Project 1
Seamcarve

Out: Thursday, May 20
In: Thursday, June 3, 11:59 PM EDT

1 Using IntelliJ

If you do not already have IntelliJ set up for CS16, follow the instructions **here**. Please post on Ed or come to hours if you have any questions!

2 Installing, Handing In, Demos

1. Click **here** to get the stencil code from GitHub (refer to the **CS16 GitHub Guide** for help with GitHub and GitHub Classroom). Navigate to your repository, click on the green Code button, and copy the project url. Then, go to your terminal in IntelliJ, `cd` to the **src** inside your **cs16** folder, and use the command:

   ```
   git clone <project url>
   ```

   Once you've cloned your personal repository from GitHub, you'll need to rename the folder from `seamcarve− <yourGitHubLogin>` to just `seamcarve`. You will have issues running your code until you make the change.

2. To hand in your project, upload your code to the Gradescope assignment through the GitHub option **(GradeScope Guide)**. Make sure your submit the **code portion to the SeamCarve assignment** and the **written problems to the SeamCarve Reflection assignment**. Remember that unlike for the homeworks, you will not be evaluating your testing suite using Gradescope.

3. To run the demo, go to this Google Drive **folder** (be sure to be logged into your brown.edu email) and download either the pkg (for MacOS users) or exe (for Windows users) file. For Mac users, if you get an error saying you cannot download from unidentified developer, go to System Preferences > Privacy & Security > General > Open anyway.

3 Introduction

In this project, you'll implement the interesting parts of the seam carving algorithm for image resizing that you learned on the first day of class. You'll write the code for pixel importance calculation as well as lowest cost seam finding.

4 Overview of Your Tasks

Your task is to fill in the `findLowestCostSeam()` method in the `MyPicturePane` class. The seam carving algorithm is not trivial so you'll want to use your program design skills to simplify your code as much as possible with helper methods. How you implement the algorithm is up to you.

4.1 Further Specifications

- The seams that your program generates do not need to exactly match the demo but they should be reasonable lowest cost seams. If your seam carver destroys obviously "important" parts of the image that the demo does not, you'll lose points.
- You must use dynamic programming to find the lowest cost seam. In other words, your code should not have a running time greater than $O(w*h)$ where w is the picture width and h is the height in pixels. If you choose to find the lowest cost seam using brute force, your seam carver will be unreasonably slow and you'll lose points.
- You're only asked to implement vertical seam carving. We do not ask you to do both, as there is no algorithmic difference between vertical and horizontal seam carving.

5 README

You're required to hand in a README text file (**must be named README**) that documents any notable design choices or known bugs in your program. Remember that clear, detailed, and concise READMEs make your TAs happier when it counts (right before grading your project). Your README should be saved in the same directory as your Java files. Please refer to the README Guide in the Docs section of the CS16 Website (link).

6 Reading

As with any assignment in this course, it is very important that you fully understand the algorithms and/or data structures that you'll have to write before you begin coding. The following resources will help you get a better grasp on seam carving or dynamic programming.

- Slides and docs on the website are your best resource.
- The original paper on seam carving is available on the website here.
- Check out the video: http://www.youtube.com/watch?v=c-SSu3tJ3ns

7 Visualizer

The visualizer consists of 3 images and a slider. The top left image is the original image that was loaded into your `MyPicturePane`, the top right image shows the seams that have been carved from the image. The color of the seams changes from white to red to black as more and more seams are carved. The bottom image is the result of removing the seams from the image and "squishing" the remaining pixels into a thinner image.

The slider simply picks how many seams the user would like to carve from the image. It ranges from 0 to the image width - 1. For the slider to work, you need to fill in the `findLowestCostSeam()` method to have it return a seam. See the next section for what a 'seam' is (i.e. how it is represented in code).

