Assignment 8 Photo Editor

1 Overview

You now have enough knowledge to complete your very own (albeit simpler) version of commercial picture manipulation software! As a result of this project, we hope that you will feel considerably more comfortable with arrays, loops, and one example of dynamic programming, all while finishing something really cool that you can share!

Seam carving was first published in this paper by S. Avidan and A. Shamir in 2007. It's a relatively accessible paper, and you'll want to have this link available to refer to it as you implement seam carving. The approach uses dynamic programming in order to quickly find the lowest-cost seam from top to bottom or from left to right in an image. Repeated removal of seams allows for image-aware resizing to any smaller size. (The paper offers a seam-insertion algorithm for resizing to larger sizes, too, though that won't be our first priority here.)

You may complete this assignment individually or in partnership.

As part of this project, we will be mainly dealing with two classes:

- The `Picture` class encapsulates information associated with a picture.
- The `Pixel` class encapsulates the information associated with each *element* of that picture (also called a pixel).

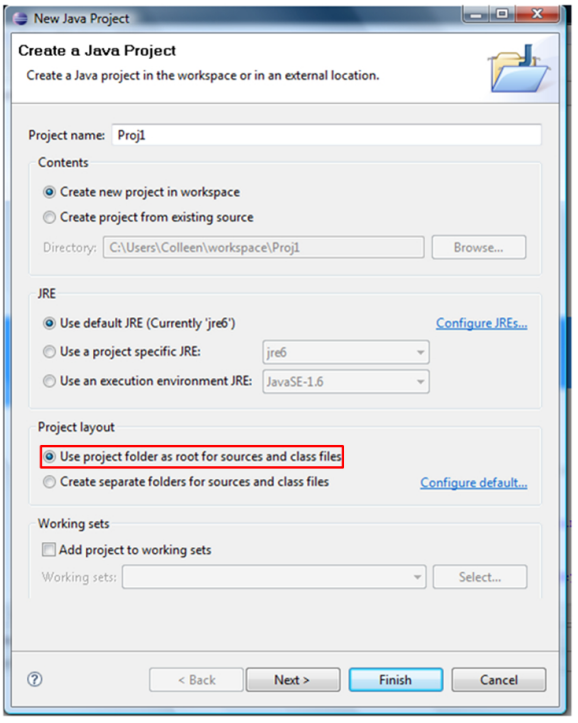
A picture is stored in a computer as a two-dimensional array of pixels, where each pixel represents a small section of the original picture. The pixel at the top left of the image has the coordinate (0,0), with the x-coordinate increasing rightwards, and the y-coordinate increasing downwards.

Every pixel in a picture has a certain color, which is represented using RGB values. The idea is that every color can be specified by three components – red, green, and blue – where the values for each component range from `0` to `255`; the higher the value, the more the contribution of that particular color. Black is the absence of color, and so all the components have value `0`. White is the presence of all color, and so all the components have value `255`.

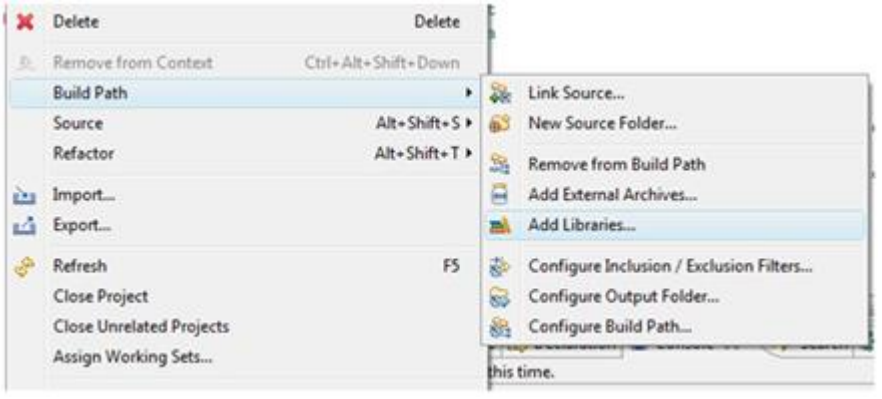
Every pixel also has an alpha value to represent its transparency: its value ranges from `0` to `255` as well, with `0` representing a pixel that is completely transparent, and `255` representing a pixel that is completely opaque.

