CS1026a: Assignment 3

Due: Novemver 14th, 2010
Weight: 10%

Purpose:
To gain experience with

- Arrays
- Nested Loops
- Conditionals
- Making "movies" from a sequence of pictures

Task:

Digital images, both pictures and video, have enabled artists, designers, filmmakers, photographers, etc. to create various kinds of special effects. One of these effects, often seen in videos (e.g. music videos) and commercials, is called ***morphing***. Basically, the artist starts with two digital images and then, using digital techniques, converts one into the other by constructing a number of intermediate images. Each of these intermediate images is constructed from the pixels in the two initial images that the artist starts with. The construction is done systematically, so that when looking at a sequence of pictures ***startingPicture, p1, p2, p3, endingPicture***, it appears that each successive intermediate picture ***p1, p2*** and ***p3*** becomes more like ***endingPicture***. The more intermediate pictures, the smoother the morphing.

In this assignment, we will create a sequence of images that can be shown as a movie. The sequence of images will be a "morphing" of one image into another.

Part A:

The first part of the assignment will be to create a sequence of morphed pictures using the sequence **startingPicture, p1, p2, p3, endingPicture**. You can pick the start and end picture, but the pictures **p1, p2 and p3** are made up pixel by pixel, using an equation. This way they are calculated to be somewhere in between **startingPicture** and **endingPicture**. For each pixel of the intermediate pictures, we compute its red, green and blue values individually using the following formula for **each** colour:

*colourValue = startValue + ((endValue – startValue)/4) * k*

where *k* is the k^{th} intermediate picture.

So, for example, to compute the red value of a pixel in picture 2, the value of k would be 2. If the starting red value of that pixel was 100, and the end value was 200, the formula would be:

*redValue of pixel = 100 + ((200 – 100)/4) * 2*

This formula must be used three times on **every** pixel in order to get the red, green and blue values for the pixel. This idea can be extended for any number of intermediate pictures. If we wanted 7 intermediate pictures instead of 3, then k would range from 1 to 7, and we would have divided by 8 in the formula. In general, the formula for each colour is:

*colourValue = startValue + ((endValue – startValue)/(numStages + 1)) * k*

where *numStages* is the total number of intermediate pictures in the morphing.

For Part A of the assignment, you will write a method in the Picture class that forms an intermediate picture in the morning sequence, as specified below. This method should be called once for each intermediate picture.

Functional Specifications for Part A

1. You are to write a method morphStage() for inclusion in the Picture class. It will have the header

public void morphStage(Picture startPicture, Picture endPicture, int numStages, int k)

where the parameters are:

- **startPicture** : the starting picture
- **endPicture** : the ending picture
- **numStages** : the total number of intermediate stages to be used in the morphing process
- **k :** the current stage in the morphing process (will be a number between 1 and numStages inclusive)

morphStage() will be invoked on a Picture object that is the "target" picture, that is, the picture that represents the kth stage in morphing from the starting picture to the ending picture. The method will form the picture for the current stage (ie intermediate stage k) be setting the red, green, and blue values for the pixels in **this** picture according to the above formula, pixel by pixel.

The method may assume that the starting and ending pictures are the same size. (The intermediate pictures will of course be the same size as the starting and ending pictures.)

2. You will write a class TestMorphing (in *TestMorphing.java*) that consists primarily of a main method that tests morphStage() for correctness. It must contain the following tests:

i. Get two pictures using FileChooser. If they are of the same size, form 3 intermediate morphed pictures and display them, along with the starting and ending pictures. (You should end up with a total of 5 pictures on the screen from this test: the starting picture, the 3 intermediate morphed pictures, and the ending picture. You should be able to move them around on the screen using the mouse so that all can be displayed.) If the two pictures are not of the same size, print an error message and do not proceed with the morphing.

ii. Get two pictures of the same size different from the ones in the first test, and perform a task similar to the first one, except that this time you will generate 9 intermediate morphed pictures. This test should result in a total of 11 pictures on the screen. Again, check that the pictures are of the same size first.

You can select two of the images from the following sets of pictures that are the same size: {flower1.jpg, flower2.jpg}, {whiteFlower.jpg, yellowFlowers.jpg, rose.jpg}, {swan.jpg, twoSwans.jpg}, or you can use your own pictures that are the same size.

Part B:

Movies are essentially sequences of pictures that are typically displayed at 24 “frames” per second, each frame being a single picture. To create the effect of morphing as seen in videos and commercials, we would need to produce many more images and then play them in rapid succession, thus creating a video in which one image is morphed into the other.

