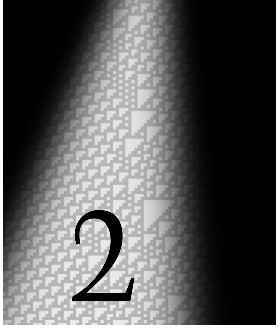
EXCERPTED FROM

STEPHEN
WOLFRAM
A NEW
KIND OF
SCIENCE

CHAPTER 2

The Crucial Experiment



2

The Crucial Experiment

How Do Simple Programs Behave?

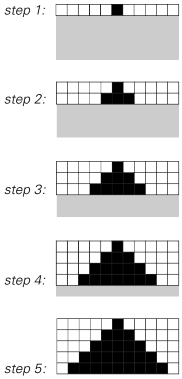
New directions in science have typically been initiated by certain central observations or experiments. And for the kind of science that I describe in this book these concerned the behavior of simple programs.

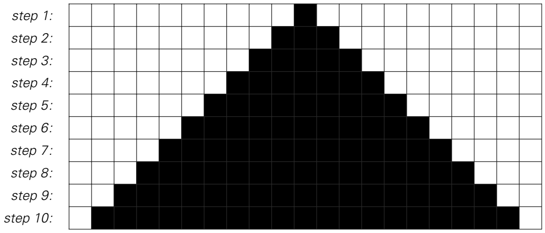
In our everyday experience with computers, the programs that we encounter are normally set up to perform very definite tasks. But the key idea that I had nearly twenty years ago—and that eventually led to the whole new kind of science in this book—was to ask what happens if one instead just looks at simple arbitrarily chosen programs, created without any specific task in mind. How do such programs typically behave?

The mathematical methods that have in the past dominated theoretical science do not help much with such a question. But with a computer it is straightforward to start doing experiments to investigate it. For all one need do is just set up a sequence of possible simple programs, and then run them and see how they behave.

Any program can at some level be thought of as consisting of a set of rules that specify what it should do at each step. There are many possible ways to set up these rules—and indeed we will study quite a few of them in the course of this book. But for now, I will consider a particular class of examples called cellular automata, that were the very first kinds of simple programs that I investigated in the early 1980s.

An important feature of cellular automata is that their behavior can readily be presented in a visual way. And so the picture below shows what one cellular automaton does over the course of ten steps.

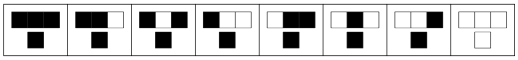




A visual representation of the behavior of a cellular automaton, with each row of cells corresponding to one step. At the first step the cell in the center is black and all other cells are white. Then on each successive step, a particular cell is made black whenever it or either of its neighbors were black on the step before. As the picture shows, this leads to a simple expanding pattern uniformly filled with black.

The cellular automaton consists of a line of cells, each colored either black or white. At every step there is then a definite rule that determines the color of a given cell from the color of that cell and its immediate left and right neighbors on the step before.

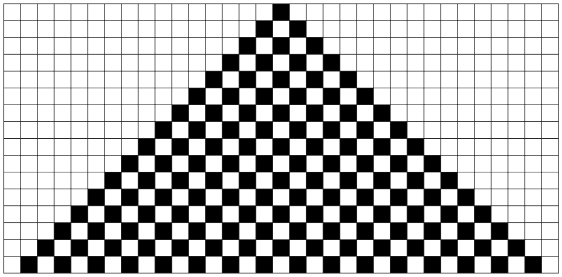
For the particular cellular automaton shown here the rule specifies—as in the picture below—that a cell should be black in all cases where it or either of its neighbors were black on the step before.

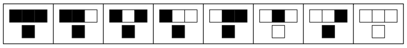


A representation of the rule for the cellular automaton shown above. The top row in each box gives one of the possible combinations of colors for a cell and its immediate neighbors. The bottom row then specifies what color the center cell should be on the next step in each of these cases. In the numbering scheme described in Chapter 3, this is cellular automaton rule 254.

And the picture at the top of the page shows that starting with a single black cell in the center this rule then leads to a simple growing pattern uniformly filled with black. But modifying the rule just slightly one can immediately get a different pattern.

As a first example, the picture at the top of the facing page shows what happens with a rule that makes a cell white whenever both of its neighbors were white on the step before—even if the cell itself was black before. And rather than producing a pattern that is uniformly filled with black, this rule now instead gives a pattern that repeatedly alternates between black and white like a checkerboard.

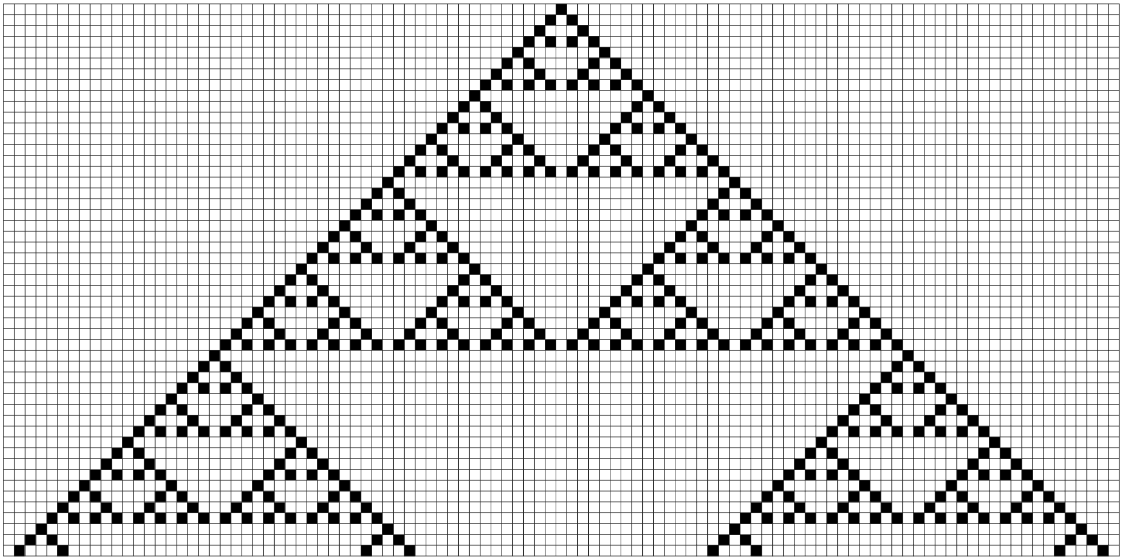


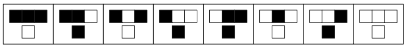


A cellular automaton with a slightly different rule. The rule makes a particular cell black if either of its neighbors was black on the step before, and makes the cell white if both its neighbors were white. Starting from a single black cell, this rule leads to a checkerboard pattern. In the numbering scheme of Chapter 3, this is cellular automaton rule 250.

This pattern is however again fairly simple. And we might assume that at least with the type of cellular automata that we are considering, any rule we might choose would always give a pattern that is quite simple. But now we are in for our first surprise.

The picture below shows the pattern produced by a cellular automaton of the same type as before, but with a slightly different rule.

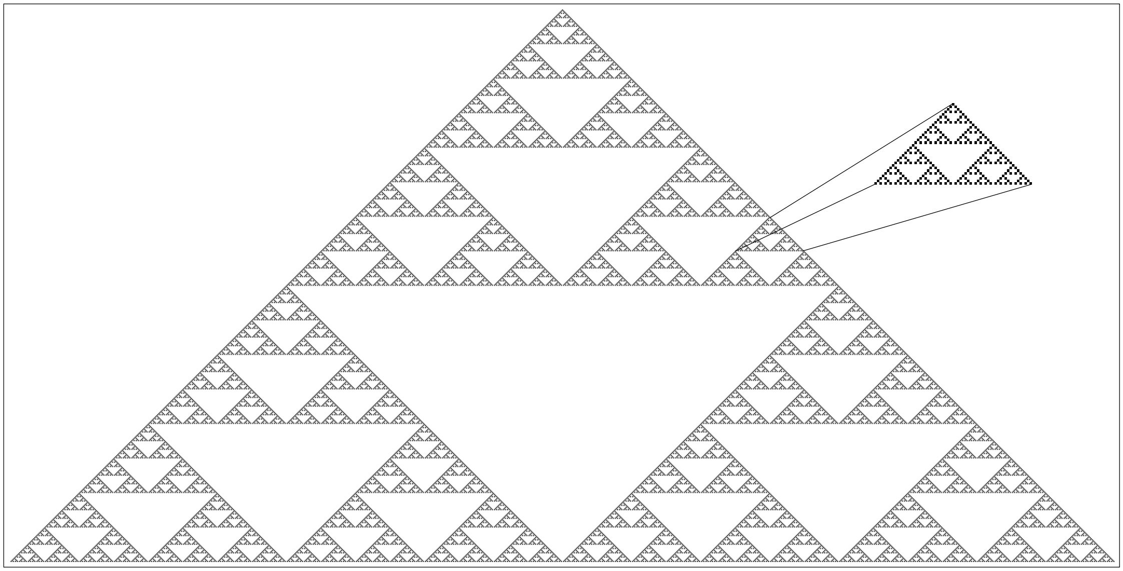




A cellular automaton that produces an intricate nested pattern. The rule in this case is that a cell should be black whenever one or the other, but not both, of its neighbors were black on the step before. Even though the rule is very simple, the picture shows that the overall pattern obtained over the course of 50 steps starting from a single black cell is not so simple. The particular rule used here can be described by the formula $a_i' = Mod[a_{i-1} + a_{i+1}, 2]$. In the numbering scheme of Chapter 3, it is cellular automaton rule 90.

This time the rule specifies that a cell should be black when either its left neighbor or its right neighbor—but not both—were black on the step before. And again this rule is undeniably quite simple. But now the picture shows that the pattern it produces is not so simple.

And if one runs the cellular automaton for more steps, as in the picture below, then a rather intricate pattern emerges. But one can now see that this pattern has very definite regularity. For even though it is intricate, one can see that it actually consists of many nested triangular pieces that all have exactly the same form. And as the picture shows, each of these pieces is essentially just a smaller copy of the whole pattern, with still smaller copies nested in a very regular way inside it.

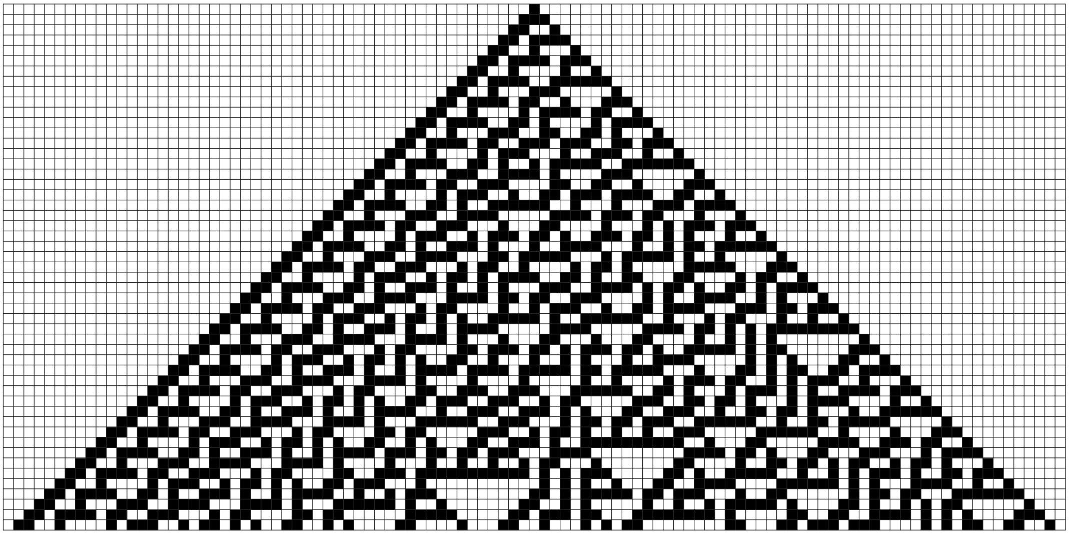


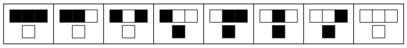
A larger version of the pattern from the previous page, now shown without a grid explicitly indicating each cell. The picture shows five hundred steps of cellular automaton evolution. The pattern obtained is intricate, but has a definite nested structure. Indeed, as the picture illustrates, each triangular section is essentially just a smaller copy of the whole pattern, with still smaller copies nested inside it. Patterns with nested structure of this kind are often called "fractal" or "self-similar".

So of the three cellular automata that we have seen so far, all ultimately yield patterns that are highly regular: the first a simple uniform pattern, the second a repetitive pattern, and the third an intricate but still nested pattern. And we might assume that at least for cellular automata with rules as simple as the ones we have been using these three forms of behavior would be all that we could ever get.

But the remarkable fact is that this turns out to be wrong.

And the picture below shows an example of this. The rule used—that I call rule 30—is of exactly the same kind as before, and can be described as follows. First, look at each cell and its right-hand neighbor. If both of these were white on the previous step, then take the new color of the cell to be whatever the previous color of its left-hand neighbor was. Otherwise, take the new color to be the opposite of that.





A cellular automaton with a simple rule that generates a pattern which seems in many respects random. The rule used is of the same type as in the previous examples, and the cellular automaton is again started from a single black cell. But now the pattern that is obtained is highly complex, and shows almost no overall regularity. This picture is our first example of the fundamental phenomenon that even with simple underlying rules and simple initial conditions, it is possible to produce behavior of great complexity. In the numbering scheme of Chapter 3, the cellular automaton shown here is rule 30.

The picture shows what happens when one starts with just one black cell and then applies this rule over and over again. And what one sees is something quite startling—and probably the single most surprising scientific discovery I have ever made. Rather than getting a simple regular pattern as we might expect, the cellular automaton instead produces a pattern that seems extremely irregular and complex.

But where does this complexity come from? We certainly did not put it into the system in any direct way when we set it up. For we just used a simple cellular automaton rule, and just started from a simple initial condition containing a single black cell.

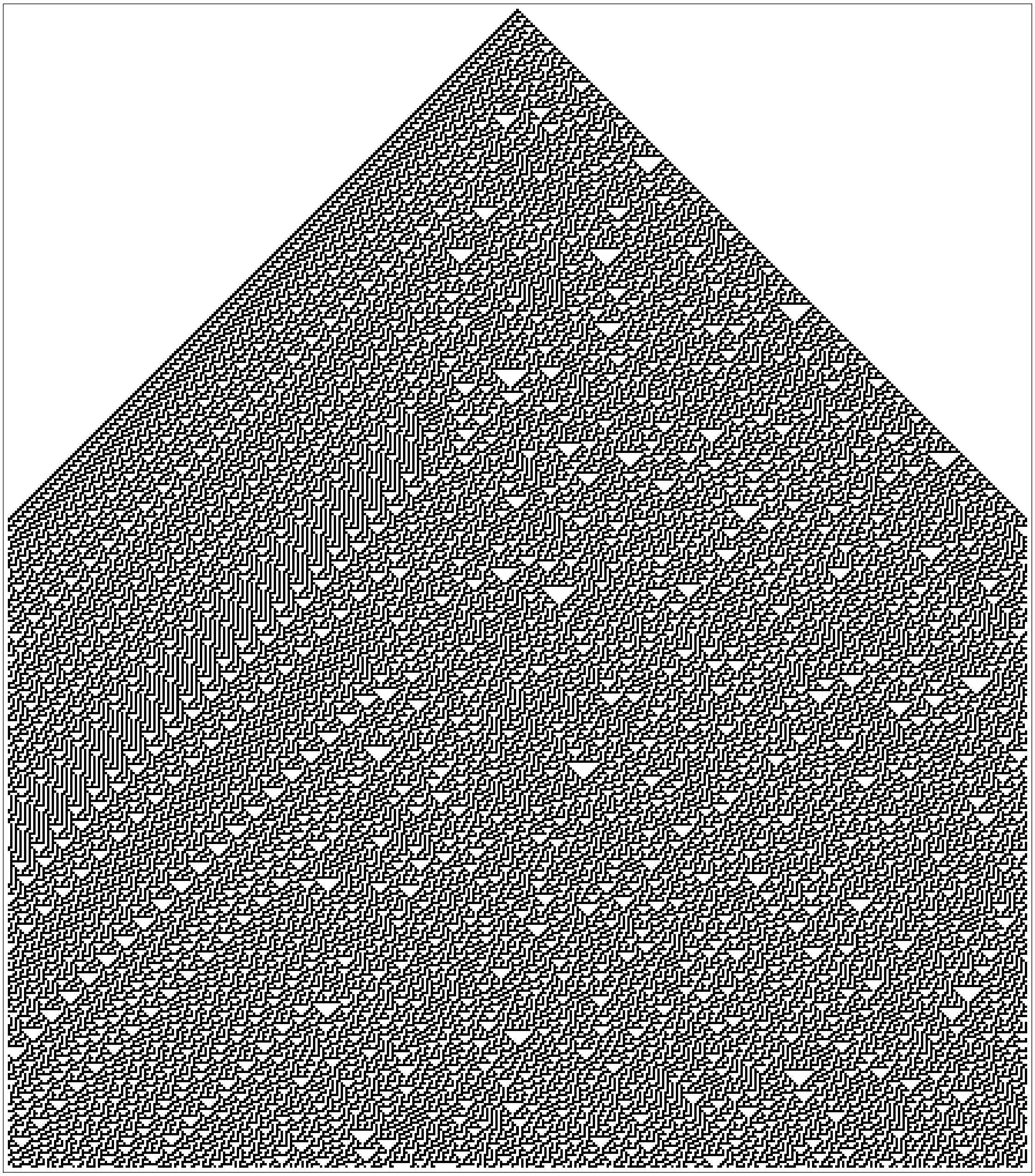
Yet the picture shows that despite this, there is great complexity in the behavior that emerges. And indeed what we have seen here is a first example of an extremely general and fundamental phenomenon that is at the very core of the new kind of science that I develop in this book. Over and over again we will see the same kind of thing: that even though the underlying rules for a system are simple, and even though the system is started from simple initial conditions, the behavior that the system shows can nevertheless be highly complex. And I will argue that it is this basic phenomenon that is ultimately responsible for most of the complexity that we see in nature.

The next two pages show progressively more steps in the evolution of the rule 30 cellular automaton from the previous page. One might have thought that after maybe a thousand steps the behavior would eventually resolve into something simple. But the pictures on the next two pages show that nothing of the sort happens.

