www.logofoundation.org

Balance

By Michael Tempel

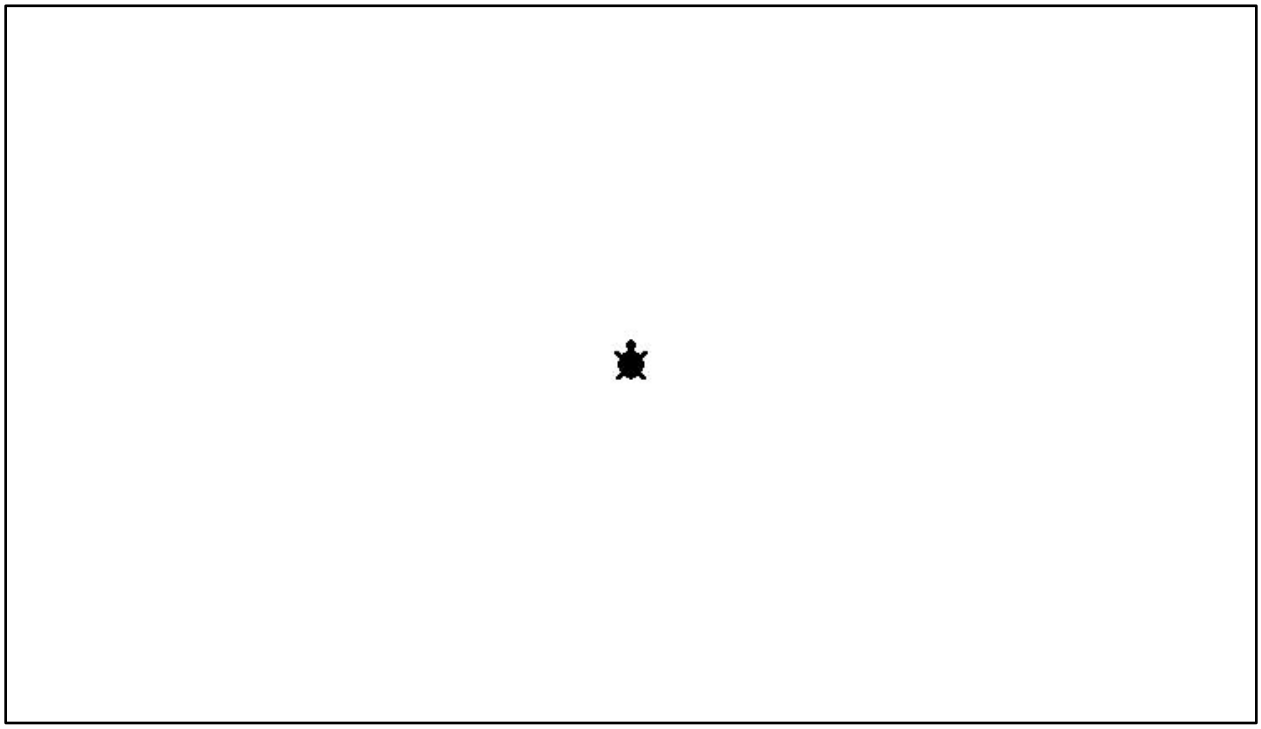
Introduction

Systems Thinking is a perspective in which one looks at the relationships and interconnections among the components of a complex whole – a system. It may apply to organizations, ecologies, and machines.

An important component of many systems is feedback, especially what is called negative-feedback or balancing-feedback. A familiar illustration is the way in which an air conditioner works: when the thermostat senses that the temperature has reached a certain elevation, the unit turns on the cool the room. When the temperature lowers to a specified point, it turns off, so the temperature rises. When it reaches the specified higher limit, it turns off, and so on. A balance is achieved where the temperature remains within a limited range. Other examples of balancing feedback include predator-prey interactions, supply and demand in markets and governors that regulate the speed of engines.

The turtle graphics component of a Logo programming environment provides a laboratory for exploring balancing feedback. Unlike real-world systems, which are very complex, the turtle system allows us to build a streamlined, simplified microworld in which to explore the phenomenon of balancing feedback.

