Kenneth Maguire

Graphics Program

Program 3B

Program Description

Since Doug said we could program this in any language we want, I chose to use Python, since I've been using it for other classes all semester and I've never worked with graphics in Python. Since Doug only provided the setup for and plotting for Racket, I figured this would be a good chance to figure it out from scratch. My program starts by calculating the dimensions of the picture Doug provided, named "BigEarth.jpg" which I had to convert to a different format before using. Python's graphics package isn't able to handle jpg format, I had to convert the image to .gif format which displayed with the same resolution. I was able to find the dimensions using the struct package (found at source https://stackoverflow.com/questions/8032642/how-to-obtain-image-size-using-standard-python-class-without-using-external-lib) which reads bytes 6-10 from the header of the "BigEarth.gif" file, and converts the bytes to width and height as int values. Using those width and height values, I could divide width by 360 for the longitude ratio, and height by 180 for the latitude ratio. These ratios were used for plotting all of my values on the map and plotting the grid points for longitude and latitude since just using the raw data for plotting and using increments of 10 for adding lat/long lines would only work if the resolution of the image was 360x180, and since the ratio of the image was 1024x512, I had to multiply each value by the appropriate ratio to correctly plot the coordinates and draw lat/long lines. After getting ratios, the program reads all the meteor data from a file and adds "GeoLocation","Mass", and "Fall" all contained in a tuple, to a list. Then program plots the latitude and longitude lines, by plotting from 0 to 360 (since the actual longitude scale is from -180 to 180) by increments of 10 for longitude, and 0 to 180 (since the actual latitude scale is from -90 to 90) by increments of 10 for longitude. After adding the lat/long lines, I started plotting my meteor data on the map as circles, using a log8 function on the mass for the size of circle, and coloring red for "Fell" meteors, and green for "Found meteors.

To get the actual location of the meteor on the map, I had to add 180 for long values in GeoLocation, and 90 for lat values in GeoLocation since my scales were from 0 to 360 and 0 to 180 instead of -180 to 180 and -90 to 90 respectively. I also had to multiple each by the ratio's pulled from width and height of the actual image, and subtract the value from height for latitude since pixel 0,0 is in the top left corner, and the plot points work for 0,0 being the bottom left since my scales were from 0 to 360 and 0 to 180. I had to use these scales and ratios so that I could locate the correct pixel coordinates for plot point, since the pixels are scaled from 0 to 1024 and 0 to 512.

Design

To learn from my mistakes in the past assignment when using really long variable names, I attempted to keep all the variable names in this assignment short, but make them easy to read when some is trying to understand my program. The program starts by handling all files ("BigEarth.jpg" and "meteorite-landings.csv") before interpreting the data and adding anything to the window. The structure of my program is fairly simple, but is as follows

1) Get dimensions of "BigEarth.gif"
2) Create Window of size (dimensions.width, dimensions.height)
3) Read all meteorite data from the file "meteorite-landings.csv"
    - Verify if GeoLocation = "(0.000000, 0.000000)"
        - If yes, skip
        - If not, add GeoLocation, Mass and Fall as tuple to list
4) Add image to window
5) Get ratios using width and height of actual window and scales of 0-360/0-180
6) Add Latitude and Longitude lines
7) Plot meteor data as circles of different colors and sizes bases on "Mass" and "Fell" values.

- For plot point:
    - lat = height - (geoLocationLat + 90) * ratioLat
    - long = (geoLocationLong + 180) * ratioLong
- For size of circle, get the value of log8(mass) //used log 8 to get reasonable and different sizes for circles. For a minimum size in case it's too small to be seen, If size < 2, set size to 2
- For color of circle:
    - If "Fell", circle = red
    - Else If "Found", circle = green
    - Else circle = white

