12/03/18 170002815 160027195 170006953 CS1006 – Practical 2 Ognjen Arandjelović
Mercurial Link: https://zz35.hg.cs.st-andrews.ac.uk/Project2/

1. Build Instructions

1.1 Program is run with command line arguments. Argument 0 specifies the path of the image that is to be retargeted by the program.
1.2 The GUI works by taking in the retargeted image in the form of an icon. Next, this Icon is converted into a label and added to the display panel.

2. Design:

2.1 Part 1

The main section of part one was to calculate the energy values for each pixel using the surrounding pixels' RGB values. To do this I created a PixelEnergiesCalculator class, inside which there is one public method and 3 private methods.

The public method is named calculateEnergies and takes the input of a BufferedImage and returns an energy matrix for the image as type of double[][]. This method calculates the energies for the top and bottom rows of pixels first to avoid ArrayOutOfBoundExceptions, and then proceeds to do the rest of the image. To calculate the energies, this method uses the private methods energyFunction and getRGBValues.

ArrayOutOfBoundExceptions, and then proceeds to do the rest of the image. To calculate the energies, this method uses the private methods energyFunction and getRGBValues.

The getRGBValues method uses the integer RGB value given by BufferedImage's getRGB method, and returns a size 3 integer array of the RGB value of the pixel. The conversion is done by bytewise operators.

The energyFunction method takes an input of the four RGB values for the surrounding pixels, calculates the energy and returns this value as type double. To calculate the energy, the third private class calculateSquareDelta is used – this class simply calculates the square difference between to RGB values; however its existence is justified, since it avoids code duplication.

2.2 Part2: Seam Identification:

First step: get the smallest energy calculation values.
Began with the second row, each element (x,y) in this row select the smallest value in the last row. The range of its selection is from (x-1, y-1), (x, y-1), and (x+1, y-1). Compare three last elements' energy calculation value to add the smallest one. The third add the total smallest energy calculation value from last tow row and get the coordinates of them. The third-row selection range is also the same as the second row the following diagram could show how this work.

(Take 3x4 for example) In every column, the first element is coordinate, and the other is energy calculation value. The third element is the total values, which includes the smallest value in the last row.)

(0, 0) 5	(1, 0) 6	(2, 0) 2	(3, 0) 1
(0, 1) 3 5+3	(1, 1) 4 2+4	(2, 1) 8 1+8	(3, 1) 2 1+2
(0, 2)	(1, 2)	(2, 2)	(3, 2)

For example for the third-row first element, "(0, 2)", it would select "2+4" to add to itself to get the smallest value. And the coordinates of the path will be stored in an array which is {(1,0), (1,1), (0,2)}.

Second step: store the energy calculation value and coordinates.
For every element in the second row, array list is created to store the details. The first element in the array is the total energy calculation value, which is used for comparison in the next step. The rest of the elements in the array will be the coordinates. However, some coordinate will be changed completely. Taken above example, in the second row, first array list will be {8, (0, 0), (0,1)}

However, in the third row the first array list will become {14, (1, 0), (1, 1), (0,2)} rather than {16, 0, 0), (0, 1), (0, 2)}, because the minimum path toward to (0, 2) has changed, which it began with (1, 0). Therefore, the coordinates in the array list need to be completely changed.

Third step: getting all the lines that need to be removed.
By using sort method in Java collection, with comparing the first value in the array, each array list will be sorted in descending order. All these arrays are stored within an array list.

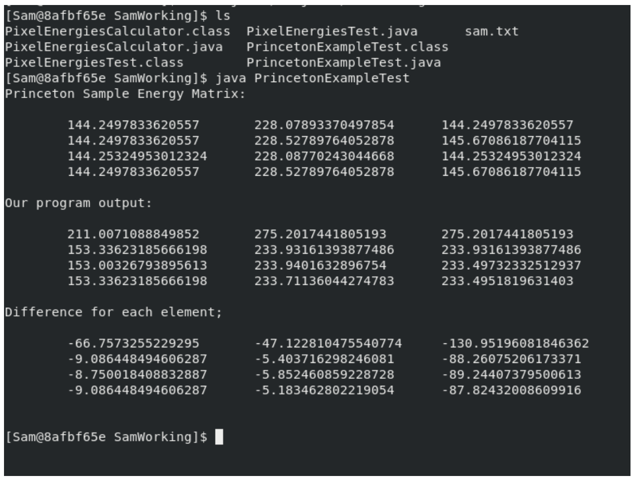
2.3 Remove the elements:
The function of part three of this project is to remove the seam specified by part two. To remove the seam I created a class called SeamCarver, which has two methods: carveVertical and carveHorizontal. These methods both take an input of a BufferedImage and a List of Lists of Integers. The image is then iterated through and copied to a new image, omitting the pixels specified in the List of Lists; the new image is then returned.