Let's start with the turtle on a white screen…



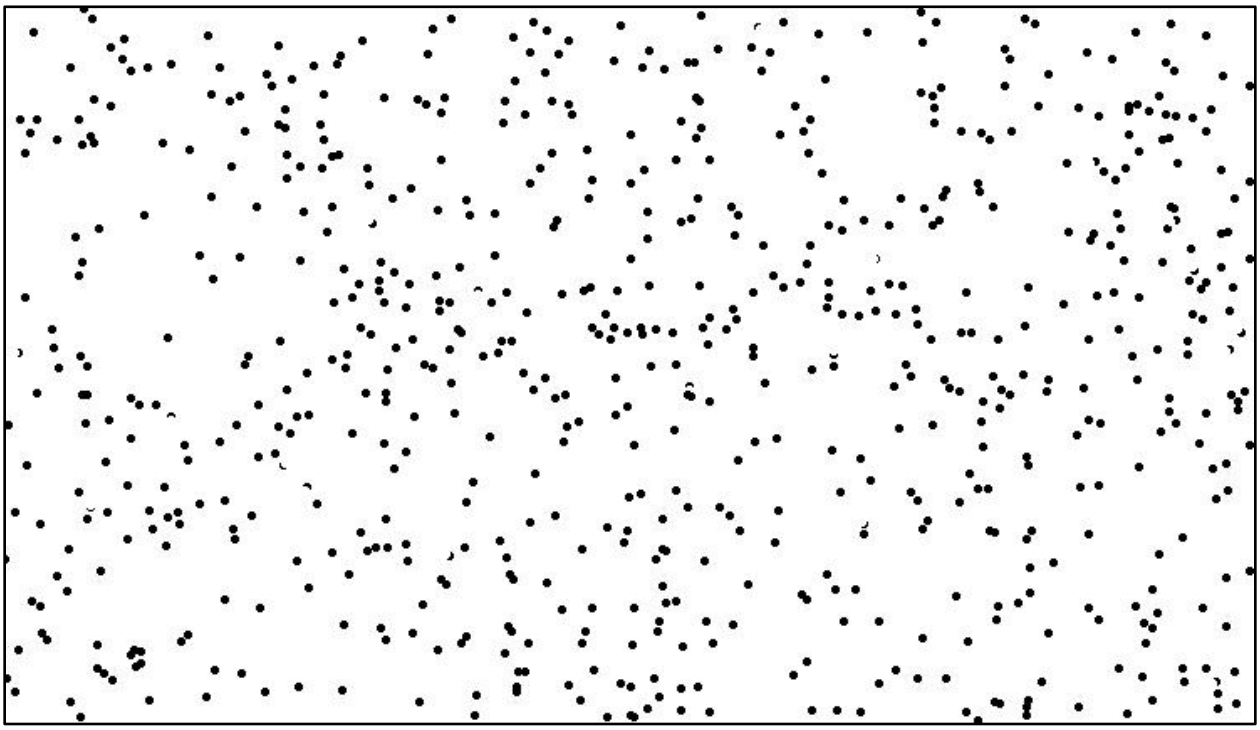
…and this Logo program…

```
to go
go-to-some-random-place
ifelse no-dot?
        [make-a-dot]
        [erase-a-dot]
end
```

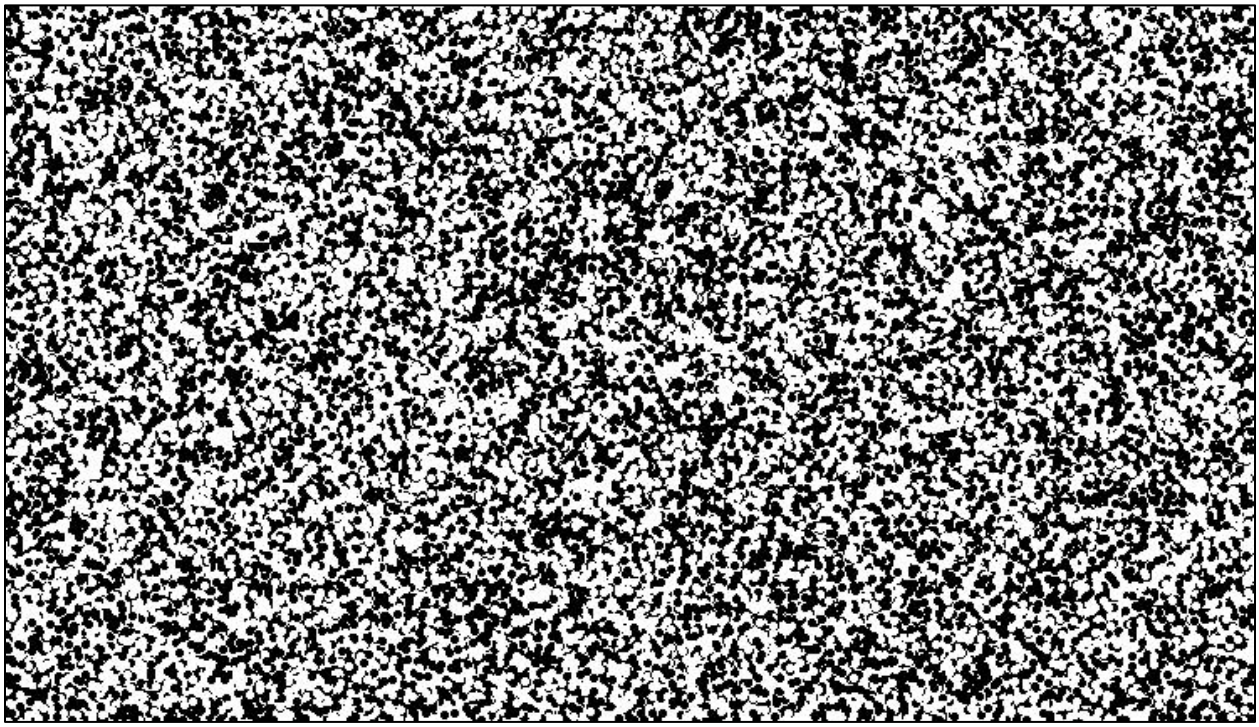
…which tells the turtle to go somewhere on the screen at random. Then, depending on what it finds, it does one of two things. If it happens to land on a black dot, it erases it. If it lands on a white space, it draws a dot.

