PRG (ETS de Ingeniería Informática) - Academic year 2019-2020

Lab practice 1. Drawing recursive figures with recursion

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Contents

1 `RSquare-A`. Recursive definition

An `RSquare` is a geometrical figure in 2D obtained by means of repeating a basic pattern: a square. A square in which it is possible to put other squares at their four corners, but with a lateral length being a fraction of the lateral length of the original square. At the same time, it is possible to put other squares in the corners of the smaller squares, still smaller, and so on. The number of times $n \geq 1$ this pattern is repeated in the drawing is referred to as the *order* of the figure.

For simplicity, these figures are going to be classified according to the way the squares are overlapped. Let `RSquare-A` be the kind of figures recursively defined as:

- An `RSquare-A` of *order* 1 is a square with lateral *length* equal to l, *centred* at point (x, y), and with the border painted with a different colour the square is filled, i.e., with highlighted border.
- An `RSquare-A` of *order* $n > 1$ is a square with lateral *length* equal to l, *centred* at point (x, y), with highlighted border, and with other four `RSquare-A` figures of order $n-1$ with lateral *length* equal to $l/2$.

  The central square is drawn over the other four at the corners.

As an example, Figure 1 shows some `RSquare-A` type squares or order 1, 2 and 3, all of them with the same lateral length l.

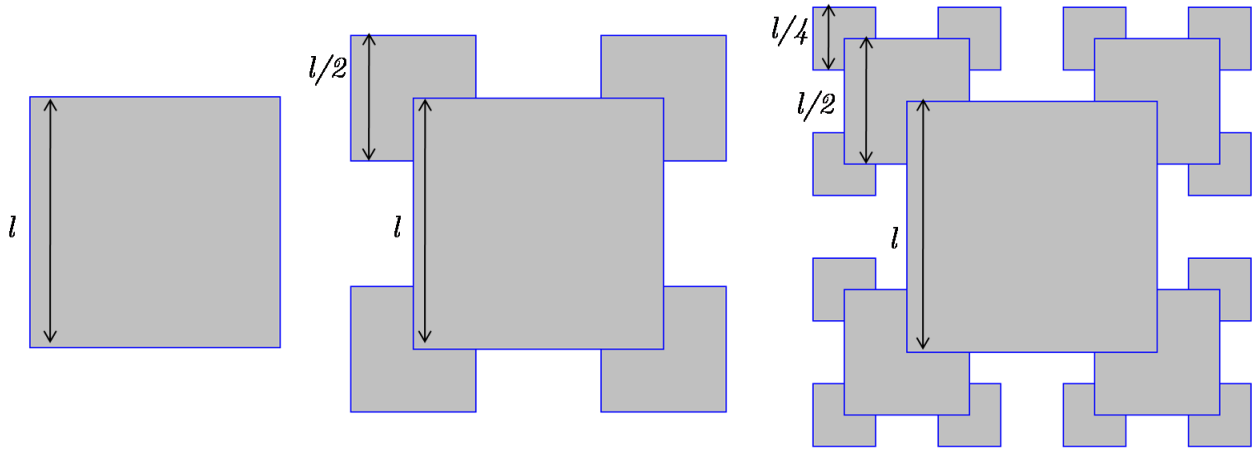


Figure 1: `RSquare-A` figures of lateral length equal to l and order 1, 2, and 3, respectively.

Notice that, following the previous definition, drawing one of these figures of *order* 4 implies to draw four figures of *order* 3 with lateral length $l/2$ located at the corners of the figure of *order* 4. By the recursive definition, drawing figures of *order 3* implies, at the same time, drawing four figures of *order* 2 with lateral length $l/4$ at each of the four corners of each of the four figures of *order* 3, and so on. The recursive drawing stops when figures of *order* 1 are drawn.

2 Drawing an `RSquare-A` figure using the library `Graph2D`

One of the tasks in this lab practice is to write a Java class for drawing `RSquare-A` figures of order $n \geq 1$. This task simplifies thanks to the use of the graphical library `Graph2D`, already used during first semester. This library facilitates to define windows of a specified size and to draw rectangles and other geometrical figures inside windows. The figures can be drawn at any arbitrary position and can be of any arbitrary size. Both, the position coordinates and the size of each figure should be specified by the programmer.

