Generative Art for All

Michael Tempel
michaelt@media.mit.edu

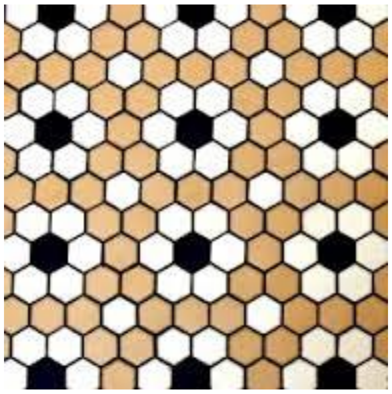
Generative Art for All

Michael Tempel
michaelt@media.mit.edu

Generative art[1] refers to art that is created by a system that operates autonomously. The artist may create the system, and/or set some parameters that affect the outcome, but the result is created, at least in part, by the system rather than directly by the artist. Generative art systems are frequently computer programs, although biological, social, or other systems may also be used to generate art.

Art that is created by using a computer is not necessarily generative. If one is using a paint or drawing application to create an image, the computer is a tool - much like a pencil or paint brush - that is controlled directly by the artist. The application is not acting autonomously.

A related concept is algorithmic art, which may be considered as one type of generative art. It generally refers to art that is created via an algorithm, implemented as a computer program, determining the outcome. But artwork involving symmetry and pattern may be implicitly algorithmic regardless of how it is created.[2] The artist is following a step by step sequence of rules, even if the algorithm is not spelled out explicitly. In this sense algorithmic art has existed for thousands of years. For example, consider how you would create this floor tile pattern from individual white, black, and brown hexagonal tiles:



You might follow this algorithm:

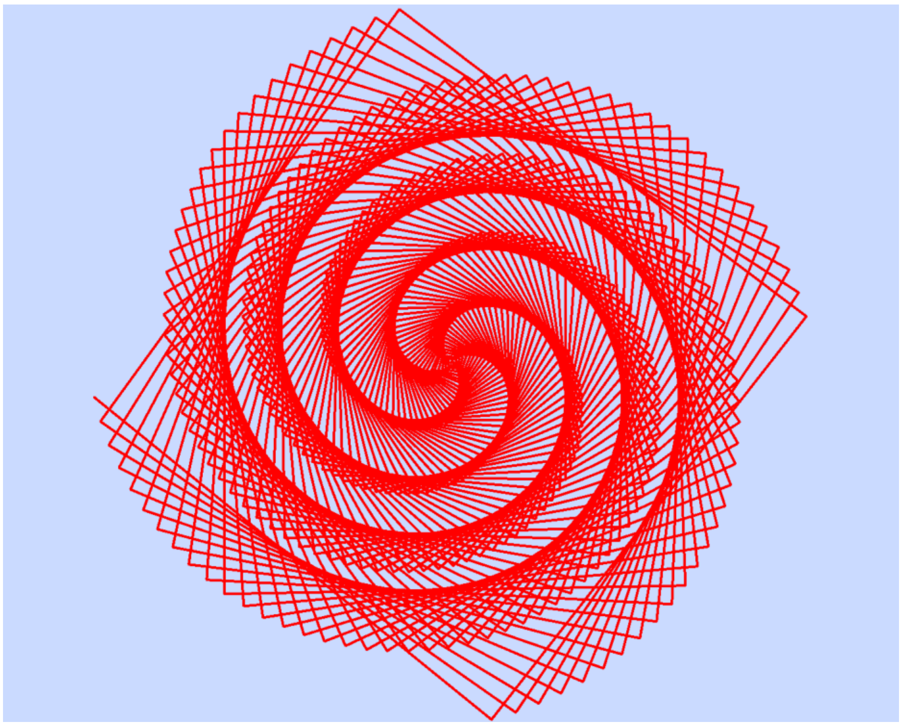
1. Place a black tile in the middle of the floor.
2. Surround the black tile with white tiles.
3. Surround the white tiles with brown tiles.
4. etc.

In practice you probably would not think the steps through explicitly, but would simply look at a picture of the desired result and proceed to lay out the tiles to duplicate the picture. You would implicitly be following an algorithm, either the one above or any one of a number of other possible algorithms that would generate the same pattern.

In reality, such a floor is generally not assembled out of individual tiles. Instead, the tiles come in sheets that are about one foot square. The tiles, with the pattern in place, are attached to a flexible mesh. These sheets are then put together to form the floor. The sheets are manufactured by a machine that is programmed to produce the pattern. In this case, the algorithm must be made explicit so as to be able instruct the machine.

Even in algorithmic art there is room for uncertainty and surprises. A musical score is an algorithm that specifies how a piece should be played. Yet different performances of the same piece vary in ways that are noticeable to listeners. This is in part due to performers not following the score exactly, but there are also aspects of performance that are not captured in the written score and come from the artist.

