A Guide to the TurtleGraphics Package for R

A. Cena, M. Gągolewski, M. Kosiński, N. Potocka,
B. Żogała-Siudem

Contents

1 The TurtleGraphics Package Introduction

The TurtleGraphics package offers to R-users functionality of the "turtle graphics" from Logo educational programming language. The main idea standing behind it is to encourage the children to learn programming and show that working with computer can be fun and creative.

The TurtleGraphics package allows to create etiher simple or more sophisticated graphics on the basis of lines. The main idea is that the Turtle, described by its location and orientation, moves with commands that are relative to its own position. The line that it leaves behind can be controlled, by disabling it or by setting its color and type.

The TurtleGraphics package offers functions to move forward or backward a given distance and to turn the Turtle in a choosen direction. The graphical parameter of the plot, for example the color, type or visibility of the line, can also be easily changed.

We strongly encourage you to try it yourself. Enjoy and have fun!

2 Installation And Usage of The Package

2.1 Installation of The Package

To install the package TurtleGraphics you should use following instructions.

```
> install_package("TurtleGraphics")
```

Then you have to load the package with the require() function, as it is shown below.

```
> require("TurtleGraphics")
```

2.2 The Basics

2.2.1 Moving The Turtle

To start using the program call the turtle_init() function. It creates a plot region and places the Turtle in the middle pointing north.

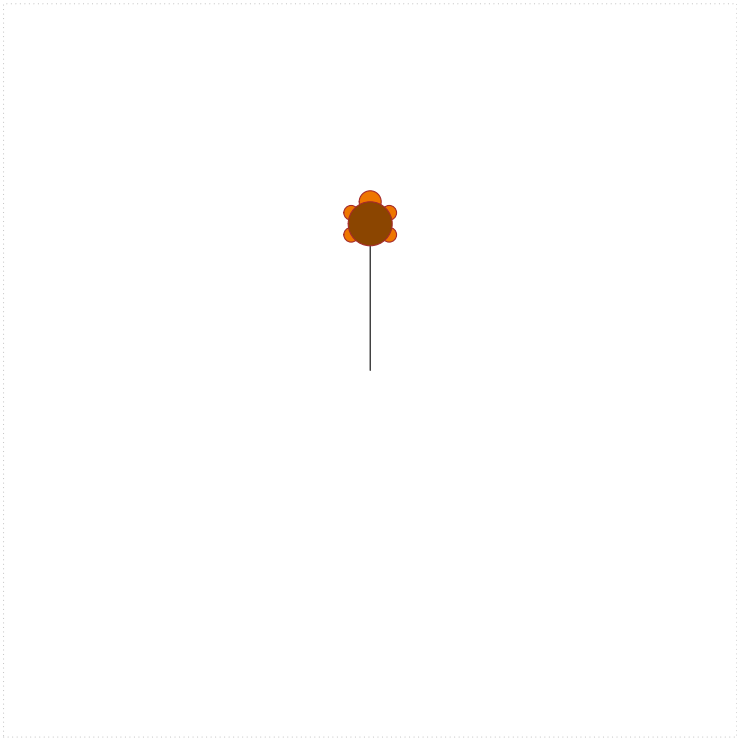
```
> turtle_init()
```

By default its size is 100 by 100 units. You can easily change it by passing as the arguments width and height (e.g. turtle_init(width=200, height=200)). To learn more about this functions try help(turtle_init).

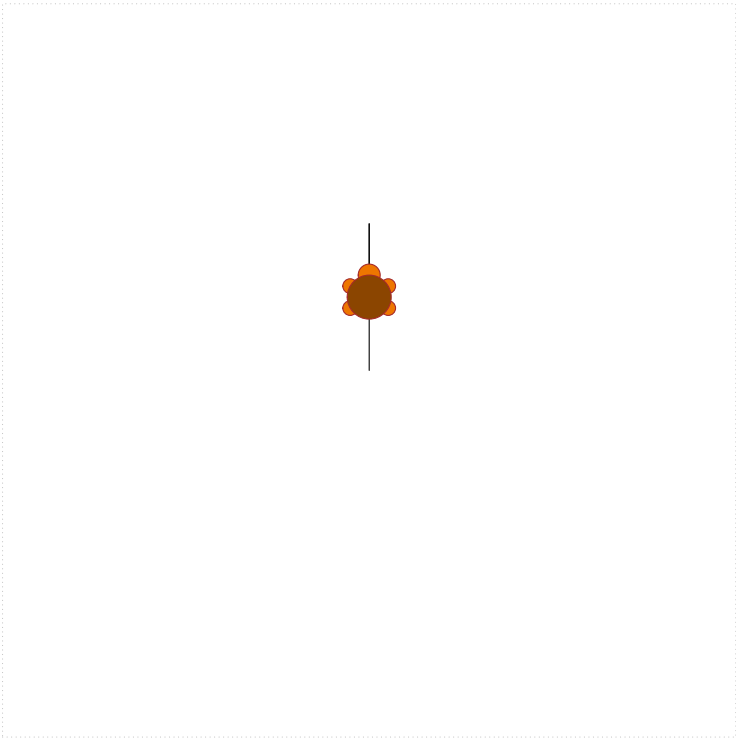
There are two main group of functions used to move the Turtle.