We get things going with the instruction **forever [go]**. This repeats **go** endlessly until we stop it.

After a little while the screen might look like this:



What will happen if we let this program run for a longer time? In a few minutes the screen will look something like this:



If you focus your attention on any small area of the screen you will see it change occasionally, but the overall pattern remains pretty much the same. The system is stable.

Why?

The screen starts out white, that is, with no dots on it. On the first **go**, the turtle is certain to land on a white spot and make a dot. On the second **go**, it is possible, but very unlikely, that the turtle will land on the dot that was made on the first **go** and erase it. Most likely it will again land on a white spot and draw a second dot.

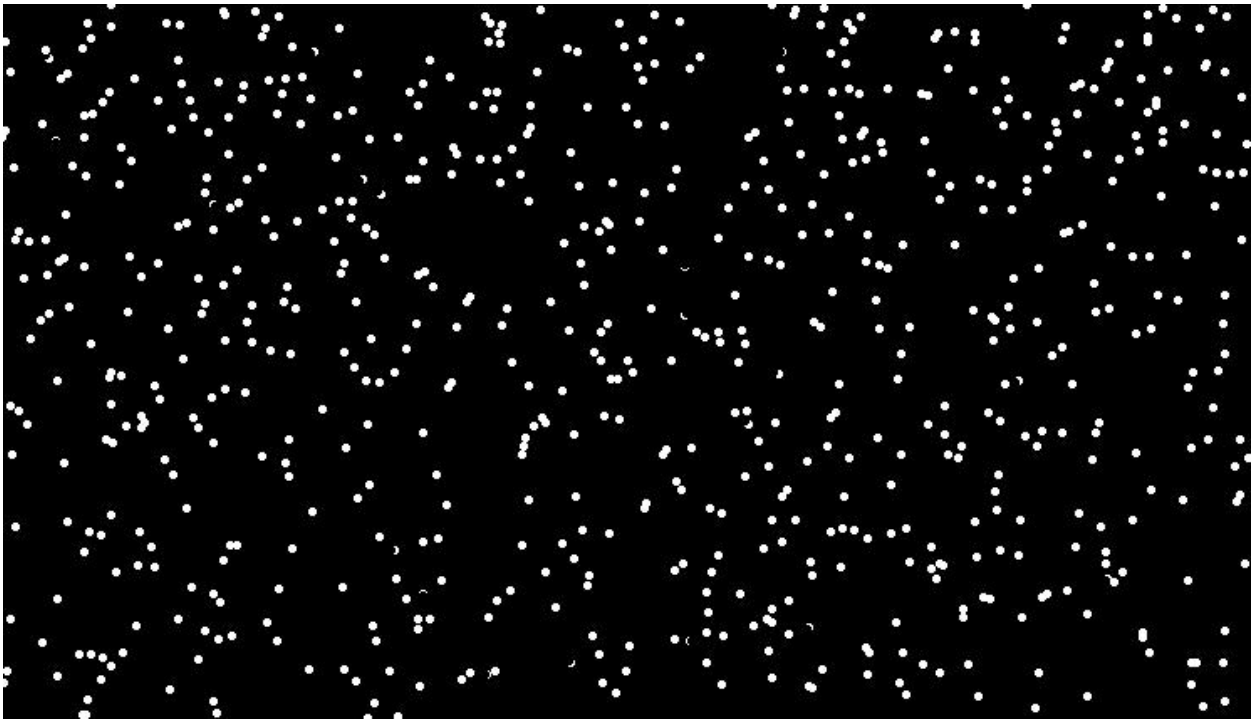
Early on, there is a much greater chance of landing on a white spot than on a black dot. But each time that happens, another dot is drawn. Over time, with more and more dots on the screen the probability of landing on a white space decreases while the probability of landing on a black dot increases.

When the screen is about half white and half black, there is approximately an equal probability of landing on a dot and erasing it or landing on a white space and drawing a dot. Therefore the half and half balance is maintained.