Implementation

The code is implemented in Python and interpreted with Python 3.7.1. The program starts by using the files for reading in data, getting data from the image, and adding the image to the window. After that, the program plots the latitude and longitude lines and then plots all of the meteors as circles on the map image in the window. The program takes awhile to run, since it has to plot over 30,000 points (after removing values with 0,0 or null for coordinates, and values with a mass of null or 0) and calculate their location, size, and find the appropriate color. The code is in screenshots below, but I've also included it in the comments, and you can find my code and image results at:

https://github.com/KennMaguire/Meteor

```python
#!/Library/Frameworks/Python.framework/Versions/3.7/bin/python3

import time
from graphics import *
from PIL import Image as Im
import struct
import csv
import math



#https://stackoverflow.com/questions/10759117/converting-jpg-images-to-png was used to convert my image, but the code is no longer needed for my program

with open("images/BigEarth.gif", "rb") as fhandle:         #https://stackoverflow.com/questions/8032642/how-to-obtain-image-size-using-standard-python-class-without-using-external-lib
    head = fhandle.read(24)
    print(head)
    width,height = struct.unpack("<HH", head[6:10])  #get width and height using struct, width = 1024, height = 512


print(width)
print(height)

win = GraphWin('Map',width, height)

pt = Point(100,50)

locMass = []
with open('data/meteorite-landings.csv', 'r') as metFile:
    next(metFile)
    metReader = csv.reader(metFile, skipinitialspace = True, delimiter=',', quotechar = '"')
    for row in metReader:
        #geolocation @ 9, mass @ 4
        #https://bytes.com/topic/python/answers/45526-convert-string-tuple
        if row[4] != "" and row[4] != "0" and row[9] != "" and row[9] != "(0.000000, 0.000000)":
            mass = float(row[4])
            fall = row[5]
            geo = row[9]
            geo = tuple(map(float, geo[1:-1].split(",")))

            locMass.append((geo, mass, fall))  #new list, geolocation @ 1, mass @ 2

myImage = Image(Point((width/2),(height/2)), "images/BigEarth.gif")         #https://stackoverflow.com/questions/19249859/importing-custom-images-into-graphics-py
ratioLong = width/360
ratioLat = height/180
print(ratioLong)
print(ratioLat)
myImage.draw(win)           #draw the image in the window


#plot longitude lines
for i in range(0, 360, 10):
    if i != 0:
        longLine = Line(Point((i*ratioLong),0), Point((i*ratioLong),height))
        longLine.setOutline("#3890e2")
        longLine.setWidth(1)
        if i == 180:
            longLine.setWidth(3)        #prime meridian is bold
        longLine.draw(win)

#plot latitude lines
for i in range(0, 180, 10):
    if i != 0:
        latLine = Line(Point(0,(i*ratioLat)), Point(width, (i*ratioLat)))
        latLine.setOutline("#3890e2")
        latLine.setWidth(1)
        if i == 90:
            latLine.setWidth(3) #equator is bold
        latLine.draw(win)
#total = 0
#plot points on the map
print("Plotting data")
for i in locMass:                                   #adding 180 to long, and 90 to lat since measurements in my space are from 0 to 360 and 0 to 180
    lat = height - (((i[0][0] + 90) * (ratioLat)))      #have to subtract from height since pixel 0,0 is in the top left, and coordinates work for bottom left position
    long = ((i[0][1] + 180) * (ratioLong))
    #print((long,lat))
    size = math.log(i[1],8)                     #using log8 for appropriate scaling
    #print(size)
    if(size < 2):
        size = 2
    cir = Circle(Point(long, lat), size)        #Point holds coordinates, size is the size of the circle
    if i[2] == "Fell":
        cir.setOutline("#6c7287")
        cir.setFill("red")
    elif i[2] == "Found":
        cir.setOutline("#6c7287")
        cir.setFill("#45cb06")
    else:
        cir.setOutline("white")
        cir.setFill("#1cba5b")
    #total += 1
    #print(total)
    cir.draw(win)

print("Finished Plotting")

win.getMouse()
win.close()
```

