Lesson #2: Turtle Graphics Interpretation of L-systems

What's It All About?

One popular means by which to render L-systems as fractal images involves **turtle graphics**. This lesson introduces turtle graphics as a visible manifestation of **turtle geometry**, a decidely procedural variant of the much more declaratively flavored **Euclidian Geometry**.

Introduction to Turtle Graphics

For a quick introduction to turtle graphics, please find your way to the Wikipedia page on turtle graphics and spend just a few minutes reading and studying the content of the page.

`https://en.wikipedia.org/wiki/Turtle_graphics`

The short story on Turtle Graphics and L-Systems

As previously mentioned, L-Systems can be rendered graphically by means of turtle graphics. Turtle graphics is a drawing system in which a creature/cursor, known as the turtle, productively maneuvers in a subset of the Cartesian plane, known as the canvas, according to well defined opeartions.

Informally, the turtle can move from one point to another, either leaving a trace as it goes, or not. Turtle can also turn to its right or to its left some number of degrees. The turtle even has a bit of stack like memory that can be relied upon for doing a certain sort of processing.

Some Turtle Graphics Commands

The following primitives, and variants of them, are generally featured in the rendering of L-Systems graphically:

1. move/draw forward some distance
2. move/draw backward some distance
3. turn right D degrees
4. turn left D degrees
5. push the state (location and heading) of the turtle onto a stack
6. pop the state (location and heading) of the turtle off of the stack

As we engage in some computations involving the rendering of various L-systems with turtle graphics, it will be useful to have a vocabulary of one letter commands corresponding to specific turtle commands. This vocabulary will serve nicely to render a number of classic L-systems:

1. F - draw forward some distance
2. M - move (but do not draw) forward some distance
3. L - turn left 90 degrees
4. R - turn right 90 degrees
5. < - turn left 120 degrees
6. > - turn right 120 degrees
7. (- turn left 30 degrees
8.) - turn right 30 degrees
9. P - push the state (location and heading) of the turtle onto a stack
10. Q - pop the state (location and heading) of the turtle off of the stack

Playing Turtle: Two Programming Exercises

Please write the following two turtle graphics programs:

1. Imagine that the turtle is in the center of its canvas facing **east**. Then, please specify a sequence of commands according to which the turtle would render a square.
2. Again, imagine that the turtle is in the center of its canvas facing **east**. This time, specify a sequence of commands by which the turtle would render a portable staircase consisting of three stairs.

Playing Turtle: Exercise K

Grab a paper and pencil. Place the pencil in the center of the paper, imagine the point is the position of the turtle, and imagine that the turtle is facing **east**. Then, simply draw a "figure" by executing the following sequence of commands:

```
F L F R F R F L F
```

Hang on to your drawing

Playing Turtle: Exercise S

This exercise may challenge your spatial intuitions a bit more than the previous exercise did. Regardless, grab a paper and pencil. Place the pencil in the center of the paper, imagine the point is the position of the turtle, and imagine that the turtle is facing **east**. Then, simply draw a "figure" by executing the following sequence of commands:

```
F > F < F < F > F > F F > F F
```

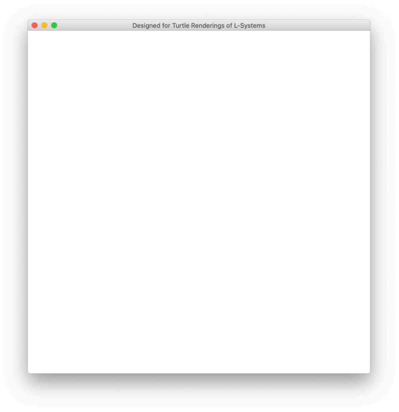
Hang on to your drawing

TGR: Turtle Graphics Renderer

The “Turtle Graphics Renderer” is a simple little Java program that can be used to render output from certain L-systems according to a specified vocabulary of turtle graphic commands. It is intended more for L-systems play than for high powered application.

You can download a copy of this program at this address:
`https://www.cs.oswego.edu/~blue/software/`

When you run the program, a graphics window like the following is presented:



The following demo shows how to run the program, and also provides an idea of what the program is capable of doing. The `help` command reveals the basic system commands. The output from `tgcommand` presents the turtle grapics commands that are available.

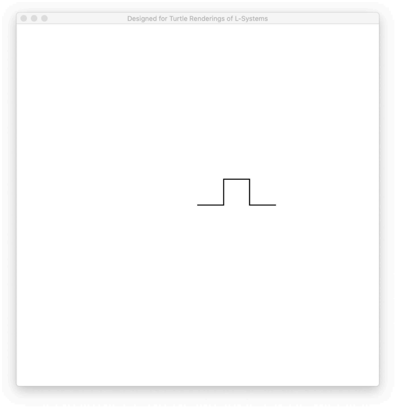
```
bash-3.2$ ls
README.TXT lib
TurtleGraphicsRenderer.jar

bash-3.2$ java -jar TurtleGraphicsRenderer.jar

tgr> help
HELP - Specify the available commands.
TGCOMMANDS - Specify the available turtle graphics commands.
EXIT - Terminate execution of the program.
RENDER U TGCS - Render the turtle graphics command sequence TGSC assuming unit U
CLEAN - Wash the screen, and move the painter to the center facing east.
LEFT - Move the painter close to the left of the canvas.
BOTTOMLEFT - Move the painter close to the bottm left of the canvas.
TOPLEFT - Move the painter close to the top left of the canvas.
FAST - No waiting between execution of turtle commands.
MEDIUM - Wait 300 ms between execution of turtle commands.
SLOW - Wait 1000 ms between execution of turtle commands.

tgr> tgcommands
F - Move forward one unit, drawing as you go.
M - Move forward one unit, without drawing as you go.
L - Turn left 90 degrees.
```

```
R - Turn right 90 degrees.
< - Turn left 120 degrees.
> - Turn right 120 degrees.
( - Turn left 30 degrees.
) - Turn right 30 degrees.
P - Push the turtle's state, it's position and heading, onto the turtle stack.
Q - Pop the turtle's state from the turtle stack, reseting position and heading.

tgr> exit
Thank you for using the turtle graphics renderer!

bash-3.2$
```

Exercise K Revisited

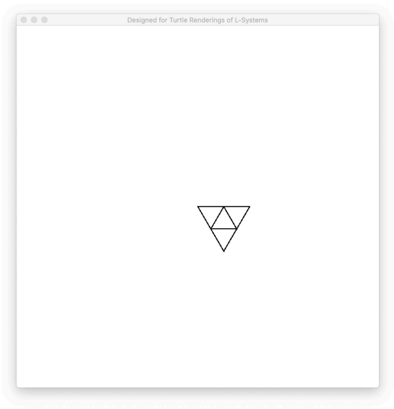
The thinking behind ...

```
tgr> clean
tgr> slow
tgr> render 50 F L F R F R F L F
tgr>
```



Exercise S Revisited

```
tgr> clean
tgr> slow
tgr> render 50 F > F < F < F > F > F F > F F
tgr>
```



How do you generate a desired sequence of tokens with an L-system?

There are two basic ways to generate a desired sequence of tokens with an L-system:

1. Select a vocabulary symbols for the L-system which are consistent with the desired tokens, and then evolve.
2. Evolve, with whatever vocabulary symbols you might be using in your L-system, and then perform a mappying from some selected generation to the desired tokens.

The sequence from Exercise K (`F L F R F R F L F`) was established by means of the first approach. The sequence from Exercise S (`F > F < F < F > F > F F > F F`) was established by means of the second approach, as will be seen shortly.

Note: The second approach provides considerable flexibility!

Example L-system: The Sierpinsky Triangle

1. Alphabet: {A,B,C,D}
2. Start string: A D B D B
3. Productions
   (a) A → A D B C A C B D A
   (b) B → B B
   (c) C → C
   (d) D → D

The first two generations of the Sierpinski triangle:

- G0: A D B D B

- G1: A D B C A C B D A D B B D B B

Turtle Graphic Interpretation of the Sierpinsky Triangle

Note that the symbols used in the conception of the Sierpinski Triangle L-system might be approapriate to playing musical notes, but they don't correspond to turtle graphics commands. If we want to render Sierpinski triangles by using our turtle graphics commands, we will ahve to perform a mapping.

The standard Turtle graphic **interpretation** for the Sierpinsky triangle L-system (with the chosen vocabulary) is this:

- A $\rightarrow$ F
- B $\rightarrow$ F
- C $\rightarrow$ <
- D $\rightarrow$ >

According to this mapping, the first two generations of the Sierpinski triangle under the turtle graphics intepretation:

- G0: F > F > F
- G1: F > F < F < F > F > F F > F F

LSG: The L-System Generator

We will make use of a little program called the "L-System Generator", or "LSG", to help us with evolving L-systems and mapping generaltional output to sequences of tokens that can be interpreted musically or graphically. This program operates on a small number of "canned" L-systems, the alphabets of which all correspond to small natural numbers. Moreover, there are maps for transforming the natural numbers in a generation into JFugue sequences, or turtle graphics sequences, or certain other sequences.

The following run of the program suggests the functionality that is available, the system level functionality, and also the turtle graphics commands:

```
bash-3.2$ ./lsg

lsg> help
help | systems | maps | display_system S | display_map M | generate N S | render N S M | exit

lsg> systems
m34
algae
cantor
sierpinski
koch

lsg> maps
sjp_algae_map ......... maps to A and B (the classic algae system map)
sjp_map_1 ............. maps to three simple JFugue notes (SJP)
sjp_glass_map_2 ....... maps to two JFugue figures of Phillp Glass's composition Two Pages (SJP)
sjp_glass_map_3 ....... maps to three JFugue figures of Phillp Glass's composition Two Pages (SJP)
mxm_map_1 ............. maps to three simple MxM note sequences (MXM)
```

```
tgr_koch_map .......... maps to turtle graphics commands to render the Koch curve (TGR)
tgr_sierpinski_map .... maps to turtle graphics commands to render the Sierpinski triangle (TGR)

lsg> exit
Terminating session ...
bash-3.2$
```

Example: LSG and Algae

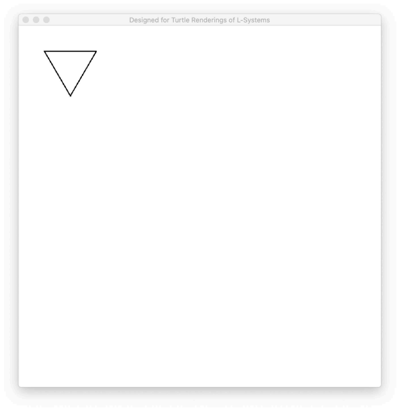
```
bash-3.2$ ./lsg

lsg> help
help | systems | maps | display_system S | display_map M | generate N S | render N S M | exit

lsg> display_system algae
Alphabet = {1,2}
Start string = [1]
Productions:
1 --> [1,2]
2 --> [1]

lsg> display_map sjp_algae_map
1 --> A
2 --> B

lsg> generate 6 algae
[1,2,1,1,2,1,2,1,1,2,1,1,2,1,2,1,1,2,1,2,1]

lsg> render 6 algae sjp_algae_map
A B A A B A B A A B A A B A B A A B A B A

lsg> exit
Terminating session ...
bash-3.2$
```

Example: LSG and the Sierpinski Triangle

```
bash-3.2$ ./lsg

lsg> help
help | systems | maps | display_system S | display_map M | generate N S | render N S M | exit

lsg> generate 0 sierpinski
[1,4,2,4,2]
lsg> render 0 sierpinski tgr_sierpinski_map
F > F > F

lsg> generate 1 sierpinski
[1,4,2,3,1,3,2,4,1,4,2,2,4,2,2]
lsg> render 1 sierpinski tgr_sierpinski_map
F > F < F < F > F > F F > F F

lsg> generate 2 sierpinski
[1,4,2,3,1,3,2,4,1,4,2,2,3,1,4,2,3,1,3,2,4,1,3,2,2,
4,1,4,2,3,1,3,2,4,1,4,2,2,2,2,4,2,2,2,2]
lsg> render 2 sierpinski tgr_sierpinski_map
F > F < F < F > F > F F < F > F < F < F > F < F F
> F > F < F < F > F > F F F F > F F F F

lsg> generate 3 sierpinski
[1,4,2,3,1,3,2,4,1,4,2,2,3,1,4,2,3,1,3,2,4,1,3,2,2,
4,1,4,2,3,1,3,2,4,1,4,2,2,2,2,3,1,4,2,3,1,3,2,4,1,4,
2,2,3,1,4,2,3,1,3,2,4,1,3,2,2,4,1,4,2,3,1,3,2,4,1,3,
2,2,2,2,4,1,4,2,3,1,3,2,4,1,4,2,2,3,1,4,2,3,1,3,2,4,
1,3,2,2,4,1,4,2,3,1,3,2,4,1,4,2,2,2,2,2,2,2,2,4,2,2,
2,2,2,2,2,2]
lsg> render 3 sierpinski tgr_sierpinski_map
F > F < F < F > F > F F < F > F < F < F > F < F F > F
> F < F < F > F > F F F F < F > F < F < F > F > F F <
F > F < F < F > F < F F > F > F < F < F > F < F F F F
> F > F < F < F > F > F F < F > F < F < F > F < F F >
F > F < F < F > F > F F F F F F F F > F F F F F F F F

lsg>
```

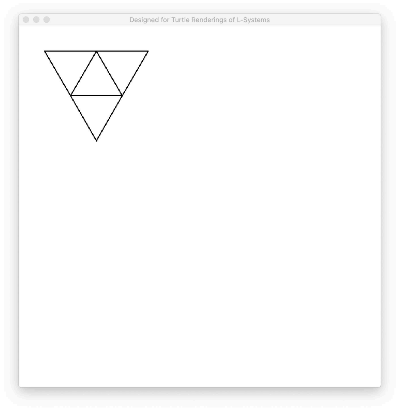
Rendering the Sierpinsky Triangle of Order 0

```
tgr> topleft
tgr> slow
tgr> render 100 F > F > F
tgr>
```



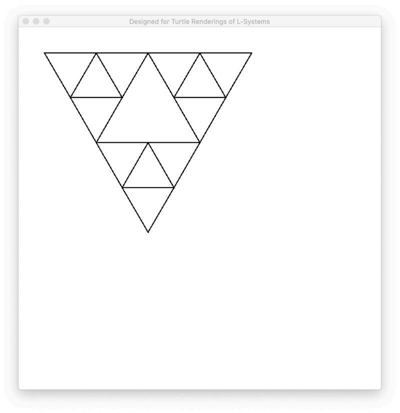
Rendering the Sierpinsky Triangle of Order 1

```
tgr> topleft
tgr> slow
tgr> render 100 F > F < F < F > F > F F > F F
tgr>
```



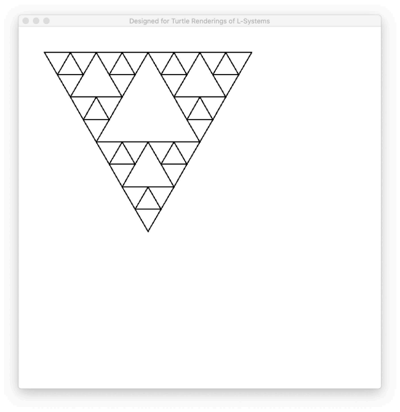
Rendering the Sierpinsky Triangle of Order 2

```
tgr> topleft
tgr> medium
tgr> render 100 F > F < F < F > F > F F < F > F < F < F > F < F F
> F > F < F < F > F > F F F F > F F F F
tgr>
```



Rendering the Sierpinsky Triangle of Order 3

```
tgr> topleft
tgr> medium
tgr> render 50 F > F < F < F > F > F F < F > F < F < F > F < F F
> F > F < F < F > F > F F F F < F > F < F < F > F > F F < F > F
< F < F > F < F F > F > F < F < F > F < F F F F > F > F < F < F
> F > F F < F > F < F < F > F < F F > F > F < F < F > F > F F F
F F F F F > F F F F F F F F
tgr>
```



Resources for the Turtle Graphics Rendering of L-systems

1. Silent film:
   `https://www.youtube.com/watch?v=LBIUagwiRMM`
2. DYI web program:
   `https://www.kevs3d.co.uk/dev/lsystems/`