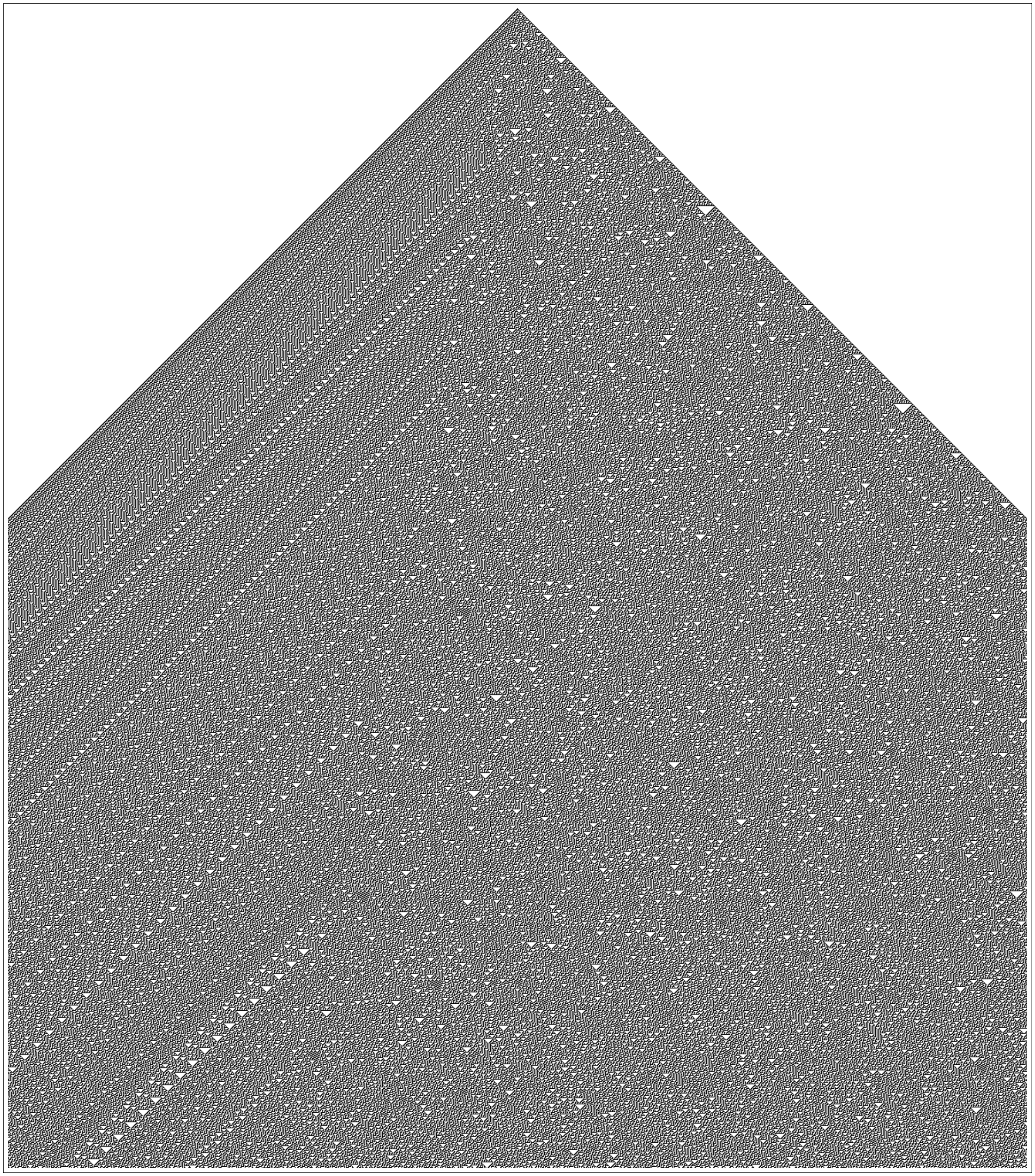
Some regularities can nevertheless be seen. On the left-hand side, for example, there are obvious diagonal bands. And dotted throughout there are various white triangles and other small structures. Yet given the simplicity of the underlying rule, one would expect vastly more regularities. And perhaps one might imagine that our failure to see any in the pictures on the next two pages is just a reflection of some kind of inadequacy in the human visual system.

But it turns out that even the most sophisticated mathematical and statistical methods of analysis seem to do no better. For example, one can look at the sequence of colors directly below the initial black cell. And in the first million steps in this sequence, for example, it never repeats, and indeed none of the tests I have ever done on it show any meaningful deviation at all from perfect randomness.

In a sense, however, there is a certain simplicity to such perfect randomness. For even though it may be impossible to predict what



Five hundred steps in the evolution of the rule 30 cellular automaton from page 27. The pattern produced continues to expand on both left and right, but only the part that fits across the page is shown here. The asymmetry between the left and right-hand sides is a direct consequence of asymmetry that exists in the particular underlying cellular automaton rule used.



Fifteen hundred steps of rule 30 evolution. Some regularities are evident, particularly on the left. But even after all these steps there are no signs of overall regularity—and indeed even continuing for a million steps many aspects of the pattern obtained seem perfectly random according to standard mathematical and statistical tests. The picture here shows a total of just under two million individual cells.

color will occur at any specific step, one still knows for example that black and white will on average always occur equally often.

But it turns out that there are cellular automata whose behavior is in effect still more complex—and in which even such averages become very difficult to predict. The pictures on the next several pages give a rather dramatic example. The basic form of the rule is just the same as before. But now the specific rule used—that I call rule 110—takes the new color of a cell to be black in every case except when the previous colors of the cell and its two neighbors were all the same, or when the left neighbor was black and the cell and its right neighbor were both white.

The pattern obtained with this rule shows a remarkable mixture of regularity and irregularity. More or less throughout, there is a very regular background texture that consists of an array of small white triangles repeating every 7 steps. And beginning near the left-hand edge, there are diagonal stripes that occur at intervals of exactly 80 steps.

But on the right-hand side, the pattern is much less regular. Indeed, for the first few hundred steps there is a region that seems essentially random. But by the bottom of the first page, all that remains of this region is three copies of a rather simple repetitive structure.

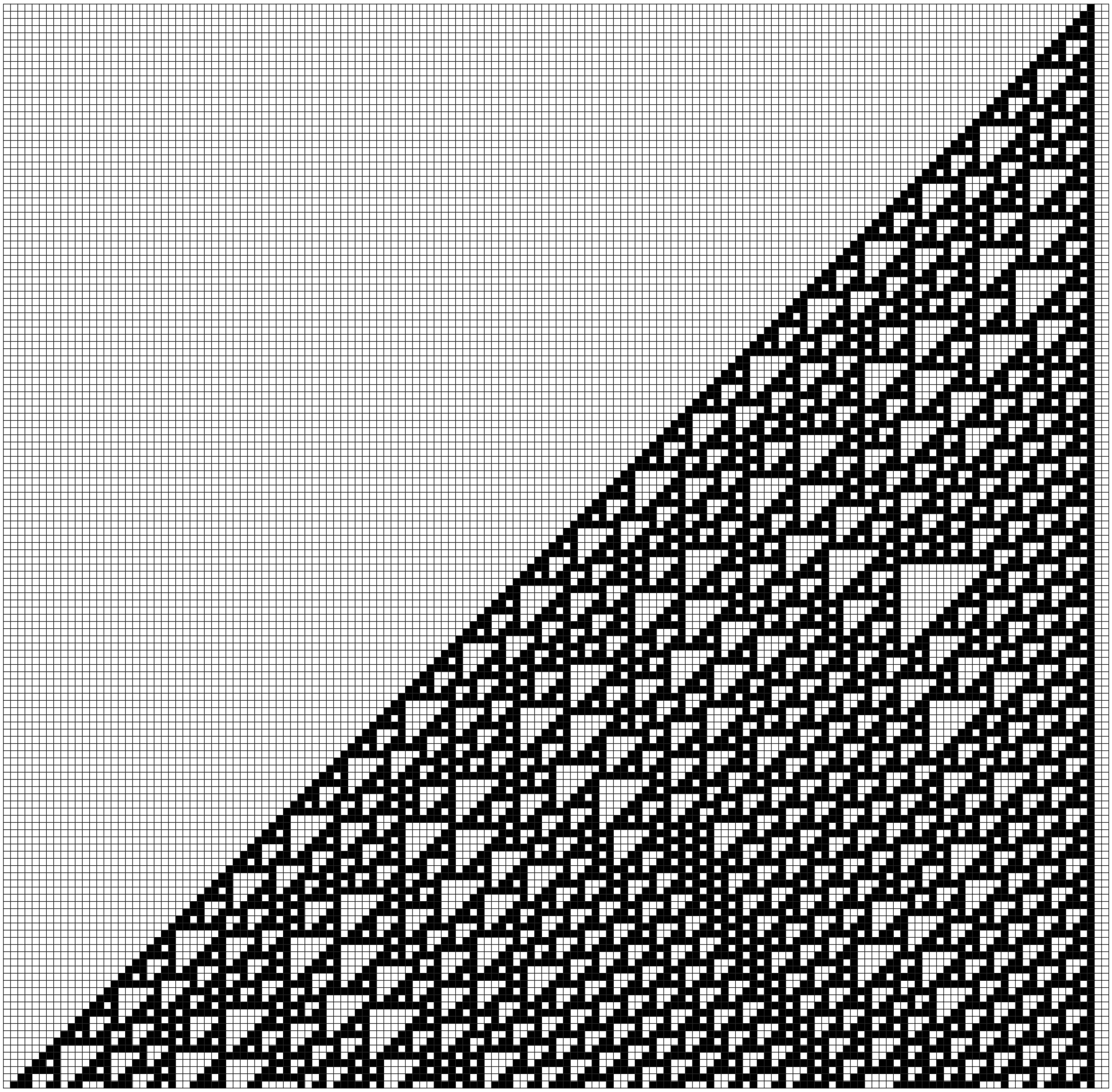
Yet at the top of the next page the arrival of a diagonal stripe from the left sets off more complicated behavior again. And as the system progresses, a variety of definite localized structures are produced.

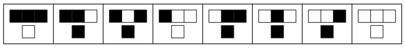
Some of these structures remain stationary, like those at the bottom of the first page, while others move steadily to the right or left at various speeds. And on their own, each of these structures works in a fairly simple way. But as the pictures illustrate, their various interactions can have very complicated effects.

And as a result it becomes almost impossible to predict—even approximately—what the cellular automaton will do.

Will all the structures that are produced eventually annihilate each other, leaving only a very regular pattern? Or will more and more structures appear until the whole pattern becomes quite random?

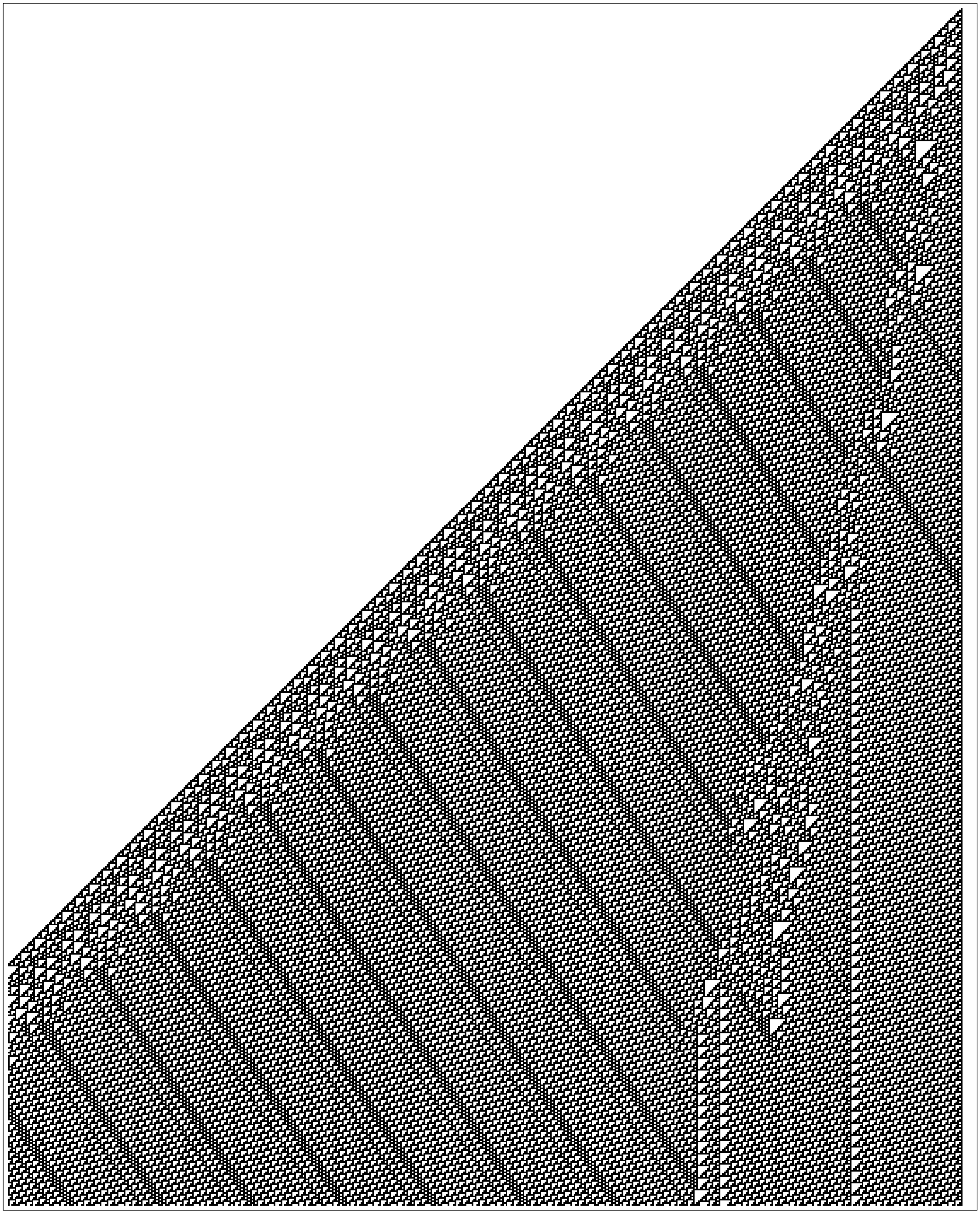
The only sure way to answer these questions, it seems, is just to run the cellular automaton for as many steps as are needed, and to