The first one consists of the turtle_forward() and the turtle_backward() functions. In its argument you have to give the distance you desire the Turtle to move. For example, to move the Turtle forward for a distance of 10 units use the turtle_forward() function. To move the Turtle backwards you can use either the turtle_forward() function with the negative number as an argument or simply use the turtle_backward() function.

```
> turtle_init()
> turtle_forward(dist=20)
```

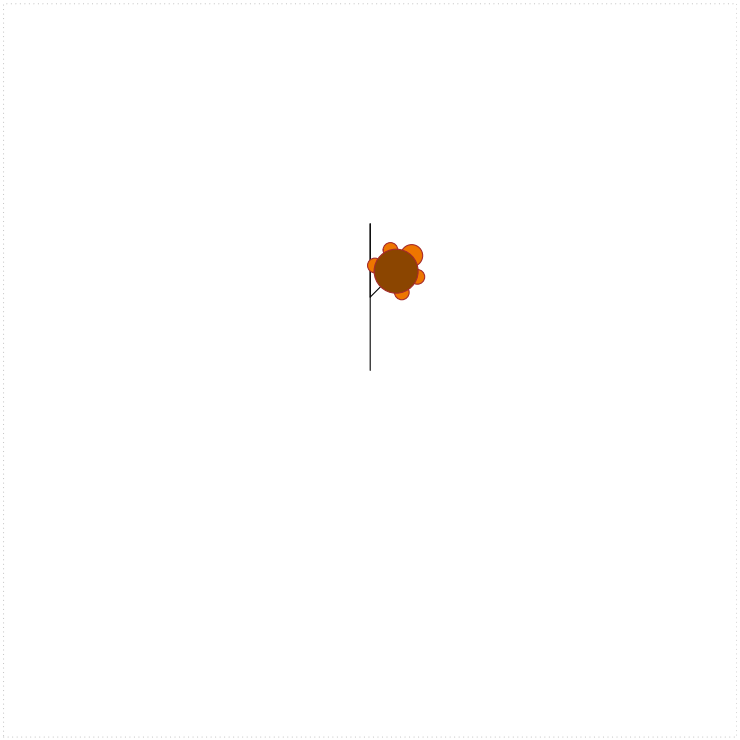


```
> turtle_backward(dist=10)
```



The other tool that helps to move the Turtle are the turtle_left and the turtle_right functions. They change the Turtle's direction by a given angle. For example, to turn the Turtle by 45 degrees to the right use the following:

```
> turtle_right(angle=45)
> turtle_forward(dist=5)
```

