TOPIC 9

MODIFYING PIXELS IN A MATRIX: COPYING, CROPPING

Notes adapted from Introduction to Computing and Programming with Java: A Multimedia Approach by M. Guzdial and B. Ericson, and instructor materials prepared by B. Ericson.

Outline

- Learn about picture manipulations using
  - More complex for loops
  - Multiple for loop variables

Copying and Transforming Pictures

3

- So far, we have taken an image and somehow changed the image itself
  - Changing colour values, mirroring
- We can do things differently: start with a ***source image*** and set pixels in a ***target image***
  - We can ***copy*** an image
  - We can also ***transform*** it in the process of copying
    - Cropping, scaling, rotating, etc.
  - Actually, we don't copy the pixels, but rather make the pixels in the target image the *same color* as the ones in the source image

Copying Pictures

4

- We need to keep track of
  - the source picture's x and y
  - the target picture's x and y
- We can use a ***blank picture*** as the target picture if we wish

Source picture

(0,0)	(1,0)
(0,1)	(1,1)
(0,2)	(1,2)

Target picture

(0,0)	(1,0)		
(0,1)	(1,1)		
(0,2)	(1,2)		

Copying Pictures

- Several blank pictures are already available:
  - 640x480.jpg (size in pixels)
  - 7inX95in.jpg (size in inches)
- Or we can make one of any size ourselves
  - Example:
    Picture targetPic = new Picture(200,200);

Copy Picture Method

- We will write a method to copy a picture:
  - The source picture will be passed as a parameter
  - The method will be invoked on the target picture
- The method header will be
  public void copyPicture (Picture sourcePicture)
- Copy algorithm:
  - Loop through the pixels of the source picture
    - Get the source and target pixels
    - Set the color of the target pixel to the color of the source pixel

For Statement Revisited

- We can do **multiple tasks** in a for statement!

```
for (start; check; step)
 {
   body of loop
 }
```

  - Initialize several loop variables in the **start area**
    - Separated by commas
  - Change several loop variables in the **change area**
  - But there is still only one test in the **check area**

Copy Picture Algorithm to Code

- Loop through the pixels of the source picture:

```
// loop through the columns
for (int sourceX = 0, targetX = 0;
     sourceX < sourcePicture.getWidth();
     sourceX++, targetX++)
{
  // loop through the rows
  for (int sourceY = 0, targetY = 0;
       sourceY < sourcePicture.getHeight();
       sourceY++, targetY++)
      {
```

- (to be continued)

Copy Picture Algorithm to Code (cont'd)

- Get the source and target pixels:

```
sourcePixel =
    sourcePicture.getPixel(sourceX,sourceY);

targetPixel = this.getPixel(targetX,targetY);
```

- Set the color of the target pixel to the color of the source pixel

```
targetPixel.setColor(sourcePixel.getColor());
```

Copy Picture Method


```
public void copyPicture (Picture sourcePicture)
{
  Pixel sourcePixel = null;
  Pixel targetPixel = null;

  // loop through the columns
  for (int sourceX = 0, targetX = 0;
       sourceX < sourcePicture.getWidth();
       sourceX++, targetX++)
  {
```

Copy Picture Method (continued)


```
  // loop through the rows
   for (int sourceY = 0, targetY = 0;
        sourceY < sourcePicture.getHeight();
        sourceY++, targetY++)
   {

     // set the target pixel color to the source pixel color
     sourcePixel = sourcePicture.getPixel(sourceX,sourceY);
     targetPixel = this.getPixel(targetX,targetY);
     targetPixel.setColor(sourcePixel.getColor());
   }
  }
}
```

Setting a Media Path

- We have been using FileChooser.pickAFile() to choose our image files
  - we may have to go through several directories to get to the right one
  - we may like to go straight to a particular image
- We can save the name of the directory that has our images in it, using FileChooser.setMediaPath(directory);
  - directory is a string that is the file path to the directory containing the image(s)
  - Example: FileChooser.setMediaPath("Z:/mediaSources/");

Setting a Media Path

- If we now use FileChooser.pickAFile() to choose our file, it will start at the saved directory name
- We can now get a particular file using FileChooser.getMediaPath(filename);
  - This generates a complete file path using the saved directory name concatenated with the filename
  - Example:

```
Picture pictureObj = new Picture(
  FileChooser.getMediaPath("caterpillar.jpg"));
```

Trying the copyPicture Method

- Example: copy an image to a larger blank canvas
  - We can use 7inX95in.jpg as the target image
  - We create a Picture object for the target image:

```
Picture targetPic = new Picture
  (FileChooser.getMediaPath("7inX95in.jpg"));
```

  - This is the Picture object on which the method will be invoked
  - We create a Picture object for the source image:

```
Picture sourcePic = new Picture
                    (FileChooser.pickAFile());
```

Trying the copyPicture Method

15

- Invoke the copy method on the target picture:

```
targetPic.copyPicture(sourcePic);
```

- Repaint the picture

```
targetPic.repaint();
```

- The result is shown on the next slide
  - Using the image in KatieFancy.jpg
  - Why was Katie's picture copied to the top left corner of the blank picture?

