Module III: Image Algorithms

Learning Objectives

- Learn about image algorithms
- Learning about how to negate an image
- Learning about how to convert an image into grayscale and how to adjust it based on perceived colors

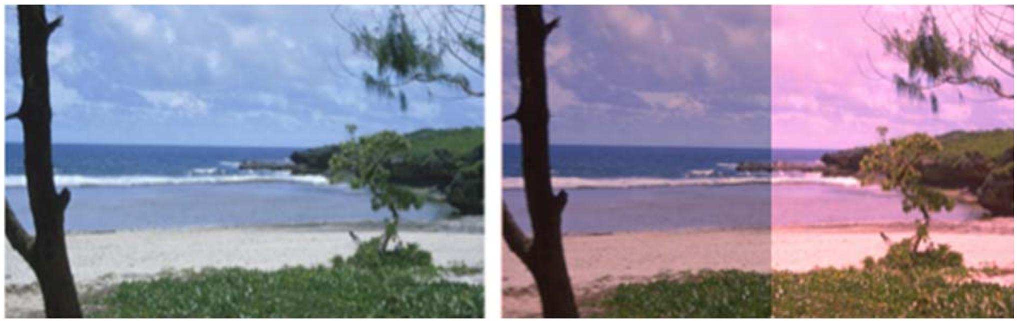
You will use the skills you have acquired so far (arrays, loops and images), and develop algorithms to manipulate images. You will also add to your understanding of media in the process. For example, you will learn how we negate an image or convert it to grayscale and adjust it based on perceived colors.

Faking a sunset

Everyone loves a good sunset scene. The warm red glow of a setting sun makes for really nice pictures. But did you know that if you don't want to wait till sunset to take a picture, you can fake a sunset.

Since sunset scenes have a red glow, let's try using the increaseRed() method to create a sunset effect. This will increase the red values of all of the pixels in the image. However, since the maximum red value of a pixel is 255, this will selectively emphasize pixels which have very little red in them over pixels which already have a lot of red in them (and will therefore either already have a red value of 255 or are already very close to 255).

An alternative to this is to reduce the blue and green values in the image. By decreasing the non-red values of each pixel, we can make the red values look more intense. Wait! you may be thinking. Won't this have the opposite problem (blue and green could already be close to 0, making it hard to decrease them further?). While that is true, if blue and green are close to 0, red is already emphasized in that pixel, so it shouldn't be a problem. Take a look at the picture below. On the left is the original. The left side of the right image shows the results of an algorithm that decreases blue and green. The right shows the results of applying increaseRed(). Which do you think looks better? Look at how different areas of the image are affected by the two algorithms (such as the grass, the sand, the ocean, and so on).



Let's see how to create a sunset method. We will need to change more than one color value at once. Let's try reducing the blue and green values of each pixel by 30%. A good first step in writing code is always to write pseudocode describing what you want to do. Here's the algorithm for faking a sunset by reducing the blue and green values:

```
1. Get the array of pixels from the picture
2. Loop through the array, one pixel at a time
    1. Set the blue value of the current pixel to 0.7 times the original value
    2. Set the green value of the current pixel to 0.7 times the original value
```

To implement the algorithm, we'll need to decide what kind of loop to use. In this case, a foreach loop would be most appropriate, since we are iterating through an array, something that foreach loops are specialized to handle. This goes into Picture.java

Here is some sample code:

```
/**
 * makeSunset: create a sunset effect by decreasing the amount of blue and green in each pixel
 */

public void makeSunset() {
   Pixel[] pixelArray = this.getPixels();
   int value = 0;

   //loop through the pixels
   for (Pixel pixelObj : pixelArray) {
      //change the blue value
      value = pixelObj.getBlue();
      pixelObj.setBlue((int) (value * 0.7));
      //change the green value
      value = pixelObj.getGreen();
      pixelObj.setGreen((int) (value * 0.7));
   }
}
```

Now that we have this method written, here's how we can test it:

```
String file = "<you file path>\\MediaSix\\src\\mediasix\\beach.jpg";
Picture pictureObj = new Picture(file);
pictureObj.makeSunset();
pictureObj.repaint();
```

It is important to note the separation of algorithm and implementation. The algorithm typically focuses on key ideas (such as changing the green and blue values of each pixel). The implementation addresses details such as which type of loop to use. You could implement the same algorithm using a for loop, for example.

did I get this?