2 Downloading and Running the Framework Code

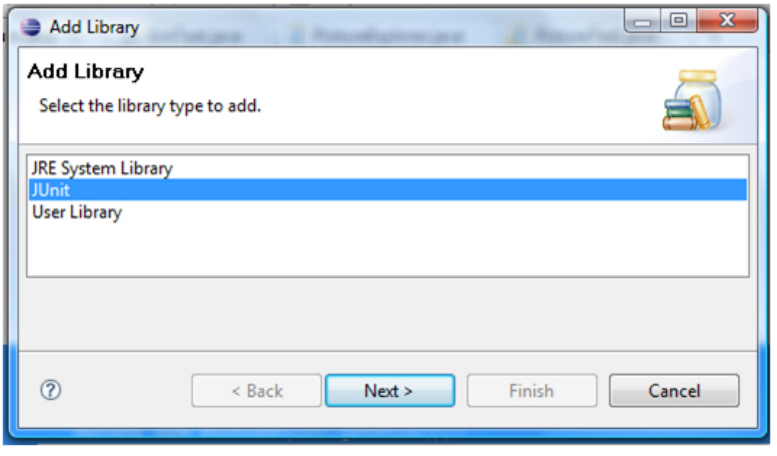
1. From the Eclipse file menu, select `New` → `Java Project`.
2. Name the project named `PhotoEditor`.
3. Select `Use project folder as root for sources and class files` under `Project Layout`, as shown below



4. Download the files from `Hw9.zip`. Unzip the files and paste them into your `PhotoEditor` folder in Eclipse. (Right-click on the project `PhotoEdistor` and select Paste.)
5. There will be errors in `PictureTest.java`. This occurs because JUnit is not in the build path. Right-click on the project and select `Build Path` → `Add Libraries`, as below



6. Then select JUnit and click `Next >` and then `Finish`, as shown in



7. Finally, run `Picture.java` as a `Java application`.

This will prompt you to open a picture file. Once you open a picture file you will see a yellow-black crosshair on the picture: moving this crosshair around allows you to view the color information of, and the color of, the pixel located at the current position of the crosshair, in the pixel information panel.

You will implement the methods that are called by the GUI when a menu option is selected. When using the GUI, you do not need to worry about how the information is provided to the method. When you are not using the GUI you can call your methods directly from inside of a `main` method or `JUnit` test.

We have provided the `grayscale` method as an example of how you can implement a picture effect. This menu option should work in the GUI. Your tasks for this project will involve completing the other methods listed below based upon the comments in the file Picture.java

3 Tasks

The public method signatures should not be modified in any way, or else the GUI will not work and you will fail our auto-grader tests.

3.1 Required Methods

Fill in the methods below, which are described in Picture.java

Change Colors Menu

```
public Picture negate()
public Picture lighten(int lightenAmount)
public Picture darken(int darkenenAmount)
public Picture addRed(int amount)
public Picture addGreen(int amount)
public Picture addBlue(int amount)
```

Rotate/Flip Menu

```
public Picture rotateRight()
```

Seam Carving Menu

```
private int luminosityOfPixel(int x, int y)
public Picture luminosity()
private int getEnergy(int x, int y)
public Picture energy()
private int[] computeSeam()
public Picture showSeam()
public Picture carve()
public Picture carveMany(int numSeams)
```

3.2 Some Extra Credit Options (up to 15 points + 5 points)

At the top of your Picture.java file, please describe any extra credit you have completed. If you complete all of the following methods, you'll receive the full 15 points of extra credit. However, doing other creative things can also get you up to the 15 points, just make sure to describe what you did. You can also be awarded an additional 5 points of extra credit for making changes to the GUI (in `PictureExplorer.java`) and adding new functionality.