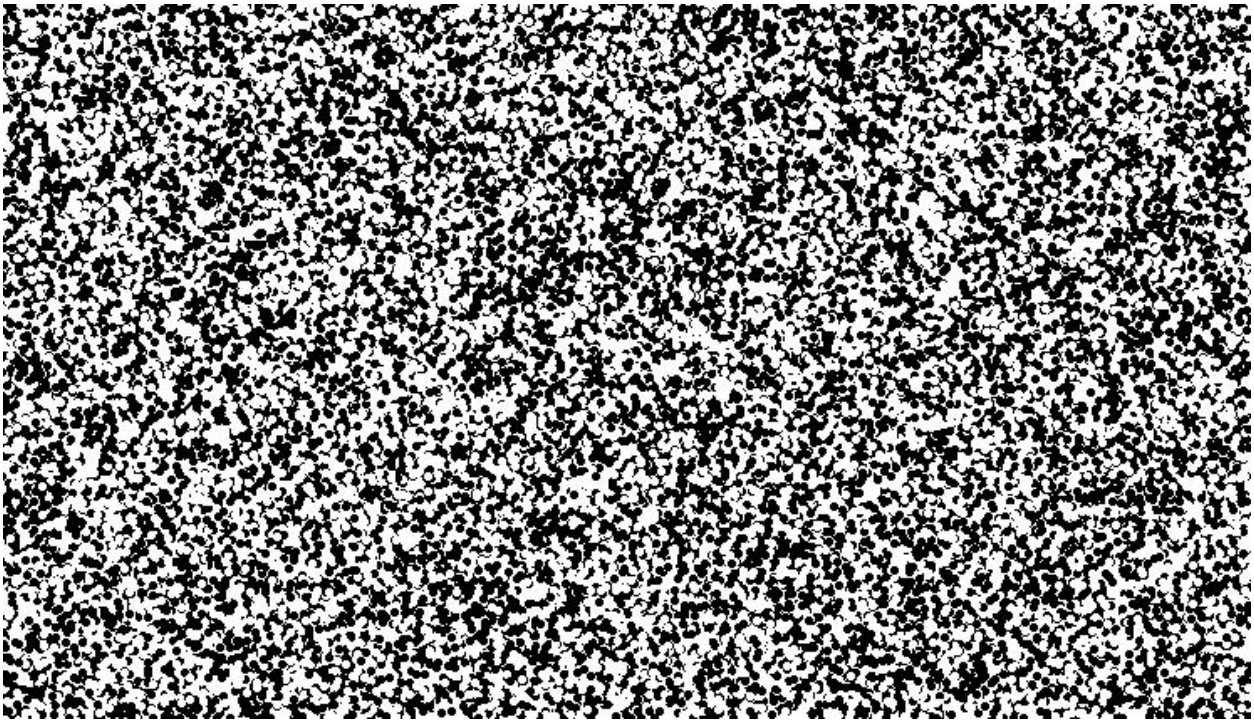
What if we start out with the screen looking like this…

…instead of being white?

After the program runs for a short while, we see something like this:



Sometime later a balance is achieved and the screen looks like this:

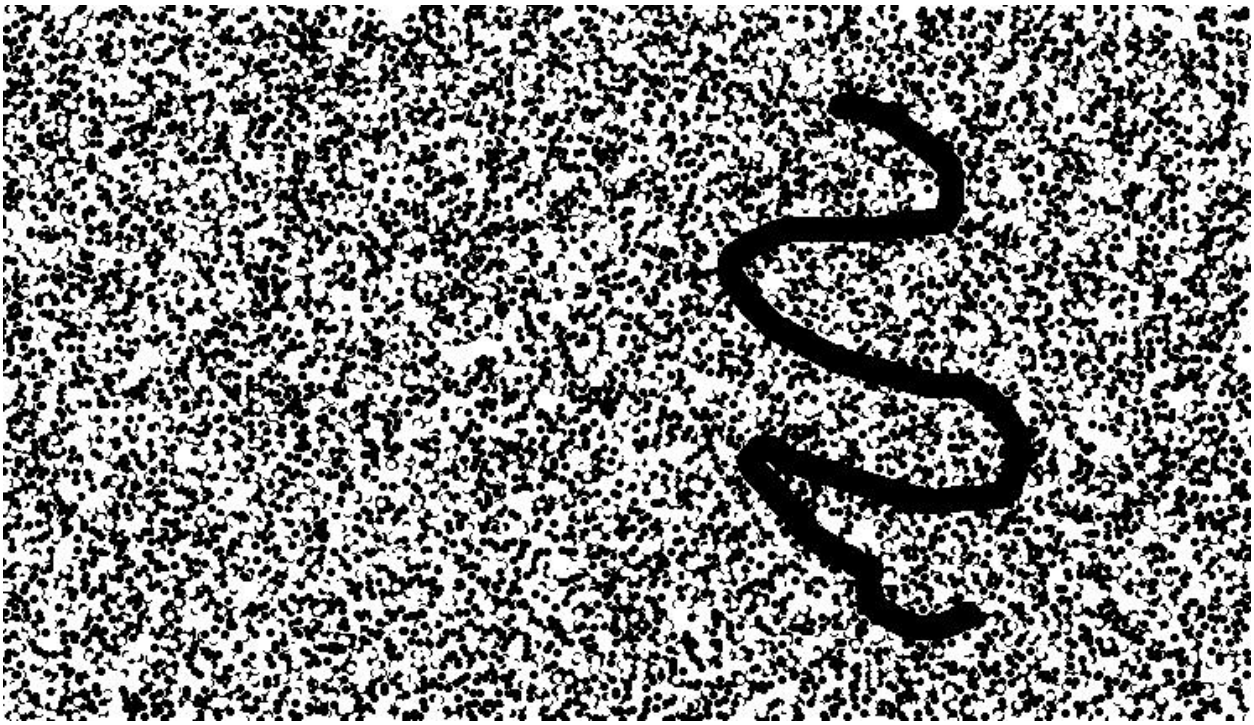


By starting with a black screen, it is certain that the first **go** will put the turtle on a black spot, and it will erase it. Early on in the process, it remains more likely that the turtle will erase a dot

than draw one. But as more and more dots are erased, the probabilities of landing on a white space or a black space become about equal.