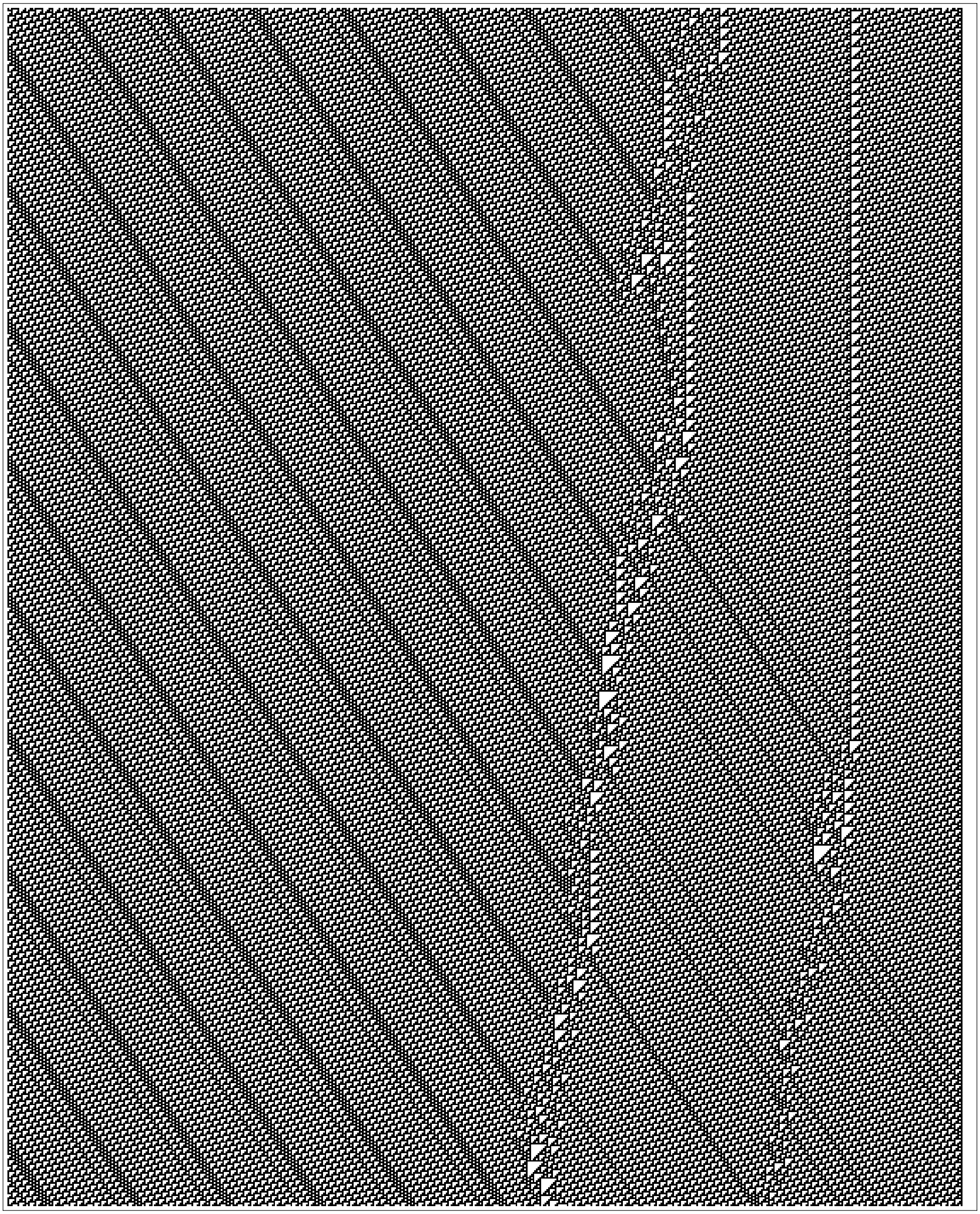


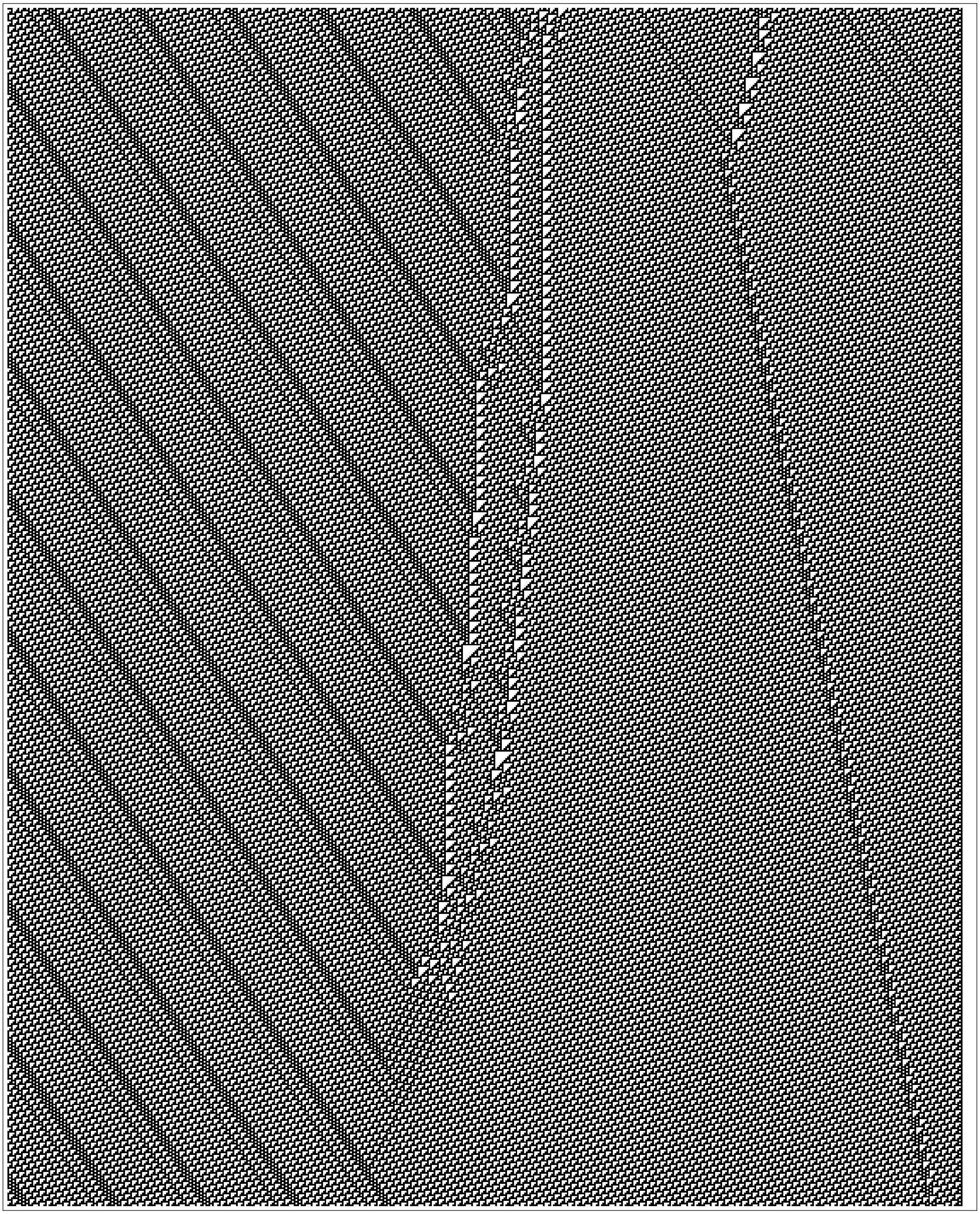


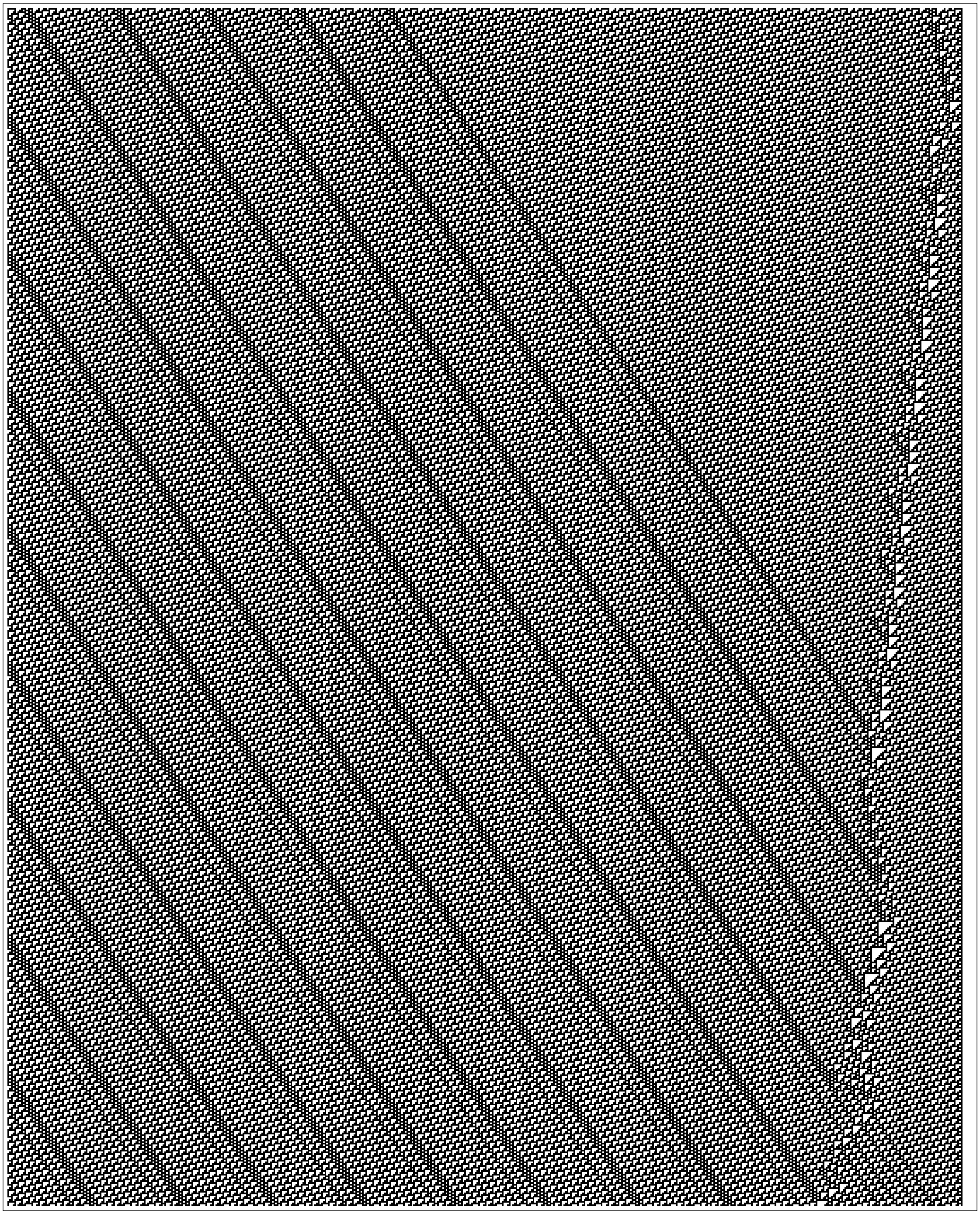
A cellular automaton whose behavior seems neither highly regular nor completely random. The picture is obtained by applying the simple rule shown for a total of 150 steps, starting with a single black cell. Note that the particular rule used here yields a pattern that expands on the left but not on the right. In the scheme defined in Chapter 3, the rule is number 110.

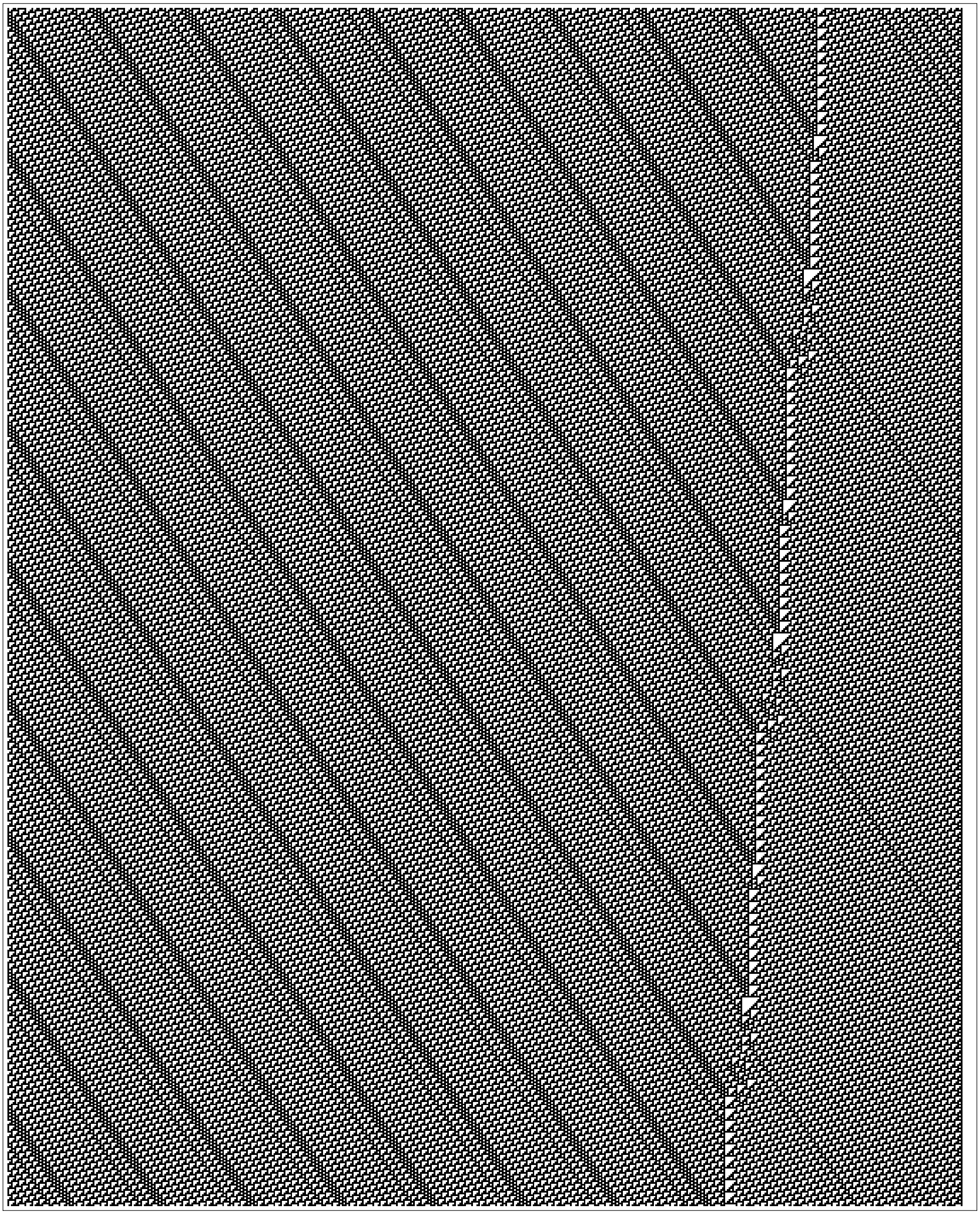
More steps in the pattern shown above. Each successive page shows a total of 700 steps. The pattern continues to expand on the left forever, but only the part that fits across each page is shown. For a long time it is not clear how the right-hand part of the pattern will eventually look. But after 2780 steps, a fairly simple repetitive structure emerges. Note that to generate the pictures that follow requires applying the underlying cellular automaton rule for individual cells a total of about 12 million times. ▶

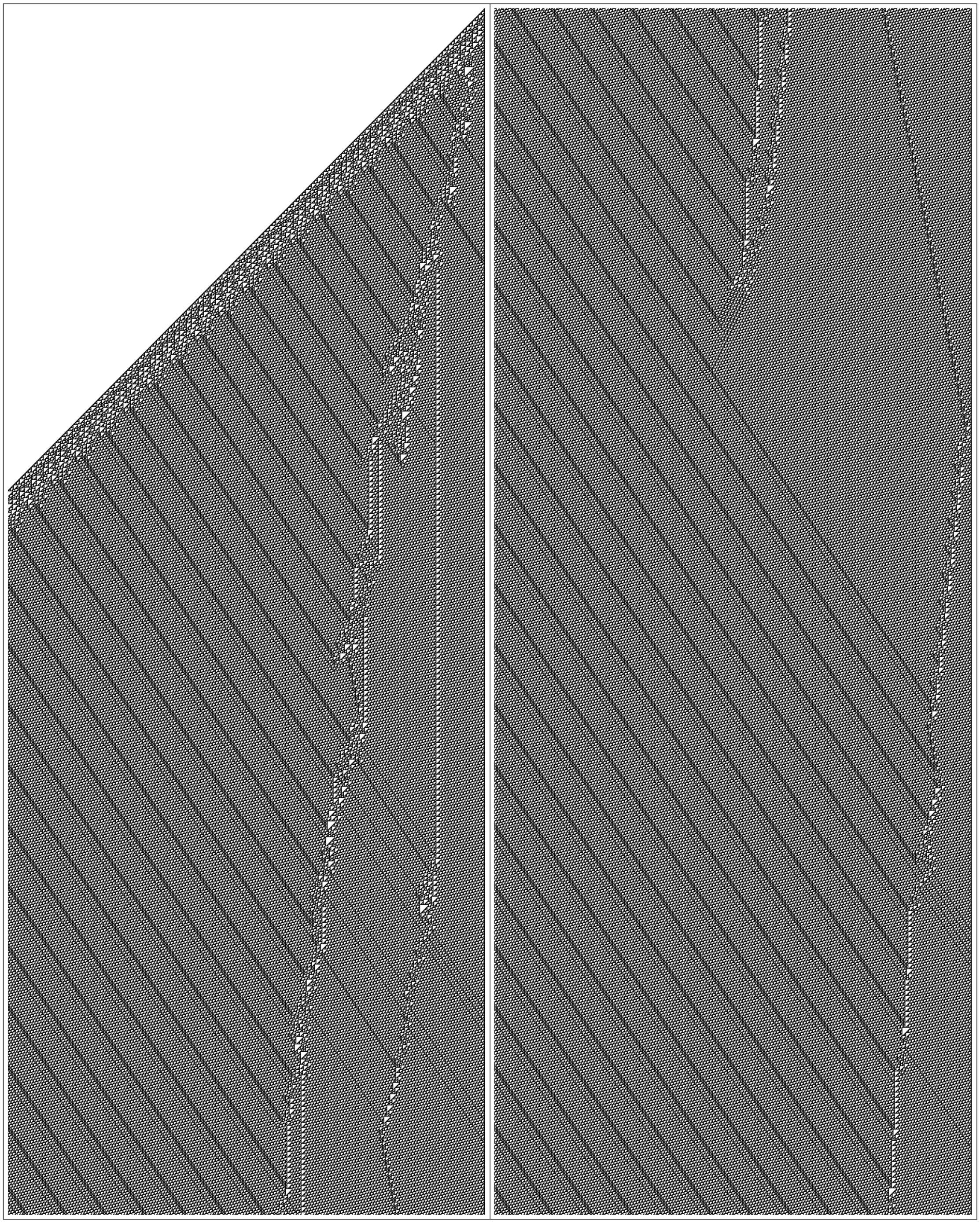












watch what happens. And as it turns out, in the particular case shown here, the outcome is finally clear after about 2780 steps: one structure survives, and that structure interacts with the periodic stripes coming from the left to produce behavior that repeats every 240 steps.

However certain one might be that simple programs could never do more than produce simple behavior, the pictures on the past few pages should forever disabuse one of that notion. And indeed, what is perhaps most bizarre about the pictures is just how little trace they ultimately show of the simplicity of the underlying cellular automaton rule that was used to produce them.

One might think, for example, that the fact that all the cells in a cellular automaton follow exactly the same rule would mean that in pictures like the last few pages all cells would somehow obviously be doing the same thing. But instead, they seem to be doing quite different things. Some of them, for example, are part of the regular background, while others are part of one or another localized structure. And what makes this possible is that even though individual cells follow the same rule, different configurations of cells with different sequences of colors can together produce all sorts of different kinds of behavior.

Looking just at the original cellular automaton rule one would have no realistic way to foresee all of this. But by doing the appropriate computer experiments one can easily find out what actually happens—and in effect begin the process of exploring a whole new world of remarkable phenomena associated with simple programs.

The Need for a New Intuition

The pictures in the previous section plainly show that it takes only very simple rules to produce highly complex behavior. Yet at first this may seem almost impossible to believe. For it goes against some of our most basic intuition about the way things normally work.

◀ A single picture of the behavior from the previous five pages. A total of 3200 steps are shown. Note that this is more than twice as many as in the picture on page 30.

For our everyday experience has led us to expect that an object that looks complicated must have been constructed in a complicated way. And so, for example, if we see a complicated mechanical device, we normally assume that the plans from which the device was built must also somehow be correspondingly complicated.

But the results at the end of the previous section show that at least sometimes such an assumption can be completely wrong. For the patterns we saw are in effect built according to very simple plans—that just tell us to start with a single black cell, and then repeatedly to apply a simple cellular automaton rule. Yet what emerges from these plans shows an immense level of complexity.

So what is it that makes our normal intuition fail? The most important point seems to be that it is mostly derived from experience with building things and doing engineering—where it so happens that one avoids encountering systems like the ones in the previous section.

For normally we start from whatever behavior we want to get, then try to design a system that will produce it. Yet to do this reliably, we have to restrict ourselves to systems whose behavior we can readily understand and predict—for unless we can foresee how a system will behave, we cannot be sure that the system will do what we want.

But unlike engineering, nature operates under no such constraint. So there is nothing to stop systems like those at the end of the previous section from showing up. And in fact one of the important conclusions of this book is that such systems are actually very common in nature.

But because the only situations in which we are routinely aware both of underlying rules and overall behavior are ones in which we are building things or doing engineering, we never normally get any intuition about systems like the ones at the end of the previous section.

So is there then any aspect of everyday experience that should give us a hint about the phenomena that occur in these systems? Probably the closest is thinking about features of practical computing.

For we know that computers can perform many complex tasks. Yet at the level of basic hardware a typical computer is capable of executing just a few tens of kinds of simple logical, arithmetic and other instructions. And to some extent the fact that by executing large numbers of such instructions one can get all sorts of complex behavior is similar to the phenomenon we have seen in cellular automata.

But there is an important difference. For while the individual machine instructions executed by a computer may be quite simple, the sequence of such instructions defined by a program may be long and complicated. And indeed—much as in other areas of engineering—the typical experience in developing software is that to make a computer do something complicated requires setting up a program that is itself somehow correspondingly complicated.

In a system like a cellular automaton the underlying rules can be thought of as rough analogs of the machine instructions for a computer, while the initial conditions can be thought of as rough analogs of the program. Yet what we saw in the previous section is that in cellular automata not only can the underlying rules be simple, but the initial conditions can also be simple—consisting say of just a single black cell—and still the behavior that is produced can be highly complex.

So while practical computing gives a hint of part of what we saw in the previous section, the whole phenomenon is something much larger and stronger. And in a sense the most puzzling aspect of it is that it seems to involve getting something from nothing.

For the cellular automata we set up are by any measure simple to describe. Yet when we ran them we ended with patterns so complex that they seemed to defy any simple description at all.

And one might hope that it would be possible to call on some existing kind of intuition to understand such a fundamental phenomenon. But in fact there seems to be no branch of everyday experience that provides what is needed. And so we have no choice but to try to develop a whole new kind of intuition.

And the only reasonable way to do this is to expose ourselves to a large number of examples. We have seen so far only a few examples, all in cellular automata. But in the next few chapters we will see many more examples, both in cellular automata and in all sorts of other systems. And by absorbing these examples, one is in the end able to develop an intuition that makes the basic phenomena that I have discovered seem somehow almost obvious and inevitable.

Why These Discoveries Were Not Made Before

The main result of this chapter—that programs based on simple rules can produce behavior of great complexity—seems so fundamental that one might assume it must have been discovered long ago. But it was not, and it is useful to understand some of the reasons why it was not.

