Image Manipulation: Multidimensional Arrays and Unit tests

Due: February 14th in class

Image Library

Your team (of, in most cases, two students) will define a *class library* to manipulate images on a computer. Also note that as part of this assignment, you will be writing unit tests using the Visual Studio testing framework. So as you are developing your code, you should keep this in mind and be working on the unit tests simultaneously to working on your code. You can even use *Test Driven Development* and write the unit tests first!

The main learning objectives for this assignment are:

1. Getting used to C# (instead of Java).
2. Properly using C# properties and indexers.
3. Using the C# unit test frameworks.

Best practices using git

Your teamwork must proceed according to the guidelines described in Protected Branch Workflow at `https://gitlab.com/dawsoncollege/CSharp411/w2018/git-tutorials/blob/master/protected_branch_workflow.md`.

Each teammate should work on one or more temporary feature branches, making small, atomic commits. Each teammate's contribution must be submitted to the team for review in a pull request against the staging branch. Each internal pull request must be reviewed by the other teammate before being merged into your team's staging branch.

The quality of your commits, commit messages, internal pull requests and code review will all count toward your individual grade. Each commit message should indicate the author and reviewer of the commit. Follow the format shown in the Protected Branch Workflow document.

Getting started

To start, one team member will:

- Go to H:/411/Projects, or whichever directory you will be using for this semester's projects.
- open a Git Bash window

- configure git if needed: check git config –list to see if your user.name and user.email details from last semester are still good. You may need to update your e-mail address to match the one used to create your GitLab account)
- clone your team's repo from `https://gitlab.com/dawsoncollege/CSharp411/w2018/section02/ass1/team0x/ImageManipulation.git` (where 0x is your team number).
- cd to the ImageManipulation folder
- create and checkout a staging branch.

```
git config --list
# set user.name ans user.email if needed:
git config --global user.name "Grace Hopper"
git config --global user.email "gracie@ilovelinux.ca"
git clone https://gitlab.com/dawsoncollege/CSharp411/w2018/section02/ass1/team0x/ImageManipulation.git
cd ImageManipulation
git checkout -b staging
```

The team member will next open Visual Studio 2017 and create a new Class Library (.NET Framework) project. DO select the Create directory for solution checkbox but Do NOT select the Create new Git Repository checkbox. Name the project ImageManipulation, browse the location to H:/411/Projects and save. You will notice that VS2017 creates .gitignore and .gitattributes files for you.

Add all files, commit and push:

```
git add .
git commit -m "first commit"
git push origin staging
```

The other teammate can now clone the repository. You can create feature branches off the staging branch and start working :)

Background Information

In this assignment, you will write code to read, write, and manipulate images.

Every point in an image is a combination of red, green, and blue light intensities. This (r,g,b) triplet is called a *pixel*. For the purposes of this assignment, each value in a pixel is constrained to be between 0 and 255 (inclusive). Note that if all of the r,g,b values are the same in each pixel, then the image will appear as a greyscale image.[1]

In this assignment, you will work with 2 different types of image formats: pnm and pgm. Pgm only supports grey images, where as pnm supports colour images.

Below is a sample pnm image file taken from Wikipedia to illustrate this idea:

```
P3
# "P3" means this is a RGB color image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value each following number can be
# The part below is image data: RGB triplets
# The first number in each triplet is the red intensity, followed by green, then blue
3 2
255
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

[1] Colloquially, we refer to images such as these as being "black and white." But in reality, these images have more than 2 colours and a more accurate terminology would be "grey image."

The first line (P3) above is a format code that is used by the computer as a flag to indicate that the image is in pnm format.

Directly after the image format may follow one or more comment lines. These are lines starting with a `#` symbol and can store image metadata, such as the author's name or date modified. Note that some files may not have a comment line.

The next line has the image width and height (in that order). The third line is used to specify what the *scale* of pixel intensities to come is. For example, if the number is 255, it means that the intensities should be interpreted on a scale from 0 to 255. So 0 is black and 255 is white. If the number was 100, then the scale would be 0 to 100. So 0 would be black and 100 would be white.

All subsequent lines contain pixel (r,b,g) values. **Note that the numbers can be on the same line or on separate lines.** The numbers are listed in "row order." This means that the first 3 numbers are the 3 values (red,green, and blue in that order) for the upper left pixel. The next 3 values are the pixel in the top row, 2nd column. This continues until the first row is complete at which point the next numbers start referring to the 2nd row.

If we know that our image is in greyscale (no colour) we can use the pgm format instead.