To tie all parts together, a class named Implementer was created, and includes one method which invloves instantiating each previously mentioned class and using their methods to complete the desired function of removing a certain number of seams.

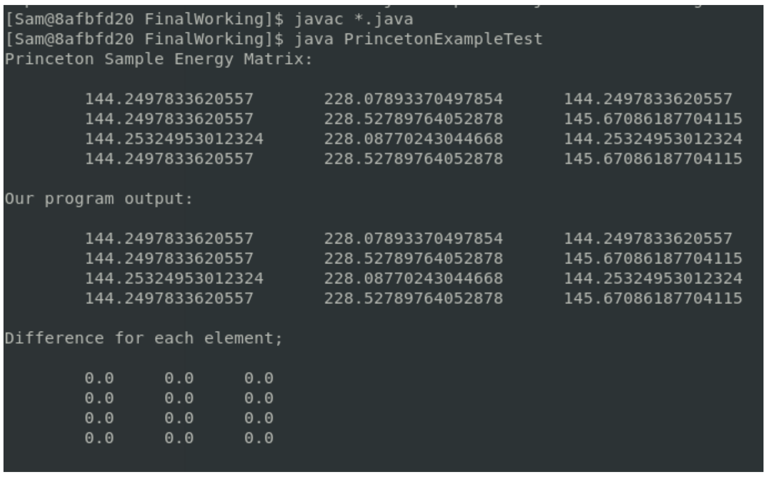
2.4 GUI Extension Part:
As can seem in the code, some of the GUI's functionality was dummied out because there was not enough time to complete the intended effect. At first conception, the hope was that by resizing the window of the GUI, a new image could be displayed reflecting this change in size. Unfortunately, we could not find a way of doing this that was compatible with run-time efficiency. Therefore, we decided to add a button that, when pushed would display a new image sized and retargeted to fit within the resized JFrame. Unfortunately, we did not have enough time to work through the bugs caused when trying to integrate this functionality with the rest of the program.

3. Testing:

3.1 Part1
To test Part 1, I used the Princeton sample data provided in the specification to compare to the results from our program. The output from the program is shown below.



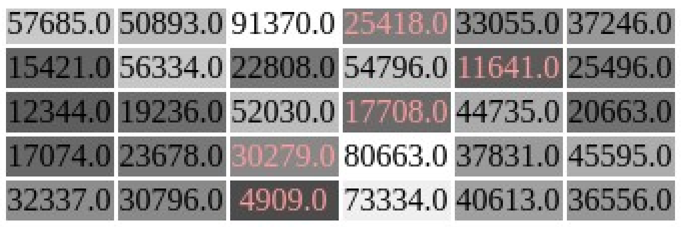
```
[Sam@8afbf65e SamWorking]$ ls
PixelEnergiesCalculator.class  PixelEnergiesTest.java      sam.txt
PixelEnergiesCalculator.java   PrincetonExampleTest.class
PixelEnergiesTest.class        PrincetonExampleTest.java
[Sam@8afbf65e SamWorking]$ java PrincetonExampleTest
Princeton Sample Energy Matrix:

        144.2497833620557       228.07893370497854      144.2497833620557
        144.2497833620557       228.52789764052878      145.67086187704115
        144.25324953012324      228.08770243044668      144.25324953012324
        144.2497833620557       228.52789764052878      145.67086187704115

Our program output:

        211.0071088849852       275.2017441805193       275.2017441805193
        153.33623185666198      233.93161393877486      233.93161393877486
        153.00326793895613      233.9401632896754       233.49732332512937
        153.33623185666198      233.71136044274783      233.4951819631403

Difference for each element;

        -66.7573255229295       -47.122810475540774     -130.95196081846362
        -9.086448494606287      -5.403716298246081      -88.26075206173371
        -8.750018408832887      -5.852460859228728      -89.24407379500613
        -9.086448494606287      -5.183462802219054      -87.82432008609916


[Sam@8afbf65e SamWorking]$ █
```



```
[Sam@8afbfd20 FinalWorking]$ javac *.java
[Sam@8afbfd20 FinalWorking]$ java PrincetonExampleTest
Princeton Sample Energy Matrix:

        144.2497833620557       228.07893370497854      144.2497833620557
        144.2497833620557       228.52789764052878      145.67086187704115
        144.25324953012324      228.08770243044668      144.25324953012324
        144.2497833620557       228.52789764052878      145.67086187704115

Our program output:

        144.2497833620557       228.07893370497854      144.2497833620557
        144.2497833620557       228.52789764052878      145.67086187704115
        144.25324953012324      228.08770243044668      144.25324953012324
        144.2497833620557       228.52789764052878      145.67086187704115

Difference for each element;

        0.0     0.0     0.0
        0.0     0.0     0.0
        0.0     0.0     0.0
        0.0     0.0     0.0
```