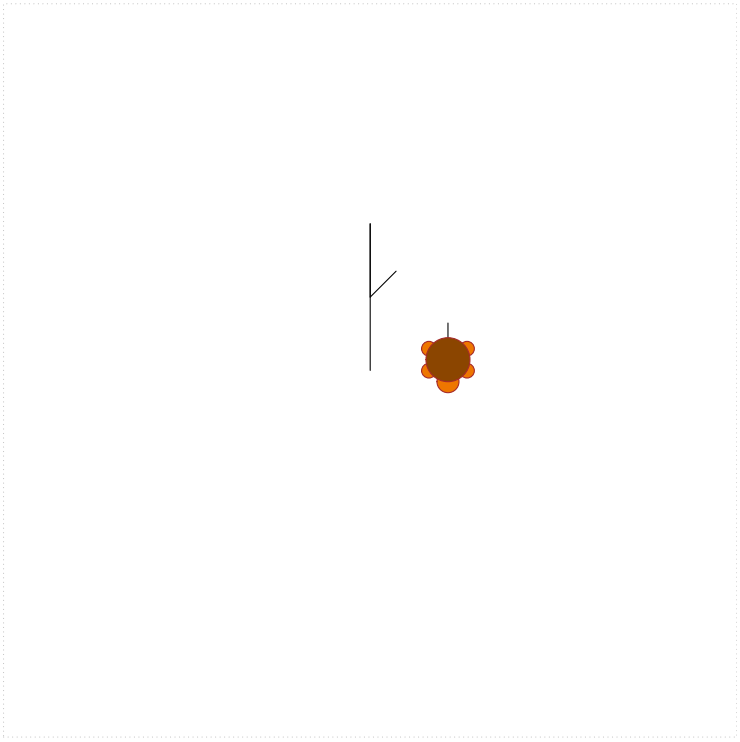


2.2.2 Additional Options

Now you know the basis and it is mostly it. Not very complicated, is it? Don't worry though, there are some additional parameters that you can play with.

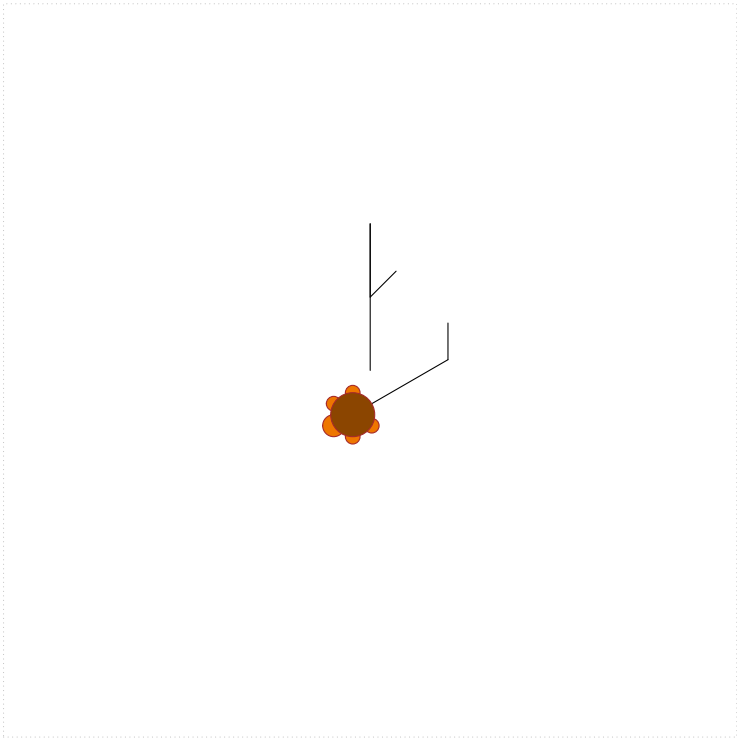
To disable the path from being drawn you can simply use the turtle_up() function. Let's consider a simple example. Turn the Turtle to the right by 90 degrees and then use the turtle_up() function. Now, when you move forward the path is not visible. If you want the path to be drawn again you should call the turtle_down() function.

```
> turtle_right(angle=90)
> turtle_up()
> turtle_forward(dist=10)
> turtle_right(angle=45)
> turtle_down()
> turtle_forward(dist=5)
```



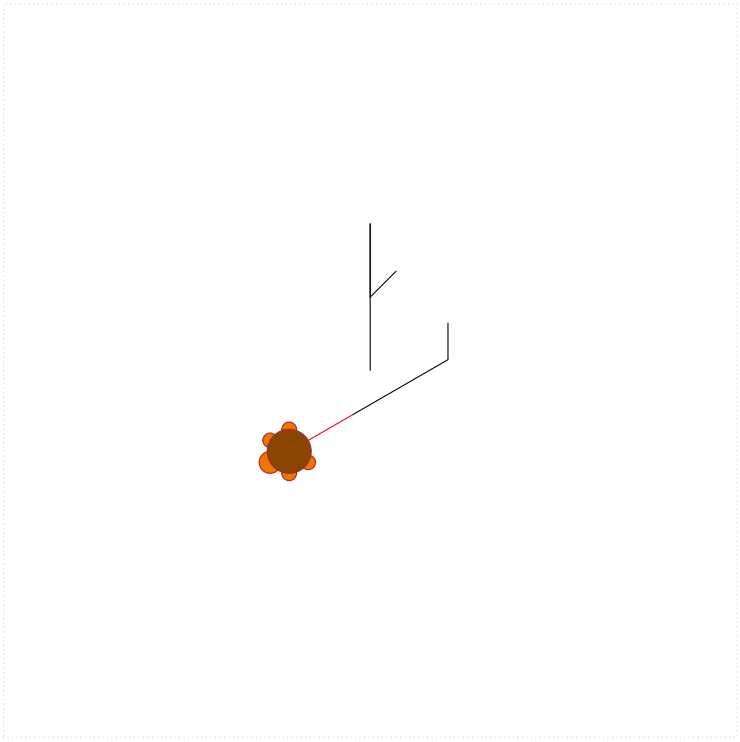
Similarly, you may show or hide the Turtle image, using the turtle_show() and turtle_hide() functions respectively. If you call a lot of functions it is strongly recommended to hide the Turtle first as it speeds up the process.

```
> turtle_hide()
> turtle_right(angle=60)
> turtle_forward(dist=15)
> turtle_show()
```