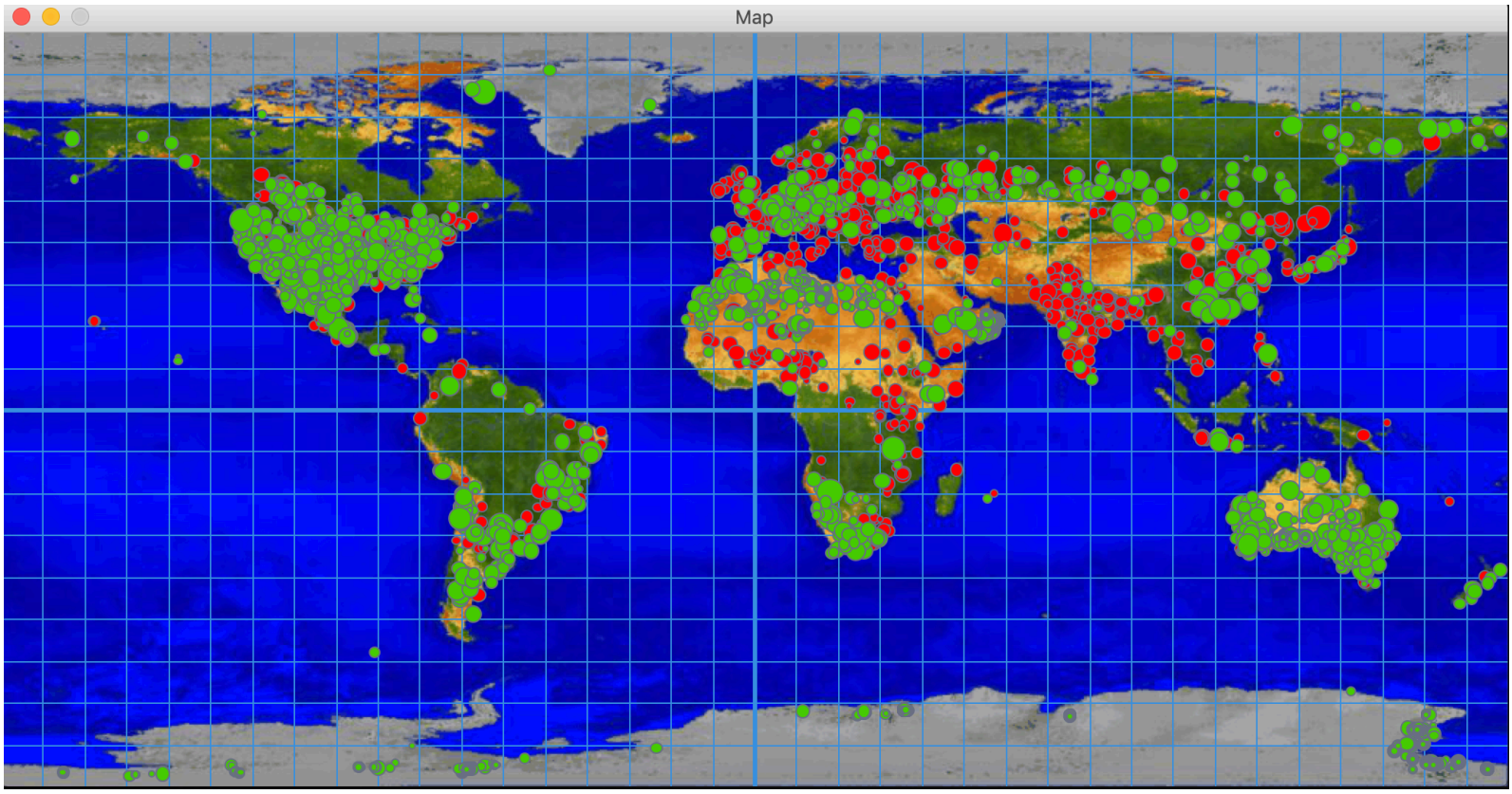
Sample Runs

The output of the program should always be the same, but without a border on my circles, it was really difficult to see the individual plot points. Each plot took around 20-25 minutes.