Rotate/Flip Menu

```
public Picture flip(int axis)
```

Picture Effects Menu:

```
public Picture chromaKey(int x, int y, Picture background, int threshold)
```

```
public Picture showEdges(int threshold)
public Picture blur(int blurThreshold)
public Picture paintBucket(int x, int y, int threshold, Color newColor)
```

3.3 Seam Carving

3.3.1 `private int luminosityOfPixel(int x, int y)`

You will implement `luminosity()`, which is like `grayscale()`, but uses a fancier formula to determine how light a pixel is. Here is a description of the difference between grayscale and luminosity:

http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

Luminosity is a weighted average of red, green, and blue; the weights take into account that people are much better at seeing differences in some colors (e.g., greens) than in others (e.g., blues):

```
luminosity = 0.21 * redness + 0.72 * greenness + 0.07 * blueness
```

3.3.2 `public Picture luminosity()`

You will implement this method, which returns a new Picture Object, where each pixel is gray (has equal values of Red, Green, and Blue) and where the level of gray is determined by the luminosity of the pixel.

3.3.3 `private int getEnergy(int x, int y)`

For implementing the `getEnergy` method, you need to compute the "energy" of a particular pixel in the image as defined by the paper that demonstrated the seam carving algorithm.

Let us abbreviate the grayscale intensity (the luminosity) at point `(x,y)` as $i(x,y)$. Moving horizontally, we can numerically approximate the **horizontal derivative** (rate of horizontal change in intensity) at `(x,y)` with any of three methods:

- The *forward difference*

$$i(x+1,y) - i(x,y)$$

- The *backwards difference*

$$i(x,y) - i(x-1,y)$$

- The *central difference*

$$\frac{i(x+1,y) - i(x-1,y)}{2}$$

where `i(x,y)` denotes the grayscale intensity at point `(x,y)`.

Numerically, the central difference is considered most accurate. For pixels on the edge, `i(x-1,y)` or `i(x+1,y)` might not exist, in which case you would need to use one of the others.

Vertical derivatives are similar. For example the vertical central difference would be computed as

$$\frac{i(x, y+1) - i(x, y-1)}{2}.$$

The test cases provided in PictureTest.java assume that when possible you will use the forward difference. And when the forward difference is not available you will use the backwards difference.

The paper defines the energy at each pixel using the sum of the absolute value (`Math.abs`) of two derivatives: the horizontal and vertical rates of change in grayscale intensity. Using the forward difference for both the horizontal and vertical directions:

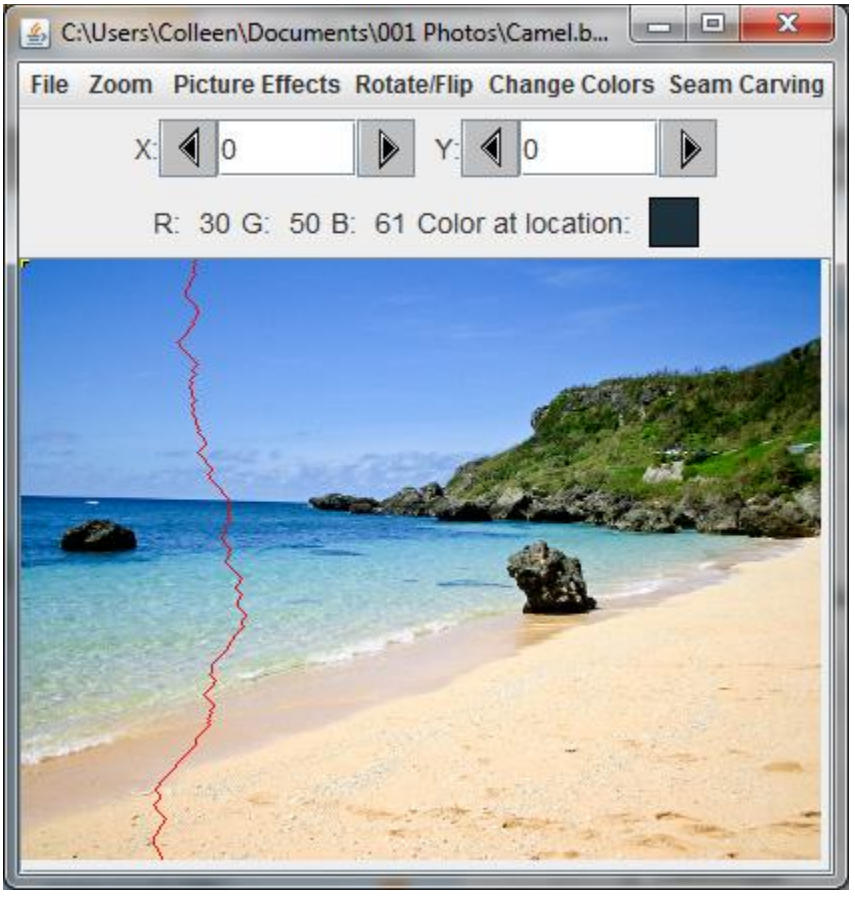
$$energy = abs(\ i(x+1,\ y) - i(x,y)) + abs(\ i(x,\ y+1) - i(x,\ y))$$

or

$$energy = abs(horizontal\_derivative) + abs(vertical\_derivative)$$

3.3.2 `private int[] computeSeam()`

You will use the method `computeSeam()` in both `showSeam()` and `carve()`. The result of calling `showSeam` on Okinawa.bmp is shown below.



This is the dynamic-programming part of the assignment. A *vertical seam* is a sequence of pixels, one per row of pixels, such that pixels in adjacent rows are no more than one column apart. Since there's exactly one pixel per row, we only have to remember the columns of each pixel; this can be represented as a single array of integers.

Step one is to create an array `table` of integers, of the same size as the image. Each entry `table[x][y]` is the total energy of all pixels in the least-energy seam starting on the top row and moving down to the ending at pixel `(x,y)`. As discussed in the paper, we can define this recursively by

$$\begin{aligned} \text{table}[x][0] &= \text{getEnergy}[x][0] \\ \text{table}[x][y] &= \text{getEnergy}[x][y] + \min \begin{cases} \text{table}[x-1][y-1] \\ \text{table}[x][y-1] \\ \text{table}[x+1][y-1] \end{cases} \end{aligned}$$

That is, the least-energy seam ending at `(x,y)` extends the best seam ending either directly above-and-to-the-left, directly above, or directly above-and-to-the-right. (Note that for pixels on the far left and far right borders, only two of these three possibilities exist.)

The dynamic programming approach is to fill out this table in order of increasing `y`: first all the entries where `y=0`, then the entries where `y=1`, etc.

We also need code to keep track, at each pixel, which of the 3 possible parent seams was chosen. We can do this basically in the same way we did in our Spampede breadth-first-search algorithms. Specifically, creating a second 2D array `parent`. The idea is that `parent[x][y]` will be the column of the previous pixel in the best seam ending at `(x,y)`. So, if while computing `table[x][y]` we decided the smallest of the three possible parent seams was `table[x+1][y-1]`, then `parent[x][y]` should be set to `x+1`. If the smallest of the three possible parent seams was `table[x][y-1]`, then `parent[x][y]` should be set to `x`. And so on. Finally, when we have filled out the `table` and `parent` arrays, we can find the best seam. First, we find the smallest `table` value in the bottom (maximum `y`) row. This will be where the seam ends. Using the parent array, we can figure out whether the seam extends up-and-left, up, or up-and-right, to find the next pixel in the seam. We continue working our way up, until we reach the top row. We can collect the columns of each pixel in the seam into an array. (The row numbers will be obvious, since there's exactly one pixel per row.)