The following recommendations should be taken into account when writing the code:

- According to the characteristics of `RSquare-A` figures, one of order n and lateral length l should be fit into a square bounded by a lateral size equal to

$$l + \frac{l}{2} + \frac{l}{2^2} + \cdots + \frac{l}{2^{n-1}} = 2l - \frac{l}{2^{n-1}}$$

  whose upper bound is $2l$. [1]

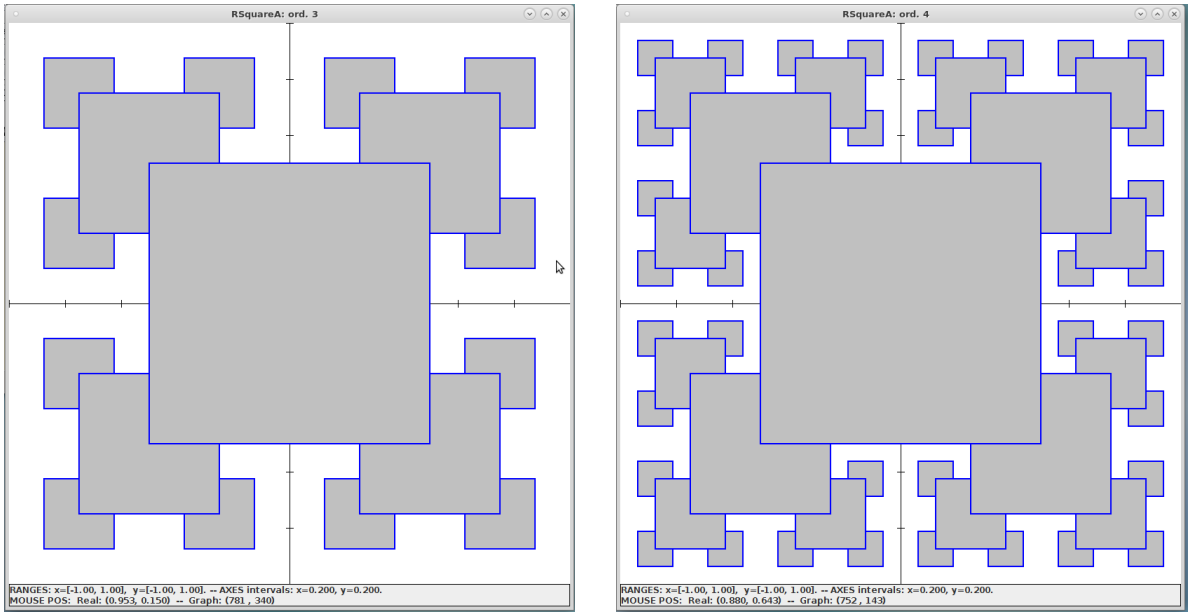
  Additionally, in order to make the solution simpler, the `RSquare-A` of higher order will be centred at the origin of coordinates, i.e., at point $(0, 0)$, and its lateral length will be the unity. These criteria have been followed when drawing the figures shown by Figure 2.

[1] The summation $l + \frac{l}{2} + \frac{l}{2^2} + \cdots + \frac{l}{2^{n-1}}$ is the sum of the n first terms of a geometric series in which l is the first term and $\frac{1}{2}$ is the common ratio. Thus, $\sum_{k=0}^{n-1} l\ \frac{1}{2^k} = l\ \frac{1 - \frac{1}{2^n}}{1 - \frac{1}{2}} = l\ \frac{2(2^n - 1)}{2^n} = 2l - \frac{l}{2^{n-1}}$

According to the previous results, this implies that if you have to draw a figure of lateral length equal to 1, then the window should have lateral length equal to 2, but with $(-1, -1)$ as the coordinates of the lower left corner, and $(1, 1)$ as the coordinates of the upper right corner. An example is shown in Figure 3.

