Image Processor

1 Introduction

Hello everyone! This week, we will be working with classes we provide to construct your very own class, as well as getting some nice 2D array practice. Behold your next assignment: the Picture Befuddler!

2 Problem Description

So here's the scoop. You're a museum thief — but not just any museum thief. You are a museum thief with a sense of humor. Your master plan to confuse the police investigators is to leave multiple fake copies of the art that youve stolen behind in your wake. But to make things even more interesting (or perhaps frustrating for your chasers), youve modified all of the fake copies to be silly alterations based on the original picture, some of which are barely recognizable. You must use your knowledge of programming and RGB pixels to create simple image filters. Here are the following ways you can choose to alter copies of your stolen picture:

1. *greyscale():* Requires that you take the average of the RGB (red, green, and blue!) color channels for each pixel and then set each of the channels to that averaged value.
2. *invert():* Subtract each of the RGB channel values from the maximum possible value allowed (255) and then set each channel to its respective difference.
3. *noRed():* Remove the red component from each pixel so only the blue and green components are affecting the image.
4. *blackAndWhite():* Like greyscale, take the average of the RGB values for each pixel. If the average for the pixel is above 127, set that pixel to white (255, 255, 255). If it is below, set it to black (0, 0, 0).
5. *maximize():* For each pixel, find the component with the highest value. Set the remaining two values to zero. If two are tied for highest, set the third to zero; if all three are tied for highest, leave it as is.

6. *overlay(Pic other):* This alteration takes in another picture and overlays the two pictures over each other, starting at the top left corner of both images.[1] To accomplish this, average the R, G, and B values for both images. Think carefully about which pixels you are going to loop through, as in this method you will need the pixel values of not one but two different images!

3 Solution Description

Create a class called ImageProcessor that implements all of these methods. Some things to note about ImageProcessor: You must have a constructor for ImageProcessor that takes in a `Pic` object as its parameter. Any additional constructors are up to you.

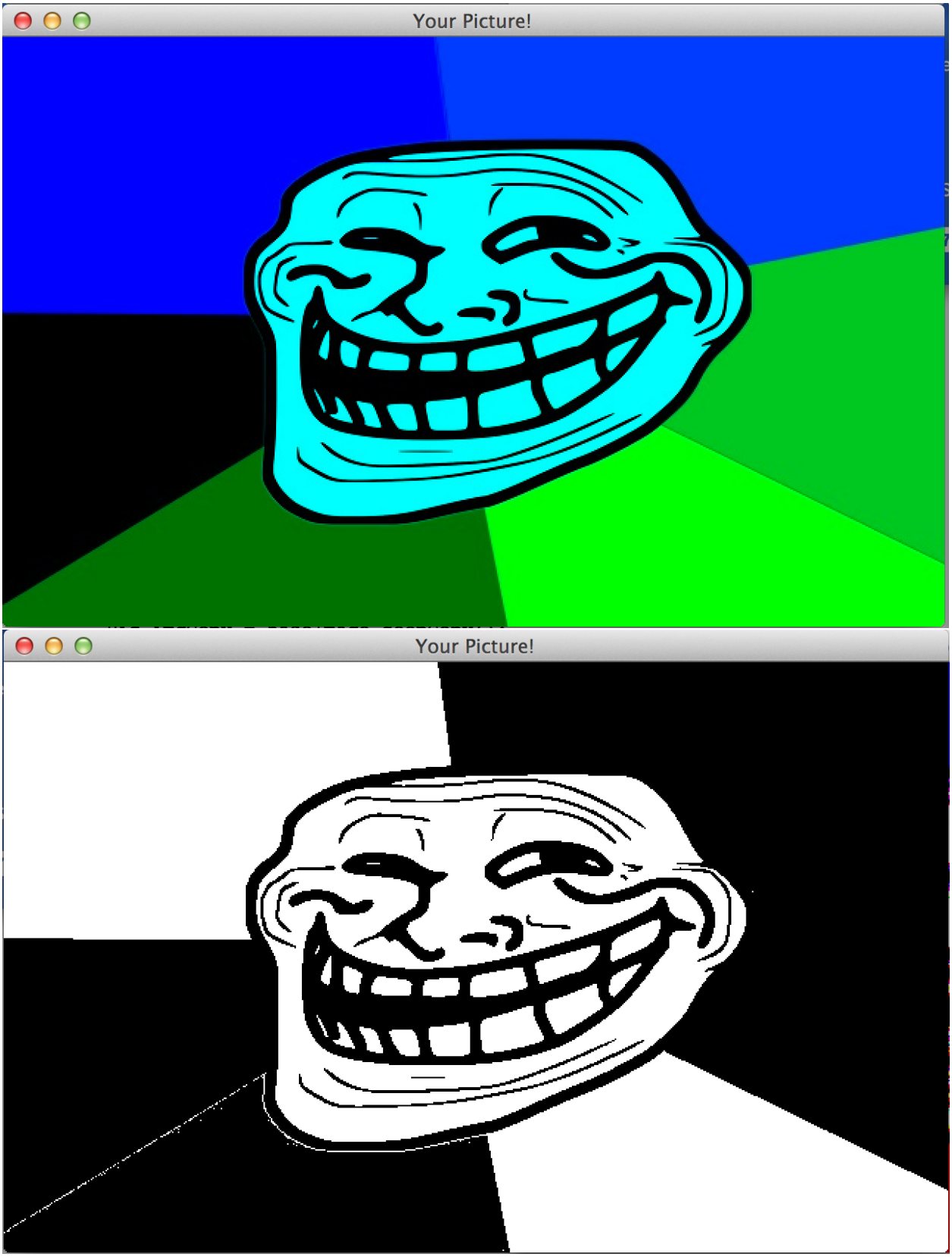
- Use the given Pixel and Pic classes to write ImageProcessor; thus, ImageProcessor is manipulating a picture of type Pic which contains a two-dimensional array of Pixel objects.
- When instantiating an ImageProcessor, you should set some `Pic` instance variable of ImageProcessor to be the clean, unmodified image. All of the manipulation methods will use this unmodified image as the base image.
- In the modification methods, do not modify the original picture! Instead, create a temporary copy of that picture which you will then modify using the methods of the Pic and Pixel classes. This allows you to call the manipulation methods one after the other without worrying about them stacking on top of each other.
- Each modification method should then return the modified Pic. These methods should be instance methods (that is, non-static methods).
- Finally, write a main method that instantiates an ImageProcessor using an image name passed in through the command-line arguments. Then, call each of your manipulation methods on that object and show the returned picture. If the user provides a **second** command line argument, call overlay() using that second argument. Yes, this means your program should open 6 different windows when it is run.