To change the nature of the Turtle's trace you can use the turtle_col(), turtle_lty() and turtle_lwd() functions. The first one, as you can easily guess, changes the color of the path the Turtle is making. For example, if you wish to change the trace into red try

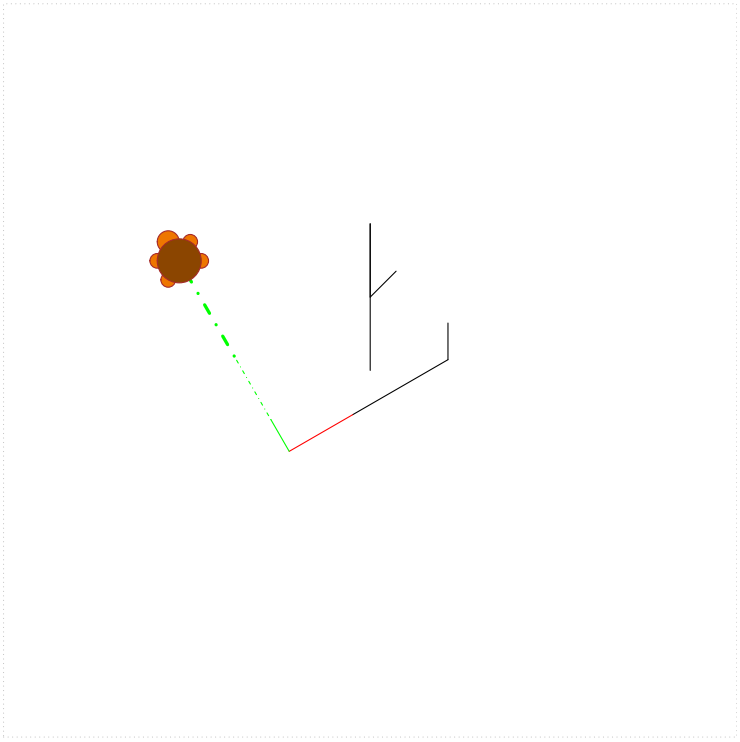
```
> turtle_col(col="red")
> turtle_forward(dist=10)
```



The full list of colors is available under the colors() function. Important! Remember that when passing an argument to this function you always have to use the quotation marks.

The turtle_lty() and turtle_lwd() functions change the type and the width of the line the Turle is making. To change the type of the path as an argument pass a number from 0 to 6 – each means a different type of the line (0 =blank, 1 =solid (default), 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash). To change the width of the line use the turtle_lwd() function. As an argument you pass a width you desire (but don't exaggerate!).

```
> turtle_right(angle=90)
> turtle_col(col="green")
> turtle_forward(dist=5)
> turtle_lty(lty=4)
> turtle_forward(dist=10)
> turtle_lwd(lwd=3)
> turtle_forward(dist=15)
```



If you got lost in the terrarium don't worry! There is a function turtle_status() which returns the parameters of your drawing. It tells you whether the Turtle and its path are visible, the width and height of the terrarium, where the Turle is placed right now and at which angle.

```
> turtle_status()

$DisplayOptions
$DisplayOptions$col
[1] "green"

$DisplayOptions$lty
[1] 4

$DisplayOptions$lwd
[1] 3

$DisplayOptions$visible
[1] TRUE

$DisplayOptions$draw
[1] TRUE
```

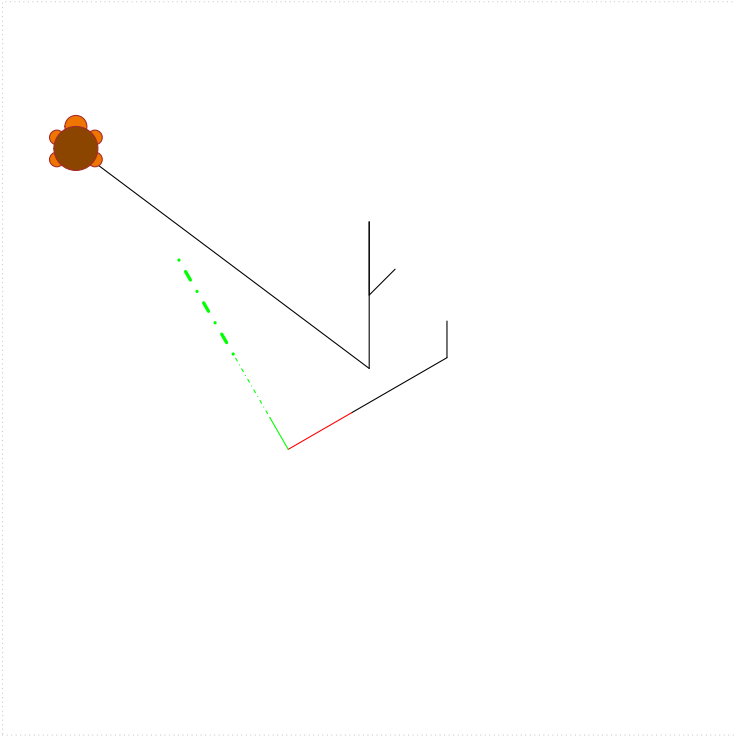
```
$Terrarium
$Terrarium$width
[1] 100

$Terrarium$height
[1] 100


$TurtleStatus
$TurtleStatus$x
[1] 23.95597

$TurtleStatus$y
[1] 64.94523

$TurtleStatus$angle
[1] 330
```

If you just want to know where the Turtle is or at which angle try turtle_getpos() and turtle_getangle() functions respectively.

```
> turtle_getpos()

       x        y
23.95597 64.94523

> turtle_getangle()

angle
  330
```

If you wish to place the Turtle back at the starting position use the turtle_reset() function. The turtle_goto() function on the other hand, makes the Turtle go to the place you tell it to in its arguments passing the x and y coordinates. Mind that this function leaves the trace while using.

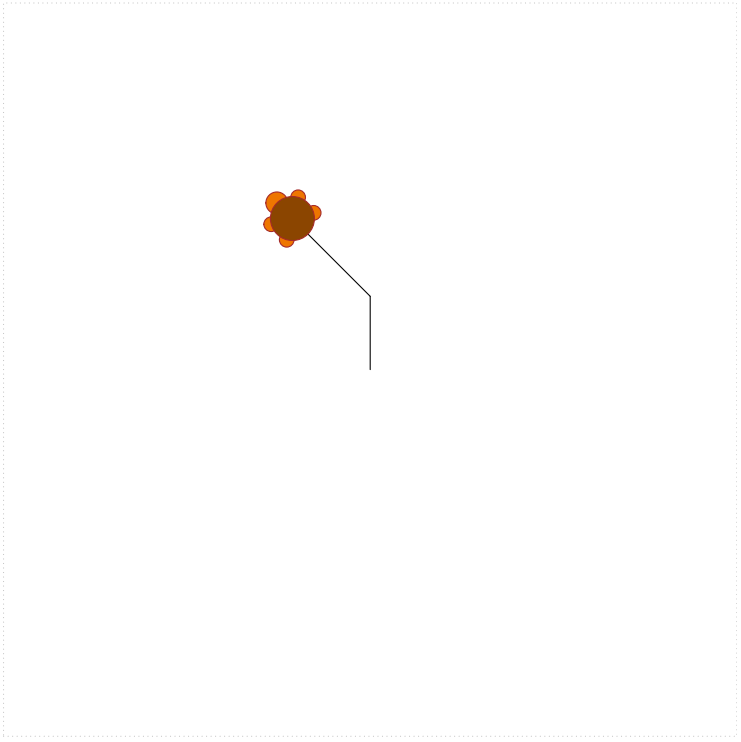
```
> turtle_reset()
> turtle_goto(x=10,y=80)
```