- Drawing a square with highlighted border is performed in two steps. First fill the area of the square by means of method `fillRect()` from the graphical library. Second, draw the border using a different colour by means of method `drawRect()`. As you can read in the documentation of the library `Graph2D`, the coordinates these two methods need are the corresponding to the upper left corner of the rectangle.
- In order to guarantee all the squares of the figure are drawn recursively in the same window, a method should be implemented with four parameters: (1) the reference to the object representing the window, (2) the order of the figure, (3) the size, and (4) the position (coordinates x and y). The position can be two parameters.

  Additionally, in the same class it should be implemented another method with a unique parameter, just the order of the figure, then, it should create a window with the correct size for fitting the figure of the specified order. Then, this method should call the previous one with initial values of the parameters, which are lateral size equal to 1 and coordinates $(0, 0)$ for indicating the origin.



(a) `RSquare-A` of order 3. (b) `RSquare-A` of order 4.

Figure 2: `RSquare-A` figures of order 3 and 4. Both figures have lateral length equal to 1, are centred in a window of size 2×2.

3 Lab activities

3.1 Activity 1: Project `prg`. Installation of the graphical library

a) Create a *BlueJ* project with name `prg` in the directory `$HOME/DiscoW`. All the remaining activities of all the lab practices will be implemented in this project.

b) Create a package within the project with name `pract1`. Obviously, for the first lab practice.

c) Download the class `RSquare` from the folder *PRG:recursos/Laboratorio/Práctica 1* of *PoliformaT*, and add this class to the package `pract1`.

d) The graphical library is provided as the class `Graph2D` in the package `graph2D`. This package is contained in the JAR file `graphLib.jar` and its documentation in the file `docGraph2D.zip` (availables in the folder *PRG:recursos/Laboratorio/Librería gráfica* of *PoliformaT*).

This library should properly loaded indicating where the JAR file is located in the file system. If you took IIP in the first semester of this academic year, you will have it loaded and, therefore, you can skip steps d.1) and d.2) described below. If you didn't do it:

d.1) Set the file `graphLib.jar` in the *BlueJ* project `prg`, and add it in *Preferencias/Librerías* of *BlueJ*.

d.2) Extract the contents of the file `docGraph2D.zip` in the folder of the project `prg` in order to have access to the documentation when needed, in particular to file `Graph2D.html`.

e) Download the class `FigRecSimple` from the folder *PRG:recursos/Laboratorio/Práctica 1* of *PoliformaT*, and add this class to the package `pract1`. This class is an example for drawing recursive figures using the graphical library.

3.2 Activity 2: Drawing `RSquare-A` figures

a) Complete the code of the following method of the class `RSquare`:

```
/** Draws in the window <code>gd</code> a grey filled square with
 *  the border highlighted in blue, centred at <code>( cX, cY )</code>
 *  and lateral length equal to <code>l</code>.  */
public static void drawCentSquare(Graph2D gd, double cX, double cY, double l)
```

Test it once completed by creating an object of the class `Graph2D` in the *code pad* using the default constructor with $[-1.0, 1.0]$ as the range for both, x-axis and y-axis. Then run it for drawing the window and one square of lateral length equal to 1 centred at $(0, 0)$. Test it also with other positions and sizes of the square.

Notice that this method calls another one, `delay()`, already implemented in the class. The method `delay()` add time delays for avoiding each square is drawn too fast. The purpose of adding time delays is to allow students to observe the order in which squares are overlapped while the figure is drawn.

b) Write a recursive method with the following profile:

```
/** Draws in the window <code>gd</code> an <code>RSquare-A</code> figure
 *  of order <code>n >= 1</code>, with the central square centred at
 *  <code>( cX, cY )</code> and lateral length equal to $l$.  */
public static void rSquareA(Graph2D gd, int n, double cX, double cY, double l)
```

c) Write a wrapper method for invoking the recursive one described above. The profile of the wrapper method should be:

```
/** Draws an <code>RSquare-A</code> figure of order <code>n >= 1</code>
 *  with lateral length equal to 1 and centred at <code>(0, 0)</code>. */
public static void rSquareA(int n)
```

This method should create a window with $[-1.0, 1.0]$ as the range for both x-axis and y-axis, title `"RSquareA: ord.  " + n`, and should invoke the recursive method `rSquareA()` for drawing a figure of order `n` with central square centred at $(0, 0)$ and lateral length equal to 1.

d) Run these methods with values for the order of the figures equal to 3, 4 and 7. Check whether the obtained drawings match with the ones shown by Figures 2(a), 2(b) and 3.

