CS 157 Lab 15 November 19, 2009
Picture File Processing

The goal for this lab is to put a subject picture into another (target) picture using the technique known as "chroma-key" or "ultimate". The way this works is to have a solid background color for the subject, usually blue or green as bright and evenly lit as possible, and then programmatically replace each pixel whose color is close to that background color with the corresponding pixel of the target picture. We will start with some helper methods. Use the Media Computation project we started in the last lab.

1. We need a method that will make both pictures the same size. There are many ways to do this. Perhaps the simplest is to make the bigger picture as small as the smaller. Create a method called **trimTo** that returns a Picture and takes three arguments: a Picture, an integer that stands for the smaller width, and an integer that stands for the smaller height. If either the width or the height of the picture passed in as an argument is smaller than the width or height arguments, print a message saying you cannot trim to a smaller dimension and return the argument picture. Otherwise, create a new picture with dimensions given by the arguments, set up nested loops as before, and set the color of the current pixel in the new picture to that of the corresponding pixel in the argument picture. Then return the new picture. If you wish, you can "offset" the pixels you use from the bigger picture by adding half the difference between the widths to each x-coordinate and a similar offset for y. Test your method by copying two picture files of different sizes into your space, and then in the main method, show the smaller picture, call trimTo using the bigger picture as an argument with width and height arguments from the smaller picture. Then show the picture that is returned. The two pictures shown should have the same size.

2. You need to figure out the red, green, and blue values of the background of the subject picture. If the background were pure black, those numbers would all be zero. But you're probably not that lucky. So you'll need a method that shows you a bunch of the numbers so that you have a good idea of what values to expect in the background. So create a method called **printBoxColor** that takes a Picture and an integer called **size** as arguments. This method should have nested loops as before, but all it does is print the red, green, and blue values for the pixel at that column and row location, separated by a space. The upper bound for the loops should be **size**. Print a blank line after each row, and make sure to add extra spaces between the different pixels so you can tell where one ends and the next begins. Try using a size of 100. Look at the output, and try to determine the range of values for each of the colors.

3. Create a boolean method called **isCloseToBackgroundColor** that takes a **Pixel** argument. Have it return **true** if the Pixel argument's color (red, green, and blue) is "close" to what you determined to be the range of the background color. Otherwise return **false**. This method will be one long "if" statement, with lots of "&&" in the condition. You may need to experiment with possible values for how close, so be sure to use named quantities rather than literal numbers for this.

4. Create a method called **insert** that takes the subject and target pictures as arguments in that order. Set up the same nested loops you've been using. Inside the inner loop, if the current subject pixel's color is close to the background color (using your method from the previous step), send a "setColor" message to the current subject Pixel and have it change color ("setColor") to that of the Pixel in the target picture located at the same column and row. This will be one long statement, but you can break it into smaller steps with temporary variables if you wish. Nothing should happen if the color of the subject Pixel is not close to the background color. Add a call to **insert** in your **main** method, and then **show** the subject picture. If everything went very well, you should see the background that was a solid color replaced by the corresponding part of the target picture. You will probably have to experiment with the numbers in your boolean method to get good results. Of course, the best results would occur if the background was a solid color that did not occur at all in the foreground.

5. If you finish this early (for example, if you had a solid black background to work with), try some other effects. For example, try decreasing the amount of red (or another color) in the picture by a fixed amount, such as 50%. Try creating a negative of a picture. This is very easy. Just change each color value to its "complement": 255 minus the value. So zero would become 255 and vice versa. Here's a useful idea: mirroring. You've already done this to some extent when you reversed the picture vertically, only now you just want to reverse half of a picture horizontally and superimpose that onto the other half. Here's an example of a good task. On Blackboard under Course Documents you will find a picture of the Temple of Hephaistos. You can right click and save it on your computer. Then you should be able to "repair" the temple by mirroring the left side and putting it on the right. The hardest part is figuring out where the middle of the temple is (which column). But this is a worthy project for you to work on during the long Thanksgiving break when you get tired of eating.