3 Advanced Usage of the Package

Now you know the basics. There some more advanced methods of usage of the package. There is one more function (turtle_do()) in the package that is designed for more complicated sequence of commands. The usage is the following: in the argument you put expr = and between the curly brackets you give the sequence. Each command should be separted by a new line. Let's look at the example:

```
> turtle_init()
> turtle_do(expr = {
+     turtle_move(10)
+     turtle_turn(45)
+     turtle_move(15)
+  })
>
```



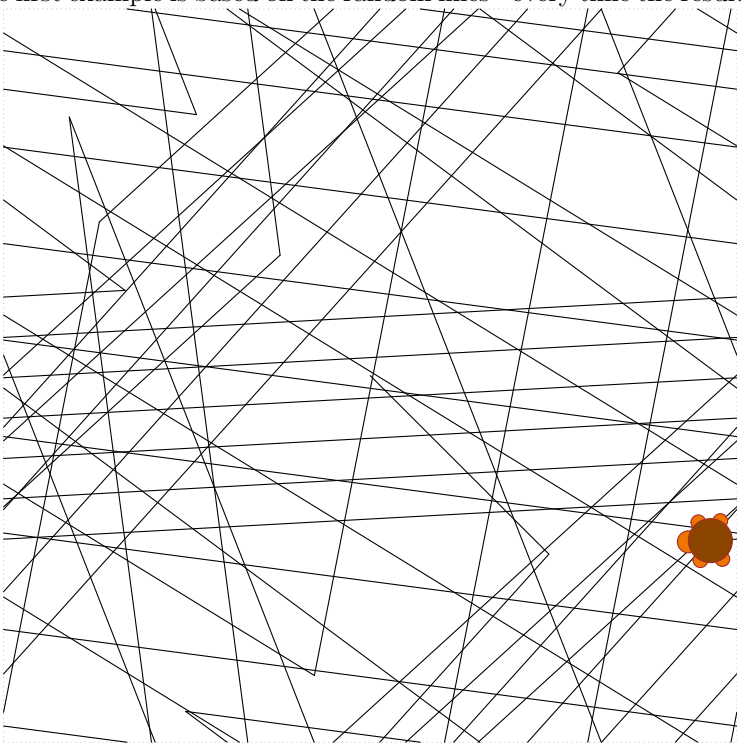
You may ask why bother using such a function if the result is the same as while using three separate commands (in this case this would be turtle_move(10); turtle_turn(45); turtle_move(15)). The thing is that this function hides the Turtle before performing the expression, thus the time that expr use is decreased.

3.1 Examples

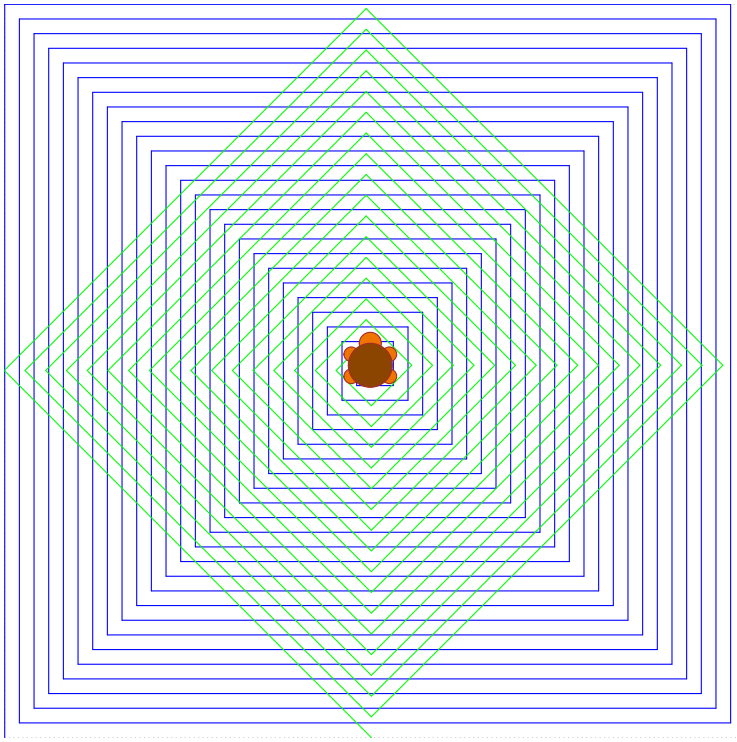
At the end of this guide we would like to present some colorful and inspiring examples. The code is available in 4 and it is not explained – those who are more inquiring will search via Internet how it works.

