeight(Canvas) // 2
Radius = 150

#-------------------------------------------------------------
# Draw horizontal green lines from screen left to screen
# right for the first 40 lines of the image.
#-------------------------------------------------------------

for Y in range(40):
    HorizontalLine(Canvas, 0, getWidth(Canvas)-1, Y, green)

#-------------------------------------------------------------
# Draw horizontal blue lines from screen left to screen
# right, starting at line 40 and stepping down 10 rows at
# a time.
#-------------------------------------------------------------

for Y in range(40, getHeight(Canvas), 10):
    HorizontalLine(Canvas, 0, getWidth(Canvas)-1, Y, blue)

#-------------------------------------------------------------
# Draw vertical blue lines from line 40 to screen bottom,
# starting at screen left and stepping right 10 columns at
# a time.
#-------------------------------------------------------------

for X in range(0, getWidth(Canvas), 10):
    VerticalLine(Canvas, X, 40, getHeight(Canvas)-1, blue)

#-------------------------------------------------------------
# Draw a big yellow filled circle at screen center, with
```

```
# a bunch of red circle outlines starting at radius 10 and
# stepping by 5 per circle.  Finally, draw a red spot at
# the center, and draw a black outline.
#-------------------------------------------------------------

BresenhamFilledCircle(Canvas, XMid, YMid, Radius, yellow)
for R in range(10, Radius, 5):
    BresenhamCircle(Canvas, XMid, YMid, R, red)
BresenhamFilledCircle(Canvas, XMid, YMid, 5, red)
BresenhamCircle(Canvas, XMid, YMid, Radius, black)

#-------------------------------------------------------------
# Draw my signature in the green area.
#-------------------------------------------------------------

addLine(Canvas, 10, 10, 20, 30)     # V
addLine(Canvas, 20, 30, 30, 10)
addLine(Canvas, 40, 10, 30, 30)     # E
addLine(Canvas, 40, 10, 60, 10)
addLine(Canvas, 35, 20, 45, 20)
addLine(Canvas, 30, 30, 50, 30)
addLine(Canvas, 70, 10, 60, 30)     # R
addLine(Canvas, 70, 10, 80, 10)
addLine(Canvas, 65, 20, 75, 20)
addLine(Canvas, 80, 10, 85, 15)
addLine(Canvas, 85, 15, 75, 20)
addLine(Canvas, 75, 20, 80, 30)
addLine(Canvas, 90, 10,110, 10)     # T
addLine(Canvas,100, 10, 90, 30)
addLine(Canvas,120, 10,130, 10)     # S
addLine(Canvas,130, 10,132, 12)
addLine(Canvas,120, 10,115, 15)
addLine(Canvas,115, 15,120, 20)
addLine(Canvas,120, 20,125, 20)
addLine(Canvas,125, 20,128, 25)
addLine(Canvas,128, 25,125, 30)
addLine(Canvas,125, 30,115, 30)
addLine(Canvas,115, 30,112, 25)

#-------------------------------------------------------------
# Save the image to file.
#-------------------------------------------------------------

print ("Pixels in Canvas = ", getWidth(Canvas)*getHeight(Canvas))
print ("Items in Canvas = ",len(Canvas))

WriteBMP('TestGraphics.bmp', Canvas)
print ("All Done")
```

Run the `TestGraphics.py` program that I provide; it should automatically create a bitmap graphics file in the same folder as `TestGraphics.py` and `Graphics.py`, called `TestGraphics.bmp` (about 900K in size). When you load that bitmap into a standard graphics program (whatever you've got on your computer), you should see the following image:



If the `.BMP` file is not correctly constructed, your computer will complain about it being corrupted, or will complain about it being un-viewable in some way. The problem is likely in ***your code***, in `WriteString`, `WriteBytes`, or in `WriteBMP` itself. Historically, the most common problem I've seen in student code has been in the `WriteBytes` function, usually due to improper indentation of the `Outfile.write` command. Just because the program runs without crashing, it doesn't mean that the `.BMP` file was created correctly! If the `.BMP` can't be viewed, you've done something wrong.

Other places that errors can occur are in the Bresenham circle routines (incomplete or segmented circles), `addLine` (lines that extend too far or not far enough, making my signature incorrect), and wrong color definitions (weird colors in the bitmap).

As you notice, the `TestGraphics.py` program calls nearly all of the functions in `Graphics.py`, so if there are any bugs (either syntax or logic) then either (1) `TestGraphics.py` will crash, or (2) the image will be different from what you see here, or (3) the `.BMP` file is corrupt and cannot be viewed.

Fix your bugs until you get the correct test image.

When you are done, submit your `Graphics.py` program as **Lab #3**. We will test your submission by running `TestGraphics.py` using your version of `Graphics.py`.

Study carefully the "Grading Codes" document on the class Web site. That is the generic rubric for all programming assignments in this class. It lays out the rules you must follow for all programs (you must have your name, lab number, and date in a comment at the top of the program, the program must run to completion without errors, it must solve the assigned problem, etc.) Each infraction gets a certain number of points removed and a letter code indicating which infraction was involved.

For this particular assignment, the *additional* error codes are as follows:

H: -5 `TestGraphics.py` runs to completion but `TestGraphics.bmp` is corrupted or contains little if anything that is recognizable.

I: -3 The `TestGraphics.bmp` image is created but the geometry of something (lines, circles, colors, etc.) is wrong (that is, the picture is wrong in some way).

Since you are basically typing in someone else's code (mine!) there is really no reason for there to be *any* errors, so the grading is very simple (and harsh).