```
P2
# "P2" means this is a greyscale image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value each number could be
# The part below is image data. Each number is the intensity value of a single pixel.
# A value of 0 would display as black and a value of 255 as white.
3 2
255
85 85 85
170 255 0
```

Further examples can be found at the wikipedia page at `https://en.wikipedia.org/wiki/Netpbm_format#PGM_example`.

The entry on the first line, P2, is the format code. If it is `P2`, then the format is pgm. If it is `P3`, then the format is pnm. The rest is very similar to the `P3` format described earlier with one difference. Since we know that P2 is grey scale, we only use one number to store the r,g,b values in a pixel. So each number in the P2 data represents an entire pixel, whereas in P3 format, we needed 3 numbers.

You can create and read these image files in a C# program using `StreamReader` and `StreamWriter` as you would any other file. You can view the contents of the files in a text editor such as `notepad++`. To view the files as pictures, you can use a software tool such as Gimp, which is installed on the lab computers and can be downloaded at `https://www.gimp.org/`. It is not required to use Gimp if you know of another software that reads .pnm or .pgm image files.

For the purposes of testing your methods, we have provided a .pnm and a .pgm image file of a cat skyping with a rodent who has a teddy bear. You can test your method on other .pnm images as well. If you create your own files to work with we suggest you keep them small (maximum 256 x 256). In this case. If you convert png, jpg images using gimp, make sure also to select "ascii" (and not raw) as the format.

Before starting this assignment, open the provided image files using both an image viewer and text editor to make sure you can see the contents. **Also, be aware that the code you write should be able to handle rectangular (not necessarily square) images**

Pixel Class

Define a new type `Pixel`. A `Pixel` has 3 integer properties, each representing a red, a green and a blue field. These properties should have public accessors but their setters should be private.

Make sure you are using proper C# *properties* as opposed to fields. Always consider whether setters are public, private, or should only be done by the constructor.

There should be two constructors:

1. One constructor should take as input 3 values: red, green, and blue (in that order). It should initialize the 3 properties of the `Pixel` to be those 3 values respectively.
2. The second constructor should take as input just 1 value `intensity`. It should initialize all 3 properties of the `Pixel` to be that same value of `intensity`.

Both of the above constructors should throw an `ArgumentException` if any of the intensities are outside of the expected range $[0 - 255]$ (inclusive).

Finally, write a method `Grey()` which returns an `int` representing the average of the 3 properties. If the average is not an integer, you should truncate the decimal point. This value will be used later to convert a colour image to greyscale.

Image Class (40 points)

Define an `Image` class which is meant to represent the abstract notion of an Image.

An `Image` consists of several properties, all of which can be read from any class, but only set from the `Image` class.

1. A `String Metadata` which can store information about the image in a text format. For example, one might include in Metadata the author's name, and the time at which the image was created.
2. An `int MaxRange` which stores the upper bound of the values of the pixels. You can assume (that is, no check is required) MaxRange is greater than or equal to all of the values in the pixel array.
3. A `Pixel[,] data` which stores the values of the various pixels. The first "row" of the array (i.e. the first dimension referenced by the first index) stores the first row of the image pixel values. This means that the length of the first dimension is the `height` of the image. The length of each 'row' (2nd dimension) is the `width` of the image. To make this property, you will need to have a backing field of a multidimensional array and add an *indexer* with public read access.

Note that the Image class does *not* store the format code. This allows an Image object to be easily exported to multiple formats.

Now add the following methods to this class.

1. A constructor that takes values for all of the attributes as input in the order in which they are listed above. Be sure to create and store a *copy* of the input `Pixel[,]` array. Your constructor should throw an `ArgumentException` if the maxRange value is negative.
2. A method `ToGrey`, which converts the `Image` to a greyscale image. A grey scale image is one in which the red, green, and blue components in each pixel are the same. This can be done by taking setting each of the 3 components equal to the average of the original three values.

   For example, if the pixel originally stored at a location is 0 for red, 100 for green, and 200 for blue, the new pixel would store 100 for all 3 attributes. Test your method using a main method to make sure that it creates a grey version of the original .pnm image.