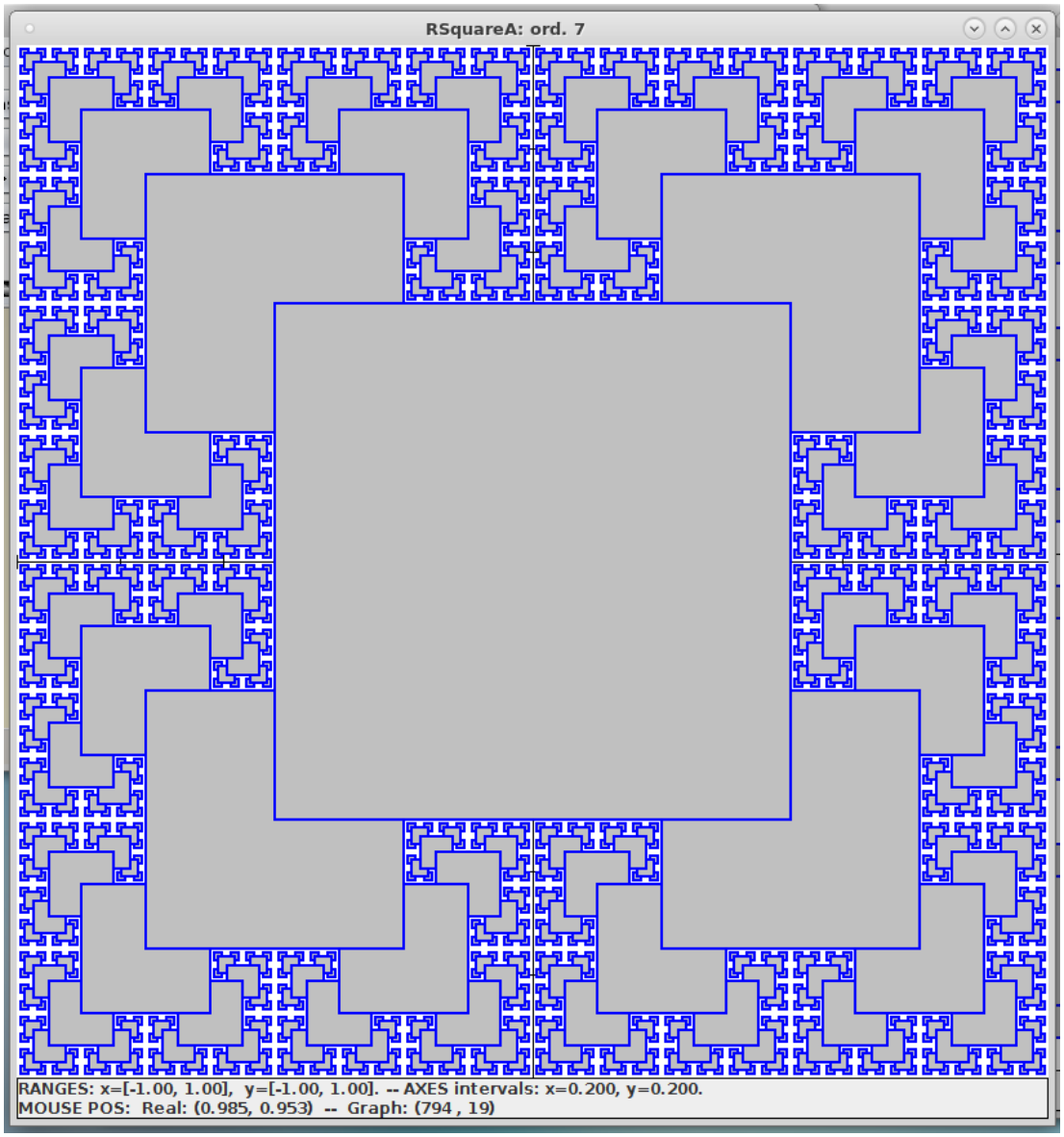
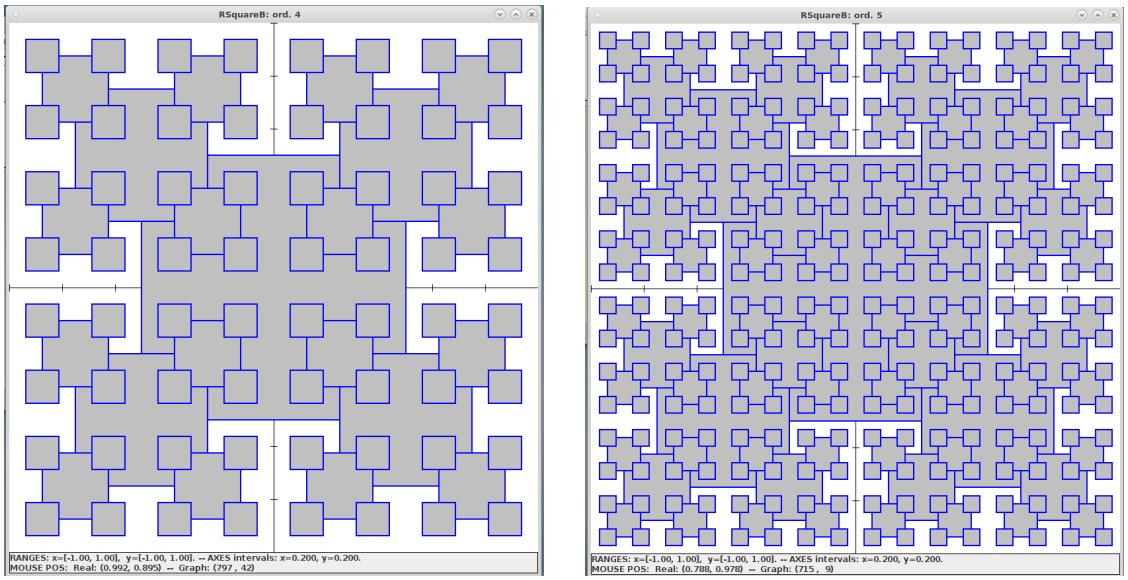


