Reflections and Rotations

Have you ever had to turn your digital camera to take just the right shot? When you download the pictures from your camera, you probably had to rotate your picture to see the object(s) with the right orientation. Rotating and reflecting pictures are sometimes useful, but can also create fun effects.

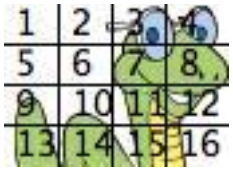
We will look at reflections first, then rotations. Suppose we want to get a mirror image of our picture. That means we would like something like this:

Original Picture:

Reflected Picture:

The reflected picture is what you would see if you looked at the original picture in a mirror.

We need to figure out how to do this through programming. The reflected picture has all of the same pixel values, but the columns are reversed. So, the color from top left pixel of the original image gets copied to the top right pixel in the new image. In our example, suppose our picture has only 16 pixels:



When we reverse this, we can see what happens to the pixels:

The pixel in the top left corner (the pixel numbered “1”) gets copied to the upper right corner in the new picture. You can see how the pixels in the top row follow this pattern. The pixels in the second row do the same thing. It is similar for pixels in the third and fourth rows.

To make this work in programming, we could use the following algorithm. (You will write the code for this in the next mini-lab.)

Vertical Reflection Algorithm:

- Make a new, empty picture the same size as the picture to be reflected
- For each row *y* in the original picture:
    - Assign the first *x*-value of the new picture (*newX*) to be the *x*-value of the last pixel in the row (*i.e.*, it should be assigned the width -1)
    - For each x-value of the original picture:
        - Get the pixel at location (*x*, *y*) of the original picture.
        - Set the color of the pixel at location (*newX*, *y*) of the new picture to be the value of the color at pixel (*x*, *y*) from the original picture.
        - Decrease the value of *newX* by 1
- Return the new picture

In the next activity, you will write the code to implement this algorithm.

We could also reflect a picture so that it is upside-down, as if you put a mirror at the bottom of the picture instead of the side of the picture. So, our reflected snake picture would look like this:

Original Picture:

Reflected Picture:

Exercise: Write an algorithm for reflecting the picture horizontally (*i.e.*, across the bottom of the picture).

We can use reflection to create some fun effects. Consider the following function that reflects the right half of a picture onto the left half:

```
# Function to reflect the right half onto the left half
def reflectRightOntoLeft(picture):
  newPic = picture.copy()
  for y in range(newPic.height):
    newX = newPic.width -1
    for x in range(newPic.width//2):
          rvalue, gvalue, bvalue = newPic.getpixel((x,y))
          newPic.putpixel((newX,y),(rvalue, gvalue, bvalue))
          newX = newX - 1

  return newPic
```

When we pass the snake picture into this function, we obtain the following two-headed snake:

The idea behind this function is to use a vertical line down the middle of the picture as the mirror.

Exercise: Modify this function to reflect the left half over the right half.

Exercise: Modify this function to use a horizontal line through the middle of the picture as the mirror point.

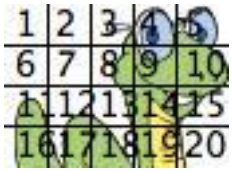
Exercise: Modify this function to use a line of your choosing as the mirror point.

Now let's turn our attention to rotating pictures. Let's see what happens to the pixels in our snake picture when we rotate the picture to the right.

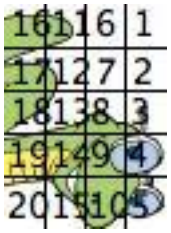
Since the snake picture is not square, we will slightly modify our pixel numbering to reflect this. (It didn't matter so much in the reflection examples, but it will be more

important to see what happens to the picture dimensions when rotating the picture.)

Original picture:



Picture rotated right:



We first notice that the width of the rotated picture is the same as the height of the original picture, and the height of the rotated picture is the same as the width of the original picture. We can also see that the top row of pixels in the original pictures becomes the last column of pixels in the rotated picture. The first column of pixels in the original picture becomes the top row of pixels in the rotated picture. When we implement this in code, we will use separate x- and y- values for each picture, rather than trying to find a (somewhat complicated) mathematical formula to relate the two sets of coordinates. Here is the actual code:

```
# Function to rotate image 90 degrees to the right
def rotateRight(picture):
  # Create a new image for the result
  # Width of new image is height of original
  # Height of new image is width of original
  newPic = Image.new('RGB', (picture.height, picture.width))
  # Create variable to keep track of x-coord in new picture
  newX = newPic.width - 1
  # Loop through original picture
  for y in range(picture.height):
    # Set up variable to keep track of y-coord in new picture
    newY = 0
    for x in range(picture.width):
      rvalue, gvalue, bvalue = picture.getpixel((x, y))
      newPic.putpixel((newX, newY), (rvalue, gvalue, bvalue))
      newY = newY + 1
    newX = newX - 1

  return newPic
```

Exercise: Test this code with several different pictures

Exercise: Modify this code to rotate a picture to the left.

Activity: Reflections and Rotations