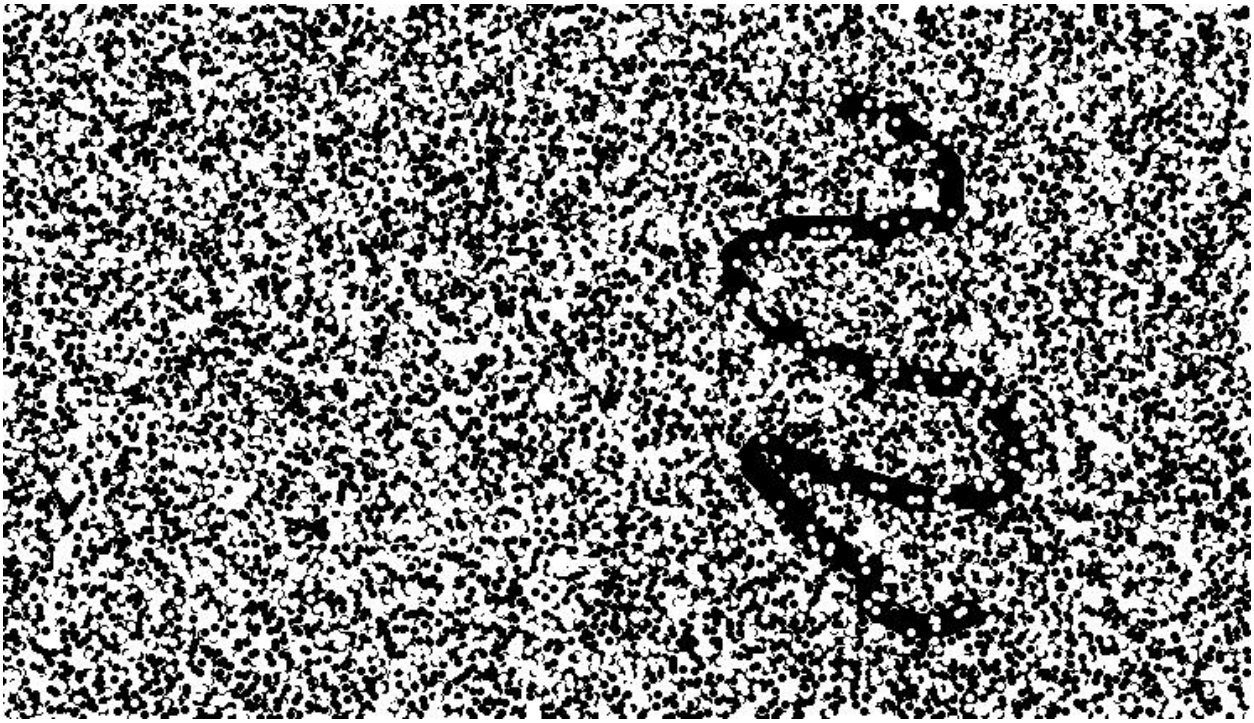
It doesn't matter whether we start with a black screen or a white screen. The same equilibrium is achieved.

The system is quite resilient. With the program running, we can scribble in black on a screen that is in balance.

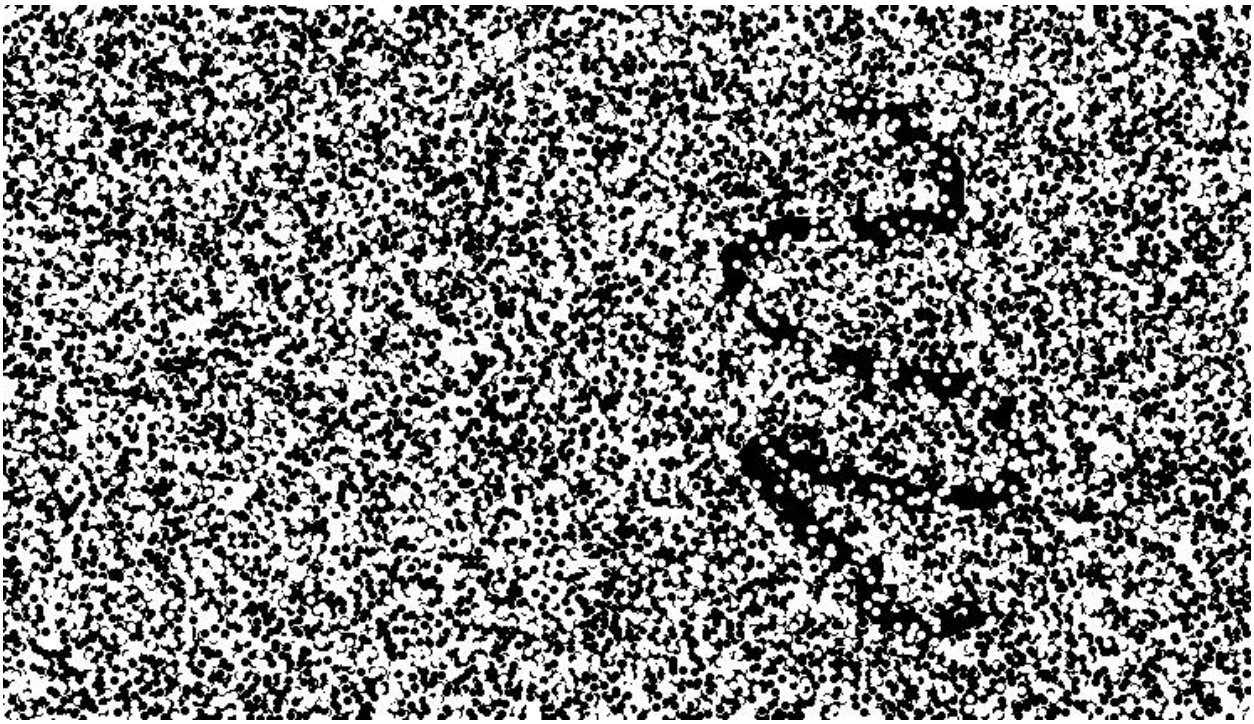


What happens?