Example: If we had a 3-by-3 image whose energies were

1	2	3
8	6	4
5	6	5

then `table` would be

1	2	3
9	7	6
12	12	11

and `parent` would be

X	*X*	*X*
0	0	1
1	2	2

(The top row of parent doesn't matter, since there are no more pixels above.)

Thus, the best seam ends in the lower right corner, in column 2, because that's the smallest value of `table` in the bottom row. Working backward through the parent array, we find that the next pixel up is also in column 2, and the pixel above that is in column 1. Thus we get the seam (from top to bottom) `1,2,2`, corresponding to the original pixels with intensities 2, 4, and 7.

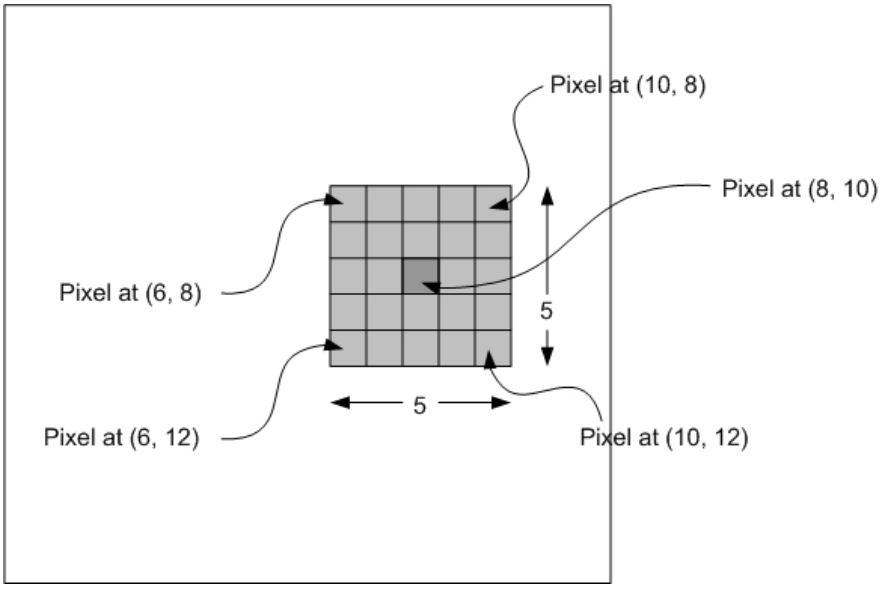
If there is a tie when you're identifying which Pixel to select as the minimum path, please select the Pixel with the same x value, then the Pixel to the left (x-1) then the pixel to the right (x+1).

Please make sure you understand where this seam comes from **before** you start coding.

3.4 Blur (part of the extra credit)

Here are some additional notes to help you understand the `blur`:

We implement this method by averaging all the pixels in a square of side (2× `blurThreshold`)+1 centered at each pixel in the current image. For example, if `blurThreshold` is `2`, and the current pixel is at location (`8`, `10`), then we are considering pixels in a 5 × 5 square that has corners at pixels (`6`, `8`), (`10`, `8`), (`6`, `12`), and (`10`, `12`); this is illustrated in the figure below. The red, blue, green and alpha values should each be averaged separately.



4 Description of Included Files

Classes you will modify:

- `Picture.java`: This class is a subclass of the `SimplePicture` class, and implements the various features that can be used to edit the picture.
- `PictureTest.java`: This class contains the JUnit test cases that test the functions in the `Picture` class.