We've seen how to create a sunset scene by manipulating the blue and green values of every pixel in an image. We implemented it using a foreach loop. For this exercise, we would like you to implement the same algorithm using a for loop.

Solution:

```
public void makeSunsetForLoop() {
   Pixel[] pixelArray = this.getPixels();
   int value = 0;
   Pixel pixelObj;

   //loop through the pixels
   for (int i =  0; i < pixelArray.length; i++) {

      pixelObj = pixelArray[i];
      //change the blue value
      value = pixelObj.getBlue();
      pixelObj.setBlue((int) (value * 0.7));
      //change the green value
      value = pixelObj.getGreen();
      pixelObj.setGreen((int) (value * 0.7));
   }
}
```

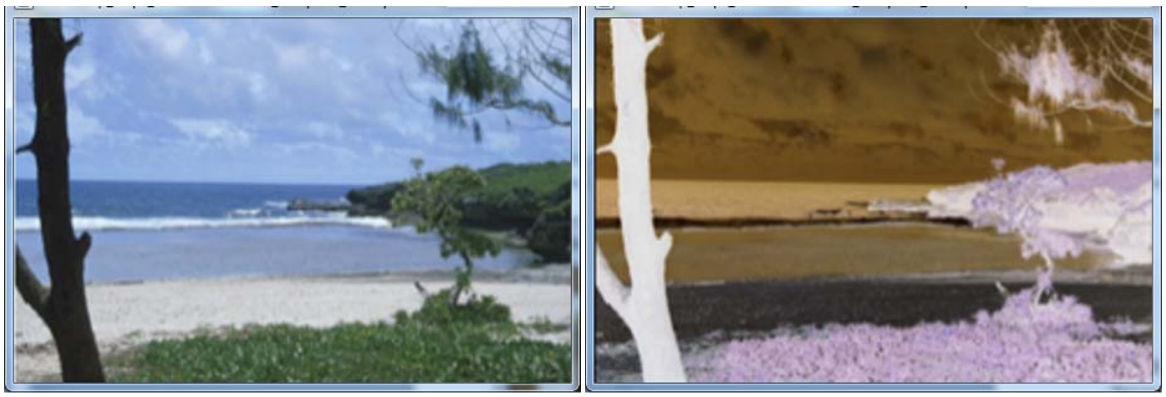
Tip...

If you are uncertain what the file path is try this to get your directory:

```
String picFile = FileChooser.pickAFile();
System.out.println(picFile);
```

Negating a Picture

Remember the old days of film cameras? When film came out of the camera, it looked strange, as if all the colors were opposite to what they were supposed to be. Film is a negative, where white becomes black, black becomes white and every color is the inverse of itself.



The basic principle of negating a picture is that White should become Black (255,255,255 becomes 0,0,0) and Black should become White (0,0,0 becomes 255,255,255). Speaking more generally, the current value of each color of each pixel should be subtracted from 255.

Here's the algorithm for negating an image:

1. Get the array of pixels from the picture
2. Declare variables to hold any temporary values inside the loop (such as the current red, green, and blue values of the current pixel)
3. Loop through each pixel
    1. Set the red value at the pixel to 255 - current red value
    2. Set the blue value at the pixel to 255 - current blue value
    3. Set the green value at the pixel to 255 - current green value

Here is an implementation of this algorithm using a for loop.

```
public void negate() {
  Pixel[] pixelArray = this.getPixels();
  Pixel pixelObj = null;

  int redValue, blueValue, greenValue = 0;

  //loop through all the pixels
  for (int i = 0; i < pixelArray.length; i++) {
    //get the current pixel
    pixelObj = pixelArray[i];

    //get the values
    redValue = pixelObj.getRed();
    greenValue = pixelObj.getGreen();
    blueValue = pixelObj.getBlue();
```

```
      //set the pixel's color
      pixelObj.setColor(new Color(255 - redValue,
          255 - greenValue,
          255 - blueValue));
    }
  }
```

Tip...