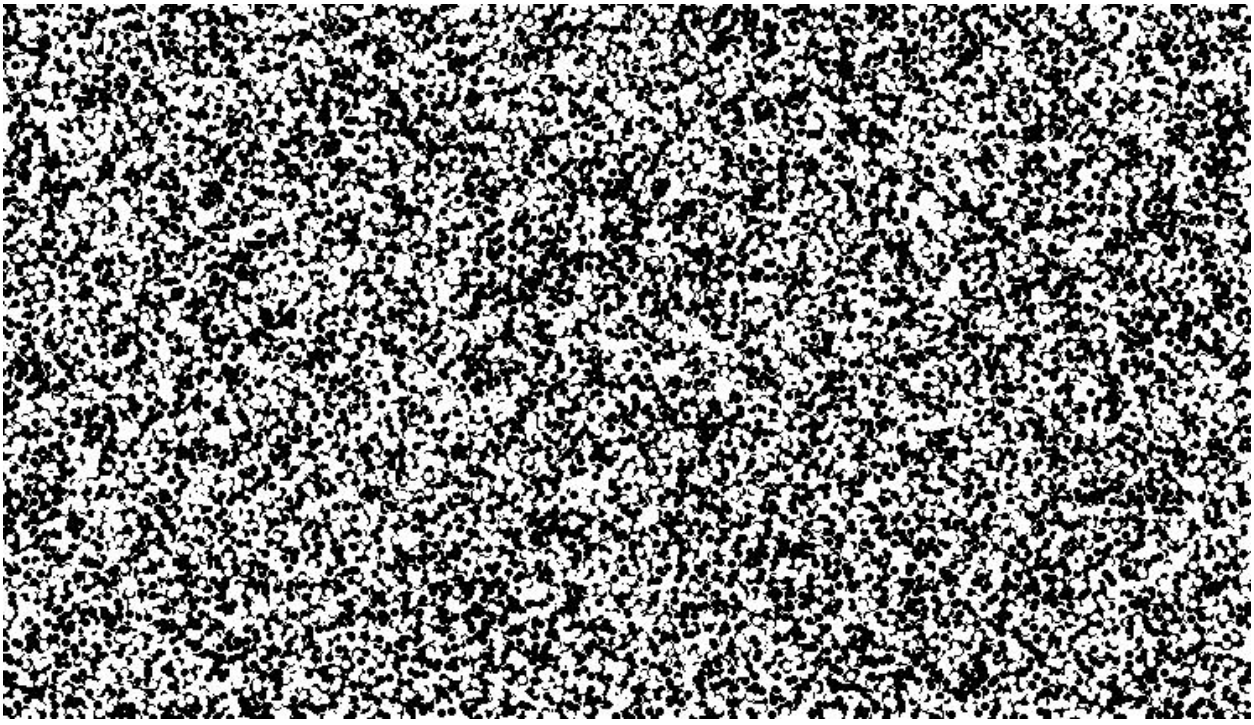
The scribble begins to fade…



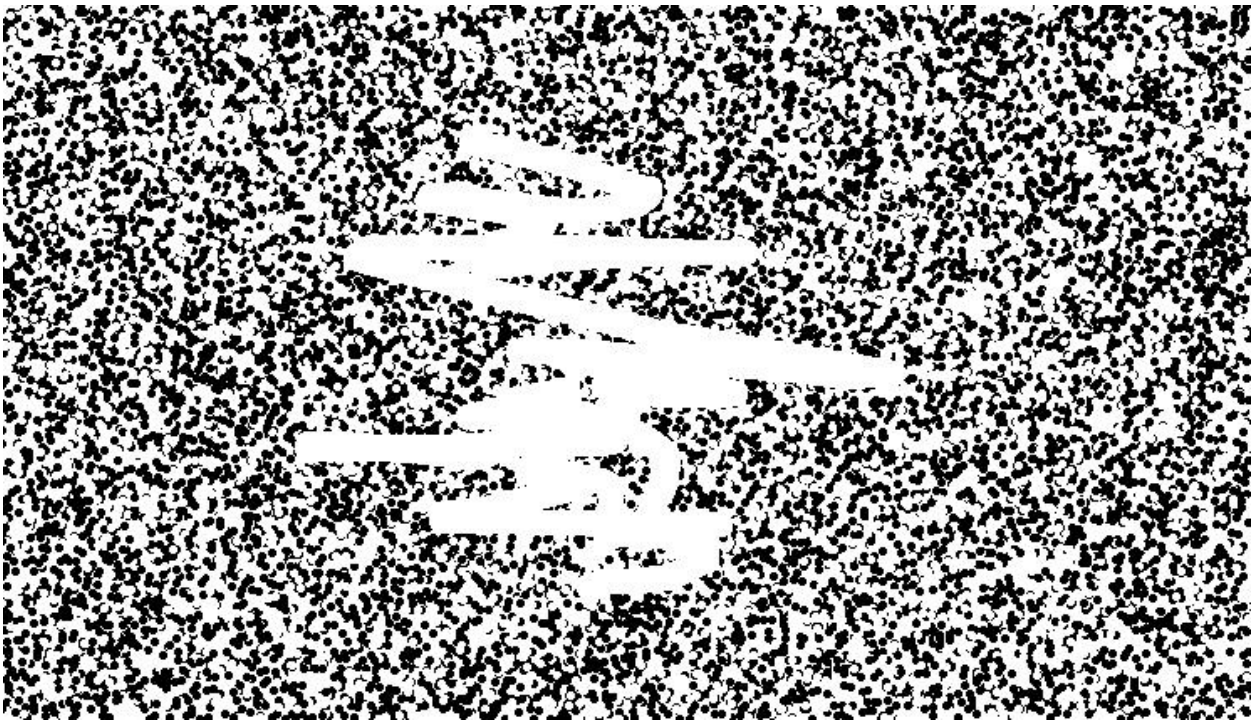