Classes that you will use:

- `Pixel.java`: This class provides static and non-static accessor methods that allow you to obtain information regarding a pixel in a picture.
- `Color` class (Java Library): http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html

Classes you can ignore:

- `FileChooser.java`: This class controls and abstracts the file choosers that allow the user to select files to open and save. You do not need to know how this class works.
- `ImageDisplay.java`: This class controls and abstracts the image display and the crosshair feature. You do not need to know how this class works.
- `PictureExplorer.java`: This class brings together and controls the various elements of the graphical user interface. You do not need to know how this class works.
- `PictureFrame.java`: This class controls and abstracts the frame containing the picture. You do not need to know how this class works.
- `SimplePicture.java`: This class is the parent of the `Picture` class, and also stores variables and methods relating to a particular picture file, allowing the `Picture` class to deal only with the various features that can be implemented on that picture file. You do not need to know how this class works.

There are also image files that are used by `PictureTest.java` and are required for the GUI to function properly:

1. **For the GUI**:
   (a) `leftArrow.gif`: The GIF file for the left arrow in the pixel information panel.
   (b) `rightArrow.gif`: The GIF file for the right arrow in the pixel information panel.

5 Testing

We have provided you with tests within `PictureTest.java`. When you download the project, only the `testGrayscale()` and `helpersWork()` test methods should pass. You are not required to add additional tests, but you will likely need to write additional code to aid you in your debugging. For example, we wrote the following code (which is provided in `Picture.java`):

```
/**
 * Method to print out a table of the intensity for each Pixel in an image
 */
public void printIntensity(){
	int pictureHeight = this.getHeight();
	int pictureWidth = this.getWidth();
	System.out.println("Intensity:");
	for(int y = 0; y < pictureHeight; y++) {
		System.out.print("[");
		for(int x = 0; x < pictureWidth; x++) {
			System.out.print(this.luminosityOfPixel(x, y) + "\t");
		}
		System.out.println("]");
	}
}

/**
 * Method to print out a table of the energy for each Pixel in an image
 */
public void printEnergy(){
	int pictureHeight = this.getHeight();
	int pictureWidth = this.getWidth();
	System.out.println("Energy:");
	for(int y = 0; y < pictureHeight; y++) {
		System.out.print("[");
		for(int x = 0; x < pictureWidth; x++) {
			System.out.print(this.getEnergy(x, y) + "\t");
		}
		System.out.println("]");
	}
}

/**
 * Prints a two dimensional array of ints
 * @param array
 */
public void printArray(int[][] array) {
	int height = array.length;
	int width = array[0].length;
	for (int r = 0; r < width; ++r) {
		for (int c = 0; c < height; ++c) {
			System.out.print(array[c][r] + "\t");
		}
		System.out.println();
	}
}
```

6 Style Guide

10% of your grade for this project will be based on your coding style. We will deduct points for code that does not match the following style, documentation and encapsulation guidelines. (This guide is based upon the style guide written by Professor Jonathan Shewchuk's at the University of California, Berkeley)

1. Each method must be preceded by a comment describing its behavior unambiguously. These comments must include descriptions of what each parameter is for, and what the method returns (if anything). They must also include a description of what the method does (though not necessarily how it does it) detailed enough that somebody else could implement a method that does the same thing from scratch using only the provided documentation. See the comments in the framework code provided for an example of what we mean.
2. All classes, fields, and methods must have the proper `public/private/protected` qualifier. We will deduct points if you make things `public` that could conceivably allow a user to corrupt the data structure.
3. Classes that contain extraneous debugging code, print statements, or meaningless comments that make the code hard to read will be penalized.
4. Your file should be indented to clearly show the structure of nested statements like loops and `if` statements. Sloppy indentation will be penalized. Eclipse will automatically indent your code. Highlight your code and press `Ctrl+I` or `Ctrl+Shift+F`.
5. All `if`, `else`, `while`, `do`, and `for` statements should use braces, even if they are not syntactically required.
6. All classes start with a capital letter, all methods and (non-final) data fields start with a lower case letter, and in both cases, each new word within the name starts with a capital letter. Constants (final fields) are all capital letters only.
7. Numerical constants with special meaning should always be represented by all-caps `final static` constants.
8. All class, method, field, and variable names should be meaningful to a human reader.
9. Methods should not exceed about 50 lines. Any method that long can probably be broken up into logical pieces. The same is probably true for any method that needs more than 7 levels of indentation.
10. Avoid unnecessary duplicated code; if you use the same (or very similar) lines of code in two different places, those lines should probably be a separate method call.
11. Programs should be easy to read.

