Name: _____

Date: _____

ACTIVITY 3

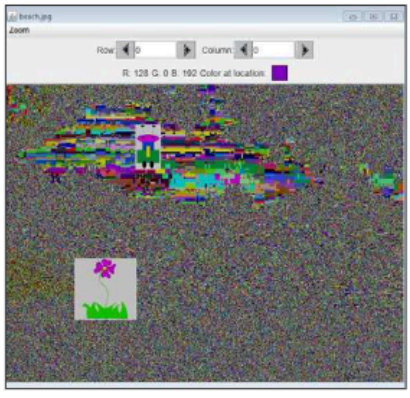
Identifying a Hidden Picture

1. In the `Steganography` class, after modifying `canHide` to allow the secret image to be smaller than the source image, write a second version of the `hidePicture` method that allows the user to place the hidden picture anywhere on the modified picture. This means adding two parameters for `startRow` and `startColumn` that represent the row and column of the upper left corner where the hidden picture will be placed.

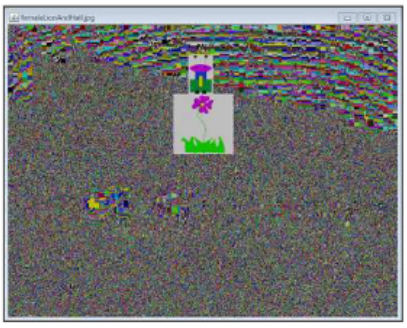
Sample code:

```
Picture beach = new Picture("beach.jpg");
Picture robot = new Picture("robot.jpg");
Picture flower1 = new Picture("flower1.jpg");
beach.explore();

// these lines hide 2 pictures
Picture hidden1 = hidePicture(beach, robot, 65, 208);
Picture hidden2 = hidePicture(hidden1, flower1, 280, 110);
hidden2.explore();

Picture unhidden = revealPicture(hidden2);
unhidden.explore();
```

Results:

`hidden2` ***With Pictures Hidden:***	`unhidden` ***After Calling*** `revealPicture`***:***
	

Results:

Bounding Rectangle	*Unhidden Pictures*
	

Check Your Understanding:

5. This activity is focused on being able to hide the secret image in a random location within the source image. One aspect that was not mentioned was how to determine an appropriate random location to hide the secret image. Assuming source and secret are both known (and therefore the height and width of each could be determined through method calls), how would you generate random `row` and `column` values to start hiding secret?

Complete the following expressions:

`int row` = (int)(Math.random*(source.length-secret.length))

`int column` = (int)(Math.random*(source[0].length-secret[0].length))

6. Are your coordinates guaranteed to fit secret within source? If not, modify the above expressions to ensure that there is room to fit the entire secret image within source.

Yes, they are guaranteed, as the top left coordinate is used for the start rows. This subtracts the length of secret to make sure it fits.

```
public static boolean isSame(Picture one, Picture two){

    Pixel[][] a = one.getPixels2D();
    Pixel[][] b = two.getPixels2D();
    for (int r = 0; r < a.length; r++)
    {
        for (int c = 0; c < a[0].length; c++) {
            if ((a[r][c].getRed()!=b[r][c].getRed()) || (a[r][c].getGreen()!=b[r][c].getGreen()) || (a[r][c].getBlue()!=b[r][c].getBlue())){
                return false;
            }
        }
    }
    return true;
}
```

```
public static ArrayList<Point> findDifferences(Picture one, Picture two){
    ArrayList<Point> diff = new ArrayList<>();
    Pixel[][] a = one.getPixels2D();
    Pixel[][] b = two.getPixels2D();
    if (canHide(one,two)|| true){
        System.out.println(a.length + " " + a[0].length);
        for (int r = 0; r < a.length; r++)
        {
            for (int c = 0; c < a[0].length; c++) {
                if ((a[r][c].getRed()!=b[r][c].getRed()) || (a[r][c].getGreen()!=b[r][c].getGreen()) || (a[r][c].getBlue()!=b[r][c].getBlue()))
                    diff.add(new Point(r,c));
                }
            }
        }
    }
    return diff;
}
public static Picture showDifferentArea(Picture source, ArrayList<Point> diff){

    Pixel[][] a = source.getPixels2D();

    for (int r = 0; r < diff.size(); r++)
    {
        for (int c = 0; c < a[0].length; c++) {
            Point z = diff.get(r);
            a[z.x][z.y].setRed(0);
            a[z.x][z.y].setGreen(0);
            a[z.x][z.y].setBlue(0);
        }
    }
    return source;
}
```