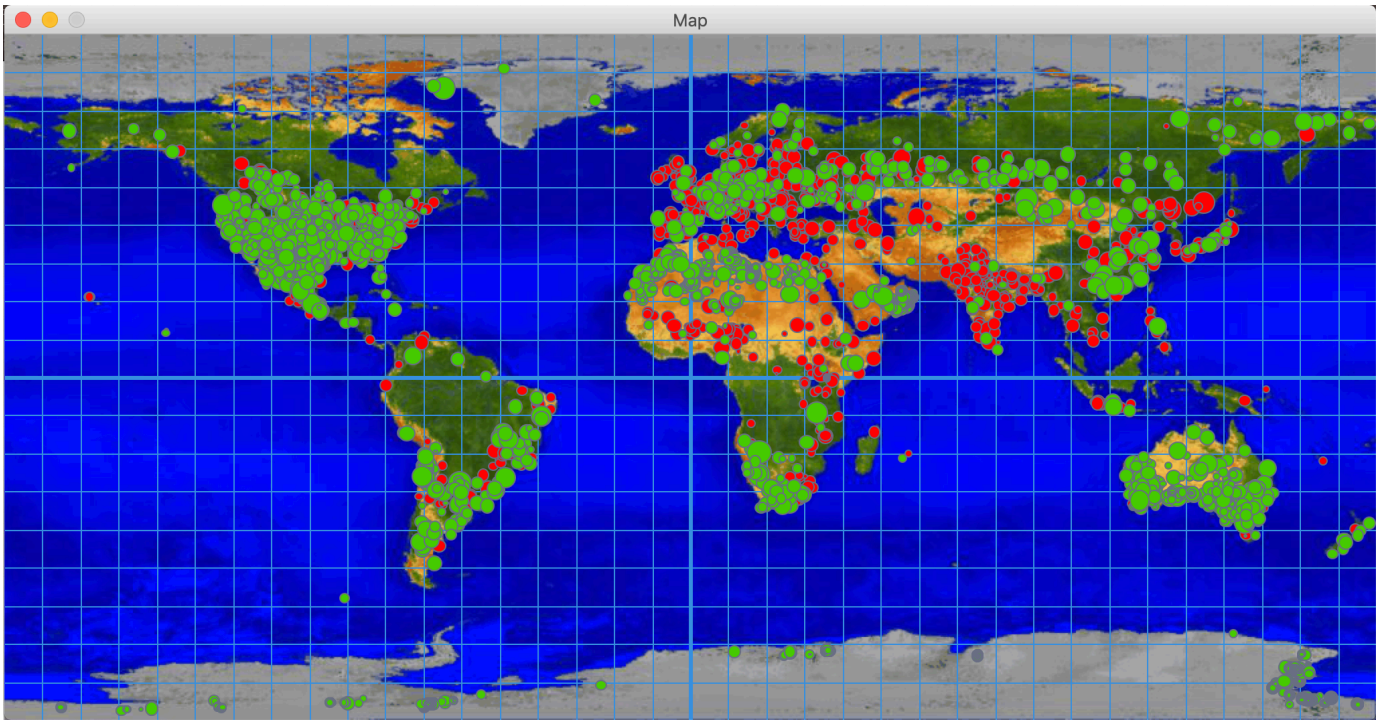
I have three plots below:

1) Plot with gray borders on circles and minimum circle size of 2
2) Plot with gray borders on circles and no minimum size
3) Plot with no borders and no minimum size