7 Submission Details

If you do not do the extra credit, you will only need to turn in `Picture.java`. If you make modifications to the GUI as part of the extra credit, please also submit the file `PictureExplorer.java`. You should not need to modify any other files.

8 Showing It Off

This section is not graded, and all the information contained herein is merely for personal use.
You have worked hard and selflessly on this project; what better way to enjoy the benefits than showing it off to family and friends? In this section, we will describe how to convert your project into a JAR file, or a **J**ava **AR**chive file.

1. From the `File` menu, select `Export`.
2. Select `JAR file`.
3. Under `Select the resources to export`, select `Proj1` (or the corresponding name for your project) and make sure that all of the resources are checked in the right-hand pane.
4. Check the boxes by `Export generated class files and resources` as well as `Export Java source files and resources`
5. Click on the `Browse` button to select a location for the JAR file to be saved and select `Next >`.

6. Uncheck the boxes next to `Export class files with compile errors` and `Export class files with compile warnings`. This means that your code will need to be error- and warning- free to be able to export it. Click `Next >`.
7. Click `Browse` to select `Picture` as the "main class" that you want to run
8. Currently, you will not be able to load the images within this directory using the JAR file. However, you will be able to use the GUI and load files from the file system.
9. When you double-click on your JAR file, it will run the `main` method within `Picture.java`. You can send this to your family and friends and they should be able to use it on their computers!

9 Frequently Asked Questions

Before posting on the newsgroup about any problem or error you might be getting, please consult this list of questions and potential solutions.

Q. My `showEdges` method does not pass the tests, even though my resultant picture and the test picture look *exactly* the same. What can I do?

A. If you are using the `Pixel.colorDistance` method, cast its result to an *integer* before comparing it with the threshold.

Q. I keep getting the following error: `Uncaught error fetching image: java.lang.NullPointerException`.

A. This error message arises because some, or all, of the images are not in the same folder as the result of your code. Please import the image files as well into your Java project.

Q. I keep getting the following error: `Exception in thread "main" java.lang.OutOfMemoryError: Java heap space`.

A. This error message arises when you are trying to load too big a file. Given the current framework, the project works best with moderately-sized images.

Q. I tried to compare two images together using the `equals` method in the `Picture` class, and even though they are obviously visually the same, the `equals` method returns `false`.

A. Check to see whether one of the image files is a JPEG file. The `equals` method fails on JPEG files because JPEG files use **lossy compression** to store images: in other words, they compress an image to store it, but doing so causes information to be lost. Since the `equals` method attempts a perfect information match, it does not work well with JPEG files. This problem will not arise with GIF, BMP, and PNG files, since these file types involve **lossless compression**.

Q: I run my program many times and after a while eclipse starts to get really slow.

A: When you close the picture explorer using the red X at the top right of the window (instead of using File->Exit), Eclipse doesn't really end your process. Go to the debug perspective in Eclipse and click on the red square to stop each process individually.

Q: How do I view my image without the entire Picture Explorer GUI?

A: Instead of calling the method `explore()` on the image, you can call the method `show()`.

Q: Any tips on these two dimensional arrays?

A: Here is a helpful resource for Arrays:
http://www.cs.hmc.edu/courses/2012/fall/cs60/assignments/sc/arrays.html

Q: If there is a tie for a minimum path – which path should we pick?

A: By default, please pick the cell above in a tie, if there is only a tie between the left and the right pixels, please pick the left pixel.

10 Acknowledgments

This original project in Summer of 2009 was written largely by Jonathan Kotker, based upon code written by BARB ERICSON (ericson@cc.gatech.edu) and KEITH MCDERMOTT (gte047w@cc.gatech.edu) from the Georgia Institute of Technology.