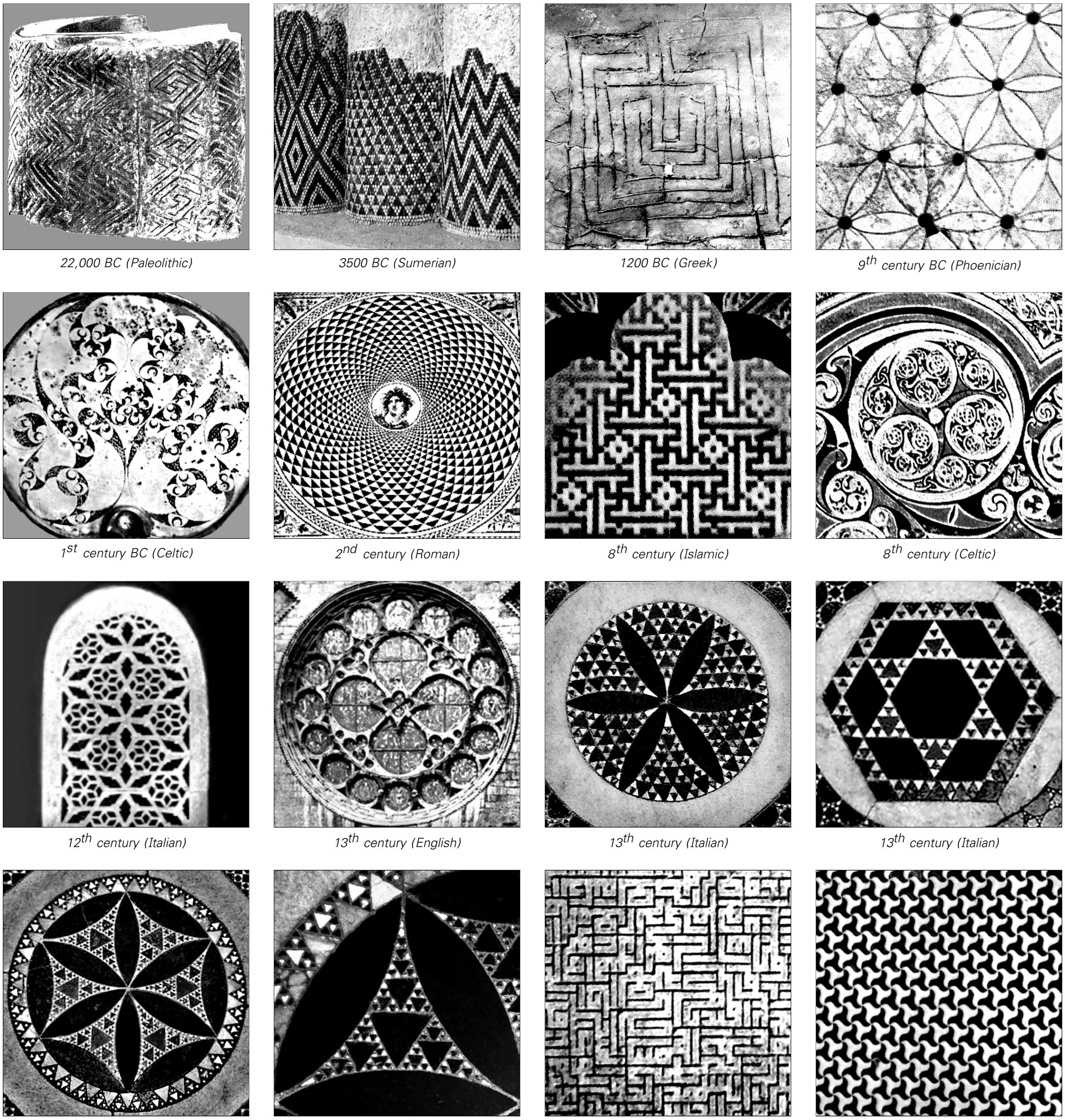
In the history of science it is fairly common that new technologies are ultimately what make new areas of basic science develop. And thus, for example, telescope technology was what led to modern astronomy, and microscope technology to modern biology. And now, in much the same way, it is computer technology that has led to the new kind of science that I describe in this book.

Indeed, this chapter and several of those that follow can in a sense be viewed as an account of some of the very simplest experiments that can be done using computers. But why is it that such simple experiments were never done before?

One reason is just that they were not in the mainstream of any existing field of science or mathematics. But a more important reason is that standard intuition in traditional science gave no reason to think that their results would be interesting.

And indeed, if it had been known that they were worthwhile, many of the experiments could actually have been done even long before computers existed. For while it may be somewhat tedious, it is certainly possible to work out the behavior of something like a cellular automaton by hand. And in fact, to do so requires absolutely no sophisticated ideas from mathematics or elsewhere: all it takes is an understanding of how to apply simple rules repeatedly.

And looking at the historical examples of ornamental art on the facing page, there seems little reason to think that the behavior of many cellular automata could not have been worked out many centuries or even millennia ago. And perhaps one day some Babylonian artifact created using the rule 30 cellular automaton from page 27 will be unearthed. But I very much doubt it. For I tend to think that if pictures like the one on page 27 had ever in fact been seen in ancient times then science would have been led down a very different path from the one it actually took.



22,000 BC (Paleolithic)

3500 BC (Sumerian)

1200 BC (Greek)

9th century BC (Phoenician)

1st century BC (Celtic)

2nd century (Roman)

8th century (Islamic)

8th century (Celtic)

12th century (Italian)

13th century (English)

13th century (Italian)

13th century (Italian)

13th century (Italian)

13th century (Italian)

14th century (Islamic)

14th century (Islamic)

Historical examples of ornamental art. Repetitive patterns are common and some nested patterns are seen, but the more complicated kinds of patterns discussed in this chapter do not ever appear to have been used. Note that the second-to-last picture is not an abstract design, but is instead text written in a highly stylized form of Arabic script.

Even early in antiquity attempts were presumably made to see whether simple abstract rules could reproduce the behavior of natural systems. But so far as one can tell the only types of rules that were tried were ones associated with standard geometry and arithmetic. And using these kinds of rules, only rather simple behavior could be obtained—adequate to explain some of the regularities observed in astronomy, but unable to capture much of what is seen elsewhere in nature.

And perhaps because of this, it typically came to be assumed that a great many aspects of the natural world are simply beyond human understanding. But finally the successes based on calculus in the late 1600s began to overthrow this belief. For with calculus there was finally real success in taking abstract rules created by human thought and using them to reproduce all sorts of phenomena in the natural world.

But the particular rules that were found to work were fairly sophisticated ones based on particular kinds of mathematical equations. And from seeing the sophistication of these rules there began to develop an implicit belief that in almost no important cases would simpler rules be useful in reproducing the behavior of natural systems.

During the 1700s and 1800s there was ever-increasing success in using rules based on mathematical equations to analyze physical phenomena. And after the spectacular results achieved in physics in the early 1900s with mathematical equations there emerged an almost universal belief that absolutely every aspect of the natural world would in the end be explained by using such equations.

Needless to say, there were many phenomena that did not readily yield to this approach, but it was generally assumed that if only the necessary calculations could be done, then an explanation in terms of mathematical equations would eventually be found.

Beginning in the 1940s, the development of electronic computers greatly broadened the range of calculations that could be done. But disappointingly enough, most of the actual calculations that were tried yielded no fundamentally new insights. And as a result many people came to believe—and in some cases still believe today—that computers could never make a real contribution to issues of basic science.

But the crucial point that was missed is that computers are not just limited to working out consequences of mathematical equations. And indeed, what we have seen in this chapter is that there are fundamental discoveries that can be made if one just studies directly the behavior of even some of the very simplest computer programs.

In retrospect it is perhaps ironic that the idea of using simple programs as models for natural systems did not surface in the early days of computing. For systems like cellular automata would have been immensely easier to handle on early computers than mathematical equations were. But the issue was that computer time was an expensive commodity, and so it was not thought worth taking the risk of trying anything but well-established mathematical models.

By the end of the 1970s, however, the situation had changed, and large amounts of computer time were becoming readily available. And this is what allowed me in 1981 to begin my experiments on cellular automata.

There is, as I mentioned above, nothing in principle that requires one to use a computer to study cellular automata. But as a practical matter, it is difficult to imagine that anyone in modern times would have the patience to generate many pictures of cellular automata by hand. For it takes roughly an hour to make the picture on page 27 by hand, and it would take a few weeks to make the picture on page 29.

Yet even with early mainframe computers, the data for these pictures could have been generated in a matter of a few seconds and a few minutes respectively. But the point is that one would be very unlikely to discover the kinds of fundamental phenomena discussed in this chapter just by looking at one or two pictures. And indeed for me to do it certainly took carrying out quite large-scale computer experiments on a considerable number of different cellular automata.

If one already has a clear idea about the basic features of a particular phenomenon, then one can often get more details by doing fairly specific experiments. But in my experience the only way to find phenomena that one does not already know exist is to do very systematic and general experiments, and then to look at the results with as few preconceptions as possible. And while it takes only rather basic computer technology to make single pictures of cellular automata, it requires considerably more to do large-scale systematic experiments.

Indeed, many of my discoveries about cellular automata came as direct consequences of using progressively better computer technology.

As one example, I discovered the classification scheme for cellular automata with random initial conditions described at the beginning of Chapter 6 when I first looked at large numbers of different cellular automata together on high-resolution graphics displays. Similarly, I discovered the randomness of rule 30 (page 27) when I was in the process of setting up large simulations for an early parallel-processing computer. And in more recent years, I have discovered a vast range of new phenomena as a result of easily being able to set up large numbers of computer experiments in *Mathematica*.

Undoubtedly, therefore, one of the main reasons that the discoveries I describe in this chapter were not made before the 1980s is just that computer technology did not yet exist powerful enough to do the kinds of exploratory experiments that were needed.

But beyond the practicalities of carrying out such experiments, it was also necessary to have the idea that the experiments might be worth doing in the first place. And here again computer technology played a crucial role. For it was from practical experience in using computers that I developed much of the necessary intuition.

As a simple example, one might have imagined that systems like cellular automata, being made up of discrete cells, would never be able to reproduce realistic natural shapes. But knowing about computer displays it is clear that this is not the case. For a computer display, like a cellular automaton, consists of a regular array of discrete cells or pixels. Yet practical experience shows that such displays can produce quite realistic images, even with fairly small numbers of pixels.

And as a more significant example, one might have imagined that the simple structure of cellular automaton programs would make it straightforward to foresee their behavior. But from experience in practical computing one knows that it is usually very difficult to foresee what even a simple program will do. Indeed, that is exactly why bugs in programs are so common. For if one could just look at a program and immediately know what it would do, then it would be an easy matter to check that the program did not contain any bugs.

Notions like the difficulty of finding bugs have no obvious connection to traditional ideas in science. And perhaps as a result of this, even after computers had been in use for several decades, essentially none of this type of intuition from practical computing had found its way into basic science. But in 1981 it so happened that I had for some years been deeply involved in both practical computing and basic science, and I was therefore in an almost unique position to apply ideas derived from practical computing to basic science.

Yet despite this, my discoveries about cellular automata still involved a substantial element of luck. For as I mentioned on page 19, my very first experiments on cellular automata showed only very simple behavior, and it was only because doing further experiments was technically very easy for me that I persisted.

And even after I had seen the first signs of complexity in cellular automata, it was several more years before I discovered the full range of examples given in this chapter, and realized just how easily complexity could be generated in systems like cellular automata.

Part of the reason that this took so long is that it involved experiments with progressively more sophisticated computer technology. But the more important reason is that it required the development of new intuition. And at almost every stage, intuition from traditional science took me in the wrong direction. But I found that intuition from practical computing did better. And even though it was sometimes misleading, it was in the end fairly important in putting me on the right track.

Thus there are two quite different reasons why it would have been difficult for the results in this chapter to be discovered much before computer technology reached the point it did in the 1980s. First, the necessary computer experiments could not be done with sufficient ease that they were likely to be tried. And second, the kinds of intuition about computation that were needed could not readily have been developed without extensive exposure to practical computing.

But now that the results of this chapter are known, one can go back and see quite a number of times in the past when they came at least somewhat close to being discovered.

It turns out that two-dimensional versions of cellular automata were already considered in the early 1950s as possible idealized models for biological systems. But until my work in the 1980s the actual investigations of cellular automata that were done consisted mainly in constructions of rather complicated sets of rules that could be shown to lead to specific kinds of fairly simple behavior.

The question of whether complex behavior could occur in cellular automata was occasionally raised, but on the basis of intuition from engineering it was generally assumed that to get any substantial complexity, one would have to have very complicated underlying rules. And as a result, the idea of studying cellular automata with simple rules never surfaced, with the result that nothing like the experiments described in this chapter were ever done.

In other areas, however, systems that are effectively based on simple rules were quite often studied, and in fact complex behavior was sometimes seen. But without a framework to understand its significance, such behavior tended either to be ignored entirely or to be treated as some kind of curiosity of no particular fundamental significance.

Indeed, even very early in the history of traditional mathematics there were already signs of the basic phenomenon of complexity. One example known for well over two thousand years concerns the distribution of prime numbers (see page 132). The rules for generating primes are simple, yet their distribution seems in many respects random. But almost without exception mathematical work on primes has concentrated not on this randomness, but rather on proving the presence of various regularities in the distribution.

Another early sign of the phenomenon of complexity could have been seen in the digit sequence of a number like $\pi \simeq 3.141592653\ldots$ (see page 136). By the 1700s more than a hundred digits of π had been computed, and they appeared quite random. But this fact was treated essentially as a curiosity, and the idea never appears to have arisen that there might be a general phenomenon whereby simple rules like those for computing π could produce complex results.

In the early 1900s various explicit examples were constructed in several areas of mathematics in which simple rules were repeatedly applied to numbers, sequences or geometrical patterns. And sometimes nested or fractal behavior was seen. And in a few cases substantially more complex behavior was also seen. But the very complexity of this behavior was usually taken to show that it could not be relevant for real mathematical work—and could only be of recreational interest.

When electronic computers began to be used in the 1940s, there were many more opportunities for the phenomenon of complexity to be seen. And indeed, looking back, significant complexity probably did occur in many scientific calculations. But these calculations were almost always based on traditional mathematical models, and since previous analyses of these models had not revealed complexity, it tended to be assumed that any complexity in the computer calculations was just a spurious consequence of the approximations used in them.

One class of systems where some types of complexity were noticed in the 1950s are so-called iterated maps. But as I will discuss on page 149, the traditional mathematics that was used to analyze such systems ended up concentrating only on certain specific features, and completely missed the main phenomenon discovered in this chapter.

It is often useful in practical computing to produce sequences of numbers that seem random. And starting in the 1940s, several simple procedures for generating such sequences were invented. But perhaps because these procedures always seemed quite ad hoc, no general conclusions about randomness and complexity were drawn from them.

Along similar lines, systems not unlike the cellular automata discussed in this chapter were studied in the late 1950s for generating random sequences to be used in cryptography. Almost all the results that were obtained are still military secrets, but I do not believe that any phenomena like the ones described in this chapter were discovered.

And in general, within the context of mainstream science, the standard intuition that had been developed made it very difficult for anyone to imagine that it would be worth studying the behavior of the very simple kinds of computer programs discussed in this chapter. But outside of mainstream science, some work along such lines was done. And for example in the 1960s early computer enthusiasts tried running various simple programs, and found that in certain cases these programs could succeed in producing nested patterns.

Then in the early 1970s, considerable recreational computing interest developed in a specific two-dimensional cellular automaton known as the Game of Life, whose behavior is in some respects similar to the rule 110 cellular automaton discussed in this chapter. Great effort was spent trying to find structures that would be sufficiently simple and predictable that they could be used as idealized components for engineering. And although complex behavior was seen it was generally treated as a nuisance, to be avoided whenever possible.

In a sense it is surprising that so much could be done on the Game of Life without the much simpler one-dimensional cellular automata in this chapter ever being investigated. And no doubt the lack of a connection to basic science was at least in part responsible.

But whatever the reasons, the fact remains that, despite many hints over the course of several centuries, the basic phenomenon that I have described in this chapter was never discovered before.

It is not uncommon in the history of science that once a general new phenomenon has been identified, one can see that there was already evidence of it much earlier. But the point is that without the framework that comes from knowing the general phenomenon, it is almost inevitable that such evidence will have been ignored.

It is also one of the ironies of progress in science that results which at one time were so unexpected that they were missed despite many hints eventually come to seem almost obvious. And having lived with the results of this chapter for nearly two decades, it is now difficult for me to imagine that things could possibly work in any other way. But the history that I have outlined in this section—like the history of many other scientific discoveries—provides a sobering reminder of just how easy it is to miss what will later seem obvious.

The Crucial Experiment

How Do Simple Programs Behave?

▪ **Implementing cellular automata.** It is convenient to represent the state of a cellular automaton at each step by a list such as `{0, 0, 1, 0, 0}`, where `0` corresponds to a white cell and `1` to a black cell. An initial condition consisting of *n* white cells with one black cell in the middle can then be obtained with the function (see below for comments on this and other *Mathematica* functions)

```
CenterList[n_Integer] :=
   ReplacePart[Table[0, {n}], 1, Ceiling[n/2]]
```

For cellular automata of the kind discussed in this chapter, the rule can also be represented by a list. Thus, for example, rule 30 on page 27 corresponds to the list `{0, 0, 0, 1, 1, 1, 1, 0}`. (The numbering of rules is discussed on page 53.) In general, the list for a particular rule can be obtained with the function

```
ElementaryRule[num_Integer] := IntegerDigits[num, 2, 8]
```

Given a rule together with a list representing the state *a* of a cellular automaton at a particular step, the following simple function gives the state at the next step:

```
CAStep[rule_List, a_List] :=
   rule[[8 - (RotateLeft[a] + 2 (a + 2 RotateRight[a]))]]
```

A list of states corresponding to evolution for *t* steps can then be obtained with

```
CAEvolveList[rule_, init_List, t_Integer] :=
   NestList[CAStep[rule, #] &, init, t]
```

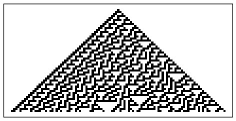
Graphics of this evolution can be generated using

```
CAGraphics[history_List] := Graphics[
      Raster[1 - Reverse[history]] AspectRatio → Automatic]
```

And having set up the definitions above, the *Mathematica* input

```
Show[CAGraphics[CAEvolveList[
        ElementaryRule[30], CenterList[103], 50]]]
```

will generate the image:



The description just given should be adequate for most cellular automaton simulations. In some earlier versions of *Mathematica* a considerably faster version of the program can be created by using the definition

```
CAStep = Compile[{{rule, _Integer, 1}, {a, _Integer, 1}},
      rule[[8 - (RotateLeft[a] + 2 (a + 2 RotateRight[a]))]]]
```

In addition, in *Mathematica* 4 and above, one can use

```
CAStep[rule_, a_] := rule[[8 - ListConvolve[{1, 2, 4}, a, 2]]]
```

or directly in terms of the rule number *num*

```
Sign[BitAnd[2^ListConvolve[{1, 2, 4}, a, 2], num]]
```

(In versions of *Mathematica* subsequent to the release of this book the built-in *CellularAutomaton* function can be used, as discussed on page 867.) It is also possible to have *CAStep* call the following external C language program via *MathLink*—though typically with successive versions of *Mathematica* the speed advantage obtained will be progressively less significant:

```
#include "mathlink.h"

main(argc, argv)
int argc; char *argv[];
{
MLMain(argc, argv);
}

void casteps(revrule, rlen, a, n, steps)
int *revrule, rlen, *a, n, steps;
{
int i, *ap, t, tp;

for (i = 0; i <steps; i++)
   {
   a[0] = a[n-2]; /* right boundary */
   a[n-1] = a[1]; /* left boundary */

   t = a[0];
   for (ap = a+1; ap <= a+n-2; ap++)
      {
      tp = ap[0];
      ap[0] = revrule[ap[1]+2*(tp + 2*t)];
      t = tp;
      }
   }

MLPutIntegerList(stdlink, a, n);
}
```

The linkage of this external program to the *Mathematica* function *CAStep* is achieved with the following *MathLink* template (note the optional third argument which allows

CAStep to perform several steps of cellular automaton evolution at a time):

```
:Begin:
:Function: casteps
:Pattern: CAStep[rule_List, a_List, steps_Integer:1]
:Arguments: {Reverse[rule], a, steps}
:ArgumentTypes: {IntegerList, IntegerList, Integer}
:ReturnType: Manual
:End:
```

There are a couple of tricky issues in the C program above. First, cellular automaton rules are always defined to use the old values of neighbors in determining the new value of any particular cell. But since the C program explicitly updates values sequentially from left to right, the left-hand neighbor of a particular cell will already have been given its new value when one tries to updates the cell itself. As a result, it is necessary to store the old value of the left-hand neighbor in a temporary variable in order to make it available for updating the cell itself. (Another approach to this problem is to maintain two copies of the array of cells, and to interchange pointers to them after every step in the cellular automaton evolution.)