4 Example Output

Using our provided picture, here is some example output for greyscale, invert, noRed, blackAndWhite, maximize, and watermark, respectively:

[1] In the case where the second image is larger than the first, you should ignore the parts that do not overlap. The final overlay dimensions should be the same as the original image.


Your Picture!
Your Picture!


Your Picture!
Your Picture!


Your Picture!
Your Picture!

5 Tips

First, make sure you understand how RGB pixel color values work. Essentially, each pixel has a value 0 and 255 for the Red, Blue, and Green intensities, respectively. All colors are made up as varying amounts of these values. Do not worry about the alpha value for this assignment (though you are free to play around with it if you would like).

- Think about how you might iterate through a 2D array to find all of the pixels. It takes one loop to find all the elements within a 1D array. How many might it take to do the same for a 2D array?
- How might you get the color channels of every pixel? Make sure you read the methods in both Pic and Pixel, you will need to use the methods in these classes to implement ImageProcessor.
- Since you are making copies of the picture youre manipulating, you will have to make sure you can return that copy. Be sure to read the Pic class to see if there are any way you can copy a Pic object.
- You are not to modify Pic.java or Pixel.java. You are given every method required to do this assignment already. Your submissions will be run with the unmodified Pic.java and Pixel.java, so if your program is relying on changes youve made then your grade will suffer.

6 Checkstyle

Review the Style Guide and download the Checkstyle jar and associated XML file. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | wc -l
     2
```

> Food for thought: is there a one-liner like above that shows you only the number of errors? Hint: `man grep`.

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | findstr /v "Starting
    audit..." | findstr /v "Audit done" | find /c /v "hascode()"
0
```

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **10** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

7 Turn-in Procedure

Submit all of the Java source files you created to T-Square. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, the `cs1331-checkstyle.xml` file, or any `.java` files we have provided you. When you're ready, double-check that you have submitted and not just saved a draft.

8 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.[2] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[2]Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is at midnight. Do not wait until the last minute!