Figure 3: An `RSquare-A` figure of order 7 and lateral length l needs a square area with lateral length less than $2l$.

3.3 Activity 3: Drawing `RSquare-B` figures

As mentioned in the first section, figures of type `RSquare` can be classified according to the way squares are overlapped. Figure 4 shows another type of `RSquare` figures named `RSquare-B`. In this type of figures, smaller squares are overlapped over the greater ones. Drawings obtained with `RSquare-B` figures show a distribution of squares which is different from the one obtained with `RSquare-A` figures. `RSquare-B` figures have a recursive definition very similar to `RSquare-A` figures, but in the general case, the central square, the greater one at each order, should be drawn before the smaller ones, then, the smaller ones are overlapped over the central square. In `RSquare-B` figures the smaller squares appear on the top, because they are drawn in the trivial case, as Figure 4 shows.



(a) `RSquare-B` figure of order 4. (b) `RSquare-B` figure of order 5.

Figure 4: `RSquare-B` figures of order 4 and 5.

In this activity new methods should be added into the class, which are very similar to the ones added in activity 2. Then, you should run similar tests.

```
/** Draws in the windows <code>gd</code> an <code>RSquare-B</code>
 *  figure of order <code>n >= 1</code>, centred at
 *  <code>( cX, cY )</code> with a central square of lateral
 *  length equal to <code>l</code>.
 */
public static void rSquareB(Graph2D gd, int n, double cX, double cY, double l)

/** Draws an <code>RSquare-B</code> figure of order <code>n >= 1</code>,
 *  with lateral length equal to <code>1</code> and centred at <code>(0, 0)</code>.
 */
public static void rSquareB(int n)
```