…and then fade some more…



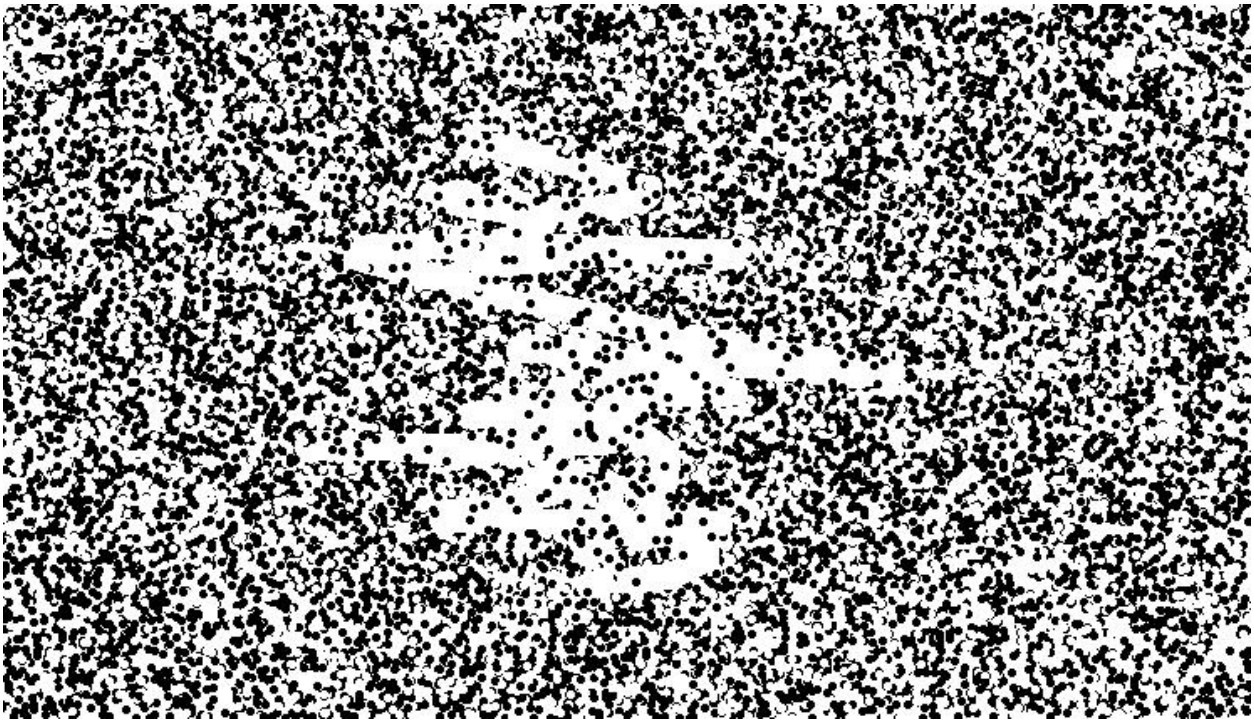
…until it is gone.