Another tricky point in cellular automaton programs concerns boundary conditions. Since in a practical computer one can use only a finite array of cells, one must decide how the cellular automaton rule is to be applied to the cells at each end of the array. In both the *Mathematica* and the C programs above, we effectively use a cyclic array, in which the left neighbor of the leftmost cell is taken to be rightmost cell, and vice versa. In the C program, this is implemented by explicitly copying the value of the leftmost cell to the rightmost position in the array, and vice versa, before updating the values in the array. (In a sense there is a bug in the program in that the update only puts new values into *n - 2* of the *n* array elements.)

▪ **Comments on *Mathematica* functions.** *CenterList* works by first creating a list of *n* 0's, then replacing the middle 0 by a 1. (In *Mathematica* 4 and above *PadLeft[{1}, n, 0, Floor[n/2]]* can be used instead.) *ElementaryRule* works by converting *num* into a base 2 digit sequence, padding with zeros on the left so as to make a list of length 8. The scheme for numbering rules works so that if the value of a particular cell is *q*, the value of its left neighbor is *p*, and the value of its right neighbor is *r*, then the element at position *8 - (r + 2 (q + 2 p))* in the list obtained from *ElementaryRule* will give the new value of the cell.

CAStep uses the fact that *Mathematica* can manipulate all the elements in a list at once. *RotateLeft[a]* and *RotateRight[a]* make shifted versions of the original list of cell values *a*. Then when these lists are added together, their corresponding elements are combined, as in *{p, q, r} + {s, t, u}⟶{p + s, q + t, r + u}*. The result is that a list is produced which specifies for each cell which element of the rule applies to that cell. The actual list of new cell values is then generated by using the fact that *{i, j, k}[[{2, 1, 1, 3, 2}]]⟶{j, i, i, k, j}*. Note that by using *RotateLeft* and *RotateRight* one automatically gets cyclic boundary conditions.

CAEvolveList applies *CAStep* *t* times. Many other evolution functions in these notes use the same mechanism. In general *NestList[s[r, #] &, i, 2]⟶{i, s[r, i], s[r, s[r, i]]}*, etc.

▪ **Bitwise optimizations.** The C program above stores each cell value in a separate element of an integer array. But since every value must be either 0 or 1, it can in fact be encoded by just a single bit. And since integer variables in practical computers typically involve 32 or 64 bits, the values of many cells can be packed into a single integer variable. The main point of this is that typical machine instructions operate in parallel on all the bits in such a variable. And thus for example the values of all cells represented by an integer variable *a* can be updated in parallel according to rule 30 by the single C statement

```
a = a>>1 ^ (a | a<<1);
```

This statement, however, will only update the specific block of cells encoded in *a*. Gluing together updates to a sequence of such blocks requires slightly intricate code. (It is much easier to implement in *Mathematica*—as discussed above—since there functions like *BitXor* can operate on integers of any length.) In general, bitwise optimizations require representing cellular automaton rules not by simple look-up tables but rather by Boolean expressions, which must be derived for each rule and can be quite complicated (see page 869). Applying the rules can however be made faster by using bitslicing to avoid shift operations. The idea is to store the cellular automaton configuration in, say, *m* variables *w[i]* whose bits correspond respectively to the cell values $\{a_1, a_{m+1}, a_{2m+1}, \ldots\}$, $\{a_2, a_{m+2}, a_{2m+2}, \ldots\}$, $\{a_3, \ldots\}$, etc. This then makes the left and right neighbors of the j^{th} bit in *w[i]* be the j^{th} bits in *w[i - 1]* and *w[i + 1]*—so that for example a step of rule 30 evolution can be achieved just by *w[i] = w[i - 1]^(w[i] | w[i + 1])* with no shift operations needed (except in boundary conditions on *w[0]* and *w[m - 1]*). If many steps of evolution are required, it is sufficient just to pack all cell values at the beginning, and unpack them at the end.

▪ **More general rules.** The programs given so far are for cellular automata with rules of the specific kind described in this chapter. In general, however, a 1D cellular automaton rule can be given as a set of explicit replacements for all possible blocks of cells in each neighborhood (see page 60). Thus, for example, rule 30 can be given as

```
{{1, 1, 1}→ 0, {1, 1, 0}→ 0, {1, 0, 1}→ 0, {1, 0, 0}→ 1,
  {0, 1, 1}→ 1, {0, 1, 0}→ 1, {0, 0, 1}→ 1, {0, 0, 0}→ 0}
```

To use rules in this form, *CAStep* can be rewritten as

```
CAStep[rule_, a_List] :=
  Transpose[{RotateRight[a], a, RotateLeft[a]}] /. rule
```

or

```
CAStep[rule_, a_List] := Partition[a, 3, 1, 2] /. rule
```

The rules that are given can now contain patterns, so that rule 90, for example, can be written as

```
{{1, _, 1}→ 0, {1, _, 0}→ 1, {0, _, 1}→ 1, {0, _, 0}→ 0}
```

But how can one set up a program that can handle rules in several different forms? A convenient approach is to put a "wrapper" around each rule that specifies what form the rule is in. Then, for example, one can define

```
CAStep[ElementaryCARule[rule_List], a_List] :=
  rule[[8 - (RotateLeft[a] + 2 (a + 2 RotateRight[a]))]]
CAStep[GeneralCARule[rule_, r_Integer : 1], a_List] :=
  Partition[a, 2 r + 1, 1, r + 1] /. rule
CAStep[FunctionCARule[f_, r_Integer : 1], a_List] :=
  Map[f, Partition[a, 2 r + 1, 1, r + 1]]
```

Note that the second two definitions have been generalized to allow rules that involve *r* neighbors on each side. In each case, the use of *Partition* could be replaced by *Transpose[Table[RotateLeft[a, i], {i, -r, r}]]*. For efficiency in early versions of *Mathematica*, explicit rule lists in the second definition can be preprocessed using *Dispatch[rules]*, and functions in the third definition preprocessed using *Compile[{{x, _Integer, 1}}, body]*.

I discuss the implementation of totalistic cellular automata on page 886, and of higher-dimensional cellular automata on page 927.

■ **Built-in cellular automaton function.** Versions of *Mathematica* subsequent to the release of this book will include a very general function for cellular automaton evolution. The description is as follows (see also page 886):

CellularAutomaton[rnum, init, t] generates a list representing the evolution of cellular automaton rule *rnum* from initial condition *init* for *t* steps.

CellularAutomaton[rnum, init, t, $\{\mathit{off}_1, \mathit{off}_x, \ldots\}$*]* keeps only the parts of the evolution list with the specified offsets.

Possible settings for *rnum* are:

n	$k = 2, r = 1$, elementary rule
{n, k}	general nearest-neighbor rule with *k* colors
{n, k, r}	general rule with *k* colors and range *r*
$\{n, k, \{r_1, r_2, \ldots, r_d\}\}$	*d*-dimensional rule with $(2 r_1 + 1) \times (2 r_2 + 1) \times \ldots \times (2 r_d + 1)$ neighborhood
$\{n, k, \{\{\mathit{off}_1\}, \{\mathit{off}_2\}, \ldots, \{\mathit{off}_s\}\}\}$	rule with neighbors at specified offsets
{n, {k, 1}}	*k*-color nearest-neighbor totalistic rule
{n, {k, 1}, r}	*k*-color range *r* totalistic rule
$\{n, \{k, \{wt_1, wt_2, \ldots\}\}, rspec\}$	rule in which neighbor *i* is assigned weight wt_i
{fun, {}, rspec}	applies the function *fun* to each list of neighbors, with a second argument of the step number

■ *CellularAutomaton[{n, k}, …]* is equivalent to *CellularAutomaton[{n, {k,* $\{k^2, k, 1\}\}\}, \ldots]$. ■ Common forms for 2D cellular automata include:

{n, {k, 1}, {1, 1}}	9-neighbor totalistic rule
{n, {k, {{0, 1, 0}, {1, 1, 1}, {0, 1, 0}}, {1, 1}}}	5-neighbor totalistic rule
{n, {k, {{0, k, 0}, {k, 1, k}, {0, k, 0}}, {1, 1}}}	5-neighbor outer totalistic rule
$\{n + k^5 (k - 1), \{k, \{\{0, 1, 0\}, \{1, 4 k + 1, 1\}, \{0, 1, 0\}\}, \{1, 1\}\}\}$	5-neighbor growth rule

■ Normally, all elements in *init* and the evolution list are integers between *0* and *k-1*. ■ But when a general function is used, the elements of *init* and the evolution list do not have to be integers. ■ The second argument passed to *fun* is the step number, starting at *0*. ■ Initial conditions are constructed from *init* as follows:

$\{a_1, a_2, \ldots\}$	explicit list of values a_i, assumed cyclic
$\{\{a_1, a_2, \ldots\}, b\}$	values a_i superimposed on a *b* background
$\{\{a_1, a_2, \ldots\}, \{b_1, b_2, \ldots\}\}$	values a_i superimposed on a background of repetitions of $b_1, b_2, \ldots$
$\{\{\{\{a_{11}, a_{12}, \ldots\}, \mathit{off}_1\}, \{a_{21}, \ldots\}, \mathit{off}_2\}, \ldots\}, bspec\}$	values a_{ij} at offsets off_i on a background
$\{\{a_{11}, a_{12}, \ldots\}, \{a_{21}, \ldots\}, \ldots\}$	explicit list of values in two dimensions
{aspec, bspec}	values in *d* dimensions with *d*-dimensional padding

■ The first element of *aspec* is superimposed on the background at the first position in the positive direction in each coordinate relative to the origin. This means that *bspec[[1,1,…]]* is aligned with *aspec[[1, 1,…]]*. ■ Time offsets off_t are specified as follows:

All	all steps *0* through *t* (default)
u	steps *0* through *u*
-1	last step (step *t*)
{u}	step *u*
$\{u_1, u_2\}$	steps u_1 through u_2
$\{u_1, u_2, du\}$	steps $u_1, u_1 + du, \ldots$

■ *CellularAutomaton[rnum, init, t]* generates an evolution list of length *t*+1. ■ The initial condition is taken to have offset *0*. ■ Space offsets off_x are specified as follows:

All	all cells that can be affected by the specified initial condition
Automatic	all cells in the region that differs from the background
0	cell aligned with beginning of *aspec*
x	cells at offsets up to *x* on the right
-x	cells at offsets up to *x* on the left
{x}	cell at offset *x* to the right
{-x}	cell at offset *x* to the left
$\{x_1, x_2\}$	cells at offsets x_1 through x_2
$\{x_1, x_2, dx\}$	cells $x_1, x_1 + dx, \ldots$

■ In one dimension, the first element of *aspec* is taken by default to have space offset *0*. ■ In any number of dimensions, *aspec[[1, 1, 1,…]]* is taken by default to have space offset *{0, 0, 0,…}*. ■ Each element of the evolution list produced by *CellularAutomaton* is always the same size. ■ With an initial condition specified by an *aspec* of width *w*, the region that can be affected after *t* steps by a cellular automaton with a rule of range r has width $w + 2 r t$. ▪ If no *bspec* background is specified, space offsets of *All* and *Automatic* will include every cell in *aspec*. ▪ A space offset of *All* includes all cells that can be affected by the initial condition. ▪ A space offset of *Automatic* can be used to trim off background from the sides of a cellular automaton pattern. ▪ In working out how wide a region to keep, *Automatic* only looks at results on steps specified by off_t.

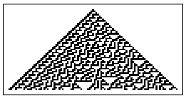
Some examples include:

This gives the array of values obtained by running rule 30 for 3 steps, starting from an initial condition consisting of a single 1 surrounded by 0's.

```
In[1] := CellularAutomaton[30, {{1}, 0}, 3]
Out[1]= {{0, 0, 0, 1, 0, 0, 0}, {0, 0, 1, 1, 1, 0, 0}, {0, 1, 1, 0, 0, 1, 0}, {1, 1, 0, 1, 1, 1, 1}}
```

This runs rule 30 for 50 steps and makes a picture of the result.

```
In[2] := Show[RasterGraphics[CellularAutomaton[30, {{1}, 0}, 50]]]
```

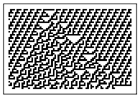


If all values in the initial condition are given explicitly, they are in effect assumed to continue cyclically. The runs rule 30 with 5 cells for 3 steps.

```
In[3] := CellularAutomaton[30, {1, 0, 0, 1, 0}, 3]
Out[3]= {{1, 0, 0, 1, 0}, {1, 1, 1, 1, 0}, {1, 0, 0, 0, 0}, {1, 1, 0, 0, 1}}
```

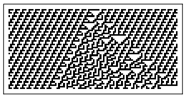
This starts from *{1,1}* on an infinite background of repeating *{1,0,1,1}* blocks. By default, only the region of the pattern affected by the *{1,1}* is given.

```
In[4] := Show[RasterGraphics[CellularAutomaton[30, {{1, 1}, {1, 0, 1, 1}}, 50]]]
```



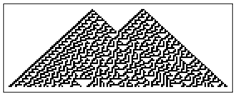
This gives all cells that could possibly be affected, whether or not they are.

```
In[5] := Show[RasterGraphics[CellularAutomaton[30,
                        {{1, 1}, {1, 0, 1, 1}}, 50, {All, All}]]]
```



This places blocks in the initial conditions at offsets -10 and 20.

```
In[6] := Show[RasterGraphics[CellularAutomaton[30,
                        {{{{1}, {-10}}, {{1, 1}, {20}}}, 0}, 50]]]
```



This gives only the last row after running for 10 steps.

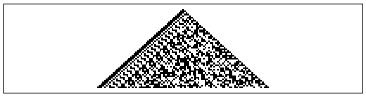
```
In[7] := CellularAutomaton[30, {{1}, 0}, 10, -1]
Out[7]= {{1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0}}
```

This runs for 5 steps, giving the cells on the 3 center columns at each step.

```
In[8] := CellularAutomaton[30, {{1}, 0}, 5, {All, {-1, 1}}]
Out[8]= {{0, 1, 0}, {1, 1, 1}, {1, 0, 0}, {0, 1, 1}, {0, 1, 0}, {1, 1, 1}}
```

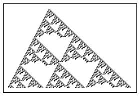
This picks out every other cell in space and time, starting 200 cells to the left.

```
In[9] := Show[RasterGraphics[CellularAutomaton[30, {{1}, 0}, 100,
                        {{1, 100, 2}, {-200, 200, 2}}]]]
```



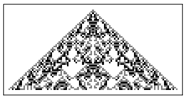
This runs the general $k=3$, $r=1$ rule with rule number 921408.

```
In[10] := Show[RasterGraphics[CellularAutomaton[{921408, 3, 1}, {{1}, 0}, 100]]]
```



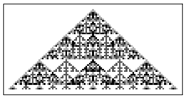
This runs the totalistic $k=3$, $r=1$ rule with code 867.

```
In[11] := Show[RasterGraphics[CellularAutomaton[{867, {3, 1}, 1}, {{1}, 0}, 50]]]
```



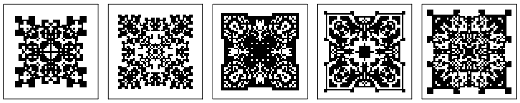
This uses a rule based on applying a function to each neighborhood of cells.

```
In[12] := Show[RasterGraphics[CellularAutomaton[
                        {Mod[Apply[Plus, #], 4] &, {}, 1}, {{1}, 0}, 50]]]
```



This runs 2D 9-neighbor totalistic code 3702 for 25 steps, giving the results for the last 5 steps.

```
In[13] := Show[GraphicsArray[Map[RasterGraphics,
                CellularAutomaton[{3702, {2, 1}, {1, 1}}, {{{1}}, 0}, 25, -5]]]]
```



▪ **Special-purpose hardware.** The simple structure of cellular automata makes it natural to think of implementing them with special-purpose hardware. And indeed from the 1950s on, a sequence of special-purpose machines have been built to implement 1D, 2D and sometimes 3D cellular automata. Two basic ideas have been used: parallelism and pipelines. Both ideas rely on the local nature of cellular automaton rules.

In the parallel approach, the machine has many separate processors, each dedicated to handling a single cell or a small group of cells. In the pipelined approach, there is just a single processor (or perhaps a few processors) through which the data on different cells is successively piped. The key point, however, is that at every stage it is easy to know what data will be needed, so this data can be prefetched, potentially through a specially built memory system.

In general, the speed increases that can be achieved depend on many details of memory and communications architecture. The increases have tended to become less significant over the years, as the on-chip memories of microprocessors have become larger, and the time necessary to send data from one chip to another has become proportionately more important.

In the future, however, new technologies may change the trade-offs, and indeed cellular automata are obvious candidates for early implementation in both nanotechnology and optical computing. (See also page 841.)

▪ **Audio representation.** A step in the evolution of a cellular automaton can be represented as a sound by treating each cell like a key on a piano, with the key taken to be pressed if the cell is black. This yields a chord such as

`Play[Evaluate[Apply[Plus, Flatten[Map[Sin[1000 # t] &,`
`N[`$2^{1/12}$`]^Position[list, 1]]]]], {t, 0, 0.2}]`

A sequence of such chords can sometimes provide a useful representation of cellular automaton evolution. (See also page 1080.)

