ACTIVITY 4

Hiding and Revealing a Text Message

Information can be represented in various ways, as seen throughout this lab with images being stored as 2D arrays of pixels. All information stored on a computer is stored as sequences of bits, and how those bits are interpreted can completely change their meaning. Consider the following eight-bit sequence—0100 1101. It's impossible to know what it represents without context. It could be the integer 77, the character 'N', the amount of red in a pixel, or any number of things.

The biggest consideration when determining how to represent information is knowing how many items need to be represented, since this will ultimately determine the number of bits that will be used. For example, early monitors were not capable of displaying the variance in color that current monitors can display, so colors were represented using far fewer bits. As monitors have improved and become capable of displaying more colors, the number of bits used to store color has increased so that all possible colors that can be displayed have a unique representation.

1. How many items can be represented with the following numbers of bits:

2 bits? 4 4 bits? 16 8 bits? 256

This activity involves hiding and then revealing a text message in a picture. The ideas are the same as those involved in hiding and revealing a picture, but the text message must be split up and reassembled more carefully.

Recall that to hide a picture, the leftmost two bits in each color for each pixel from the secret image are stored in the source image as the rightmost two bits in each matching color/pixel. The picture can be revealed by taking those rightmost bits and making them the leftmost bits again.

In this activity, you're going to store messages consisting of uppercase letters and spaces in a picture. To do this, the 26 letters will be represented by numbers (1–26 where A = 1, etc.), a space will be represented using the value 27, and a 0 will represent the end of the string. In this way, messages will be able to be coded using 28 distinct integer values.

By encoding a message in this way, five bits will be needed to hold each letter or space. Recall that five bits can hold the decimal values ranging from 0 to 31 so there are even a few extra bits in case punctuation is desired. The desire is for the text to be truly hidden in the picture, so only the rightmost two bits in each color value of each pixel are used to store the coded message. Since there are three color components per pixel, each coded character will be split into three pairs of two bits where the leftmost bit is always 0. Again, more characters such as punctuation can be added

2. You're now ready to create the method that hides the message. Complete the following method in the `Steganography` class.

```
/**
 * Hide a string (must be only capital letters and spaces) in a
 *  picture.
 *  The string always starts in the upper left corner.
 *  @param source picture to hide string in
 * @param s string to hide
 * @return picture with hidden string
 */
public static void hideText(Picture source, String s)
```

3. Now, complete the following method to be able to return the secret message hidden in the image.

```
/**
 * Returns a string hidden in the picture
 * @param source picture with hidden string
 * @return revealed string
 */
public static String revealText(Picture source)
```

4. In the `main` method, add code to test hiding and revealing a message.

Check Your Understanding

5. Given the representation scheme used, would it be possible to represent both uppercase and lowercase letters? What about digits? Exactly how many characters can be represented using the six-bit encoding scheme?

Yes, using 6 bits total, you could add a set of lowercase letters as well as uppercase. The digits can also fit within these bounds. The exact number of characters that can be represented is 63, including an end bit to signify the end of the message.

6. When storing the secret message, a special value was used to signify the end of the message. Discuss with a partner what would happen if there was no way to signal the end of a message. Which methods would change? Describe how the behavior would be different.

There would be no way to end the message, and it would translate the whole picture or row instead of just the message. The method for revealing the text would change.

```
//CODE BY AKSHAT GARG
public static void main(String[] args)
{
    Picture arch = new Picture ("arch.jpg");
    Picture arch2 = new Picture ("arch.jpg");
    Picture koala = new Picture ("koala.jpg");
    Picture robot1 = new Picture ("robot.jpg");
    ArrayList<Point> pointList = findDifferences (arch,arch2);
    System.out.println("Point List after comparing two identical pictures" + " has a size of " + pointList.size());
    pointList = findDifferences(arch, koala);
    System.out.println("Point List after comparing two different-sized pictures" + " has a size of " + pointList.size());
    arch2 = hidePicture(arch2, robot1, 65, 102);
    pointList = findDifferences (arch, arch2);

    System.out.println("Point list after hiding a picture has a size of "+ pointList.size());
    //arch.explore();
    //arch2.explore();

    hideText(arch,"HELLO WORLD");
    System.out.println(revealText(arch));
}
```

```
//CODE BY AKSHAT GARG
public static void hideText(Picture source, String s)
{
    ArrayList<Integer> message  = encodeString(s);
    int[] code = new int[message.size()*3];
    Pixel[][] pixels = source.getPixels2D();
    for (int i = 0; i < message.size();i++)
    {
        pixels[i][0].setRed(getBitPairs(message.get(i))[2]+pixels[i][0].getRed()/4*4);
        pixels[i][0].setGreen(getBitPairs(message.get(i))[1]+pixels[i][0].getGreen()/4*4);
        pixels[i][0].setBlue(getBitPairs(message.get(i))[0]+pixels[i][0].getBlue()/4*4);
    }
}
public static String revealText(Picture source)
{
    Pixel[][] pixels = source.getPixels2D();
    ArrayList<Integer> codes = new ArrayList<>();
    int j = 0;
    while ((((pixels[j][0].getRed()%4)*16+(pixels[j][0].getGreen()%4)*4+pixels[j][0].getBlue()%4)!=0)&&(j<15))
    {
        codes.add(((pixels[j][0].getRed()%4)*16+(pixels[j][0].getGreen()%4)*4+pixels[j][0].getBlue()%4));
        System.out.println(pixels[j][0].getRed()%4 + " " + pixels[j][0].getGreen()%4 + " " + pixels[j][0].getBlue()%4);
        j++;
    }
    return decodeString(codes);
}
```