An algorithm may determine a result very precisely, but there may still be surprises. This image, drawn by a program written in TurtleArt, which we will elaborate on below, includes only straight lines. There are no curves.[3]



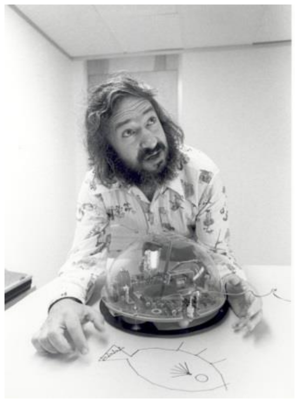
Generative art often has an element of uncertainty. This may be due to the inclusion of randomness in the algorithm used to produce the result, but can also be the result of the unpredictable nature of some parameter such as the number of people viewing the artwork, or the price of crude oil. Generative art systems may be very complex. For example, Electric Sheep[4] is running on thousands of computers generating animations, or "sheep" that morph and reproduce based on algorithms, but also affected by the popularity of individual sheep among members the worldwide community of users.

A challenge for educators is to make a generative art experience available to young students and to people with limited technical expertise. Can we construct accessible creative environments that bring the learner- artist in touch with the concepts around generative art? Such an introduction should lay the groundwork for people to move to mainstream systems more comfortably if they choose to do so. An example is TurtleArt[5]

Long Live the Turtle

TurtleArt is a contemporary version of the Logo programming language with a Logo-speaking Turtle as its starring character. Logo development began a half century ago and there have been more than 300 versions implemented to date.[6] Most of these include Turtle Geometry, which was conceived of as a form of geometry that was more accessible to young learners than the traditional school geometries of Euclid and Descartes.[7]

The Turtle was originally a robotic creature that travelled around on the floor in response to commands from a computer to move forward and back, and to turn to the right or left. It had a pen, positioned vertically in its center, which when down, allowed the Turtle to draw as it moved. By the early 1970s the turtle had migrated to the computer screen. Looking at a screen turtle is like having a view from above of a robot floor turtle.


Seymour Papert with a robotic turtle, c.1972

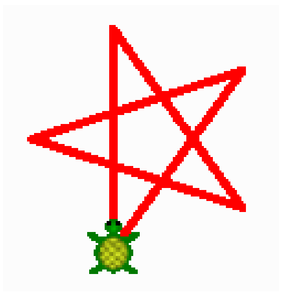


image drawn by a screen turtle

The goal for most Logo users has been to learn programming and mathematics, usually with visual results. But there has also been a long tradition of Logo programming where creating a visual effect is the goal and programming is the means.

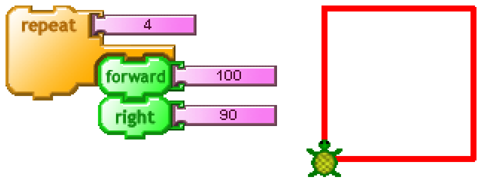
With TurtleArt the aim is to create visual art, specifically drawings that may be viewed on the computer screen or printed, framed, and hung on the wall. Programming and Turtle Geometry are the means by which those drawing are created.

TurtleArt is entirely algorithmic. The only way to generate an image on the screen is by writing and executing a program. This is in contrast to paint and draw applications in which images are created and manipulated directly using a mouse or touchpad. There are also environments, Scratch[8] and MicroWorlds,[9] for example, which incorporate both approaches.

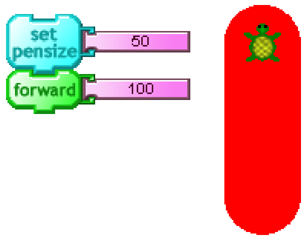
TurtleArt uses Blocks Programming[10] in which programs are created by snapping together graphical pieces on the screen, like putting together a jigsaw puzzle. The function of each block and the overall structure of the program are expressed by the shapes and colors of the blocks, and how they fit together. These visual clues help make a program more understandable.

Another advantage of this style of programming, which is especially important for beginners, is that in contrast to conventional text-based languages, it is impossible to make syntax errors. Over the past few years Blocks Programming has become widely used, driven largely by the popularity of Scratch.

The TurtleArt snippet of code to the right draws a square 100 pixels on a side.

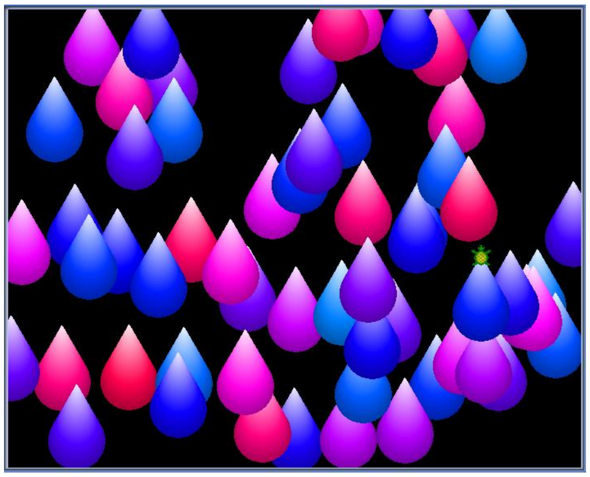


The size of the turtle's pen may be specified. The square is drawn with the pen set to its default size of four pixels. The line to the right is drawn with the pen set to a width of 50 pixels.