Trying the copyPicture Method

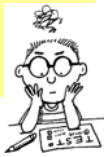
16

Copy Picture to Position Algorithm

- We will now create a more general method that copies from a source picture passed as a parameter
  - And also passes the position in the target picture at which the copy will have its upper left corner
- The method header will be
  public void copyPictureTo(Picture sourcePicture,
  int xStart, int yStart)
- We can model this method on the copyPicture method
  - What do we need to add / change?

Copy Picture to Position Method


```
public void copyPictureTo(Picture sourcePicture,
                          int xStart, int yStart)
{
  Pixel sourcePixel = null;
  Pixel targetPixel = null;

  //loop through the columns
  for (int sourceX = 0, targetX = xStart;
       sourceX < sourcePicture.getWidth();
       sourceX++, targetX++)
  {
```



Copy Picture to Position Method (cont'd)


```
// loop through the rows
   for (int sourceY = 0, targetY = yStart;
        sourceY < sourcePicture.getHeight();
        sourceY++, targetY++)
   {
    // set the target pixel color to the source pixel color
    sourcePixel = sourcePicture.getPixel(sourceX,sourceY);
    targetPixel = this.getPixel(targetX,targetY);
    targetPixel.setColor(sourcePixel.getColor());
    }
  }
}
```

Copy Picture to Position

- What would happen if the source picture did not fit into the target picture?
- How could we fix this?
  - We can stop copying if the target x value goes past the width of the target picture or the target y value goes past the height
  - Change the for loop test for the x loop:
    (sourceX < sourcePicture.getWidth()) &&
    (targetX < this.getWidth())
  - What would be the new test for the y loop?

Cropping a Picture

- We can copy just part of a picture to a new picture instead of the entire picture
  - Just change the starting and ending x and y values of the source picture to the desired values
  - How do you find the right values? You can use pictureObj.explore() to find the x and y values inside the picture

Example of Cropping a Picture

- Coordinates of the face of the girl in KatieFancy.jpg?
  - Upper left corner at (70, 3)
  - Bottom right corner at (135, 80)

Example of Cropping a Picture

- Here is the result of copying just the part of the source picture with
  - x going from 70 to 135
  - y going from 3 to 80
- And copying to position (100,100) in the target picture

An Even More General Copy Algorithm

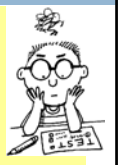
- We will now create an even more general method that copies pixels from a source picture to the current picture object, with parameters
  - The source picture (as before)
  - A start x, y and end x, y for the source picture
    - If the start x, y and end x, y cover the entire picture, then the whole picture will be copied
    - If the start x, y and end x, y are only part of the picture, then ***cropping*** will occur
  - The position in the target picture to copy to (as before)

An Even More General Copy Algorithm

- Loop through the x values between the start x and end x
- Loop through the y values between the start y and end y
- Get the pixel from the source picture for the current x and y values
  - Get the pixel from the target picture for the targetStartX + x and targetStartY + y values
  - Set the color in the target pixel to the color in the source pixel

Copy Picture to Position with Cropping

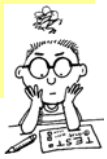

```
public void copyPictureTo(Picture sourcePicture, int startX, int startY,
                          int endX, int endY, int targetStartX, int targetStartY)
 {
   Pixel sourcePixel = null;
   Pixel targetPixel = null;

   // loop through the x values
   for (int x = startX, tx = targetStartX;
        x < endX && x < sourcePicture.getWidth() && tx < this.getWidth();
        x++, tx++)
   {
     // loop through the y values
     for (int y = startY, ty = targetStartY;
          y < endY && y < sourcePicture.getHeight() && ty < this.getHeight();
          y++, ty++)
     {
```



Copy Picture to Position with Cropping

(continued)

```
// copy the source color to the target color
        sourcePixel = sourcePicture.getPixel(x,y);
        targetPixel = this.getPixel(tx,ty);
        targetPixel.setColor(sourcePixel.getColor());
      }
    }
  }
```



Method Overloading

- Notice that we have two methods named copyPictureTo() in Picture.java, and Java did not complain about this!
- Java allows you to have methods with the same name as long as they take different parameters
  - Either a different number of parameters or different types of parameters
- This is called **method overloading** in Java

Method Overloading

- Note that the return type of a method does *not* count towards being able to overload a method
- You cannot have two methods with the same names and parameters, with the only difference being the return type

New Concepts in this Section

- For loops with multiple loop variables
- Method overloading