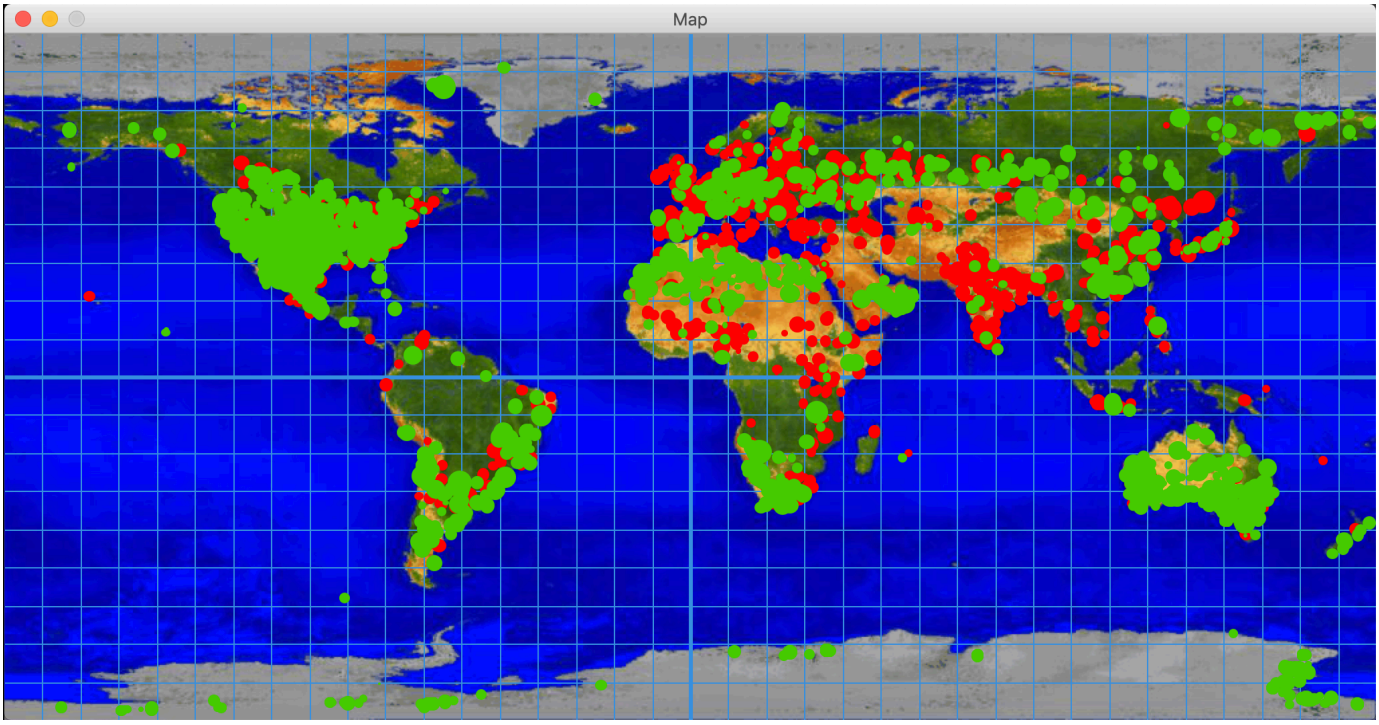
#1



#2



#3



Conclusions

After writing this program, I'm curious if the it would've run faster in Racket, or maybe another compiled language like C of Java. Getting all of the data from the image and files is fairly quick, but actually plotting the data takes 25-30 minutes. Writing the code was fairly quick even with the learning curve with the Python Graphics package, but it would've likely been much faster to plot the points in another language. I'll likely play with this program more in the future to see if I can increase the efficiency of plotting by having all of the values ahead of time, but I don't think it will help the overall time of running the program. If that doesn't help, I might try programming this again in Racket or Java to see if that increase the plotting time.

Sources

https://bytes.com/topic/python/answers/45526-convert-string-tuple

https://stackoverflow.com/questions/19249859/importing-custom-images-into-graphics-py

https://stackoverflow.com/questions/19249859/importing-custom-images-into-graphics-py

http://anh.cs.luc.edu/handsonPythonTutorial/graphics.html

mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf