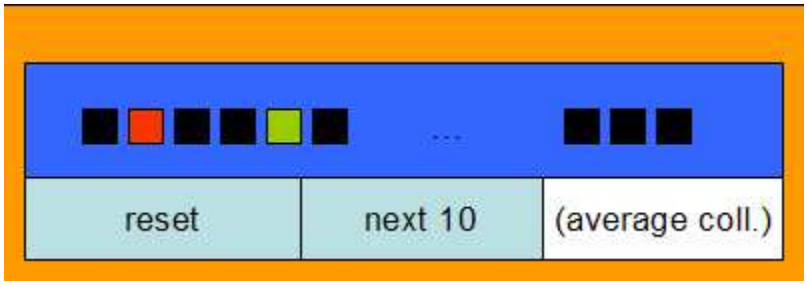
Assignment 9: Visual Hashtable

This assignment lets you visually experience the decreasing performance of an open addressing style hashtable, the more it is filled.

Your task: build a hashtable, size = 100 elements, and visualize it in the following way (or similar, use SimpleGUI):



The upper part shows little boxes in black, red and green. This is the important part, as it visualizes each cell of your hashtable. More precisely: The color of the box corresponds to the number of collisions that occurred to the element in the cell. The following coloring scheme should be used:

- an empty cell should be black
- a cell with an element that didn't collide at all should be shown in green,
- a cell containing an element that collided >=7 times should be red,
- a cell containing an element that collided 0<n<7 times should interpolate accordingly between green and red.

Again: the boxes do NOT show the cell content, but the number of collisions that occurred before the respective cell was used.

Color Generation

The following code shows how to compute the color , given the numbers of collisions ("-1" is used for empty cell). "Color" is a class in jawa.awt.

```
//-----------------------------------------------------
public Color computeCellColor(int collisions){
    int r = (int)(255.0/7.0*collisions);     // collisions = 0: r = 0. coll >=7: r>=255.
    r = (int)Math.min(r,255);                // same as: if (r>255) r = 255;
    int g = 255-r;                           // green value: the 'opposite' of r
    int b = 0;                 // sets blue to zero, only red/green values are displayed
    return(new Color(r,g,b));
}
//---------------------------------------------------------------------
```

Explanation of the color scheme:

- 0 collisions will create green,
- >=7 collision will be red,
- everything in between will be between red and green, which is yellowish.

Note 1: The value 7 (for red) is chosen because it is about log(100). This means that red elements would have performed better in a balanced binary search tree.
Note 2: no values in (Computer) Science are chosen randomly. There is always a reason.

The GUI elements:

· RESET resets the system (empties the array).
· NEXT10 creates 10 random numbers and puts them into the hashtable.
· (av. collisions) is just a text output.

remark: currently you are not able to use two buttons at the same time in SimpleGUI, since we did not learn about even-driven programming. Therefore you either program the "next 10" button only (omit the "reset" button), or, for 3 bonus points, program both buttons, event driven. There are examples in the SimpleGUI manual how to do so.

Specifications for the hashtable:

- The hashtable should be able to contain 100 double type numbers. When the "Next10" button is pressed, please generate 10 random numbers in the range of between [0..1000], use v = Math.random()*1000.
- The hash function is H(v)=round(v). The index is H(v)%100.
- For collision handling use linear probing
- In this assignment, you will need 2 arrays: one that contains the values (the actual hash table), and one that contains the number of collisions. The visualization only uses the collision-array!
- About collision counting: when two values collide, only the collision counter for the NEW number is increased. Or: once a value is in the hash table, its collision number will never change.

...and of course you can fill max. 100 elements into an array of size 100, i.e. you need to stop when the table is full.

What do we expect to see?

Filling in the first 10 elements ("next10" clicked once), all elements should be green(ish), since not too many collision should occur. With an increasing number, the newly inserted elements should become more and more red, since, in their process of insertion, they collided more often.