In Part B of this assignment, we will create two short “movies” of our morphed images:

- One that morphs from the starting picture to the ending picture
- Another one that morphs from the starting picture to the ending picture and then back again.

You will make use of the FrameSequencer and MoviePlayer classes provided with our textbook in order to create and display a movie. The FrameSequencer class has methods that will take a sequence of pictures and store them in a directory; the pictures can then be displayed in rapid succession, thereby creating a “movie”. The MoviePlayer class has methods to play back a sequence of pictures already created (e.g. by FrameSequencer).

Here is sample code to create a FrameSequencer object that will store the movie frames in the directory referenced by a String variable directoryName, created from an array of pictures referenced by a Picture[] variable pictureSequence:

```
FrameSequencer frameSequencer = new FrameSequencer(directoryName);
for (int i = 0; i < pictureSequence.length; i++)
{
   frameSequencer.addFrame(pictureSequence[i]);
}
```

Once a movie has been made with FrameSequencer, we can use the MoviePlayer class to show the movie.

The MoviePlayer class

A MoviePlayer object plays an existing movie. Here are the important details about this class:

- The constructor for a MoviePlayer object takes as a parameter a String that is the name of a directory containing the images to be displayed in the movie
- The playMovie() method opens a window in which it then displays the movie. If the playMovie() method is called without a parameter, it plays the movie at high speed. It can also accept an int parameter that specifies the number of frames to be displayed per second.

So to play a movie, we create a MoviePlayer object and invoke the playMovie() method on it.

Functional Specifications for Part B

1. You will create and play your movies in a Java program with the class name MorphingMovie that will be saved in a file named *MorphingMovie.java*. This class will consist primarily of a main method whose algorithm is given below.
2. The algorithm for the main method is:

- Get the starting picture and ending picture for the morphing, using FileChooser to get the file names. If the starting picture and ending picture are not the same size, print an error message and do not proceed with the rest of the program. (Suggestion: use the showInformation() method of the SimpleOutput class (see Lab 7) for your error message.)
- Input the number of intermediate stages for the morphing from the user (using the SimpleInput class).
- Input the name of the directory in which to store the movie. (Note that the directory name that the user enters should end with a slash, for example Z:/*Movie1/*)
- Create an array of Picture objects that will hold the sequence of pictures for the morphing (Note: its length should be the number of intermediate stages + 2)
- Store the starting and ending pictures in the array.
- Create intermediate pictures in the array, using a loop and the morphStage() method from Part A.
- Create the first morphing movie (which morphs from the starting picture to the ending picture) from the array of pictures, using a FrameSequencer object.
- Play the movie using a MoviePlayer object and playMovie().
- Now create the second morphing movie (which morphs from the starting picture to the ending picture and back again to the starting picture). You can do this in either of two ways:
    i. Add the "backwards" morphing sequence to the same directory (in which case you no longer will be able to play your first movie)
    ii. Use a new directory for all the frames of the second movie. You would then input the directory name for this movie from the user.
- Play the second movie.

3. The first movie that you make should contain at least 20 frames.

4. For the second movie, to show the ending picture morphing back to the starting picture, do NOT do a reverse morphing from the ending picture to the starting picture using morphStage(). Why not? You already have that sequence of pictures in the array; you just want to play them backwards!

5. Caution: If you use your own pictures, you may run out of memory space if they are too large. Try your program on small pictures first, such as those suggested for Part A.

Non-functional Specifications:

1. Include brief comments in your code that explain what each method is doing.

2. Assignments are to be done individually and must be your own work. Software will be used to detect cheating.

3. Use Java conventions and good Java programming techniques, for example:
    i. Meaningful variable names
    ii. Conventions for naming variables and constants
    iii. Named constants instead of "magic numbers"
    iv. Appropriate loop structures
    v. Readability: indentation, white space, consistency

What to Hand In:

1. *Picture.java* (with **your** morphStage() method added), *TestMorphing.java*, *MorphingMovie.java*
2. The .jpg files for the starting picture and the ending picture for your morphing, if you wish the marker to use your own pictures

What You Will Be Marked On:

1. Functional specifications:
    - ✓ Are the required methods written according to specifications?
    - ✓ Do they work as specified? Pay attention to details!
    - ✓ Are they called as specified?
2. Non-functional specifications: as described above.