3.2 Part 2 Testing: Find out the minimum way in horizontal and vertical.

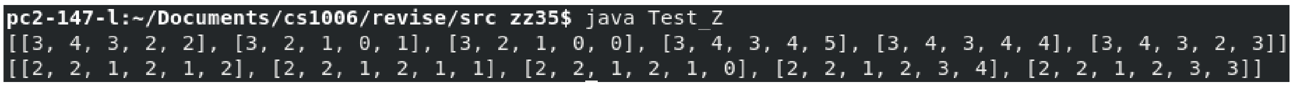
3.2.1 Input data:



57685.0	50893.0	91370.0	25418.0	33055.0	37246.0
15421.0	56334.0	22808.0	54796.0	11641.0	25496.0
12344.0	19236.0	52030.0	17708.0	44735.0	20663.0
17074.0	23678.0	30279.0	80663.0	37831.0	45595.0
32337.0	30796.0	4909.0	73334.0	40613.0	36556.0

(Hug, Ginsburg and Wayne, n.d.)
Explain: the numbers highlighted in red is the minimum value path from the top to the bottom.

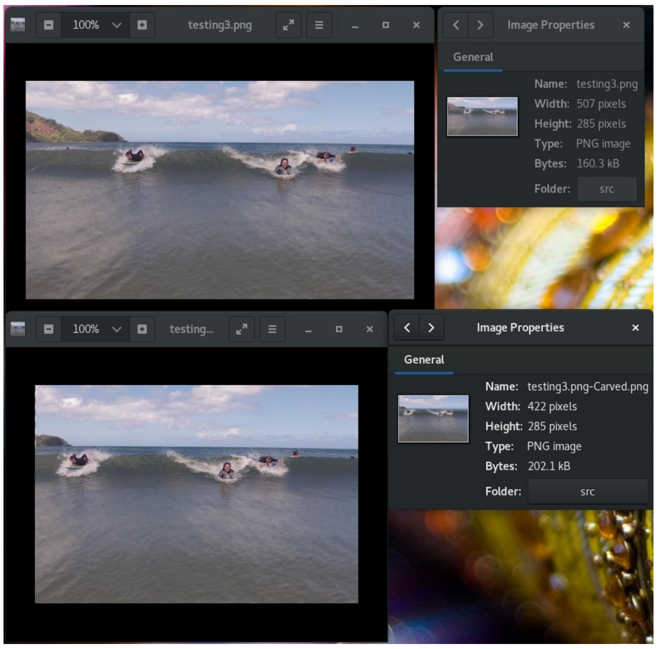
3.2.2 Testing output:



```
pc2-147-l:~/Documents/cs1006/revise/src zz35$ java Test_Z
[[3, 4, 3, 2, 2], [3, 2, 1, 0, 1], [3, 2, 1, 0, 0], [3, 4, 3, 4, 5], [3, 4, 3, 4, 4], [3, 4, 3, 2, 3]]
[[2, 2, 1, 2, 1, 2], [2, 2, 1, 2, 1, 1], [2, 2, 1, 2, 1, 0], [2, 2, 1, 2, 3, 4], [2, 2, 1, 2, 3, 3]]
```

The first row of output is to find the minimum path in descending order for vertical
The second row of output is to find the minimum path in descending order for horizontal.

3.3 General Testing:



Name: testing3.png
Width: 507 pixels
Height: 285 pixels
Type: PNG image
Bytes: 160.3 kB
Folder: src

Name: testing3.png-Carved.png
Width: 422 pixels
Height: 285 pixels
Type: PNG image
Bytes: 202.1 kB
Folder: src

References List
Hug, J., Ginsburg, M. and Wayne, K. (n.d.). Seam Carving. [image] Available at: http://www.cs.princeton.edu/courses/archive/fall13/cos226/assignments/seamCarving.html [Accessed 12 Mar. 2018].