3. A method `Flip`, which will be used to flip an image either vertically or horizontally. The method should take as input a `boolean horizontal` and return void. If the boolean is true, it will flip the image horizontally. If the boolean value is false, the flip should be done vertically. Here, we consider a vertical flip to be one in which we swap values across the `x-axis`. A horizontal flip is one in which we swap across the `y-axis`.

4. A method `Crop` which crops a rectangular section of the original `Image`. The method should take as input 4 int values: `startX` (inclusive), `startY` (inclusive), `endX` (exclusive), and `endY` (exclusive). The method should then "cut" the Image so that only the parts of the Image within the coordinates specified by the input are left. If the input arguments are invalid, then your method should throw an `ArgumentException`.

   Note we count the pixels in an image starting from 0. Note also which parameters are inclusive and which are exclusive. For example, to take the first 10 rows and 15 columns, you could call the method with input `Crop(0,0,10,15)`

Image Serialization Classes

In this section, you will create an interface plus two implementing classes to read files.

First create an interface `IImageSerializer`. This interface should require two methods:

1. `String Serialize(Image i)` : This method converts an `Image` object to a String
2. `Image Parse(String imageData)`: This method takes a `String` containing image data and converts it into an `Image` object as above.

These two methods are essentially inverses of each other.

Next, you will write 2 classes to implement this interface: `PgmSerializer` and `PnmSerializer`. These two classes should be very similar except that the `Parse` method for the `PgmSerializer` will assume the format is P2 and the `PnmSerializer` will assume the format is P3. Similarly the `Serialize` methods will produce Strings with P2 and P3 formats respectively.

Remember while loading the format of the Strings: First we have a format code (either P2 or P3). Next we have one or more lines starting with a # symbol which represent comments or metadata. Then we have 3 entries which are to be interpreted as the width, height, and maxRange of the image, respectively. You may assume that these values all exist (ie, there is a format code, there is at least one comment, there are values for height, width, and maxRange). You may also assume that the file contains the correct number of pixel values, and that each pixel is between 0 and the specified `maxRange`.

After this we have several numbers with two possible meanings:

- If the format is P3, then the numbers are to be viewed in groups of 3 numbers: the red, green, blue triplets for each pixel in the image, which have integer values between 0 and maxRange. In order to construct a single pixel, we need to read in three numbers.
- If the format is `P2`, then the numbers are to be viewed individually as the number to distribute to all 3 values of an individual `Pixel` (since a grey scale image has the same value for red, green, and blue.) In order to construct a single pixel, we only need to read in one number.

Note that the `Parse` method should throw an `InvalidDataException` if the String given is not good. For example, if the numbers are bigger than the maximum range or if there are not the correct amount of numbers.

Hint: You may use the *Split* method for this question.

Directory Reading

Lastly, you will now do some image processing by writing a few methods. This code should go in a class `ImageUtilities`.

1. Write a method `LoadFolder` which takes as input a `String` representing a directory path and returns an `Image[]`. This method should look at all the files with the extension `pnm` or `pgm` inside a directory (Hint: use the library method Directory.GetFiles) and create an Image[] based on them, using the

serializers you created above, choosing the correct one to use based on the extension of the file. To do the actual reading, you should use the `StreamReader` class.

2. Write a method `SaveFolder` which takes as input a `String` representing a directory path as well as an `Image[]` and a String `format`. This method should export every `Image` in the array to the folder given using the specified format (assumed to be "pnm" or "pgm"). To do the actual writing, you should use the `StreamWriter` class.

Unit testing

You should write unit tests for all classes except for `ImageUtilities` class by adding a second project, a Test project, in Visual Studio called `ImageTests`. (The ImageUtilities class is being omitted from the test requirement because of the dependence on file input and output.) Your grade for this part will be based on the thoroughness of your tests with some guidelines written below (these will be discussed in class as well)

1. Each method you write must have at least one test in it. More generally, each branch in your code should have at least one test in it.
2. You should have tests to ensure that your methods throw exceptions when they are supposed to.

What To Submit

To submit your team's code, open a merge request of your repo's staging branch against master. When creating the merge request:

Set the title of the merge request to Complete

Set @j-nila or @dpomerantz as the Assignee

@-mention your teammate in the description Add a label called grade (you should be able to create a new label if it doesn't exist)