3.1.1 Random Lines

The first example is based on the random lines - every time the result is different.

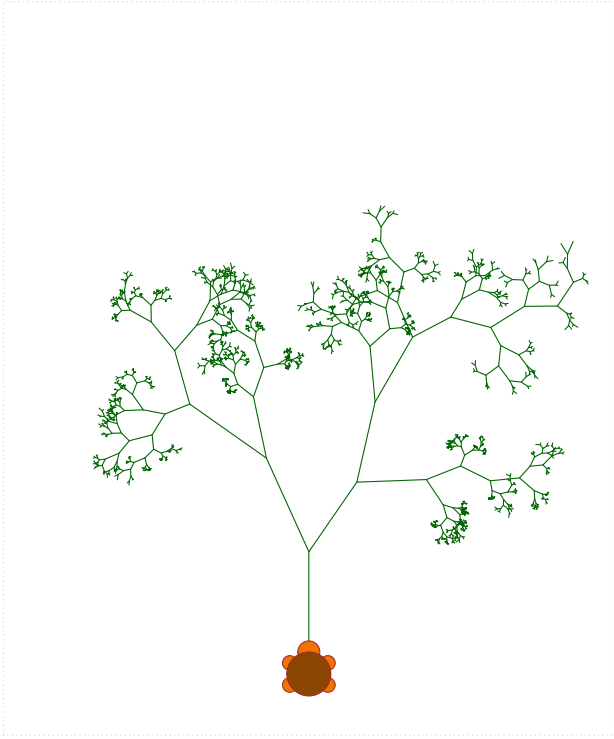


3.1.2 The Spiral

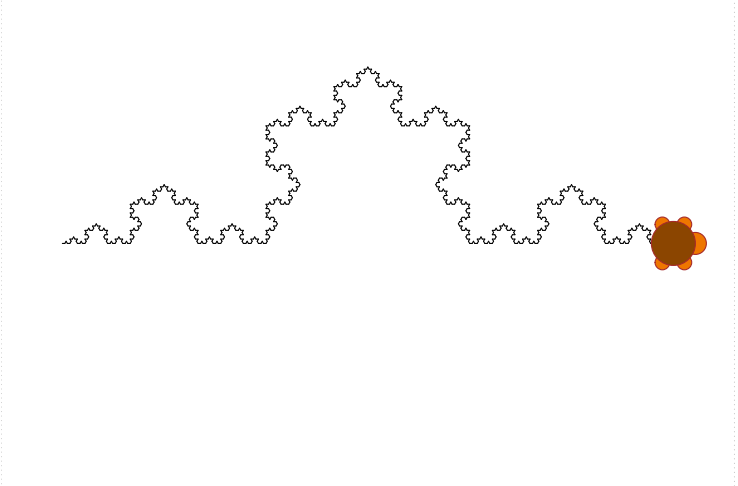


3.1.3 The Fractals

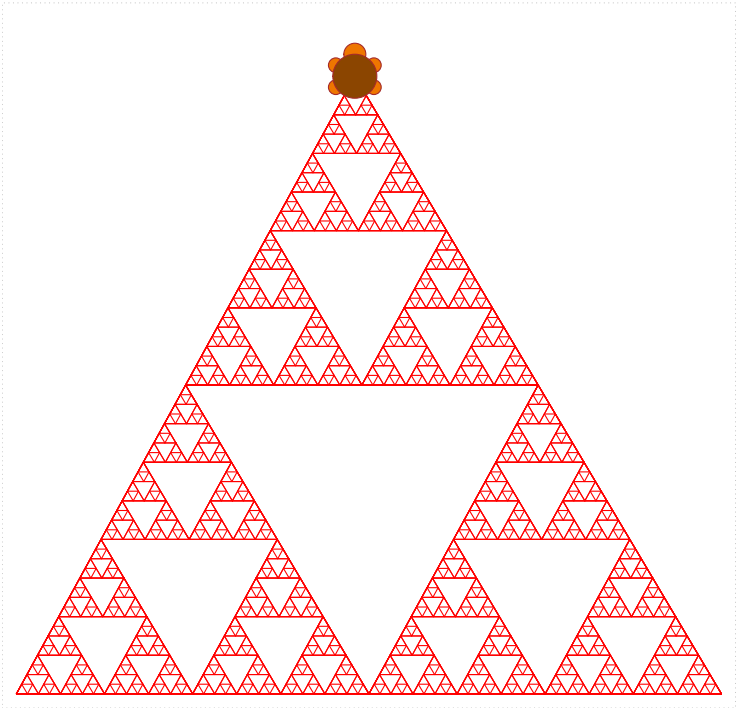
The next three examples are based on fractals - in the assumption they display self-similar pattern.