▪ **Cellular automaton rules as formulas.** The value `a[t, i]` for a cell on step `t` at position `i` in any of the cellular automata in this chapter can be obtained from the definition

```
a[t_, i_] := f[a[t - 1, i - 1], a[t - 1, i], a[t - 1, i + 1]]
```

Different rules correspond to different choices of the function `f`. For example, rule 90 on page 25 corresponds to

```
f[1, _, 1] = 0; f[0, _, 1] = 1; f[1, _, 0] = 1; f[0, _, 0] = 0
```

One can specify initial conditions for example by

```
a[0, 0] = 1; a[0, _] = 0
```

(the cell on step 0 at position 0 has value 1, but all other cells on that step have value 0). Then just asking for `a[4, 0]` one will immediately get the value after 4 steps of the cell at position 0. (For efficiency, the main definition should in practice be given as

```
a[t_, i_] := a[t, i] = f[a[t - 1, i - 1], a[t - 1, i], a[t - 1, i + 1]]
```

so that all intermediate values which are computed are automatically stored.)

The definition of the function `f` for rule 90 that we gave above is essentially just a look-up table. But it is also possible to define this function in an algebraic way

```
f[p_, q_, r_] := Mod[p + r, 2]
```

Algebraic definitions can also be given for other rules:

- Rule 254 (page 24): `1 - (1 - p) (1 - q) (1 - r)`
- Rule 250 (page 25): `p + r - p r`
- Rule 30 (page 27): `Mod[p + q + r + q r, 2]`
- Rule 110 (page 32): `Mod[(1 + p) q r + q + r, 2]`

In these definitions, we represent the values of cells by the numbers 1 or 0. If values +1 and -1 are used instead, different formulas are obtained; rule 90, for example, corresponds to `p r`. It is also possible to represent values of cells as `True` and `False`. And in this case cellular automaton rules become logic expressions:

- Rule 254: `Or[p, q, r]`
- Rule 250: `Or[p, r]`
- Rule 90: `Xor[p, r]`
- Rule 30: `Xor[p, Or[q, r]]`
- Rule 110: `Xor[Or[p, q], And[p, q, r]]`

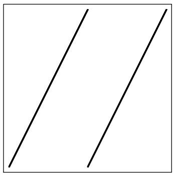
(Note that `Not[p]` corresponds to `1 - p`, `And[p, q]` to `p q`, `Xor[p, q]` to `Mod[p + q, 2]` and `Or[p, q]` to `Mod[p q + p + q, 2]`.)

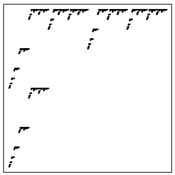
Given either the algebraic or logical form of a cellular automaton rule, it is possible at least in principle to generate symbolic formulas for the results of cellular automaton evolution. Thus, for example, one can use initial conditions

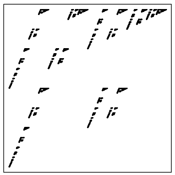
```
a[0, -1] = p; a[0, 0] = q; a[0, 1] = r; a[0, _] = 0
```

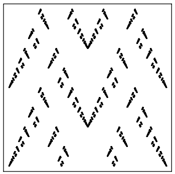
to generate a formula for the value of a cell that holds for any choice of values for the three initial center cells. In practice, however, most such formulas rapidly become very complicated, as discussed on page 618.

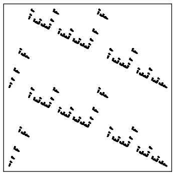
▪ **Mathematical interpretation of cellular automata.** In the context of pure mathematics, the state space of a 1D cellular automaton with an infinite number of cells can be viewed as a Cantor set. The cellular automaton rule then corresponds to a continuous mapping of this Cantor set to itself (continuity follows from the locality of the rule). (Compare page 959.)

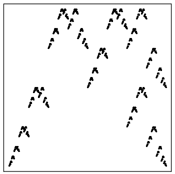

rule 170


rule 254


rule 250


rule 90


rule 30


rule 110

The pictures above show representations of the mappings corresponding to various rules, obtained by plotting `Sum[a[t + 1, i]` 2^{-i}`, {i, -n, n}]` against `Sum[a[t, i]` 2^{-i}`, {i, -n, n}]` for all possible choices of the *a[t, i]*. (Periodic boundary conditions are used, so that the *a[t, i]* can be viewed as corresponding precisely to digits of rational numbers.) Rule 170 is the classic shift map which shifts all cell values one position to the left without changing them. In the pictures below, this map has the form *Mod[2 x, 1]* (compare page 153).

■ **Page 26 · Pascal's triangle and rule 90.** As shown on page 611 the pattern produced by rule 90 is exactly Pascal's triangle of binomial coefficients reduced modulo 2: black cells correspond to odd binomial coefficients.

The number of black cells on row *t* is given by *2^DigitCount[t, 2, 1]*, where *DigitCount[t, 2, 1]* is plotted on page 902. The positions of the black cells are given by (and this establishes the connection with the picture on page 117)

```
Fold[Flatten[{#1 - #2, #1 + #2}] &, 0, 2^DigitPositions[t]]

DigitPositions[n_] :=
  Flatten[Position[Reverse[IntegerDigits[n, 2]], 1]] - 1
```

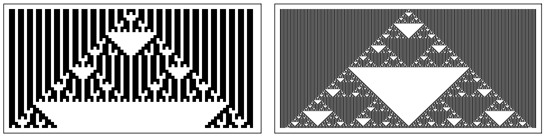
The actual pattern generated by rule 90 corresponds to the coefficients in *PolynomialMod[Expand[(1/x + x)*^{t}*], 2]* (see page 1091); the color of a particular cell is thus given by *Mod[Binomial[t, (n + t)/2], 2] /; EvenQ[n + t]*.

Mod[Binomial[t, n], 2] yields a distorted pattern that is the one produced by rule 60 (see page 58). In this pattern, the color of a particular cell can be obtained directly from the digit sequences for *t* and *n* by *1 - Sign[BitAnd[-t, n]]* or (see page 583)

```
With[{d = Ceiling[Log[2, Max[t, n] + 1]]}, If[FreeQ[
    IntegerDigits[t, 2, d] - IntegerDigits[n, 2, d], -1], 1, 0]]
```

■ **Self-similarity.** The pattern generated by rule 90 after a given number of steps has the property that it is identical to what one would get by going twice as many steps, and then keeping only every other row and column. After 2^m steps the triangular region outlined by the pattern contains altogether 4^m cells, but only 3^m of these are black. In the limit of an infinite number of steps one gets a fractal known as a Sierpiński pattern (see page 934), with fractal dimension *Log[2, 3] ≃ 1.59* (see page 933). Nesting occurs in all cellular automata with additive rules (see page 955).

■ **Another initial condition.** Inserting a single ■ in a background of ■□ blocks in rule 90 yields the pattern below in which both the white and striped regions have fractal dimension 2.



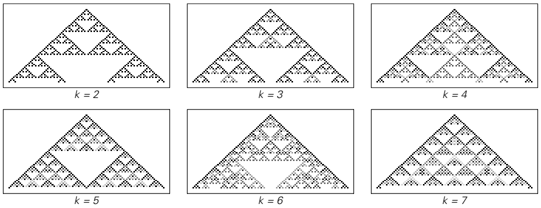
■ **More colors.** The pictures below show generalizations of rule 90 to *k* possible colors using the rule

```
CAStep[k_Integer, a_List] :=
  Mod[RotateLeft[a] + RotateRight[a], k]
```

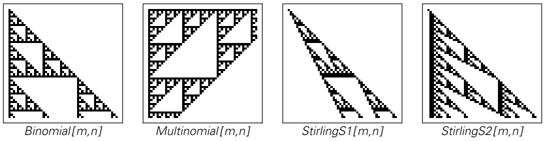
or equivalently *Mod[ListCorrelate[{1, 0, 1}, a, 2], k]*. The number of cells that are not white on row *t* in this case is given by *Apply[Times, 1 + IntegerDigits[t, k]]*. (For non-prime *k*, the patterns are obtained by superimposing the patterns corresponding to the factors of *k*.) A related result is that *IntegerExponent[Binomial[t, n], k]* is given by the number of borrows in the base *k* subtraction of *n* from *t*. *Mod[Binomial[t, n], k]* is given for prime *k* by

```
With[{d = Ceiling[Log[k, Max[t, n] + 1]]},
 Mod[Apply[Times, Apply[Binomial, Transpose[
     {IntegerDigits[t, k, d], IntegerDigits[n, k, d]}], {1}]], k]]
```

The patterns obtained for any *k* are nested. For prime *k* the total number of non-white cells down to step k^m is *(1/2 k (k + 1))*^{m} and the patterns have fractal dimension *1 + Log[k, (k + 1)/2]* (see page 955). These are examples of additive rules, discussed further on page 952. (See also page 922 for the continuous case.)

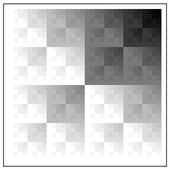

k = 2 k = 3 k = 4
k = 5 k = 6 k = 7

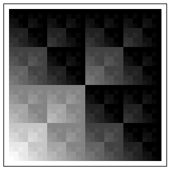
■ **History.** Pascal's triangle probably dates from antiquity; it was known in China in the 1200s, and was discussed in some detail by Blaise Pascal in 1654, particularly in connection with probability theory. The digit-based approach to finding binomial coefficients modulo *k* has been invented independently many times since the mid-1800s, notably by Edouard Lucas in 1877 and James Glaisher in 1899. The fact that the odd binomial coefficients form a nested geometrical pattern had apparently not been widely noticed before I emphasized it in 1982.

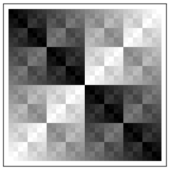

Binomial[m,n] *Multinomial[m,n]* *StirlingS1[m,n]* *StirlingS2[m,n]*

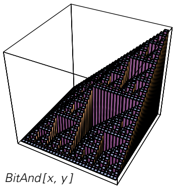
■ **Other integer functions.** The pictures above show patterns produced by reducing several integer functions modulo 2. With *d* arguments *Multinomial* yields a nested pattern in *d* dimensions. Note that *GCD[m, n]* yields a more complicated pattern (see page 613), as do `JacobiSymbol[m, 2n - 1]` (see page 1081) and various combinations of functions (see page 747).

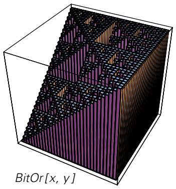
- **Bitwise functions.** Bitwise functions typically yield nested patterns. (As discussed above, any cellular automaton rule can be represented as an appropriate combination of bitwise functions.) Note that `BitOr[x, y] + BitAnd[x, y] == x + y` and `BitOr[x, y] - BitAnd[x, y] == BitXor[x, y]`.

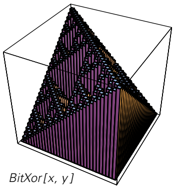

BitAnd[x, y]


BitOr[x, y]

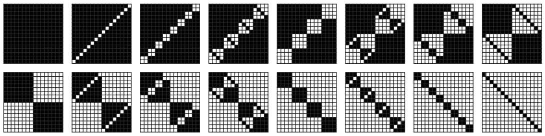

BitXor[x, y]


BitAnd[x, y]

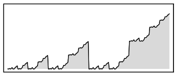

BitOr[x, y]

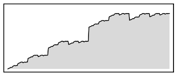

BitXor[x, y]

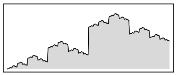
The patterns below show where `BitXor[x, y] == t` for successive `t` and correspond to steps in the "munching squares" program studied on the PDP-1 computer in 1962.



Nesting is also seen in curves obtained by applying bitwise functions to `n` and `2n` for successive `n`. Note that `2n` has the same digits as `n`, but shifted one position to the left.

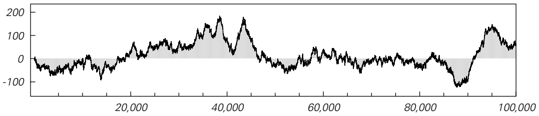

BitAnd[n, 2 n]


BitOr[n, 2 n]


BitXor[n, 2 n]

- **Page 28 · Tests of randomness.** The statistical tests that I have performed include the eight listed on page 1084.

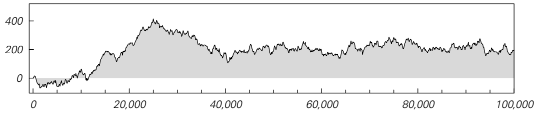
- **Page 29 · Rule 30.** The left-hand side of the pattern shown has an obvious repetitive character. In general, if one looks along a diagonal n cells in from either edge of the pattern, then the period of repetition can be at most 2^n. On the right-hand edge, the first few periods that are seen are `{1, 2, 2, 4, 8, 8, 16, 32, 32, 64, 64, 64, 64, 64, 128, 256}` and in general the period seems to increase exponentially with depth. On the left-hand edge, the period increases only extremely slowly: period 2 is first achieved at depth 3, period 4 at depth 8, 8 at 29, 16 at 400, 32 at 87,867, 64 at 2,107,985,255 or more, and so on. (Each period doubling turns out to occur exactly when a diagonal in the pattern eventually becomes a white stripe, and the diagonal to its left has an odd number of black cells in each repeating block.) The boundary that separates repetition on the left from randomness on the right moves an average of about 0.252 cells to the left at every step (compare page 949). The picture below shows the fluctuations around this average.

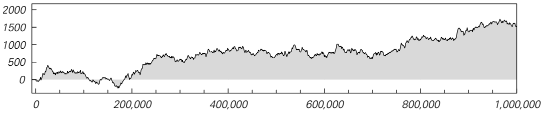


Complete pattern. All possible blocks appear to occur eventually (see page 725). The probability for a block of n adjacent white cells (corresponding to a row in a white triangle) seems quite accurately to approach 2^{-n}, with the first length 10 such block occurring at step 67 and the first length 20 one occurring at step 515.

Center column. The pictures below show the excess of black over white cells in the center column. Out of the first 100,000 cells, a total of 50,098 are black, and out of the first million 500,768 are. The longest run of identical colors in the first 100,000 cells consists of 21 black cells, and in the first million elements 22 black cells. The first n elements can be found efficiently using

```
Module[{a = 1}, Table[First[IntegerDigits[
        a, a = BitXor[a, BitOr[2 a, 4 a]]; 2, i]], {i, n}]]
```





The sequence does not repeat in at least its first million steps, and I would amazed if it ever repeats, but as of now I know of no rigorous proof of this. (Erica Jen showed in 1986 that no pair of columns can ever repeat, and the arguments on page 1087 suggest that neither can the center column together with occasional neighboring cells.)

- **Page 32 · Rule 110.** Many more details of rule 110 are discussed on pages 229 and 675. Localized structures that can occur are shown on page 292. Note that of the 8 cases in the basic rule for rule 110, only one differs from rule 102—which is a simple additive rule obtained by reflecting rule 60.

The Need for a New Intuition

▪ **Reactions of scientists.** Many scientists find the complexity of the pictures in this chapter so surprising that at first they assume it cannot be real. Typically they imagine that while the pictures may look complicated, they would actually seem simple if only they were subjected to the appropriate kind of analysis. In Chapter 10 I will give extensive evidence that this is not the case. But suffice it to say here that when it comes to finding regularities even the most advanced methods from mathematics and statistics tend to be no more powerful than our eyes. And whatever formal definition one may use for complexity (see page 557), the fact that our eyes perceive it in the systems discussed in this chapter is already very significant.

▪ **Intuition from practical computing.** Everyday experience with computers and programming leads to observations like the following:

- General-purpose computers and general-purpose programming languages can be built.
- Different programs for doing all sorts of different things can be set up.
- Any given program can be implemented in many ways.
- Programs can behave in complicated and seemingly random ways—particularly when they are not working properly.
- Debugging a program can be difficult.
- It is often difficult to foresee what a program can do by reading its code.
- The lower the level of representation of the code for a program the more difficult it tends to be to understand.
- Some computational problems are easy to state but hard to solve.
- Programs that simulate natural systems are among the most computationally expensive.
- It is possible for people to create large programs—at least in pieces.
- It is almost always possible to optimize a program more, but the optimized version may be more difficult to understand.
- Shorter programs are sometimes more efficient, but optimizations often require many cases to be treated separately, making programs longer.
- If programs are patched too much, they typically stop working at all.

▪ **Applications to design.** Many of the pictures in this book look strikingly similar to artistic designs of various styles. Probably this reflects not so much a similarity in underlying rules, but rather similarity in features that are most noticeable to the human visual system. Note that square grids of colored cells as in the cellular automata in this chapter can be used quite directly as weaving patterns. (See also page 929.)

Why These Discoveries Were Not Made Before

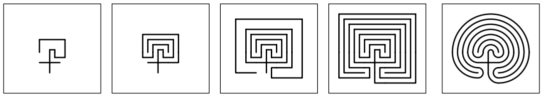
▪ **Page 43 · Ornamental art.** Almost all major cultural periods are associated with certain characteristic forms of ornament. Often the forms of ornament used on particular kinds of objects probably arose as idealized imitations of earlier or more natural forms for such objects—so that, for example, imitations of weaving, bricks and various plant forms are common. Large-scale purely abstract patterns were also central to art in such cultural traditions as Islam where natural forms were considered works of God that must not be shown directly. Once established, styles of ornament tend to be repeated extensively as a way of providing certain comfort and familiarity—especially in architecture. The vast majority of elaborate ornament seems to have been created by artisans with little or no formal theoretical discussion, although particularly since the 1800s there have been various attempts to find systematic ways to catalog forms of ornament, sometimes based on analogies with grammar. (Issues of proportion have however long been the subject of considerable theoretical discussion.) It is notable that whereas repetitive patterns have been used extensively in ornament, even nesting is rather rare. And even though for example elaborate symmetry rules have been devised, nothing like cellular automaton rules appear to have ever arisen. The results in this book now show that such rules can capture the essence of many complex processes that occur in nature—so that even though they lack historical context such rules can potentially provide a basis for forms of ornament that are familiar as idealizations of nature. (Compare page 929.)

The pictures in the main text show a sequence of early examples of various characteristic forms of ornament.

22,000 BC (Paleolithic). Mammoth ivory bracelet from Mezin, Ukraine. Similar zig-zag designs are seen in other objects from the same period. In the example shown, it is notable that the angle of the zig-zags is comparable to the angle of the Schreger lines that occur naturally in mammoth dentin.

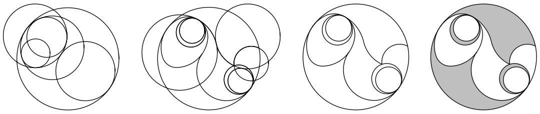
3000 BC (Sumerian). Columns with three colors of clay pegs set in mud from a wall of the Eanna temple in Uruk, Mesopotamia (Warka, Iraq)—perhaps mentioned in the Epic of Gilgamesh. (Now in the Staatliche Museum, Berlin.) This is the earliest known explicit example of mosaic.

1200 BC (Greek). The back of a clay accounting tablet from Pylos, Greece. The pattern was presumably made by the procedure shown below. Legend has it that it was the plan for the labyrinth housing the minotaur in the palace at Knossos, Crete, and that it was designed by Daedalus. It is also said that it was a logo for the city of Troy—or perhaps the plan of some of its walls. The pattern—in either its square or rounded form—has appeared with remarkably little variation in a huge variety of places all over the world—from Cretan coins, to graffiti at Pompeii, to the floor of the cathedral at Chartres, to carvings in Peru, to logos for aboriginal tribes. For probably three thousand years, it has been the single most common design used for mazes.

