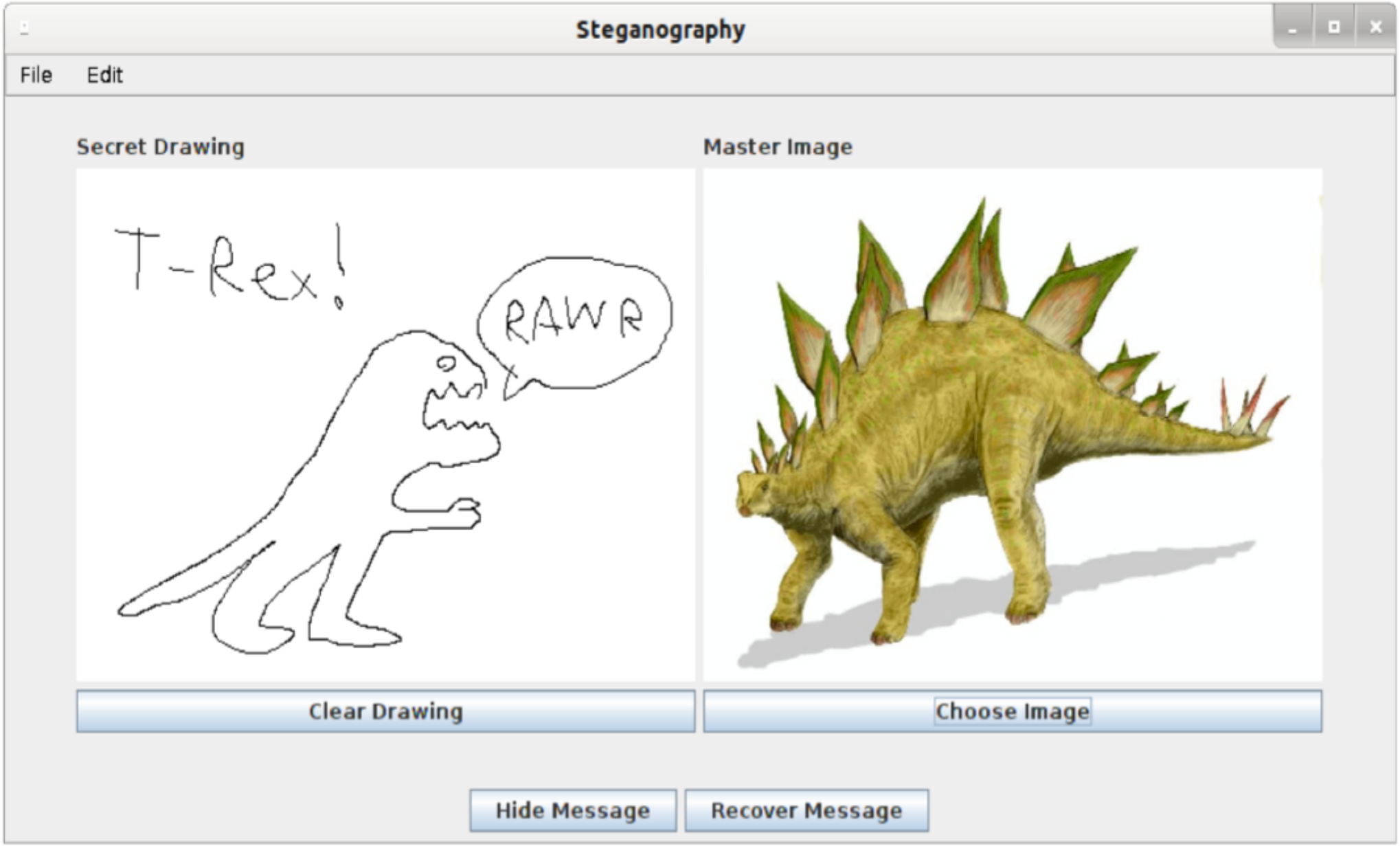


106A assignment
review #5

23 Feb 2014
7p-8p

Miles Seiver

Review session schedule

Topic	Date	Time	Location
assignment 5	today!	now!	here!
midterm 2	Sun 2 Mar	1p - 3p	Hewlett 200
assignment 6	Thu 6 Mar	5:30p - 6:30p	Hewlett 200
assignment 7	Sun 16 Mar	7p - 8p	Hewlett 200

ArrayList

Why `ArrayList`?

- There are situations where you don’t know how much data you will have

- Number of variables is fixed

ArrayList

- Can grow as big as you need
- Can check if something is contained in it with **`.contains()`**

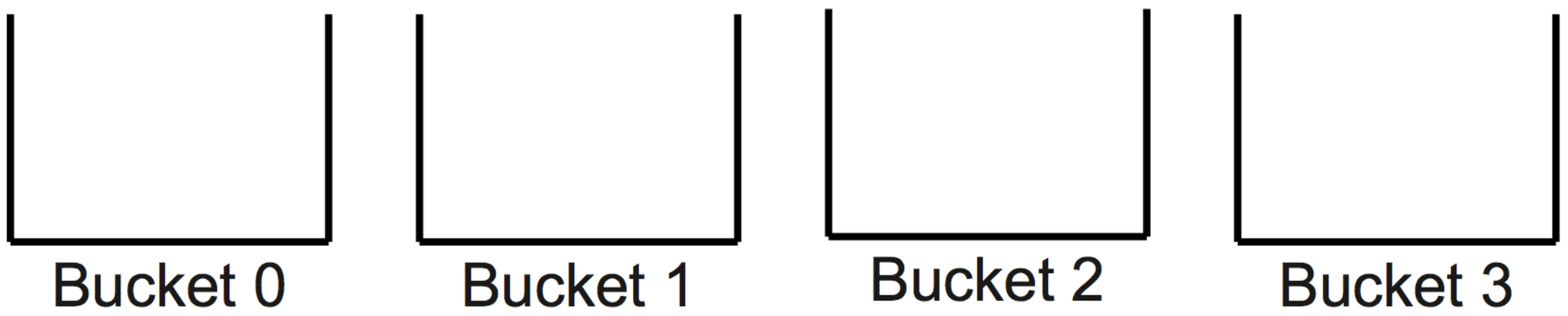
ArrayList

`boolean add(<T> element)` Adds a new element to the end of the **`ArrayList`**; the return value is always **`true`**.
`void add(int index, <T> element)` Inserts a new element into the **`ArrayList`** before the position specified by **`index`**.
`<T> remove(int index)` Removes the element at the specified position and returns that value.
`boolean remove(<T> element)` Removes the first instance of **`element`**, if it appears; returns **`true`** if a match is found.
`void clear()` Removes all elements from the **`ArrayList`**.
`int size()` Returns the number of elements in the **`ArrayList`**.
`<T> get(int index)` Returns the object at the specified index.
`<T> set(int index, <T> value)` Sets the element at the specified index to the new value and returns the old value.
`int indexOf(<T> value)` Returns the index of the first occurrence of the specified value, or −1 if it does not appear.
`boolean contains(<T> value)` Returns **`true`** if the **`ArrayList`** contains the specified value.
`boolean isEmpty()` Returns **`true`** if the **`ArrayList`** contains no elements.

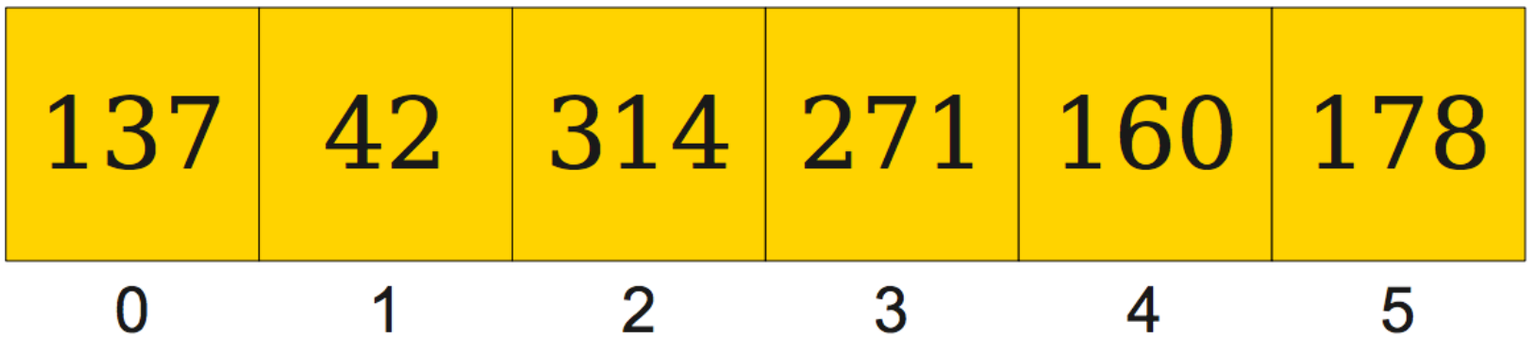
1-D arrays

Why Arrays?

- Arrays are excellent for representing a fixed-size list of **buckets**.
- We can store values in the appropriate bucket by looking up the bucket by index.



Arrays



- An array stores a **sequence** of multiple objects.
  - Can access objects by index using [].
- All stored objects have the same type.
  - You get to choose the type!
- Can store *any* type, even primitive types.
- Size is fixed; cannot grow once created.

Basic Array Operations

- To create a new array, specify the type of the array and the size in the call to **`new`**:

```
Type[] arr = new Type[size]
```

- To access an element of the array, use the square brackets to choose the index:

```
arr[index]
```

- To read the length of an array, you can read the **`length`** field:

```
arr.length
```

2-D arrays

```
Type[][] a = new Type[rows][cols];
```

Interpreting Multidimensional Arrays

- There are two main ways of intuiting a multidimensional array.
- **As a 2D Grid:**
  - Looking up ***arr*`[`*row*`]` `[`*col*`]` selects the element in the array at position (***row***, ***col***).
- **As an array of arrays:**
  - Looking up ***arr*`[`*row*`]` gives back a one-dimensional consisting of the columns in row ***row***.

Iterating through a 2-D array

```
Type[][] arr = /* … */

for (int row = 0; row < arr.length; row++) {
   for (int col = 0; col < arr[row].length; col++) {
      /* … access arr[row][col] … */
   }
}
```

```java
int[][] arr = new int[4][5];
for (int row = 0; row < arr.length; row++) {
   for (int col = 0; col < arr[row].length; col++) {
      arr[row][col] = row + col;
   }
}
```

	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	5
2	2	3	4	5	6
3	3	4	5	6	7

the assignment

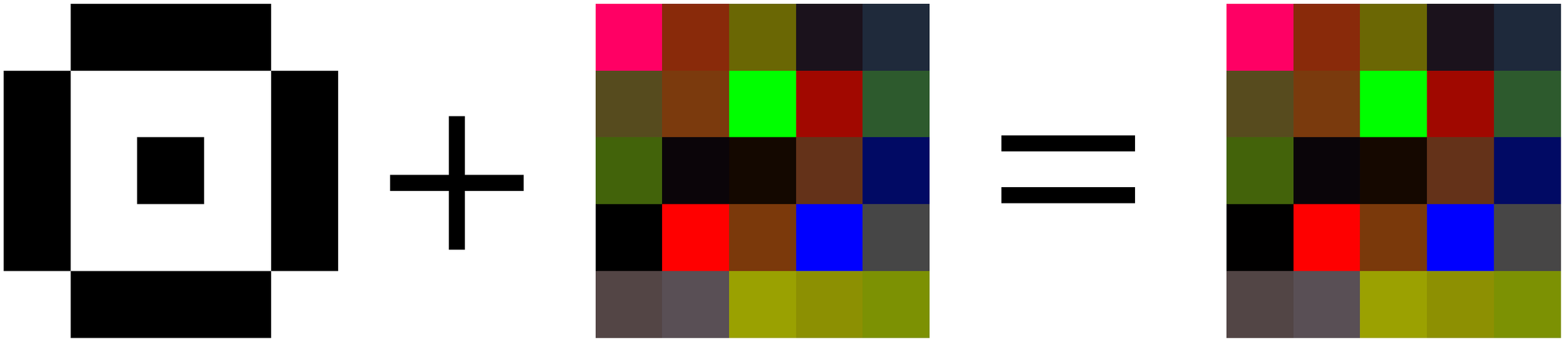
Array Algorithms

due Fri, 28 Feb
@ 3:15pm

Three parts:

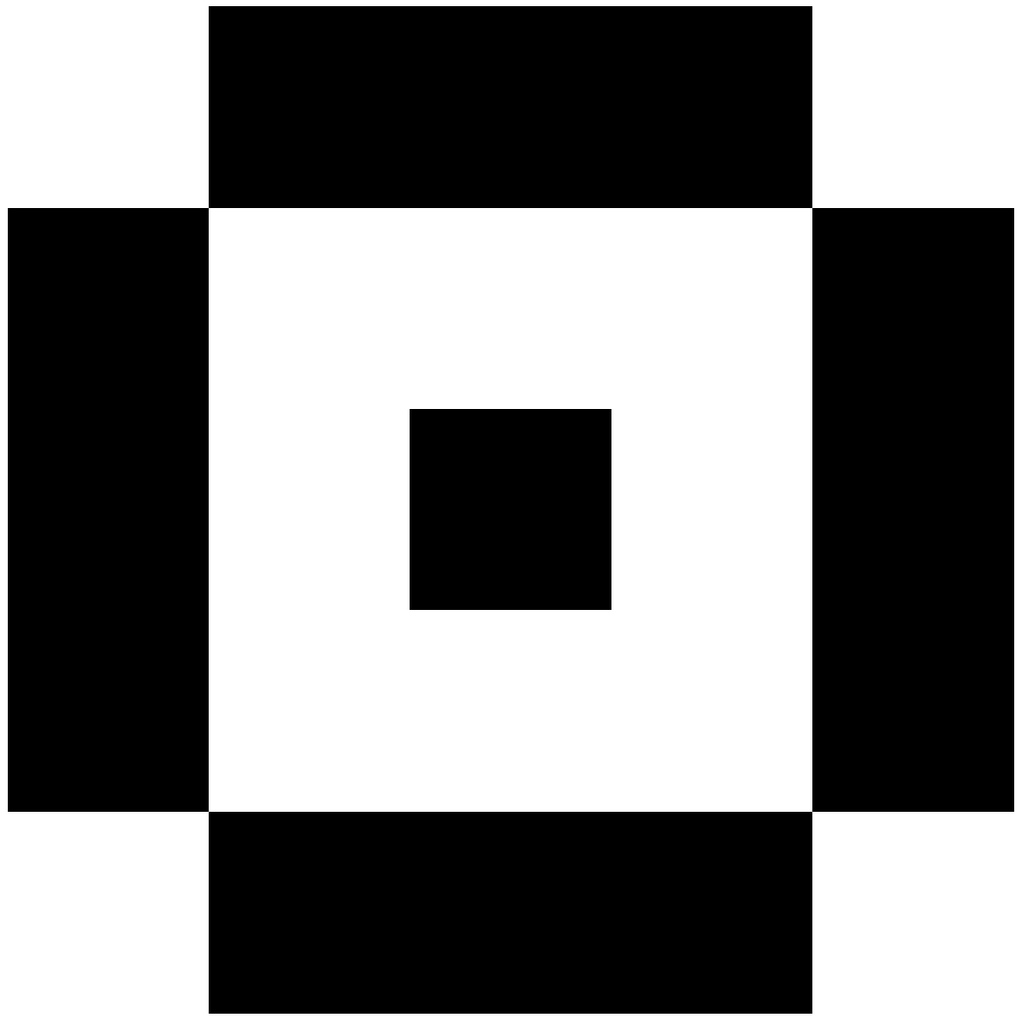
1. steganography
2. tone matrix
3. histogram equalization

steganography



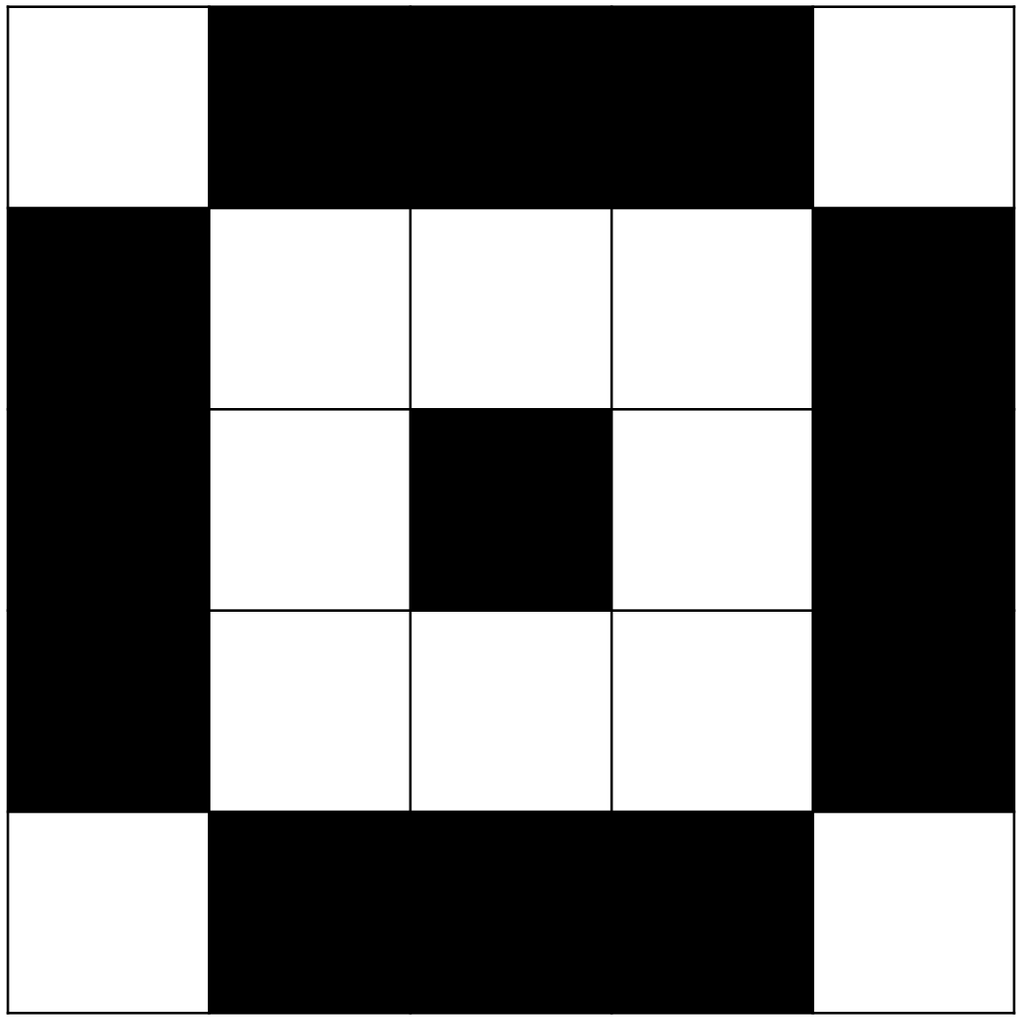
message source return

```
public static GImage hideMessage(boolean[][] message, GImage source)
```



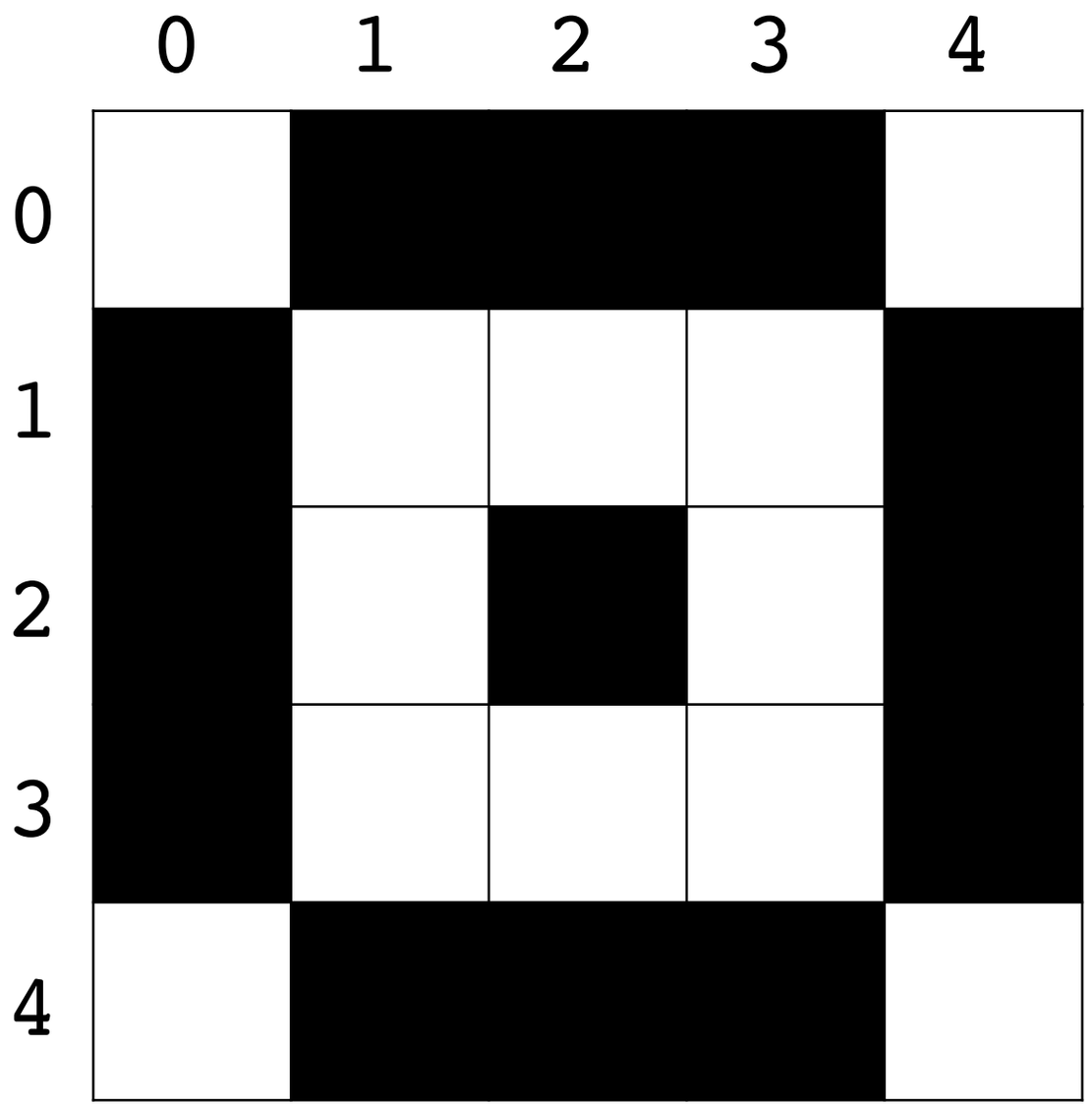
`message`

(“the secret”)



`message`

(“the secret”)

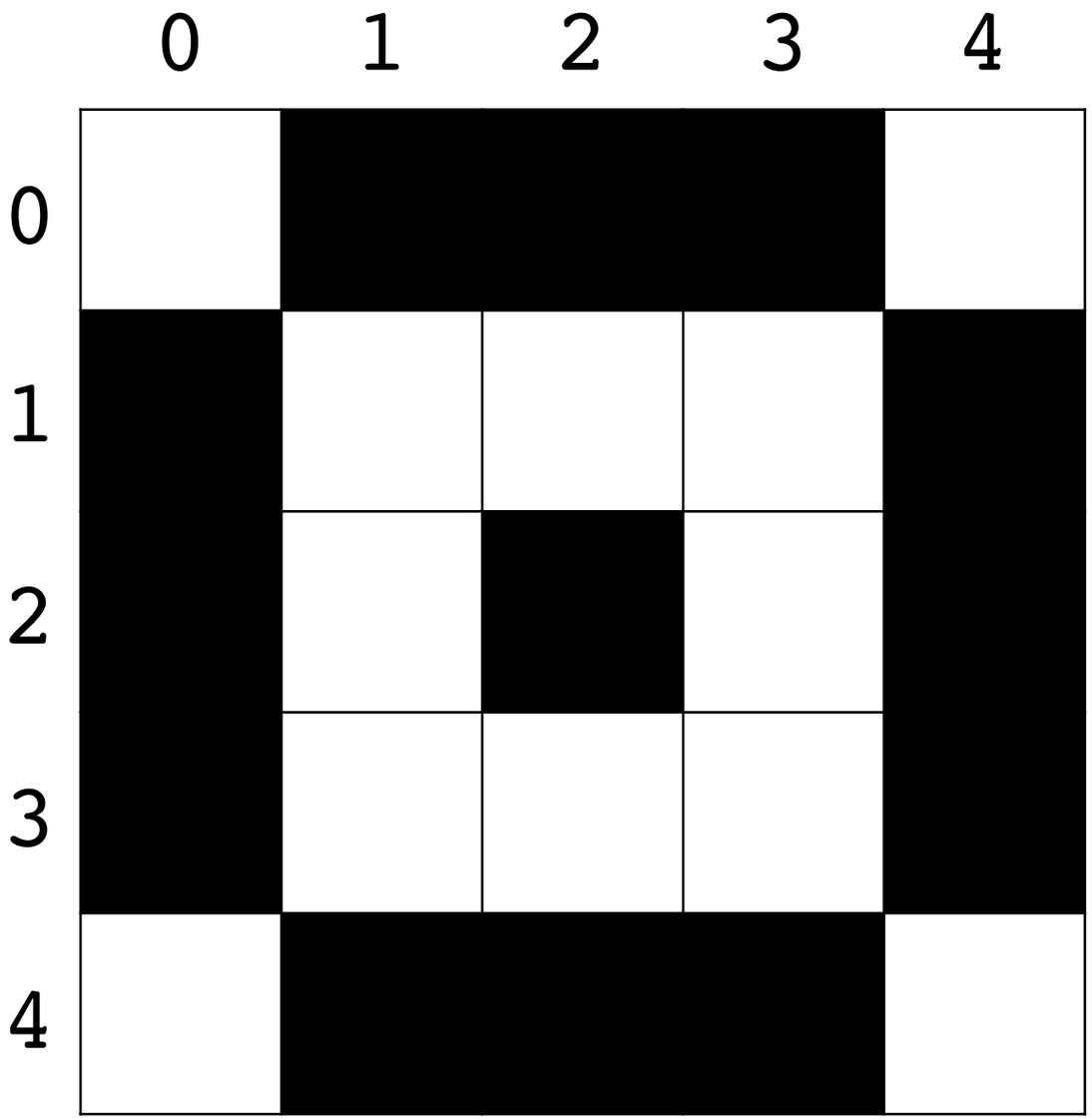


`message`

(“the secret”)

If the secret pixel is ***black***, it is represented as `true`, and you should make the red channel ***odd***.

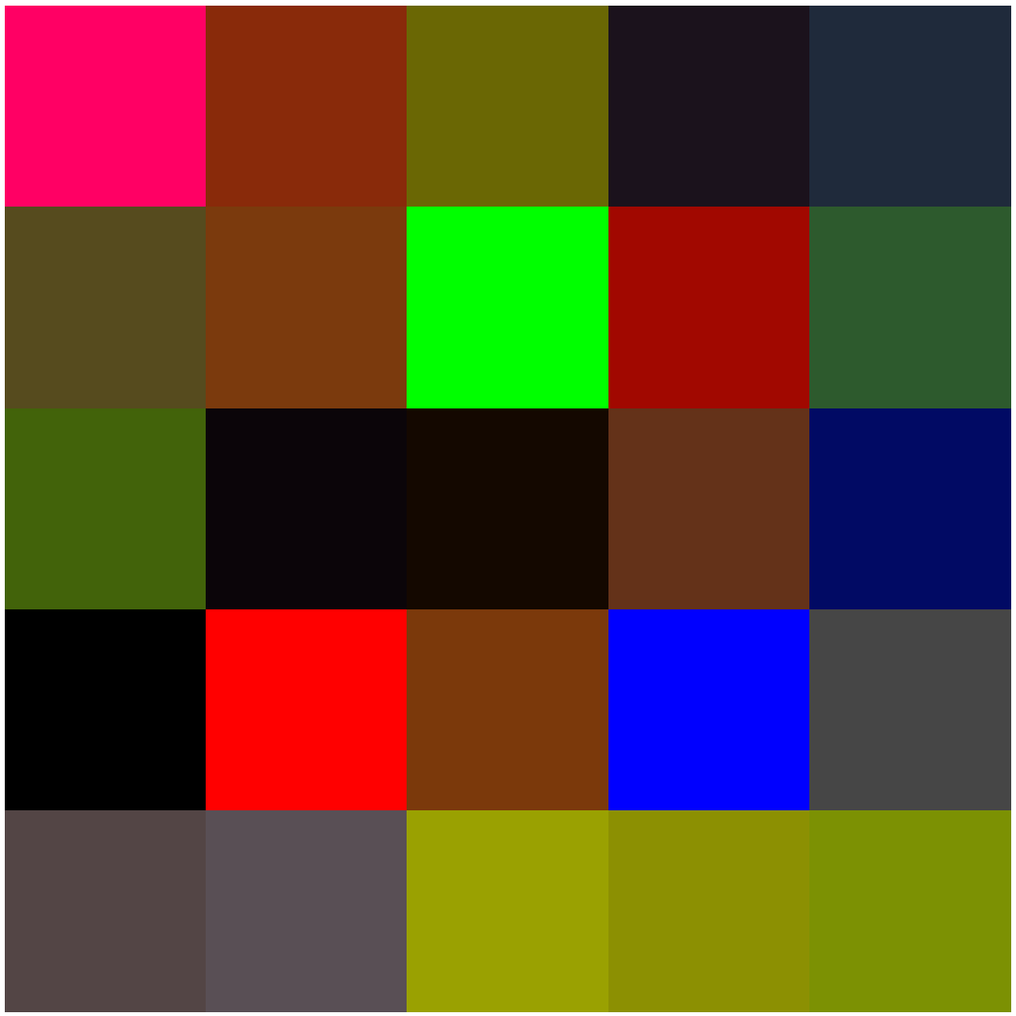
If the secret pixel is ***white***, it is represented as `false`, and you should make the red channel ***even***.



	0	1	2	3	4
0	FALSE	TRUE	TRUE	TRUE	FALSE
1	TRUE	FALSE	FALSE	FALSE	TRUE
2	TRUE	FALSE	TRUE	FALSE	TRUE
3	TRUE	FALSE	FALSE	FALSE	TRUE
4	FALSE	TRUE	TRUE	TRUE	FALSE

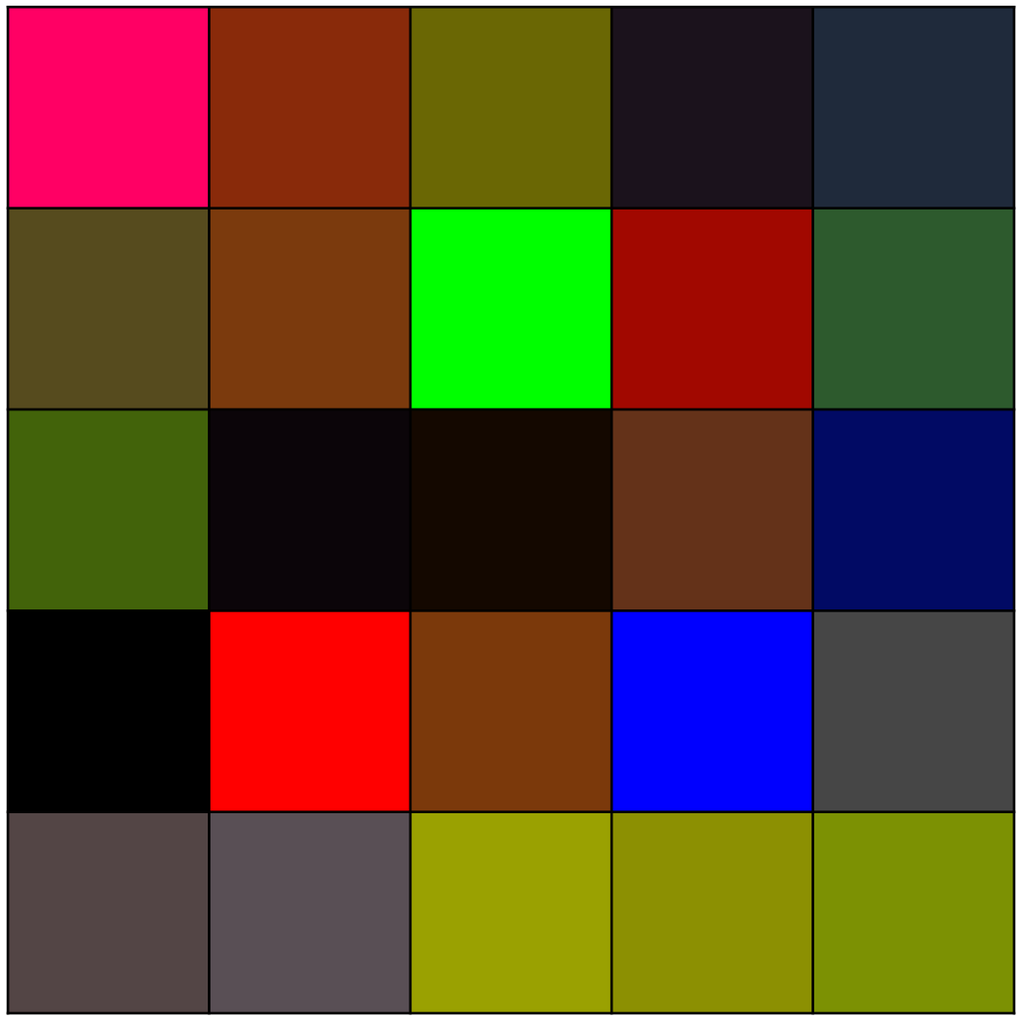
`message`
("the secret")

`boolean[][] message`



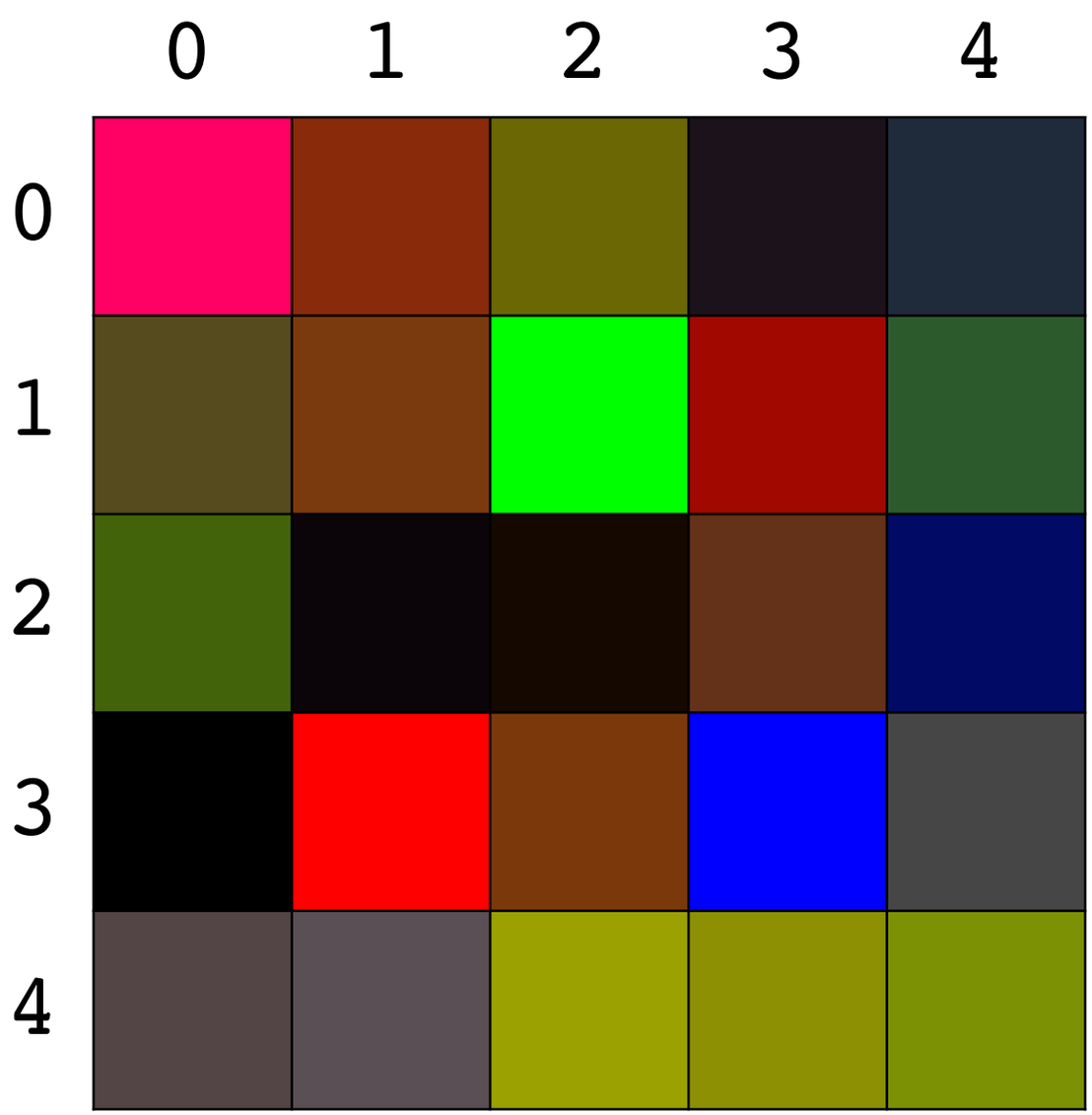
`GImage source`

(“where to hide the secret”)



`GImage source`

(“where to hide the secret”)



`GImage source`

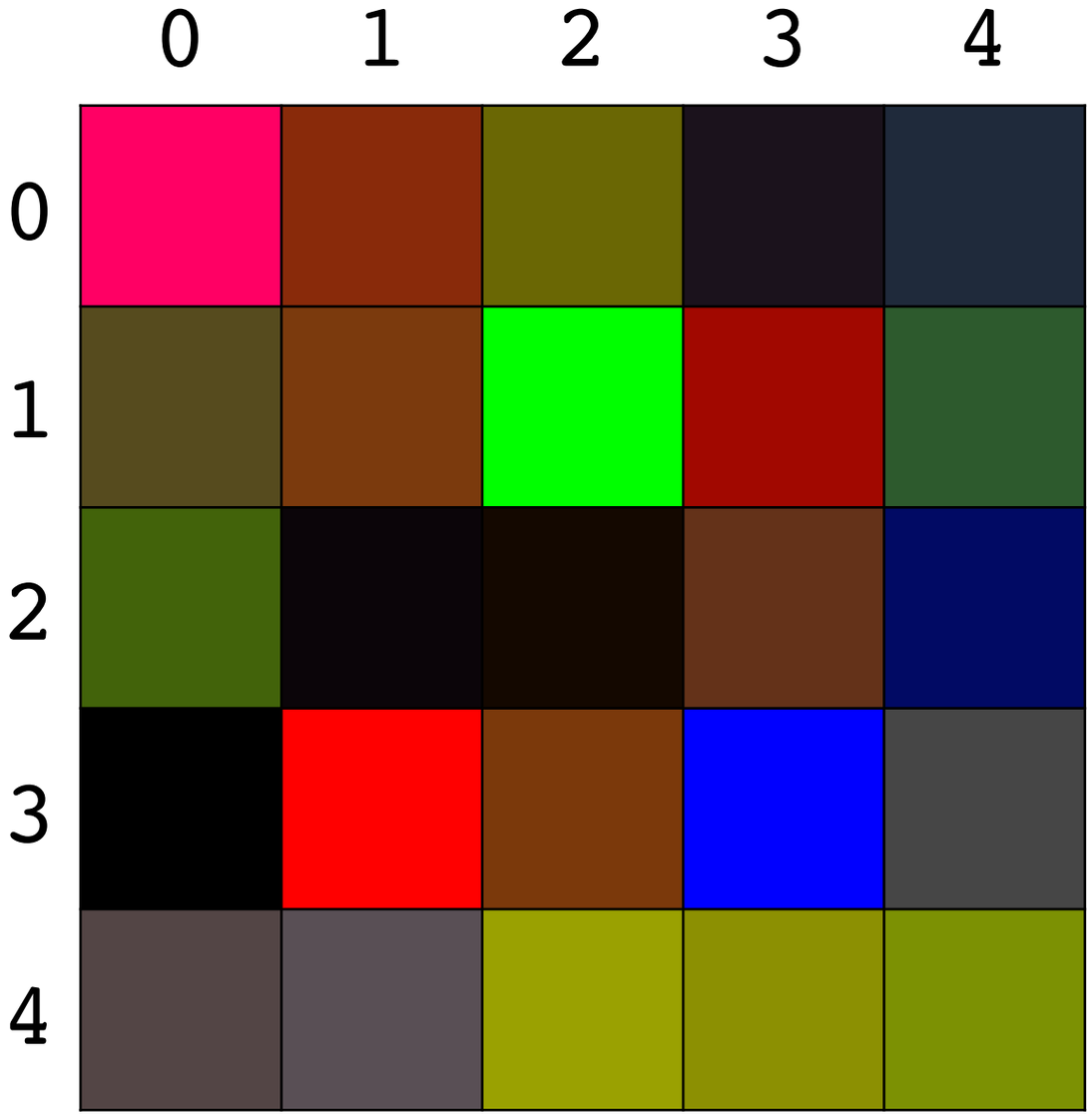
(“where to hide the secret”)

`GImage.getRed(__)`
`GImage.getGreen(__)`
`GImage.getBlue(__)`

`.getPixelArray()`

(see section 5 solution)

(see section 5 solution)



`GImage source`

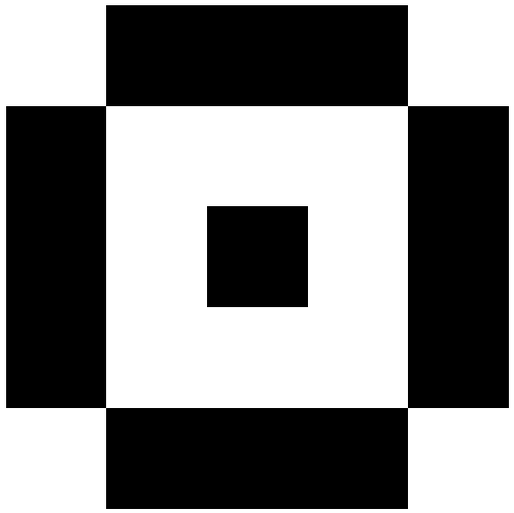
(“where to hide the secret”)

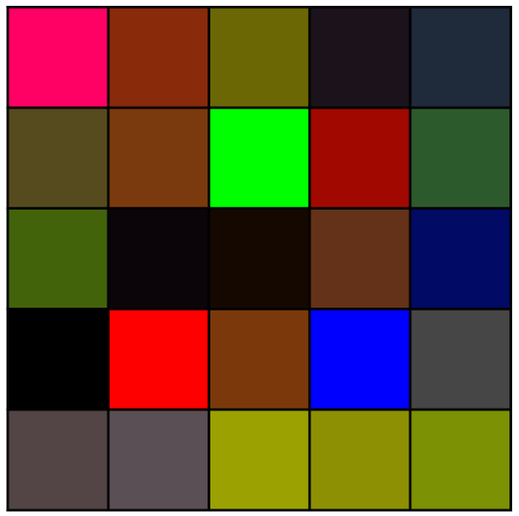
red,green,blue (0-255 for each)

	0	1	2	3	4
0	255,0,100	137,42,10	106,103,3	27,18,28	31,41,59
1	86,75,30	123,58,13	0,255,0	161,8,0	45,90,45
2	66,99,10	11,5,9	20,8,0	100,50,25	1,10,100
3	0,0,0	255,0,0	123,57,11	0,0,255	70,70,70
4	83,69,69	89,79,85	154,161,1	140,144,2	124,145,3

`int[][] pixels`

each cell is really one **`int`**, it’s the **`GImage`** methods above that turn convert each into the three components (red, green, blue)





given as a parameter

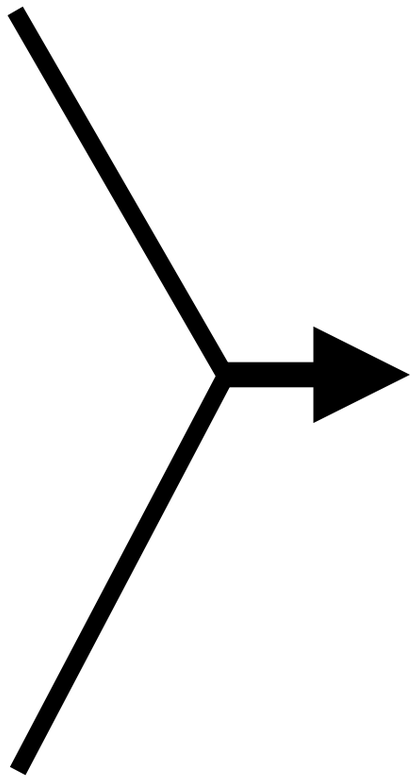
given as a parameter

FALSE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE
FALSE	TRUE	TRUE	TRUE	FALSE

255,0,100	137,42,10	106,103,3	27,18,28	31,41,59
86,75,30	123,58,13	0,255,0	161,8,0	45,90,45
66,99,10	11,5,9	20,8,0	100,50,25	1,10,100
0,0,0	255,0,0	123,57,11	0,0,255	70,70,70
83,69,69	89,79,85	154,161,1	140,144,2	124,145,3

FALSE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	FALSE	TRUE
FALSE	TRUE	TRUE	TRUE	FALSE

255,0,100	**137**,42,10	**106**,103,3	**27**,18,28	**31**,41,59
86,75,30	**123**,58,13	**0**,255,0	**161**,8,0	**45**,90,45
66,99,10	**11**,5,9	**20**,8,0	**100**,50,25	**1**,10,100
0,0,0	**255**,0,0	**123**,57,11	**0**,0,255	**70**,70,70
83,69,69	**89**,79,85	**154**,161,1	**140**,144,2	**124**,145,3



int[][] modified

254,0,100	**137**,42,10	**107**,103,3	**27**,18,28	**30**,41,59
87,75,30	**122**,58,13	**0**,255,0	**160**,8,0	**45**,90,45
67,99,10	**10**,5,9	**21**,8,0	**100**,50,25	**1**,10,100
1,0,0	**254**,0,0	**122**,57,11	**0**,0,255	**71**,70,70
82,69,69	**89**,79,85	**155**,161,1	**141**,144,2	**124**,145,3

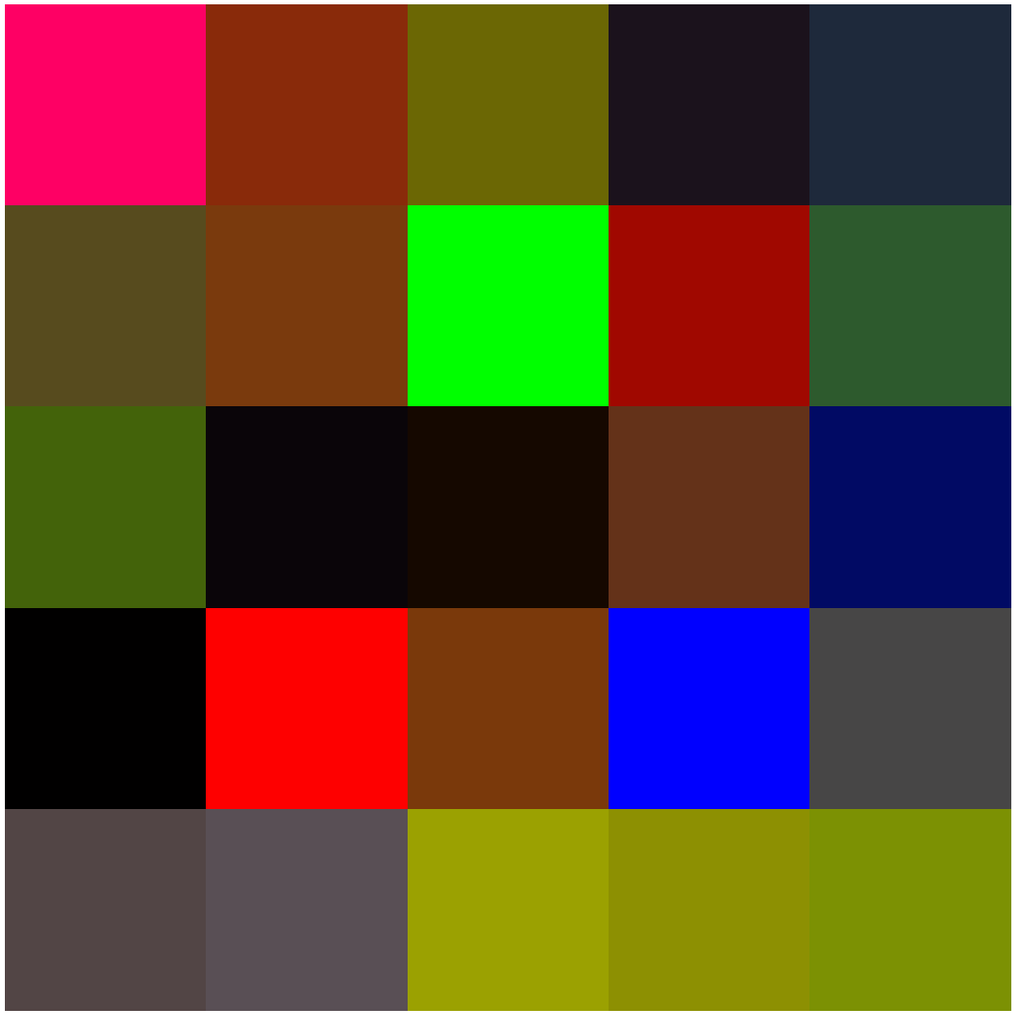
If the secret pixel is ***black***, it is represented as `true`, and you should make the red channel ***odd***.
If the secret pixel is ***white***, it is represented as `false`, and you should make the red channel ***even***.

```
int newPix = GImage.createRGBPixel(___,___,___)
```

int[][] modified

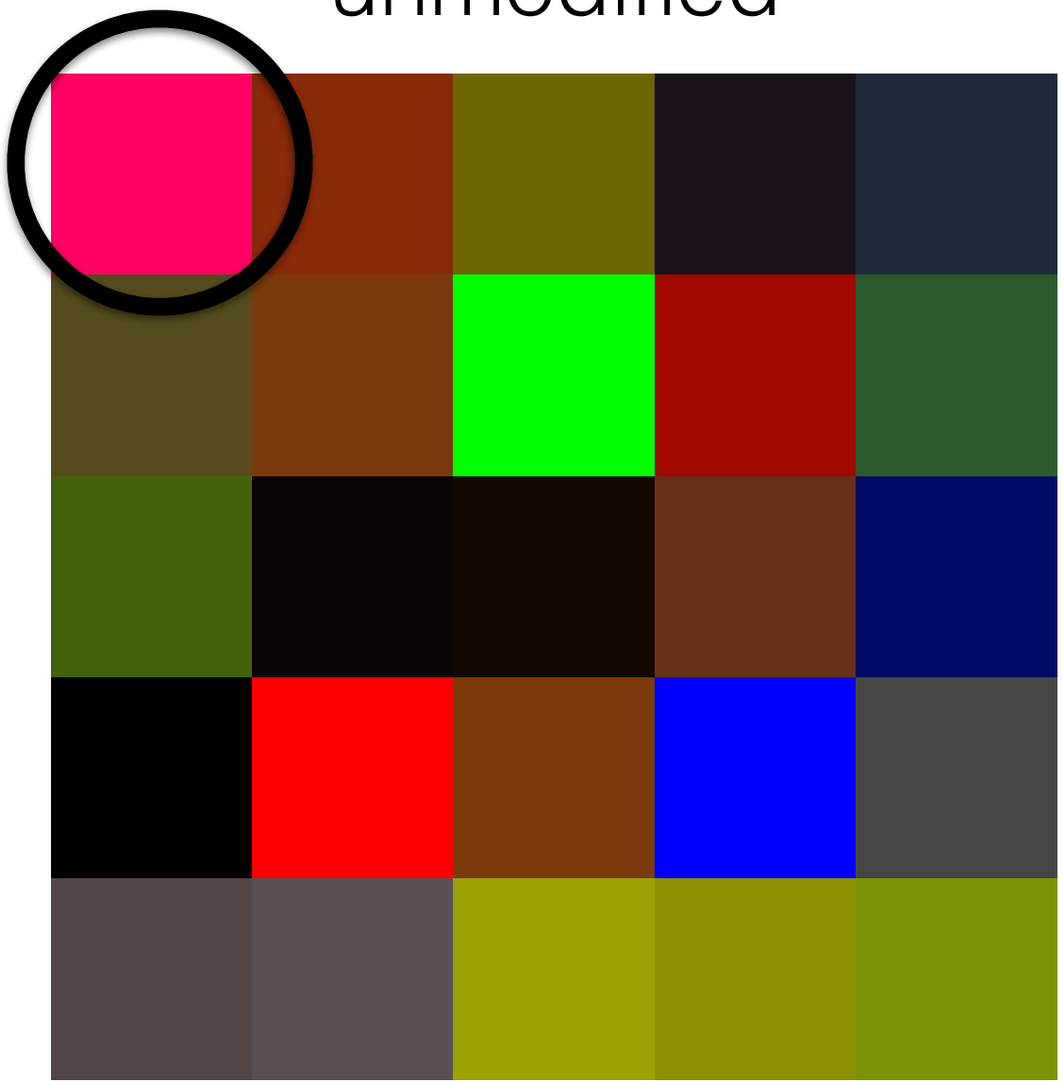
254,0,100	**137**,42,10	**107**,103,3	**27**,18,28	**30**,41,59
87,75,30	**122**,58,13	**0**,255,0	**160**,8,0	**45**,90,45
67,99,10	**10**,5,9	**21**,8,0	**100**,50,25	**1**,10,100
1,0,0	**254**,0,0	**122**,57,11	**0**,0,255	**71**,70,70
82,69,69	**89**,79,85	**155**,161,1	**141**,144,2	**124**,145,3

GImage return value

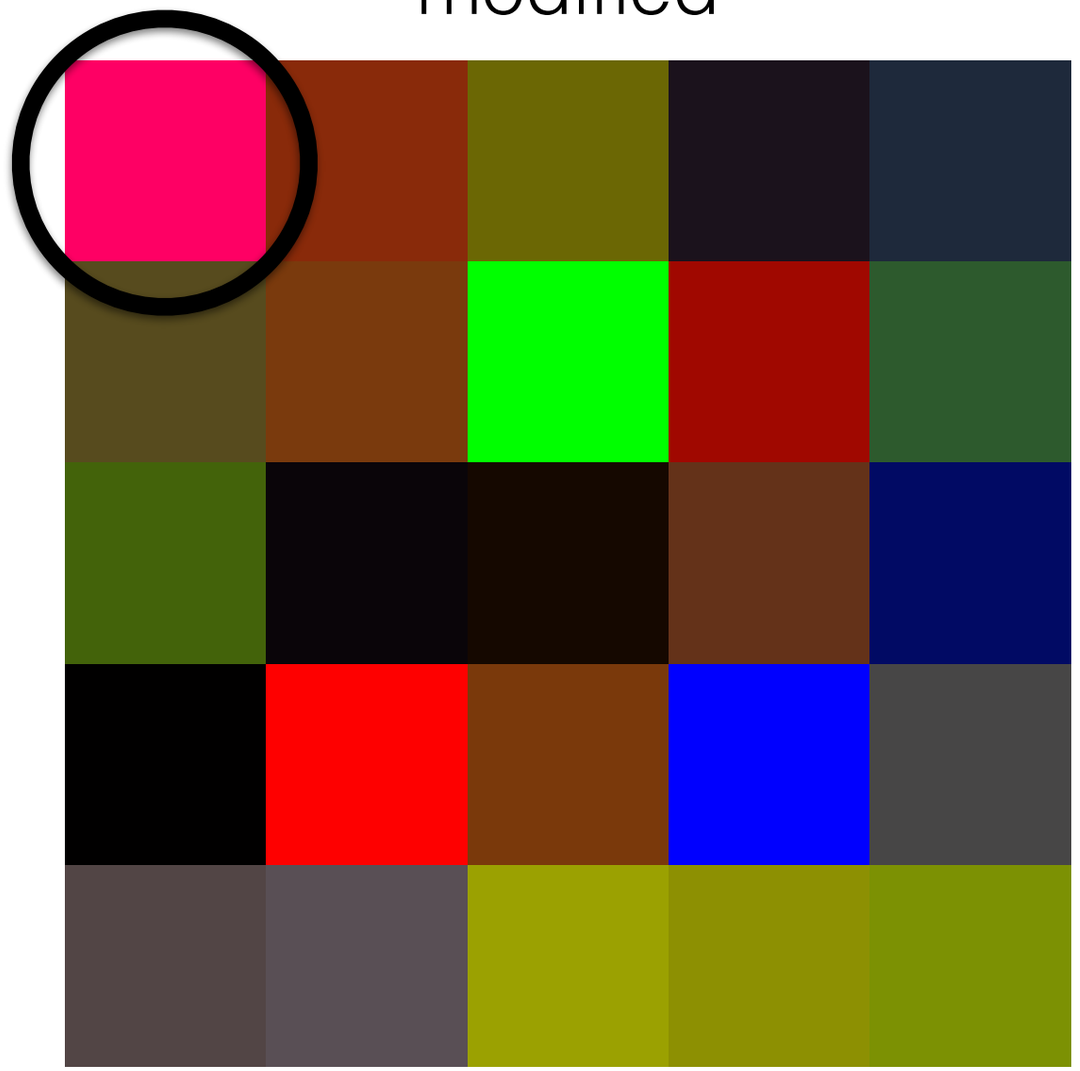


```
GImage modifiedGImage = new GImage(_____);
```

unmodified



modified



red: 255
green: 0
blue: 100

red: 254
green: 0
blue: 100

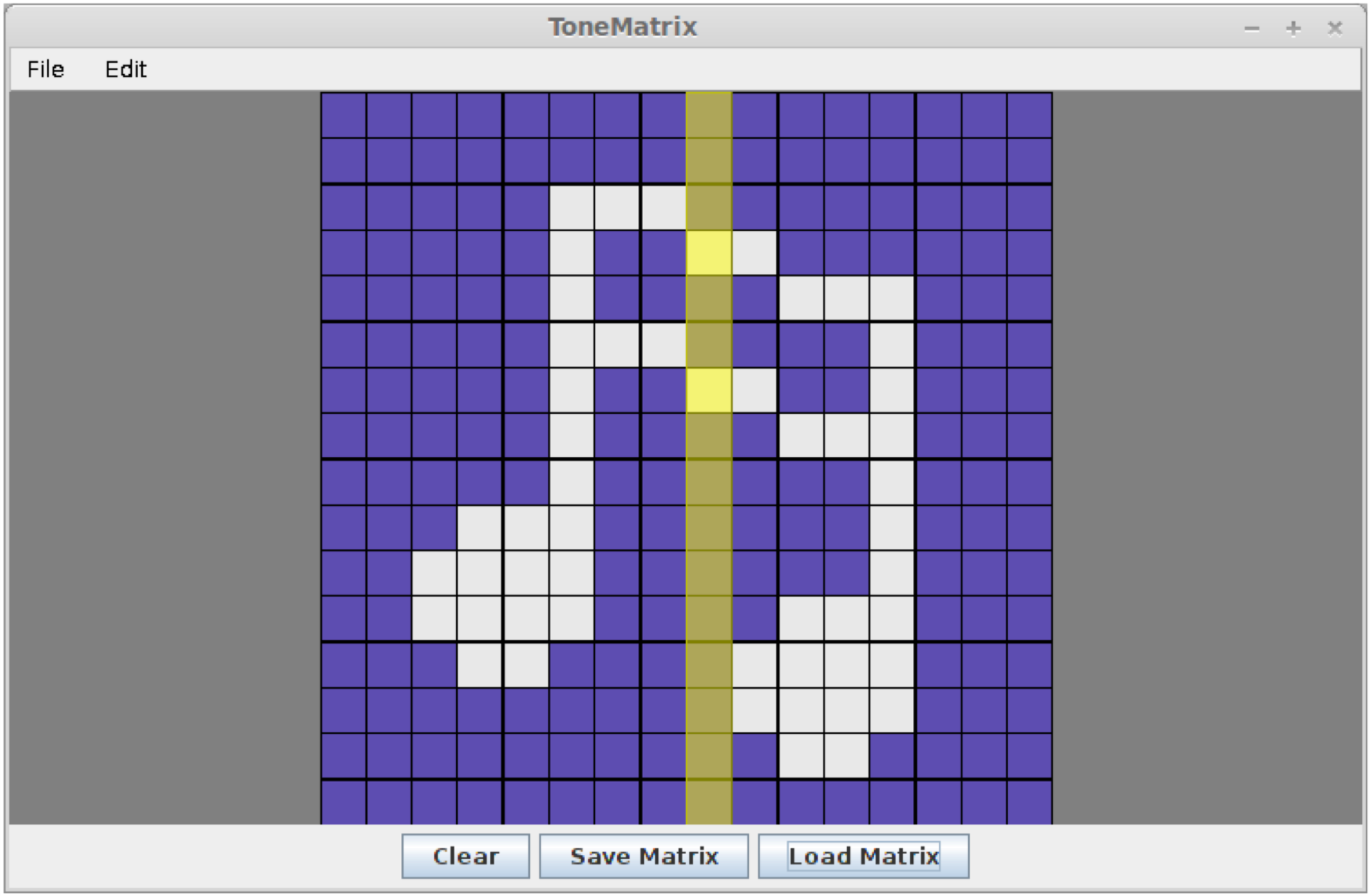
Tips

- **`findMessage`** is like **`hideMessage`** but in reverse (inspect each pixel for even or odd red channel and set the boolean in **`secretMessage`** accordingly)

- Try the given files and create your own

- In **`hideMessage`**, make sure you don't make red less than 0 or more than 255 when changing odd to even or even to odd

tone matrix



try the demo on the class website!