The Fractal Tree



The Koch Snowflake



The Sierpinski Triangle

3.1.4 Turtle Rainbow Star Rain

4 Source Code

4.0.5 Random Lines

```
> turtle_init(100, 100, mode = "cycle")
> for(i in 1:10){
+   turtle_left(runif(1,0,360))
+   turtle_forward(runif(1, 0, 1000))
+ }
```

4.0.6 The Spiral

```
> drawSpiral <- function(lineLen){
+    if (lineLen > 0){
+       turtle_forward(lineLen)
+       turtle_right(90)
+       drawSpiral(lineLen-5)
+    }
+    invisible(NULL)
+ }
> turtle_init(500, 500, mode="clip")
> turtle_setpos(x=0, y=0)
> turtle_col("blue")
> turtle_do(drawSpiral(500))
> turtle_setpos(x=250, y=0)
> turtle_left(45)
> turtle_col("green")
> turtle_do(drawSpiral(354))
> turtle_setangle(0)
```

4.0.7 The Fractals

The Fractal Tree

```
> fractal_tree <- function(s=100, n=2) {
+    if (n <= 1) {
+       turtle_forward(s)
+       turtle_up()
+       turtle_backward(s)
+       turtle_down()
+    }
+    else {
+       turtle_forward(s)
+
+       a1 <- runif(1, 10, 60)
+       turtle_left(a1)
+       fractal_tree(s*runif(1, 0.25, 1), n-1)
```

```
+       turtle_right(a1)
+
+       a2 <- runif(1, 10, 60)
+       turtle_right(a2)
+       fractal_tree(s*runif(1, 0.25, 1), n-1)
+       turtle_left(a2)
+
+       turtle_up()
+       turtle_backward(s)
+       turtle_down()
+    }
+ }
> set.seed(123)
> turtle_init(500, 600, 'clip')
> turtle_do({
+ turtle_up()
+ turtle_backward(250)
+ turtle_down()
+ turtle_col("darkgreen")
+ fractal_tree(100, 12)
+ })
```

The Koch Snowflake

```
> turtle_init(600, 400, 'error')
> turtle_up()
> turtle_left(90)
> turtle_forward(250)
> turtle_right(180)
> turtle_down()
> koch <- function(s=50, n=6) {
+    if (n <= 1)
+       turtle_forward(s)
+    else {
+       koch(s/3, n-1)
+       turtle_left(60)
+       koch(s/3, n-1)
+       turtle_right(120)
+       koch(s/3, n-1)
+       turtle_left(60)
+       koch(s/3, n-1)
+    }
+ }
> turtle_hide()
> koch(500, 6)
> turtle_show()
```

The Sierpinski Triangle

```
> drawTriangle<- function(points){
+    turtle_setpos(points[1,1],points[1,2])
+    turtle_goto(points[2,1],points[2,2])
+    turtle_goto(points[3,1],points[3,2])
+    turtle_goto(points[1,1],points[1,2])
+ }
> getMid<- function(p1,p2) c((p1[1]+p2[1])/2, c(p1[2]+p2[2])/2)
> sierpinski <- function(points, degree){
+    drawTriangle(points)
+    if (degree > 0){
+       p1 <- matrix(c(points[1,], getMid(points[1,], points[2,]),
+                      getMid(points[1,], points[3,])), nrow=3, byrow=TRUE)
+
+       sierpinski(p1, degree-1)
+       p2 <- matrix(c(points[2,], getMid(points[1,], points[2,]),
+                      getMid(points[2,], points[3,])), nrow=3, byrow=TRUE)
+
+       sierpinski(p2, degree-1)
+       p3 <- matrix(c(points[3,], getMid(points[3,], points[2,]),
+                      getMid(points[1,], points[3,])), nrow=3, byrow=TRUE)
+       sierpinski(p3, degree-1)
+    }
+    invisible(NULL)
+ }
> turtle_init(520, 500, "clip")
> p <- matrix(c(10, 10, 510, 10, 250, 448), nrow=3, byrow=TRUE)
> turtle_col("red")
> turtle_do(sierpinski(p, 6))
> turtle_setpos(250, 448)
```

4.0.8 Turtle Rainbow Star Rain