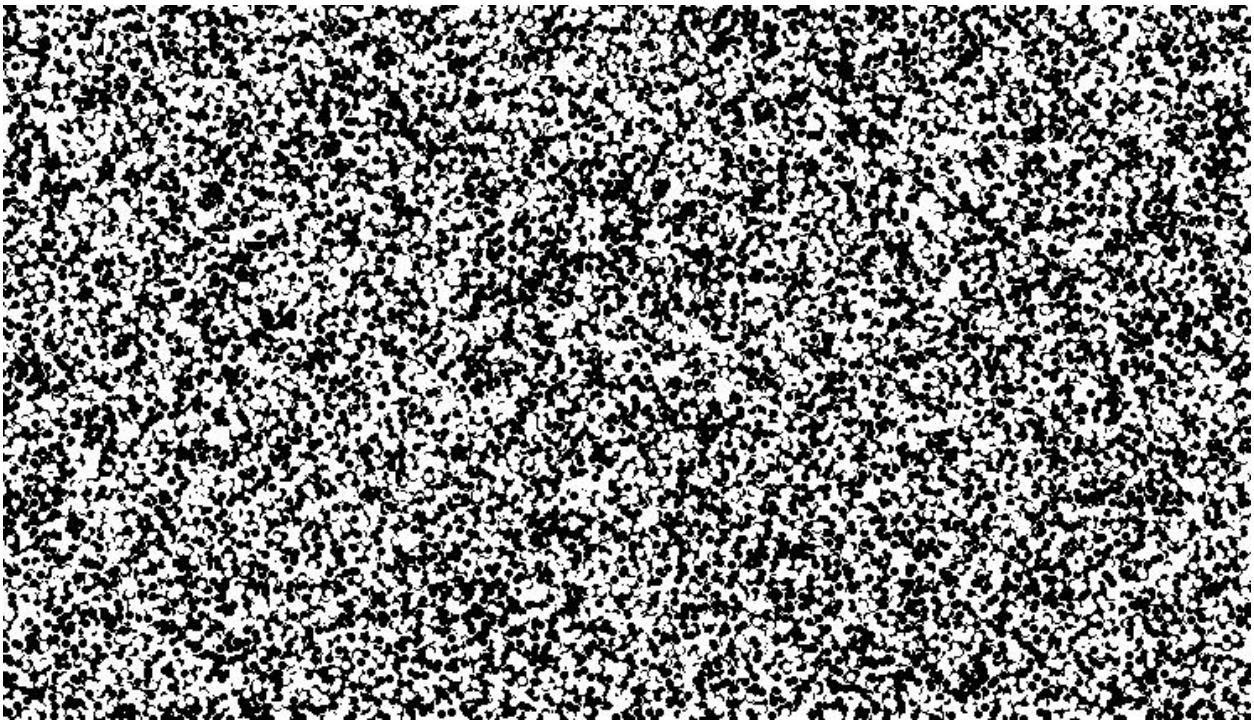
The same thing happens if we erase part of the screen while the program is running:



The erased area starts to fill in with dots…



…and eventually disappears.



This system is an example of a feedback loop. A mechanism in the system (the turtle) detects, or senses something about it (whether the location is white or black) and acts on that information. This is a negative feedback loop because the action moves the system in the direction that is opposite to what is sensed. If white is detected, the screen becomes less white. If black is detected, the screen becomes less black. The action tends to negate the state that is sensed.

This is also called a balancing feedback loop because the process tends to result in a balanced state — a state of equilibrium. As we saw, the screen settles into a stable state where it is approximately half white and half black.

There are many examples of negative or balancing feedback loops in the real world:

- The human body needs to maintain a temperature of 37˚. If you become too hot, you start to sweat and the evaporation of the moisture tends to reduce body temperature. If you become too cold, you shiver. The rapid muscle motion tends to raise body temperature.
- An air conditioner has a thermostat that turns the unit on when the temperature rises above a specified temperature. This causes the room to get cooler, so the unit turns off, which causes the temperature to rise, which causes the unit to turn on… . A balance is achieved with the temperature fluctuating in a narrow range around the set point.
- An increase in the amount of sugar in your blood causes increased secretion of insulin, which reduces blood sugar. Reduced blood sugar, in turn, reduces the secretion of insulin.
- Rabbits eat grass. If there is plenty of grass available, the population of rabbits is healthy and increases. But this means more grass is eaten, so there is less available. The rabbit population decreases. This allows more grass to grow…

But the real world is not so simple. Unlike our computer model, feedback loops in the real world do not exist in isolation. The amount of grass is determined by many factors in addition to how much the rabbits eat. A lack of rain would reduce the amount of grass. This would affect the rabbit population. There may be foxes or other predators that reduce the rabbit population. The rabbits and foxes are in a balancing feedback loop as well. The two loops interact with each other.

The computer-based activity presented here may be built upon and extended in many ways. We could alter the Logo program so that balance is achieved at approximately three-fourths black and one-fourth white rather than half an half. The program could also be used as the starting point for a simulation of a real-world feedback loop. For example, we could add more turtles and turn them into rabbits. The black dots could become green blades of grass. Then we would need to program rules that determine when and how rabbits are born and die. We could add more turtles and turn them into foxes…

Technical Details

The details of the Logo program are not included in the text of the article above, where only the main procedure, **go**, is listed. Here is the full program, which is written in MicroWorlds Logo:

```
to go
setheading random 360
forward random 1000
ifelse no-dot?
        [make-a-dot]
        [erase-a-dot]
end

to make-a-dot
pd fd 0 pu
end

to erase-a-dot
pe fd 0 pu
end

to dot?
output colorunder = 9
end

to no-dot?
output colorunder = 0
end
```

The size of the dots is determined by the turtle's pen size, which is set to 5. This is done by running the command

```
setpensize 5
```

in the command center before running the program the first time. This setting remains in effect.

Each color has a unique number: 9 is black, 0 is white. This information is used in the procedures **dot?** and **no-dot?**