8 Your Code

For this project, you'll only be required to implement the `findLowestCostSeam()` method. You're allowed to write any number of helper methods or extra classes that you want. Below is an overview of what you need to do:

- Calculate "importance" values for each pixel in the image.
  - Take a look at the Support Code section of this handout to see how to get the color of a pixel.
  - Pixels that are very different from their neighbors should have high importances, and pixels that are similar in color to their neighbors should have low importances.
  - You can store your pixel importances in a 2D array that corresponds to the pixels of the image.
- Compute the lowest cost of each vertical seam. This is the dynamic programming step of the algorithm.
  - This is the trickiest part of the project. Remember that when performing the seam carving algorithm we need to keep track of two things for every pixel. The first is the lowest cost of a seam from the bottom of the image to this pixel. The second is which direction that seam came from (the pixel directly below, below and to the left, or below and to the right).
  - Again, you should probably use an array (or two) that corresponds to the image pixels.
- Find the lowest cost seam and return it.
  - Using the costs and directions you stored, you need to determine and return the seam itself.

- There are two steps to finding the lowest cost seam. First, find the top of the seam by looking at the costs associated with the top row of the image. Second, follow the seam down through the image using the directions that you stored.
- The seam that you return is represented by an array of ints. The size of this array is the height of the image. Each index of the seam array corresponds to one row of the image. The data at each index should be the column index of the seam in this row.
- For example, given the below "image" where '*s*' is a seam pixel and '-' is a non-seam pixel:

```
- s - -
s - - -
- s - -
- - s -
```

The following code will properly return a seam:

```
int[] currSeam = new int[4];
currSeam[0] = 1;
currSeam[1] = 0;
currSeam[2] = 1;
currSeam[3] = 2;
return currSeam;
```

9 Support Code

The following are methods of `MyPicturePane` inherited from `PicturePane` that you'll need to use:

- `int getPicHeight()` returns the current picture height in pixels.
- `int getPicWidth()` returns the current picture width in pixels.
- `javafx.scene.paint.Color getPixelColor(int row, int col)` returns the color of the pixel (row, col).

Note: We use zero-based indexing, so the pixel in the upper-left corner of the image is located at (0, 0) and the pixel in the lower-right corner of the image is located at (`getPicHeight() - 1`, `getPicWidth() - 1`).

Another note: Because the image is constantly changing width, you should be sure to use the `getPicWidth()` and `getPicHeight()` methods for setting your loop bounds and initializing your arrays.

10 On `javafx.scene.paint.Color`

In the support code we've provided you three methods: `getColorRed()`, `getColorGreen()`, and `getColorBlue()`. Each of these methods takes in a javafx `Color` and returns an integer between 0 and 255 representing either the Red, Green, or Blue value of the color taken in. Your importance values should probably take into account all three of these values in some way. Hint: RGB color differences can be positive or negative, but for calculating importance, it is the magnitude that matters.

11 Testing

The best way to test your code is to compare your results to the demo. Remember that your results do not need to be identical but should be similar (less important parts of the image are removed first). For help with debugging, we have put extra images into the course directory. You can also use any picture you want. To load a new image, just click `File` and pick your image from within the seamcarve application.

The images we're providing are located in `/seamcarve-images/`.

12 Compiling and Running

To compile and run your code, use the green play button next to `public class App`. If your play button does not work, please refer to the IntelliJ Guide section "Using IntelliJ".

13 What to Hand In

1. A filled in and commented `MyPicturePane` class and an unchanged `App` class.
2. Any other classes you wrote to help along the way.
3. A README (see the **README Guide** for help).

14 Reflection

14.1 Your Task, Handing In

Note that there are no new code files to hand in or install. Your task for this part of the project is to answer the following questions below in a PDF.

14.2 Reading

You may find a brief elaboration on the object removal process here helpful.

14.3 Written Questions

In a separate document, please answer these questions thoroughly. Each answer should contain 2-3 sentences and will be graded on thoughtfulness.

Submit your file as a PDF onto GradeScope. Make sure you are submitting to the **Seam-Carve Reflection** assignment.

1. What are some real world consequences of photo manipulation? Please be specific and respond with a thoughtful answer.
2. How could you as the programmer of this algorithm mitigate malicious use of seam carving? If you believe that you cannot prevent misuse of this algorithm, please explain why.
3. Are there situations in which removing someone from a photo *would* be justified? If so, give an example and explain why.

14.4 What to Hand In

Submit a PDF containing your answers to the written questions above. Note that this is separate from the handin for your seamcarve code.