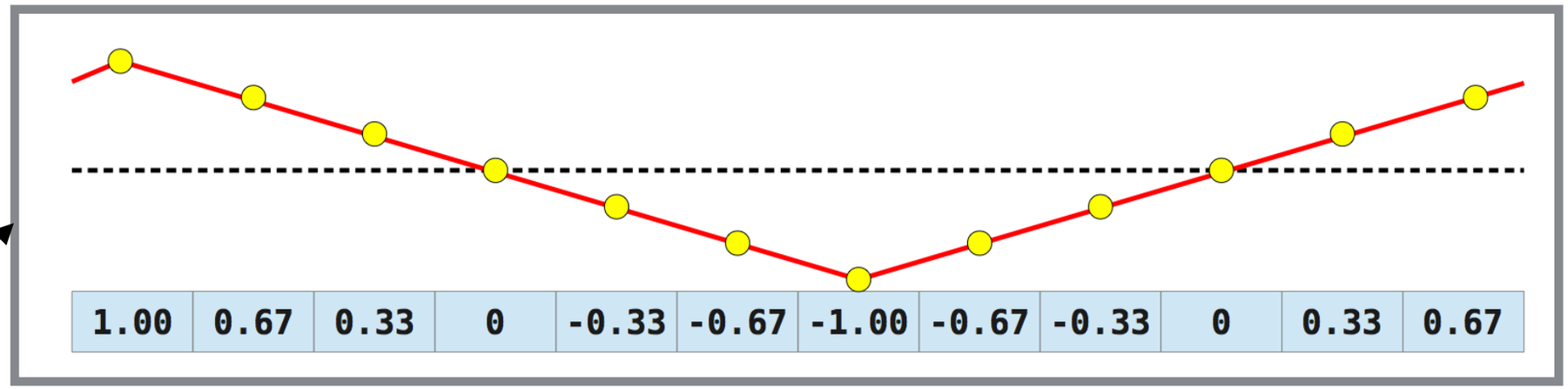
8 (column)

each light corresponds to an entire row in ***`samples`***

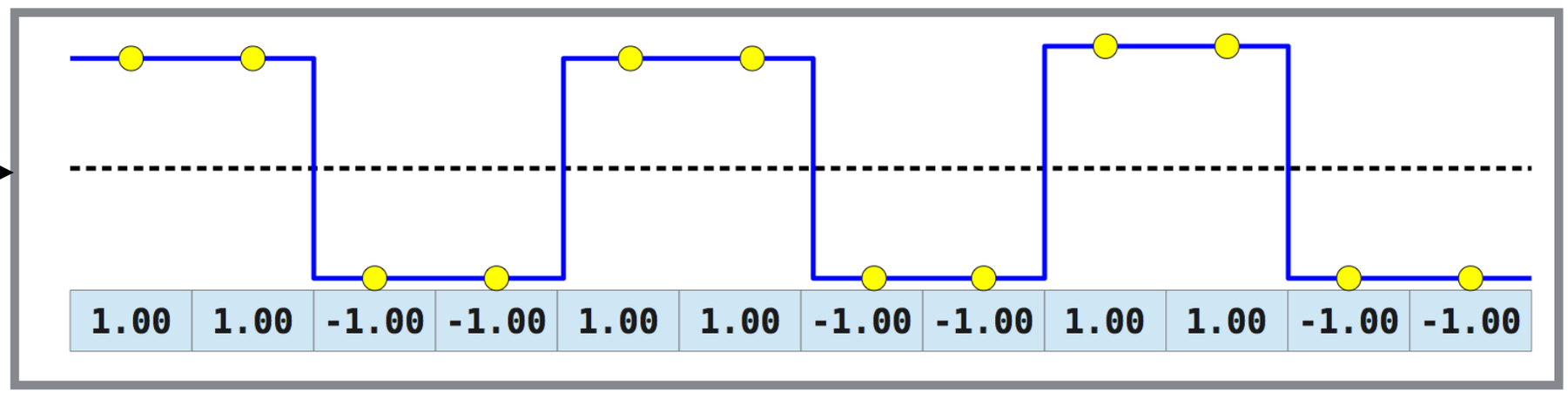
(row)

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

rix

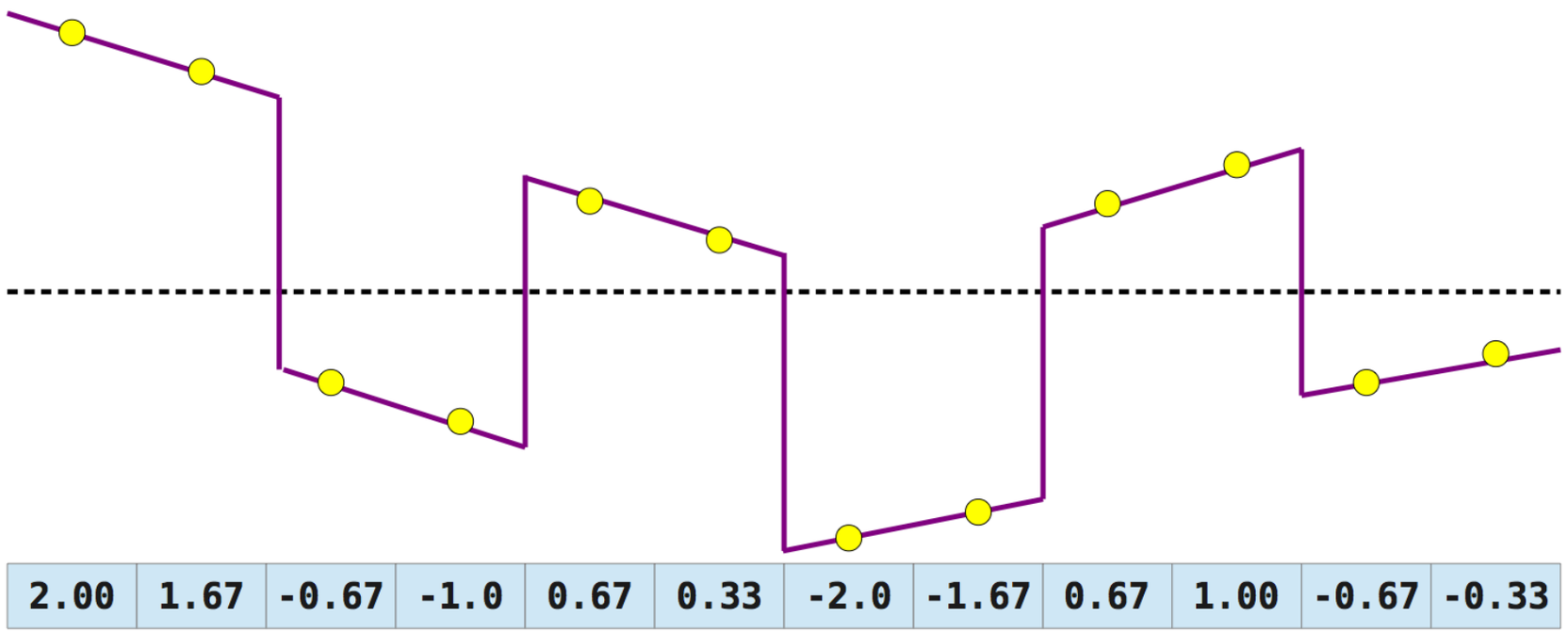


`samples[3]`



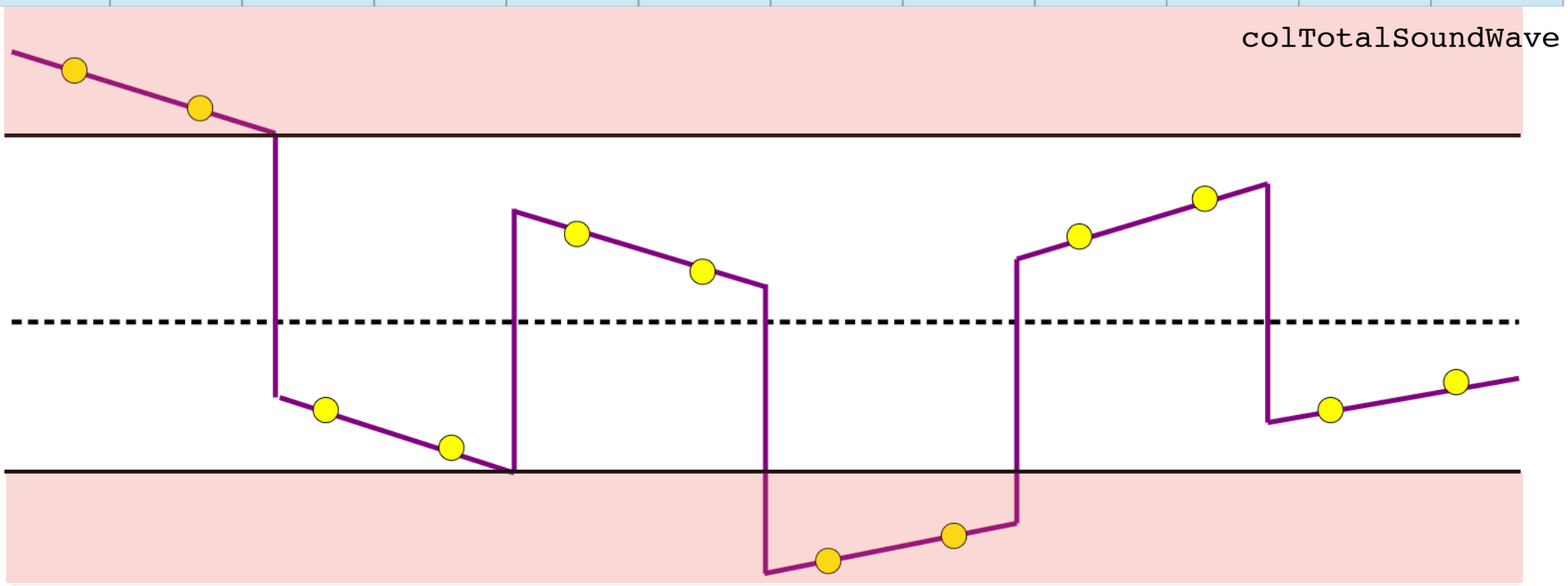
`samples[6]`

+

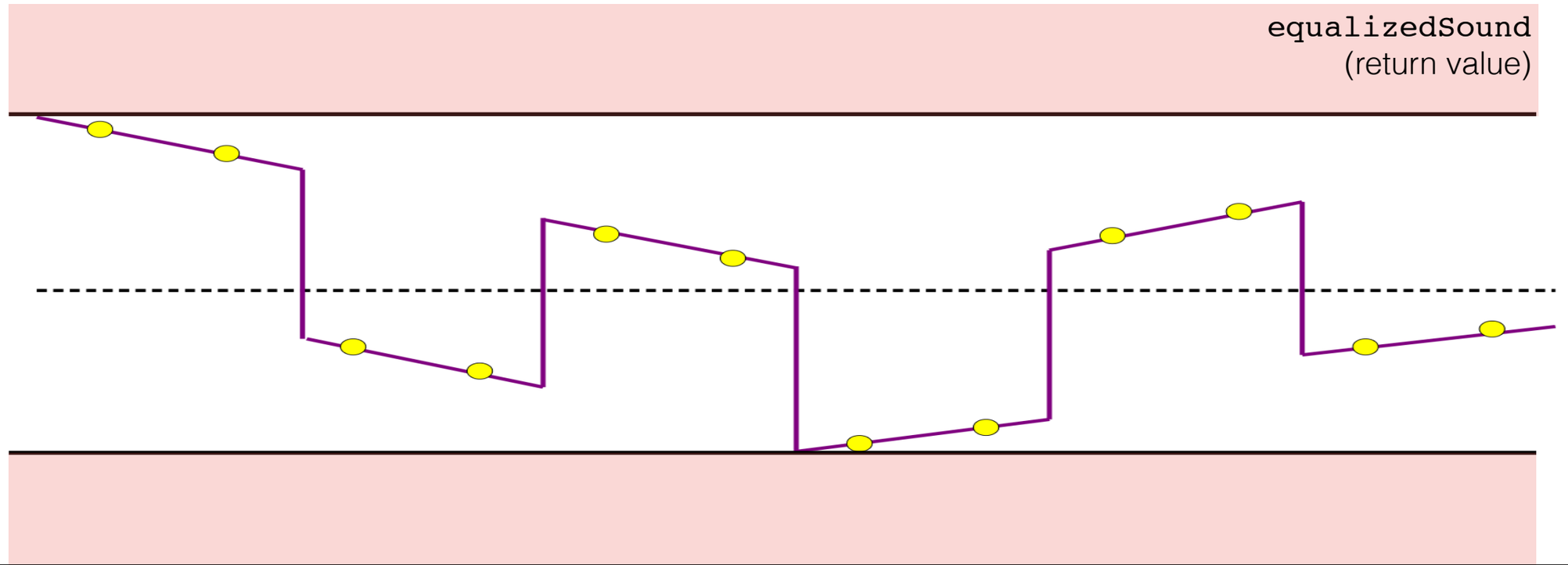


`colTotalSoundWave`

2.00	1.67	-0.67	-1.0	0.67	0.33	-2.0	-1.67	0.67	1.00	-0.67	-0.33



normalize the sound wave by “squashing” it to fit inside of the range [-1, +1] by finding the maximum intensity of the sound at any point (where the intensity of the sound at a single point in time is the absolute value of the sample at that point), then dividing all of the sample values in the sound by this maximum value.