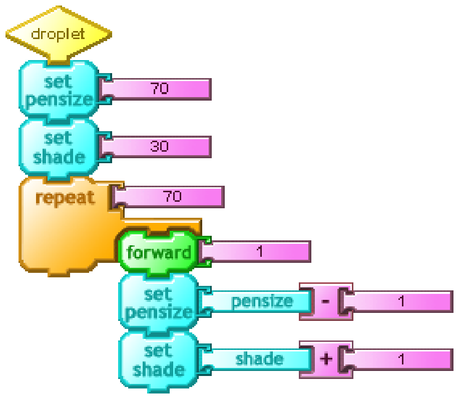


One can also set the pen's color, with 0 being red, 90 purple, and the rest of the rainbow in between. Shade may be set as well, with a number ranging from 0 to 100. Lower numbers are very dark and higher numbers are pastel shades. The default value is 50.

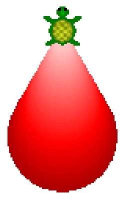
By manipulating these three parameters - pen size, color, and shade - a wide variety of visual effects can be achieved. A TurtleArt program may be entirely deterministic. The image that is created is the same every time the program is run. But there may be an element of randomness in the program so that the image is generated somewhat differently each time. Here's an example called Titanic Rain:[11] The droplets are all the same size and shape, but vary in color.



Here is the code that draws a single droplet:

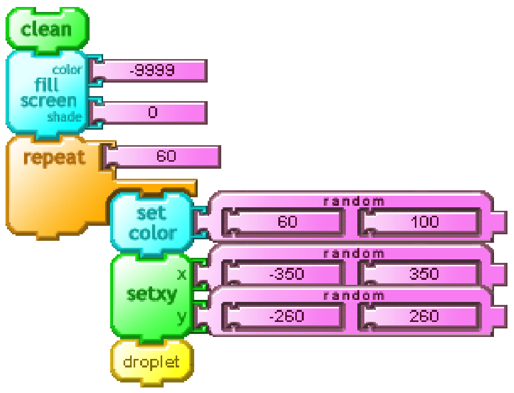


```
droplet
set pensize 70
set shade 30
repeat 70
    forward 1
    set pensize pensize - 1
    set shade shade + 1
```



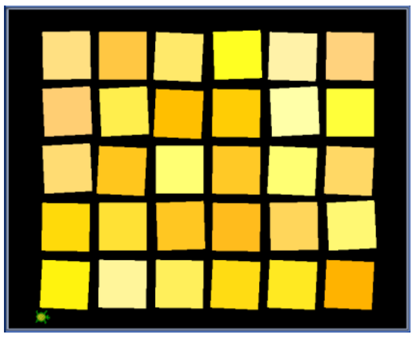
The pen size is set to 70, which is fairly wide. The shade is set to 30, which is a bit on the dark side. Then the turtle takes 70 steps forward. With each step, the pen size is narrowed by one pixel and the shade is lightened. The result is that the droplet starts out fat and dark, but lightens and tapers to a point as the turtle move up the canvass.

So far there is no randomness in the algorithm. But here's the code that creates the full image. First the screen is filled with black. Then 60 droplets are drawn. The color of each is set randomly in a range of blue through purple to red. The position of each droplet is set to a random XY coordinate.[12] Running the program multiple times will produce similar, but different images. There will be variation in the placement of the droplets, and some results will tend more to the red, while others further toward blue or purple.



```
clean
fill screen color -9999
            shade 0
repeat 60
    set color random 60 100
    setxy x random -350 350
          y random -260 260
    droplet
```

In this next example, called Patio, we see a regular pattern of five rows of six squares. There is no randomness in this arrangement on the XY grid. There is some randomness in the elements, with slight variations in color and shade. And some squares are lined up with the edges of the canvass, but most are rotated a bit clockwise or counterclockwise.



These examples bring us back to the earlier discussion about different styles of geometry. The turtle is egocentric. It moves and turns relative to where it is and which way it's pointing. It doesn't know or care about the larger context. The floor turtle can be put in rooms of different sizes. It won't know where the walls are unless it bumps into one. But when the turtle lives on a computer screen we can specify a position using X and Y coordinates. TurtleArt includes Cartesian Geometry as well as Turtle geometry.