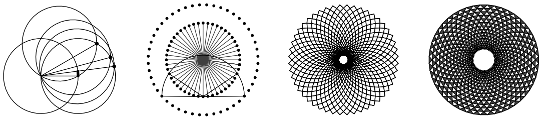


900 BC (Phoenician). Ivory carving presumably from the Mediterranean area. (Now in the British Museum.) This was a common decorative pattern, formed by drawing circles centered at holes arranged in a triangular array. It is also found in Egyptian and other art. Such patterns were discussed by Euclid and later Leonardo da Vinci in connection with the theory of lunes.

1st century BC (Celtic). The back of the so-called Desborough Mirror—a bronze mirror from Desborough, England made in the Iron Age sometime between 50 BC and 50 AD. (Now in the British Museum.) The engraved pattern is made of parts of circles that just touch each other, as in the picture below.



2nd century AD (Roman). A mosaic from a complex in Rome, Italy. (Now in the National Museum, Rome.) The geometrical pattern was presumably made by first constructing 48 regularly spaced spokes by repeated angle bisection, as in the first picture below, then drawing semicircles centered at the end of each spoke, and finally adding concentric circles through the intersection points. Similar rosette patterns may have been used in Greece around 350 BC; they became popular in churches in the 1500s.



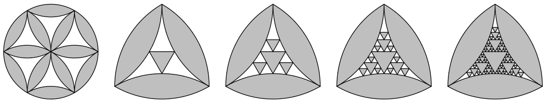
8th century (Islamic). A detail on the outside wall of the Great Mosque of Córdoba, Spain, built around 785 AD.

8th century (Celtic). An area less than 2 inches square from inside the letter ρ on the extremely elaborate chi-rho page of the Book of Kells, an illuminated gospel manuscript created over a period of years at various monasteries, probably starting around 800 AD at the Irish monastery on the island of Iona, Scotland. Even on this one page there are perhaps a dozen other very similar nested structures.

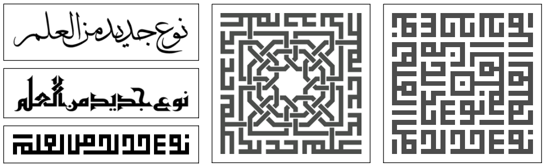
12th century (Italian). A window in the Palatine Chapel in Palermo, Sicily, presumably built around 1140 AD. The chapel is characteristic of so-called Arab-Norman style.

13th century (English). The Dean's Eye rose window of the Lincoln cathedral in England, built around 1225 AD. Similar tree-like patterns are seen in many Gothic windows from the same general period.

13th century (Italian) (4 pictures). Marble mosaics on the floor of the cathedral at Anagni, Italy, made around 1226 AD by Cosmas of the Cosmati group. (The fourth picture is a close-up of the third.) The third picture—particularly the part magnified in the fourth picture—shows an approximate nested structure, presumably created as in the pictures below. The triangles are all equilateral, with the result that at a given step several different sizes of triangles occur—though the basic structure of the pattern is still the same as from the rule 90 cellular automaton. (Compare the Apollonian packing of page 986.) The Cosmati group—mostly four generations of one family—made elaborate geometrical and other mosaics with a mixture of Byzantine, Islamic and other influences from about 1190 to 1300, mostly in and around Rome, but also for example in Westminster Abbey in England. Triangular shapes with one level of nesting are quite common in their work; three levels of nesting as shown here are rare. It is notable that in later imitations of Cosmati mosaics, these kinds of patterns were almost never used.



14th century (Islamic). Wall decoration in the Pir-i-Bakran mausoleum in Linjan, Iran, built around 1299–1312. The pattern is square Kufi calligraphy for a widely quoted verse of the Koran. Starting from traditional Naskhi Arabic script, as in the picture below, the Kufi style began to develop around 900 AD, with square Kufi being used in architectural ornamental by about 1100 AD.



14th century (Islamic). Tiled wall in the Alcázar of Seville, Spain, built in 1364. (The same pattern was used at about the same time in the Alhambra in Granada, Spain.) The pattern can be made by starting with a grid of triangles, then consistently pushing in or out the sides of each one. (Notable uses of such patterns were made by Maurits Escher starting in the 1930s.)

Other cases. The cases that are known inevitably tend to be ones created out of stone or ceramic materials that survive; no doubt there were others created for example with wood or textiles. One case with wood is Chinese lattice. What has survived mostly shows repetitive patterns, but the ice-ray style, probably going back to 100 AD, has approximate nesting, though with many random elements. The patterns shown are all basically two-dimensional. An example of 1D ornamental patterns are molding profiles. Ever since antiquity these have often been quite elaborate, and it is conceivable that they can sometimes be interpreted as showing nesting.

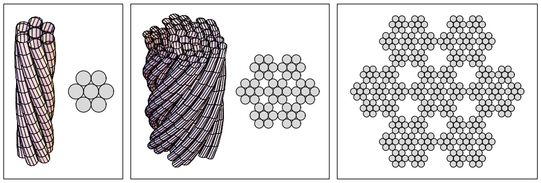
■ **Recognition of art.** One bizarre possibility is that forms like those from rule 30 could have been created as art long ago but not be recognized now. For while it is easy to tell that a cave painting of an animal is a piece of purposeful art, dots carved into a rock in an approximate rule 30 pattern might not even be noticed as something of human origin. But although there are many seemingly random painted patterns in caves from perhaps 30,000 BC, I would be amazed if any of them were actually produced by definite simple rules. (See page 839.)

■ **The concept of rules.** Processes based on rules occur in a great many areas of human endeavor. Sometimes the rules serve mainly as a constraint. But it is not uncommon for them to be used—like in a cellular automaton—as a way of specifying how structures should be built up. Almost without exception, however, the rules have in the past been chosen to yield only rather specific and simple results. Beyond ornamental art, examples with long histories include:

Architecture. Structures such as ziggurats and pyramids were presumably constructed by assembling collections of stones according to simple rules. The Great Pyramid in Egypt was built around 2500 BC and contains about two million large stones. (By comparison, the pictures of rule 30 on pages 29 and 30 contain a total of about a million cells.) Starting perhaps as long ago as 1000 BC Hindu temples were constructed with similar elements on different scales, yielding a form of approximate nesting. In Roman and later architecture, rooms in buildings have quite often been arranged in roughly nested patterns (an extreme example being the Castel del Monte from the 1200s). From the Middle Ages many Persian gardens (such as those of the Taj Mahal from around 1650) have had fairly regular nested structures obtained by a few successive fourfold subdivisions. And starting in the early 1200s, Gothic windows were often constructed with levels of roughly tree-like nested forms (see above). Nesting does not appear to have been used in physical city plans (except to a small extent in Vauban star fortifications), though it is common in organizational structures. (As indicated above, architectural ornament has also often in effect been constructed using definite rules.)

Textile making. Since early in human history there appear to have been definite rules used for weaving. But insofar as the purpose is to produce fabric the basic arrangement of threads is normally always repetitive.

Rope. Since at least 3000 BC rope has been made by twisting together strands themselves made by twisting, yielding cross-sections with some nesting, as in the second picture below. (Since the development of wire rope in the 1870s precise designs have been used, including at least recently the 7×7×7 one shown last below.)



Knots and string figures. For many thousands of years definite rules have been used for tying knots and presumably also for making string figures. But when the rules have more than a few steps they tend to be repetitive.

Paperfolding. Although paperfolding has presumably been practiced for at least 2000 years, even the nested form on page 892 seems to have been noticed only very recently.

Mathematics. Ever since Babylonian times arithmetic has been done by repeatedly applying simple rules to digits in numbers. And ever since ancient Greek times iterative methods have been used to construct geometrical figures. In the late 1600s the idea also emerged that mathematical proofs could be thought of as consisting of repeated applications of definite rules. But the idea of studying possible simple rules independent of their purpose in generating results seems never to have arisen. And as mathematics began to focus on continuous systems the notion of enumerating possible rules became progressively more difficult.

Logic. Rules of logic have been used since around 400 BC. But beyond forms like syllogisms little seems to have been studied in the way of generating identifiable patterns from them. (See page 1099.)

Grammar. The idea that human language is constructed from words according to definite grammatical rules has existed since at least around perhaps 500 BC when Panini gave a grammar for Sanskrit. (Less formal versions of the idea were also common in ancient Greek times.) But for the most part it was not until about the 1950s that rules of grammar began to be viewed as specifications for generating structures, rather than just constraints. (See page 1103.)

Poetry. Definite rules for rhythm in poetry were already well developed in antiquity—and by perhaps 200 BC Indian work on enumerating their possible forms appears to have led to both Pascal's triangle and Fibonacci numbers. Patterns of rhyme involving iterated length-6 permutations (sestina) and interleaved repetitive sequences (terza rima) were in use by the 1300s, notably by Dante.

Music. Simple progressions and various forms of repetition have presumably been used in music since at least the time of Pythagoras. Beginning in the 1200s more complex forms of interleaving such as those of canons have occasionally been used. And in the past century a few composers have implicitly or explicitly used structures based on simple Fibonacci and other substitution systems. Note that rules such as those of counterpoint are used mainly as constraints, not as ways of generating structure.

Military drill. The notion of using definite rules to organize and maneuver formations of soldiers appears to have existed in Babylonian and Assyrian times, and to be well codified by Roman times. Fairly elaborate cases were described for example by Niccolò Machiavelli in 1521, but all were set up to yield only rather simple behavior, such as a column of soldiers being rearranged into lines. (See the firing squad problem on page 1035.)

Games. Games are normally based on definite rules, but are set up so that at each step they involve choosing one of many possibilities, either by skill or randomness. The game of Go, which originated before 500 BC and perhaps as early as 2300 BC, is a case where particularly simple rules manage to allow remarkably complex patterns of play to occur. (Go involves putting black and white stones on a grid, making it visually similar to a cellular automaton.)

Puzzles. Geometric and arithmetic puzzles surprisingly close to those common today seem to have existed since as long ago as 2000 BC. Usually they are based on constraints, and occasionally they can be thought of as providing evidence that simple constraints can have complicated solutions.

Cryptography. Rules for encrypting messages have been used since perhaps 2000 BC, with non-trivial repetitive schemes becoming common in the 1500s, but more complex schemes not appearing until well into the 1900s. (See page 1085.)

Maze designs. From antiquity until about the 1500s the majority of mazes followed a small number of designs—most often based directly on the one shown on page 873, or with subunits like it. (It is now known that there are many other designs that are also possible.)

Rule-based pictures. It is rather common for geometric doodles to be based on definite rules, but it is rare for the rules to be carried far, or for the doodles to be preserved. Some of Leonardo da Vinci's planned book on "Geometrical Play" from the early 1500s has, however, survived, and shows elaborate patterns satisfying particular constraints. Various attempts to enumerate all possible patterns of particular simple kinds have been made—a notable example being Sébastien Truchet in 1704 drawing 2D patterns formed by combining ⬕, ◩, ⬔, ◪ in various possible ways.

■ **Page 44 · Understanding nature.** In Greek times it was noted that simple geometrical rules could explain many features of astronomy—the most obvious being the apparent revolution of the stars and the circular shapes of the Sun and Moon. But it was noted that with few exceptions—like beehives—natural objects that occur terrestrially did not appear to follow any simple geometrical rules. (The most complicated curves in Greek geometry were things like cissoids and conchoids.) So from this it was concluded that only certain supposedly perfect objects like the heavenly bodies could be expected to be fully amenable to human understanding. What rules for natural objects might in effect have been tried in the Judeo-Christian tradition is less clear—though for example the Book of Job does comment on the difficulty of "numbering the clouds by wisdom". And with the notable exception of the alchemists it continued to be believed throughout the Middle Ages that the wonders of nature were beyond human understanding.

▪ **Atomism.** The idea that everything might be made up from large numbers of discrete elements was discussed around perhaps 450 BC by Leucippus and Democritus. Sometime later the Epicureans then suggested that a few types of elements might suffice, and an analogy was made (notably by Lucretius around 100 AD) to the fact that different configurations of letters can make up all the words in a language. But only some schools of Greek philosophy ever supported atomism, and it soon fell out of favor. It was revived in the late 1600s, when corpuscular theories of both light and matter began to be widely discussed. In the early 1800s arguments based on atoms led to success in chemistry, and in the late 1800s statistical mechanics of large assemblies of atoms were used to explain properties of matter (see page 1019). With the rise of quantum theory in the early 1900s it became firmly established that physical systems contain discrete particles. But it was normally assumed that one should think only about explicit particles with realistic mechanical properties—so that abstract idealizations like cellular automata did not arise. (See also pages 1027 and 1043.)

▪ **History of cellular automata.** Despite their very simple construction, nothing like general cellular automata appear to have been considered before about the 1950s. Yet in the 1950s—inspired in various ways by the advent of electronic computers—several different kinds of systems equivalent to cellular automata were independently introduced. A variety of precursors can be identified. Operations on sequences of digits had been used since antiquity in doing arithmetic. Finite difference approximations to differential equations began to emerge in the early 1900s and were fairly well known by the 1930s. And Turing machines invented in 1936 were based on thinking about arbitrary operations on sequences of discrete elements. (Notions in physics like the Ising model do not appear to have had a direct influence.)

The best-known way in which cellular automata were introduced (and which eventually led to their name) was through work by John von Neumann in trying to develop an abstract model of self-reproduction in biology—a topic which had emerged from investigations in cybernetics. Around 1947—perhaps based on chemical engineering—von Neumann began by thinking about models based on 3D factories described by partial differential equations. Soon he changed to thinking about robotics and imagined perhaps implementing an example using a toy construction set. By analogy to electronic circuit layouts he realized however that 2D should be enough. And following a 1951 suggestion from Stanislaw Ulam (who may have already independently considered the problem) he simplified his model and ended up with a 2D cellular automaton (he apparently hoped later to convert the results back to differential equations). The particular cellular automaton he constructed in 1952–3 had 29 possible colors for each cell, and complicated rules specifically set up to emulate the operations of components of an electronic computer and various mechanical devices. To give a mathematical proof of the possibility of self-reproduction, von Neumann then outlined the construction of a 200,000 cell configuration which would reproduce itself (details were filled in by Arthur Burks in the early 1960s). Von Neumann appears to have believed—presumably in part from seeing the complexity of actual biological organisms and electronic computers—that something like this level of complexity would inevitably be necessary for a system to exhibit sophisticated capabilities such as self-reproduction. In this book I show that this is absolutely not the case, but with the intuition he had from existing mathematics and engineering von Neumann presumably never imagined this.

Two immediate threads emerged from von Neumann's work. The first, mostly in the 1960s, was increasingly whimsical discussion of building actual self-reproducing automata—often in the form of spacecraft. The second was an attempt to capture more of the essence of self-reproduction by mathematical studies of detailed properties of cellular automata. Over the course of the 1960s constructions were found for progressively simpler cellular automata capable of self-reproduction (see page 1179) and universal computation (see page 1115). Starting in the early 1960s a few rather simple general features of cellular automata thought to be relevant to self-reproduction were noticed—and were studied with increasingly elaborate technical formalism. (An example was the so-called Garden of Eden result that there can be configurations in cellular automata that arise only as initial conditions; see page 961.) There were also various explicit constructions done of cellular automata whose behavior showed particular simple features perhaps relevant to self-reproduction (such as so-called firing squad synchronization, as on page 1035).

By the end of the 1950s it had been noted that cellular automata could be viewed as parallel computers, and particularly in the 1960s a sequence of increasingly detailed and technical theorems—often analogous to ones about

Turing machines—were proved about their formal computational capabilities. At the end of the 1960s there then began to be attempts to connect cellular automata to mathematical discussions of dynamical systems—although as discussed below this had in fact already been done a decade earlier, with different terminology. And by the mid-1970s work on cellular automata had mostly become quite esoteric, and interest in it largely waned. (Some work nevertheless continued, particularly in Russia and Japan.) Note that even in computer science various names for cellular automata were used, including tessellation automata, cellular spaces, iterative automata, homogeneous structures and universal spaces.

As mentioned in the main text, there were by the late 1950s already all sorts of general-purpose computers on which simulations of cellular automata would have been easy to perform. But for the most part these computers were used to study traditional much more complicated systems such as partial differential equations. Around 1960, however, there were a couple of simulations related to 2D cellular automata done. Stanislaw Ulam and others used computers at Los Alamos to produce a handful of examples of what they called recursively defined geometrical objects—essentially the results of evolving generalized 2D cellular automata from single black cells (see page 928). Especially after obtaining larger pictures in 1967, Ulam noted that in at least one case fairly simple growth rules generated a complicated pattern, and mentioned that this might be relevant to biology. But perhaps because almost no progress was made on this with traditional mathematical methods, the result was not widely known, and was never pursued. (Ulam tried to construct a 1D analog, but ended up not with a cellular automaton, but instead with the sequences based on numbers discussed on page 908.) Around 1961 Edward Fredkin simulated the 2D analog of rule 90 on a PDP-1 computer, and noted its self-reproduction properties (see page 1179), but was generally more interested in finding simple physics-like features.

Despite the lack of investigation in science, one example of a cellular automaton did enter recreational computing in a major way in the early 1970s. Apparently motivated in part by questions in mathematical logic, and in part by work on "simulation games" by Ulam and others, John Conway in 1968 began doing experiments (mostly by hand, but later on a PDP-7 computer) with a variety of different 2D cellular automaton rules, and by 1970 had come up with a simple set of rules he called "The Game of Life", that exhibit a range of complex behavior (see page 249). Largely through popularization in *Scientific American* by Martin Gardner, Life became widely known. An immense amount of effort was spent finding special initial conditions that give particular forms of repetitive or other behavior, but virtually no systematic scientific work was done (perhaps in part because even Conway treated the system largely as a recreation), and almost without exception only the very specific rules of Life were ever investigated. (In 1978 as a possible 1D analog of Life easier to implement on early personal computers Jonathan Millen did however briefly consider what turns out to be the code 20 $k = 2$, $r = 2$ totalistic rule from page 283.)

Quite disconnected from all this, even in the 1950s, specific types of 2D and 1D cellular automata were already being used in various electronic devices and special-purpose computers. In fact, when digital image processing began to be done in the mid-1950s (for such applications as optical character recognition and microscopic particle counting) 2D cellular automaton rules were usually what was used to remove noise. And for several decades starting in 1960 a long line of so-called cellular logic systems were built to implement 2D cellular automata, mainly for image processing. Most of the rules used were specifically set up to have simple behavior, but occasionally it was noted as a largely recreational matter that for example patterns of alternating stripes ("custering") could be generated.

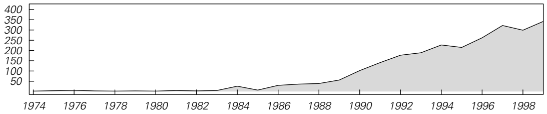
In the late 1950s and early 1960s schemes for electronic miniaturization and early integrated circuits were often based on having identical logical elements laid out on lines or grids to form so-called cellular arrays. In the early 1960s there was for a time interest in iterative arrays in which data would be run repeatedly through such systems. But few design principles emerged, and the technology for making chips with more elaborate and less uniform circuits developed rapidly. Ever since the 1960s the idea of making array or parallel computers has nevertheless resurfaced repeatedly, notably in systems like the ILLIAC IV from the 1960s and 1970s, and systolic arrays and various massively parallel computers from the 1980s. Typically the rules imagined for each element of such systems are however immensely more complicated than for any of the simple cellular automata I consider.