`colTotalSoundWave`

2.00	1.67	-0.67	-1.0	0.67	0.33	-2.0	-1.67	0.67	1.00	-0.67	-0.33

1. iterate to find the max value (absolute value):

```
2.0
```

2. equalize the array (each entry above divided by max):

1.00	0.84	-0.34	-0.50	0.34	0.17	-1.00	-0.84	0.34	0.50	-0.34	-0.17

`equalizedSound`
(return value)

Tips

- Start off by testing your Tone Matrix with only one note playing per column

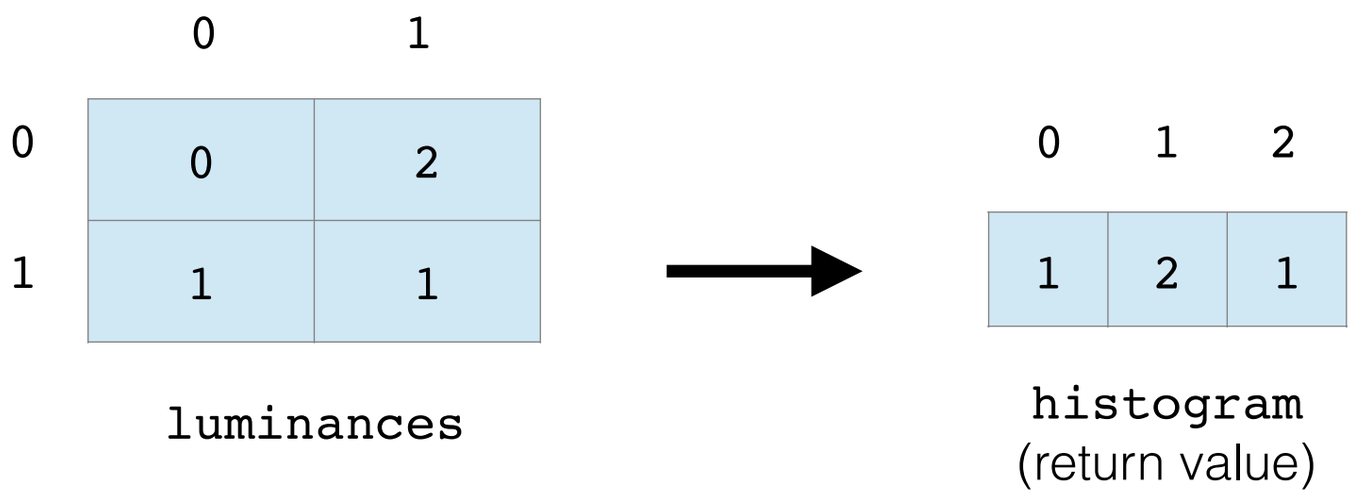
- Watch out for the case where no sounds are being played…you might divide by zero

histogram equalization

```java
/**
 * Given the luminances of the pixels in an image, returns a histogram of
 * the frequencies of those luminances.
 * <p>
 * You can assume that pixel luminances range from 0 to MAX_LUMINANCE,
 * inclusive.
 *
 * @param luminances The luminances in the picture.
 * @return A histogram of those luminances.
 */
public static int[] histogramFor(int[][] luminances) {
        /* TODO: Implement this! */
}
```

---

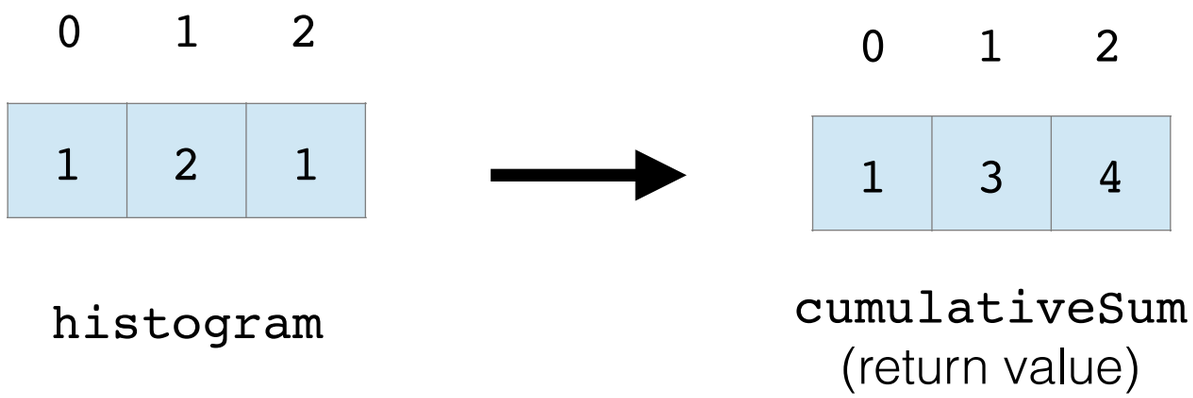
an example for a 4-pixel image with **`MAX_LUMINANCE = 3`**



```java
/**
 * Given a histogram of the luminances in an image, returns an array of the
 * cumulative frequencies of that image. Each entry of this array should be
 * equal to the sum of all the array entries up to and including its index
 * in the input histogram array.
 * <p>
 * For example, given the array [1, 2, 3, 4, 5], the result should be
 * [1, 3, 6, 10, 15].
 *
 * @param histogram The input histogram.
 * @return The cumulative frequency array.
 */
public static int[] cumulativeSumFor(int[] histogram) {
        /* TODO: Implement this! */
}
```

---

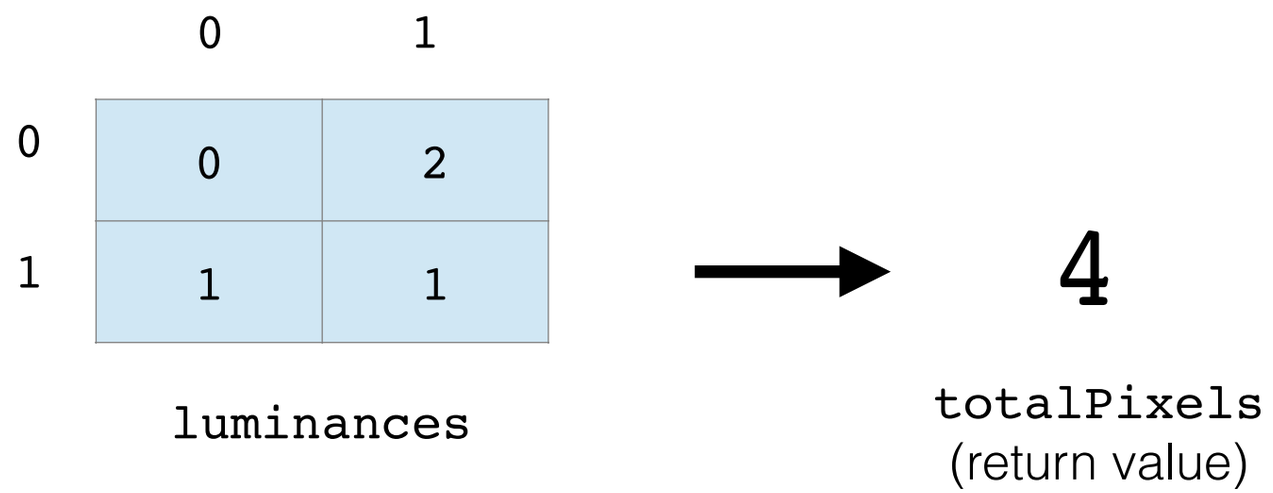
an example



```java
/**
 * Returns the total number of pixels in the given image.
 *
 * @param luminances A matrix of the luminances within an image.
 * @return The total number of pixels in that image.
 */
public static int totalPixelsIn(int[][] luminances) {
        /* TODO: Implement this! */
}
```

---

an example for a 4-pixel image



```java
/**
 * Applies the histogram equalization algorithm to the given image,
 * represented by a matrix of its luminances.
 * <p>
 * You are strongly encouraged to use the three methods you have implemented
 * above in order to implement this method.
 *
 * @param luminances The luminances of the input image.
 * @return The luminances of the image formed by applying histogram
 *         equalization.
 */
public static int[][] equalize(int[][] luminances) {
      /* TODO: Implement this! */
}
```

$$newLuminance = \frac{\text{MAX\_LUMINANCE} \cdot cumulativeHistogram[L]}{totalPixels}$$

HistogramEqualizationTests

File Edit

histogramFor Tests

Fail	Result array should have proper number of entries.
Fail	Testing image of all black pixels.
Fail	Testing image of all white pixels.
Fail	Testing image with one pixel of each color.
Fail	Testing image with half black pixels and half white pixels.

cumulativeSumFor Tests

Fail	Result array should have proper number of entries.
Fail	Testing sum of histogram of all 1s.
Fail	Testing histogram of all black pixels.
Fail	Testing histogram of only white and black pixels.
Fail	Testing histogram of increasing frequencies.

totalPixelsIn Tests

Fail	Test of 40 x 40 image.
Fail	Test of 50 x 30 image.
Fail	Test of 30 x 50 image.
Fail	Test of 1 x 1 image.
Fail	Test of 1 x 8 image.
Fail	Test of 8 x 1 image.

Tips

- Implement the methods in the order they appear in the starter code

- Use the testing harness to debug each method before going on to the next one

- Follow the specifications carefully
- Comment
- Go to the LaIR if you get stuck
- **Incorporate IG feedback!**
- Have fun!