In both Titanic Rain and Patio, the elements of the composition - droplets or squares – are created using Turtle geometry. This allows the element to be drawn at different locations using the same code. The placement of the elements on the canvas is done using Cartesian Geometry.

Turtle Art in Context

We have seen how TurtleArt is a programming environment designed for artistic expression, specifically to create drawings, and for exploring generative art. As a contemporary instance of Logo it is also a vehicle for exploring and learning mathematics and programming. A guiding principle in the decades of Logo development has been to create programming environments that have a "low threshold and high ceiling". A beginner should be able to enter easily without obstacles or a big step up. But the language should also allow for sophisticated programming.

With the development of Scratch beginning in the mid-2000s, an additional design criteria was emphasized, that of "wide walls." People with differing interests should all be able to express themselves by creating projects in various domains – art, music, mathematics, science, storytelling, games, and animations.[13]

How does TurtleArt fit into this framework? It has a very low threshold, arguably lower than that of Scratch because of its simplicity. But as a programming language it is limited, so the ceiling is also low. And the walls are extremely narrow, focusing on a specific area of artistic expression. Brian Silverman, one of the developers of TurtleArt argues that this narrowness encourages "going deep" into the domain of algorithmic drawing. In part this is a result of the highly focused environment and lack of distractions. But there are also details in the design of TurtleArt that make it especially well suited to the domain.[14]

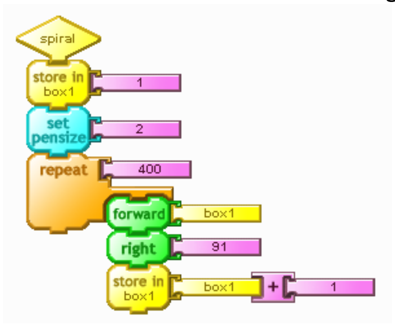
One can look at the low threshold, high ceiling goal in different ways. An environment can be deigned to be broadly inclusive, like Scratch. But one can also imagine a family of programming languages, with different dialects and vocabularies, but enough similarity between them so as to make it easy for people to move from one to the other.

One would choose an appropriate environment for the exploration or project at hand. For TurtleArt, that domain is generative art, specifically drawing, at the intersection of art, mathematics, and computing.

[1] For an overview of generative art see *What is Generative Art?* By Phillip Galanter http://www.philipgalanter.com/downloads/ga2003_paper.pdf

[2] Art that includes mathematical specification may or may not be algorithmic. For example, in classical Greek architecture, the proportions of the components of columns are precisely spelled out for each order, but this does not provide a step by step procedure for building the column. Weaving is an example of algorithmic art that has existed for thousands of years and has been automated for more than two centuries, initially with the Jacquard loom and its predecessors.

[3] This is the code that drew the image:



```
spiral
store in box1  1
set pensize  2
repeat  400
    forward  box1
    right  91
    store in box1  box1 + 1
```

[4] http://www.electricsheep.org/

[5] TurtleArt is a product of the Playful Invention Company. The software, galleries of images, and tutorials are available from www.turtleart.org

[6] See *What is Logo* http://el.media.mit.edu/logo-foundation/what_is_logo/. Pavel Boytchev has compiled a list of past and current versions of Logo as the *Logo Tree Project* http://www.elica.net/download/papers/LogoTreeProject.pdf

[7] See chapter 3 of *Mindstorms* by Seymour Papert (1980 Basic Books)

[8] http://scratch.mit.edu

[9] www.microworlds.com

[10] A brief overview of this style of visual programming, and an explanation for its rationale, is *Blocks Programming* by Michael Tempel, http://el.media.mit.edu/logo-foundation/resources/papers/pdf/blocks_programming.pdf and in *CSTA Voice*, Vol. 9 No. 1, March 2013

[11] This example and the subsequent one, Patio, were created by Brian Silverman. These drawings and many more may be found in the galleries on the TurtleArt website - www.turtleart.org

[12] The XY coordinates [0 0] are at the center of the canvas. X values are in the range of -350 to 350. Y values range from -260 to 260.

[13] For a discussion of these issues and more see *Some Reflections on Designing Construction Kits for Kids* by Mitchel Resnick and Brian Silverman http://web.media.mit.edu/~mres/papers/IDC-2005.pdf

[14] For example, the floor tile pattern that appears earlier in this article can be drawn with TurtleArt as a pattern of repeated hexagons. As with any infinite tile pattern, when a border is placed around a portion of the design, some of the elements may appear only partially. TurtleArt allows one to draw a hexagon near the edge of the canvass with the turtle leaving the screen and returning as if the non-visible part of the shape is drawn beyond the border. The hexagon pattern could also be drawn in Scratch, but the turtle cannot leave the stage, so the tile pattern is disturbed at the edges. This is not a defect in Scratch, but rather a reflection of design criteria that are better suited to animation and game projects.