From at least the early 1940s, electronic or other digital delay lines or shift registers were a common way to store data such as digits of numbers, and by the late 1940s it had been noted that so-called linear feedback shift registers (see page 974) could generate complicated output sequences. These systems turn out to be essentially 1D additive cellular automata (like rule 90) with a limited number of cells (compare page 259). Extensive algebraic analysis of their behavior was done starting in the mid-1950s, but most of it concentrated on issues like repetition periods, and did not even explicitly uncover nested patterns. (Related analysis of linear recurrences over finite fields had been done in a few cases in the 1800s, and in some detail in the 1930s.) General 1D cellular automata are related to nonlinear feedback shift registers, and some explorations of these—including ones surprisingly close to rule 30 (see page 1088)—were made using special-purpose hardware by Solomon Golomb in 1956–9 for applications in jamming-resistant radio control—though again concentrating on issues like repetition periods. Linear feedback shift registers quickly became widely used in communications applications. Nonlinear feedback shift registers seem to have been used extensively for military cryptography, but despite persistent rumors the details of what was done continue to be secret.

In pure mathematics, infinite sequences of 0's and 1's have been considered in various forms since at least the late 1800s. Starting in the 1930s the development of symbolic dynamics (see page 960) led to the investigation of mappings of such sequences to themselves. And by the mid-1950s studies were being made (notably by Gustav Hedlund) of so-called shift-commuting block maps—which turn out to be exactly 1D cellular automata (see page 961). In the 1950s and early 1960s there was work in this area (at least in the U.S.) by a number of distinguished pure mathematicians, but since it was in large part for application to cryptography, much of it was kept secret. And what was published was mostly abstract theorems about features too global to reveal any of the kind of complexity I discuss.

Specific types of cellular automata have also arisen—usually under different names—in a vast range of situations. In the late 1950s and early 1960s what were essentially 1D cellular automata were studied as a way to optimize circuits for arithmetic and other operations. From the 1960s onward simulations of idealized neural networks sometimes had neurons connected to neighbors on a grid, yielding a 2D cellular automaton. Similarly, various models of active media—particularly heart and other muscles—and reaction-diffusion processes used a discrete grid and discrete excitation states, corresponding to a 2D cellular automaton. (In physics, discrete idealizations of statistical mechanics and dynamic versions of systems like the Ising model were sometimes close to cellular automata, except for the crucial difference of having randomness built into their underlying rules.) Additive cellular automata such as rule 90 had implicitly arisen in studies of binomial coefficient modulo primes in the 1800s (see page 870), but also appeared in various settings such as the "forests of stunted trees" studied around 1970.

Yet by the late 1970s, despite all these different directions, research on systems equivalent to cellular automata had largely petered out. That this should have happened just around the time when computers were first becoming widely available for exploratory work is ironic. But in a sense it was fortunate, because it allowed me when I started working on cellular automata in 1981 to define the field in a new way (though somewhat to my later regret I chose—in an attempt to recognize history—to use the name "cellular automata" for the systems I was studying). The publication of my first paper on cellular automata in 1983 (see page 881) led to a rapid increase of interest in the field, and over the years since then a steadily increasing number of papers (as indicated by the number of source documents in the Science Citation Index shown below) have been published on cellular automata—almost all following the directions I defined.



■ **Close approaches.** The basic phenomena in this chapter have come at least somewhat close to being discovered many times in the past. The historical progression of primary examples of this seem to be as follows:

- 500s–200s BC: Simply-stated problems such as finding primes or perfect numbers are presumably seen to have complicated solutions, but no general significance is attached to this (see pages 132 and 910).
- 1200s: Fibonacci sequences, Pascal's triangle and other rule-based numerical constructions are studied, but are found to show only simple behavior.
- 1500s: Leonardo da Vinci experiments with rules corresponding to simple geometrical constraints (see page 875), but finds only simple forms satisfying these constraints.
- 1700s: Leonhard Euler and others compute continued fraction representations for numbers with simple formulas (see pages 143 and 915), noting regularity in some cases, but making no comment in other cases.
- 1700s and 1800s: The digits of π and other transcendental numbers are seen to exhibit apparent randomness (see page 136), but the idea of thinking about this randomness as coming from the process of calculation does not arise.
- 1800s: The distribution of primes is studied extensively—but mostly its regularities, rather than its irregularities, are considered. (See page 132.)

- 1800s: Complicated behavior is found in the three-body problem, but it is assumed that with better mathematical techniques it will eventually be resolved. (See page 972.)
- 1880s: John Venn and others note the apparent randomness of the digits of π, but somehow take it for granted.
- 1906: Axel Thue studies simple substitution systems (see page 893) and finds behavior that seems complicated—though it turns out to be nested.
- 1910s: Gaston Julia and others study iterated maps, but concentrate on properties amenable to simple description.
- 1920: Moses Schönfinkel introduces combinators (see page 1121) but considers mostly cases specifically constructed to correspond to ordinary logical functions.
- 1921: Emil Post looks at a simple tag system (see page 894) whose behavior is difficult to predict, but failing to prove anything about it, goes on to other problems.
- 1920: The Ising model is introduced, but only statistics of configurations, and not any dynamics, are studied.
- 1931: Kurt Gödel establishes Gödel's Theorem (see page 782), but the constructions he uses are so complicated that he and others assume that simple systems can never exhibit similar phenomena.
- Mid-1930s: Alan Turing, Alonzo Church, Emil Post, etc. introduce various models of computation, but use them in constructing proofs, and do not investigate the actual behavior of simple examples.
- 1930s: The $3n+1$ problem (see page 904) is posed, and unpredictable behavior is found, but the main focus is on proving a simple result about it.
- Late 1940s and 1950s: Pseudorandom number generators are developed (see page 974), but are viewed as tricks whose behavior has no particular scientific significance.
- Late 1940s and early 1950s: Complex behavior is occasionally observed in fairly simple electronic devices built to illustrate ideas of cybernetics, but is usually viewed as something to avoid.
- 1952: Alan Turing applies computers to studying biological systems, but uses traditional mathematical models rather than, say, Turing machines.
- 1952–1953: John von Neumann makes theoretical studies of complicated cellular automata, but does not try looking at simpler cases, or simulating the systems on a computer.
- Mid-1950s: Enrico Fermi and collaborators simulate simple systems of nonlinear springs on a computer, but do not notice that simple initial conditions can lead to complicated behavior.
- Mid-1950s to mid-1960s: Specific 2D cellular automata are used for image processing; a few rules showing slightly complex behavior are noticed, but are considered of purely recreational interest.
- Late 1950s: Computer simulations of iterated maps are done, but concentrate mostly on repetitive behavior. (See page 918.)
- Late 1950s: Ideas from dynamical systems theory begin to be applied to systems equivalent to 1D cellular automata, but details of specific behavior are not studied except in trivial cases.
- Late 1950s: Idealized neural networks are simulated on digital computers, but the somewhat complicated behavior seen is considered mainly a distraction from the phenomena of interest, and is not investigated. (See page 1099.)
- Late 1950s: Berni Alder and Thomas Wainwright do computer simulations of dynamics of hard sphere idealized molecules, but concentrate on large-scale features that do not show complexity. (See page 999.)
- 1956–1959: Solomon Golomb simulates nonlinear feedback shift registers—some with rules close to rule 30—but studies mainly their repetition periods not their detailed complex behavior. (See page 1088.)
- 1960, 1967: Stanislaw Ulam and collaborators simulate systems close to 2D cellular automata, and note the appearance of complicated patterns (see above).
- 1961: Edward Fredkin simulates the 2D analog of rule 90 and notes features that amount to nesting (see above).
- Early 1960s: Students at MIT try running many small computer programs, and in some cases visualizing their output. They discover various examples (such as "munching foos") that produce nested behavior (see page 871), but do not go further.
- 1962: Marvin Minsky and others study many simple Turing machines, but do not go far enough to discover the complex behavior shown on page 81.
- 1963: Edward Lorenz simulates a differential equation that shows complex behavior (see page 971), but concentrates on its lack of periodicity and sensitive dependence on initial conditions.
- Mid-1960s: Simulations of random Boolean networks are done (see page 936), but concentrate on simple average properties.

- 1970: John Conway introduces the Game of Life 2D cellular automaton (see above).
- 1971: Michael Paterson considers a class of simple 2D Turing machines that he calls worms and that exhibit complicated behavior (see page 930).
- 1973: I look at some 2D cellular automata, but force the rules to have properties that prevent complex behavior (see page 864).
- Mid-1970s: Benoit Mandelbrot develops the idea of fractals (see page 934), and emphasizes the importance of computer graphics in studying complex forms.
- Mid-1970s: Tommaso Toffoli simulates all 4096 2D cellular automata of the simplest type, but studies mainly just their stabilization from random initial conditions.
- Late 1970s: Douglas Hofstadter studies a recursive sequence with complicated behavior (see page 907), but does not take it far enough to conclude much.
- 1979: Benoit Mandelbrot discovers the Mandelbrot set (see page 934) but concentrates on its nested structure, not its overall complexity.
- 1981: I begin to study 1D cellular automata, and generate a small picture analogous to the one of rule 30 on page 27, but fail to study it.
- 1984: I make a detailed study of rule 30, and begin to understand the significance of it and systems like it.

▪ **The importance of explicitness.** Looking through this book, one striking difference with most previous scientific accounts is the presence of so many explicit pictures that show how every element in a system behaves. In the past, people have tended to consider it more scientific to give only numerical summaries of such data. But most of the phenomena I discuss in this book could not have been found without such explicit pictures. (See also page 108.)

▪ **My work on cellular automata.** I began serious work on cellular automata in the middle of 1981. I had been thinking for some time about how complicated patterns could arise in natural systems—in apparent violation of the Second Law of Thermodynamics. I had been particularly interested in self-gravitating gases where the basic physics seemed clear, but where complex phenomena like galaxy formation seemed to occur. I had also been interested in neural networks, where there had been fairly simple models developed by Warren McCulloch and Walter Pitts in the 1940s. I came up with cellular automata as an attempt to capture the essential features of a range of systems, from self-gravitating gases to neural networks. I wanted to find models that had a simple structure like the Ising model in statistical mechanics (studied since the 1920s), but which had definite rules for time evolution and could easily be simulated on a computer. Ironically enough, while cellular automata are good for many things, they turn out to be rather unsuitable for modelling either self-gravitating gases or neural networks. (See page 1021). But by the time I realized this, it was clear that cellular automata were of great interest for many other purposes.

I did my first major computer experiments on cellular automata late in 1981 (see page 19). Two features initially struck me most. First, that starting from random initial conditions, cellular automata could organize themselves to produce complex patterns. And second, that in cases like rule 90 simple initial conditions led to nested or fractal patterns. During the first half of 1982, I worked hard to analyze the behavior of cellular automata using ideas from statistical mechanics, dynamical systems theory and discrete mathematics. And in June 1982, I finished my first paper on cellular automata, entitled "Statistical Mechanics of Cellular Automata". Published in the journal *Reviews of Modern Physics* in July 1983, this paper already presents in raw form many of the key ideas that led to the development of the science described in this book. It discusses the fact that by not using traditional mathematical equations, simple models can potentially be made to reproduce complex phenomena, and it mentions some of the consequences of viewing models like cellular automata as computational systems. The paper also contained a small picture of rule 30 started from a single black cell. But at the time, I did not study this picture in detail, and I tacitly assumed that whenever I saw randomness it must come from the random initial conditions that I used. (See page 112.)

It was some time in the fall of 1981 that I first found out (at a dinner with some then-young MIT computer scientists) that a version of the systems I had invented had been studied before under the name of "cellular automata". (I had been aware of the Game of Life, but its recreational emphasis had put me off studying it.) Knowing the name cellular automata, I was able to track down quite a number of relevant papers from the 1950s and 1960s. But I found that active research on what had been called cellular automata had more or less petered out (with the slight exception of a group at MIT at that time mainly concerned with building special-purpose hardware for 2D cellular automata). By late 1982 preprints of my paper on cellular automata had created quite a stir, and I got involved in organizing a conference held in March 1983 at Los Alamos to bring together many people newly interested in cellular automata with earlier workers in the field.

As part of preparing for that conference, I decided to use the graphics capabilities of the new workstation computer I had just obtained (a very early unit from Sun Microsystems) to investigate in a systematic way the behavior of a large collection of different cellular automata. And after spending several weeks looking at screen after screen of patterns—and trying to analyze their properties—I came to the conclusion that one could identify in the behavior of cellular automata with random initial conditions just four basic classes, each with its own characteristic features (see page 231).

In 1982 and early 1983, my efforts to analyze cellular automata were mainly based on ideas from discrete mathematics and dynamical systems theory. In the course of 1983, I also began to make serious use of formal language theory and the theory of computation. But for the most part I concentrated on characterizing behavior obtained from all possible initial conditions. And in fact I still vaguely assumed that if simple initial conditions were used, only fairly simple behavior would be obtained. Several of my papers had actually shown quite detailed pictures where this was not the case. I had noticed them, but they had never been among the examples I had studied in depth, partly for the superficial reason that the rules they involved were not symmetrical, or inevitably led to patterns that were otherwise not convenient for display. I do not know exactly what made me start looking more carefully at simple initial conditions, though I believe that I first systematically generated high-resolution pictures of all the $k = 2$, $r = 1$ cellular automata as an exercise for an early laserprinter—probably at the beginning of 1984. And I do know that for example on June 1, 1984 I printed out pictures of rule 30, rule 110 and $k = 2$, $r = 2$ totalistic code 10 (see note below), took them with me on a flight from New York to London, and a few days later was in Sweden talking about randomness in rule 30 and its potential significance.

A month or so later, writing an article for *Scientific American*—nominally on the subject of software in science and mathematics—led me to think more carefully about basic issues of computation and modelling, and to describe for the first time the idea of computational irreducibility (see page 737). In the fall of 1984 I began to investigate some of the implications of what I had discovered about cellular automata for foundational questions in science. And by early 1985 I had written what I consider to be my two most fundamental (if excessively short) papers from the period: one on undecidability and intractability in theoretical physics, and the other on intrinsic randomness generation and the origins of randomness in physical systems.

In the early summer of 1985 I was doing consulting at a startup company called Thinking Machines Corporation, which had developed a massively parallel computer called the Connection Machine that was fairly well suited to cellular automaton simulation. Partly as an application for this computer I then ended up making a detailed study of rule 30 and its randomness—among other things proposing it as a practical random sequence generator and cryptosystem.

I had always thought that cellular automata could be a way to get at foundational questions in thermodynamics and hydrodynamics. And in mid-1985, partly in an attempt to find uses for the Connection Machine, I devised a practical scheme for doing fluid mechanics with cellular automata (see page 378). Then over the course of that winter and the following spring I analyzed the scheme and worked out its correspondence to the traditional continuum approach.

By 1986, however, I felt that I had answered at least the first round of obvious questions about cellular automata, and it increasingly seemed that it would not be easier to go further with the computational tools available. In June 1986 I organized one last conference on cellular automata—then in August 1986 essentially left the field to begin the development of *Mathematica*.

Over the years, I have come back to look at cellular automata again and again, and every time I have been amazed and delighted by the richness of the phenomena they exhibit. As I argue in this book, a vast range of systems must in the end show the same basic phenomena. But cellular automata—and especially 1D ones—make the phenomena particularly clear, which is why even after investigating all sorts of other systems 1D cellular automata are still the most common examples that I use in this book.

▪ **My papers.** The primary papers that I published about cellular automata and other issues related to this book were (the dates indicate when I finished my work on each paper; the papers were actually published 6–12 months later):

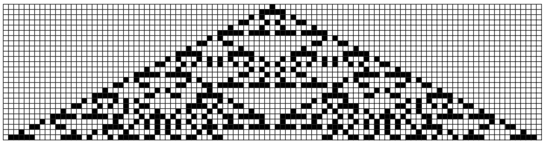
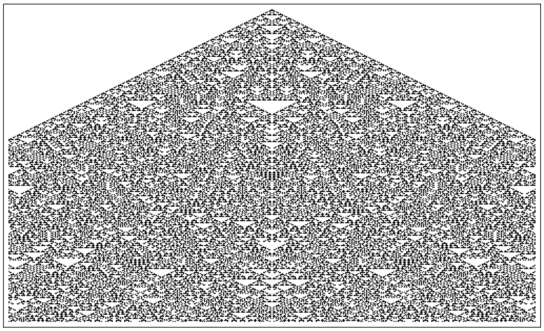
- "Statistical mechanics of cellular automata" (June 1982) (introducing 1D cellular automata and studying many of their properties)
- "Algebraic properties of cellular automata" (with Olivier Martin and Andrew Odlyzko) (February 1983) (analyzing additive cellular automata such as rule 90)
- "Universality and complexity in cellular automata" (April 1983) (classifying cellular automaton behavior)
- "Computation theory of cellular automata" (November 1983) (characterizing behavior using formal language theory)

- “Two-dimensional cellular automata” (with Norman Packard) (October 1984) (extending results to two dimensions)
- “Undecidability and intractability in theoretical physics” (October 1984) (introducing computational irreducibility)
- “Origins of randomness in physical systems” (February 1985) (introducing intrinsic randomness generation)
- “Random sequence generation by cellular automata” (July 1985) (a detailed study of rule 30)
- “Thermodynamics and hydrodynamics of cellular automata” (with James Salem) (November 1985) (continuum behavior from cellular automata)
- “Approaches to complexity engineering” (December 1985) (finding systems that achieve specified goals)
- “Cellular automaton fluids: Basic theory” (March 1986) (deriving the Navier-Stokes equations from cellular automata)

The ideas in the first five and the very last of these papers have been reasonably well absorbed over the past fifteen or so years. But those in the other five have not, and indeed seem to require the whole development of this book to be able to present in an appropriate way.

Other significant publications of mine providing relevant summaries were (the dates here are for actual publication—sometimes close to writing, but sometimes long delayed):

- “Computers in science and mathematics” (September 1984) (*Scientific American* article about foundations of the computational approach to science and mathematics)
- “Cellular automata as models of complexity” (October 1984) (*Nature* article introducing cellular automata)
- “Geometry of binomial coefficients” (November 1984) (additive cellular automata and nested patterns)
- “Twenty problems in the theory of cellular automata” (1985) (a list of unsolved problems to attack—most now finally resolved in this book)
- “Tables of cellular automaton properties” (June 1986) (features of elementary cellular automata)
- “Cryptography with cellular automata” (1986) (using rule 30 as a cryptosystem)
- “Complex systems theory” (1988) (1984 speech suggesting the research direction for the new Santa Fe Institute)

■ **Code 10.** Rule 30 is by many measures the simplest cellular automaton that generates randomness from a single black initial cell. But there are other simple examples—that historically I noticed slightly earlier than rule 30, though did not study—that occur in $k = 2$, $r = 2$ totalistic rules. And indeed among the 64 such rules, 13 show randomness. An example shown below is code 10, which specifies that if 1 or 3 cells out of 5 are black then the next cell is black; otherwise it is white.