> You'll want to make sure that the picture you choose isn't too large, since otherwise you may run out of memory. Why is this? Because a larger picture will have a larger pixel array.
>
> You can fix this by making an image smaller using:
>
> ```
> Picture smallerPict = p.getPictureWithHeight(480);
> ```

Changing an image to grayscale

Another common image manipulation algorithm is changing an image to grayscale. Devices such as the Amazon Kindle are limited to grayscale, and before color screens, grayscale was the best kind of display option available. Instead of viewing the world in black and white, people could now see pictures in shades of gray.

Grayscale (as the name kind of implies), ranges from black to white, and all shades of gray in between. In terms of our representation of pixel color, this means that red, green and blue values of the color will all be the same. So, how can we change any color to gray? What number should we use for all three values? Well, we can use the *intensity* of the color for our grayscale "magic number".

The intensity of a color is the average of all the color components. The equation that we want is: intensity = (red + green + blue) / 3.

Here's the algorithm for converting an image to grayscale:

1. Get the array of pixels from the picture
2. Declare a variable to hold the intensity of the current pixel
3. Loop through from the start to end of the array
    - Calculate the average of the current values
        - (redValue + greenValue + blueValue) / 3
    - Set the red value to the average
    - Set the green value to the average
    - Set the blue value to the average

Here is what the code would look like, using a for loop:

```
public void grayscale() {
    Pixel[] pixelArray = this.getPixels();
    Pixel pixelObj = null;

    int intensity = 0;

    //loop through all the pixels
    for (int i = 0; i < pixelArray.length; i++) {
      //get the current pixel
      pixelObj = pixelArray[i];

      intensity = (pixelObj.getRed()
          + pixelObj.getGreen()
```

```
            + pixelObj.getBlue()) / 3;

        //set the pixel's color
        pixelObj.setColor(
            new Color(intensity, intensity, intensity));
    }
}
```

This implementation of grayscale was a very naive implementation. Our eyes perceive blue to be darker than red or green, even when the same amount of light is reflected from an object. Thus, a better grayscale algorithm should take this into account.

How can we build a better algorithm? Instead of taking a simple average of all three color values, we can apply a weighted average, where a certain percentage of each color value is used, instead of being divided equally. The resulting value is termed *luminance*. Luminance can make the grayscale image look better. In our updated algorithm with luminance, we can weight green the highest (58.7%), red a bit less (29.9%) and blue the very least (11.4%) emphasis. Which algorithm do you think works better?

did I get this?

Let's use our new model, and create a new method called grayscaleWithLuminance, that generates a grayscale image. Remember to use the new algorithm for calculating intensity!

Solution:

```
public void grayscaleWithLuminance() {
    Pixel[] pixelArray = this.getPixels();
    Pixel pixelObj = null;

    int intensity = 0;

    //loop through all the pixels
    for (int i = 0; i < pixelArray.length; i++) {
        //get the current pixel
        pixelObj = pixelArray[i];

        intensity = (int) (pixelObj.getRed()*0.299
            + pixelObj.getGreen() * 0.587
            + pixelObj.getBlue() * 0.114) / 3;

        //set the pixel's color
        pixelObj.setColor(
            new Color(intensity, intensity, intensity));
    }
}
```

Summary

In this module, we learned how to use the skills we have learnt about arrays, loops and images and apply them to image manipulation algorithms. We learnt how to fake a sunset, negate a picture, convert a picture to grayscale, as well as a better grayscale algorithm that weights the colors of a pixel based on luminance.