Oct 83

COMPUTER RECREATIONS

Introducing a department concerned with the pleasures of computation

by Brian Hayes

Let us calculate.
—GOTTFRIED WILHELM VON LEIBNIZ

Some of the livelier games played with a microcomputer now come equipped with a function designated TBIC. The letters stand for "The boss is coming," and when the key assigned to the function is pressed, the battlefield on the glass screen immediately goes silent and dark. Here, it seems, are the two poles of the public response to the recent proliferation of inexpensive computers. On the one hand the computer is an engine of business, a capitalist tool; on the other it is a medium for entertainments so frivolous that they must be hidden from view, like comic books.

I do not mean to belittle either the practical applications of computers in business and industry or the genre of computer games whose principal aim is to test the player's reflexes. Utilitarian computing is unquestionably important. As for the video games, their construction may well be among the highest expressions of the programmer's art. It should be observed, however, that neither of these uses of the computer engages very deeply the question of what a computer is and what it can do.

There is a vast territory between business programming and video games, between VisiCalc and Space Invaders. The territory includes the applications of the computer in all the arts and sciences, and perhaps most obviously in mathematics. It includes the use of the computer to simulate aspects of the natural world and of human societies. Furthermore, it includes many pursuits that properly speaking are not "uses" of the computer at all but rather serve to focus attention on the computer itself and on the nature of mechanized computation. It is this realm between stern practicality and mere diversion that "Computer Recreations" will undertake to explore. Contributions from readers are welcome. Given that thinking remains a good deal harder than computing, those without access to computing machinery should be at only a slight disadvantage.

Even the tools of the businessman can sometimes be applied to problems in the theory and practice of computing. Here I shall consider some questions raised by novel applications of the programs called electronic spreadsheets.

A paper spreadsheet is a large page ruled into many columns and rows. It might be employed for analyzing the budget of a company. Each department could be given a column and each category of income or expense a row. Totals and percentages for each department and each category could be entered in additional columns and rows.

The electronic spreadsheet reproduces this structure on the screen of a cathode-ray tube, but with a few notable differences. On paper a given cell (defined as the intersection of a column and a row) can hold either a label, such as the name of a department, or a number. In an electronic spreadsheet a cell can also be assigned a mathematical formula. Thus the cell at the end of a row might hold a formula that calls for summing the values entered into all the other cells in the row. What is displayed on the screen is the number that results from evaluating the formula, in this case the total, but the underlying content of the cell is the formula itself rather than the number. If one of the other entries in the row is changed, the total is recalculated automatically.

The first of the electronic-spreadsheet programs was VisiCalc, developed in 1978 by Daniel Bricklin, who was then a student at the Harvard University School of Business, Robert Frankston and Dan Fylstra. It is said to have sold more copies than any other computer program. Dozens of other programs operating on similar principles have been introduced since then, and VisiCalc itself has been revised several times. Most of the experiments described here were done with two later spreadsheet programs: 1–2–3, conceived by Mitchell Kapor and Jonathan Sachs of the Lotus Development Corporation of Cambridge, Mass., and Multiplan, a product of the Microsoft Corporation of Bellevue, Wash. In most cases other spreadsheet programs would serve as well.

Although the electronic spreadsheet was designed for financial analysis, it is capable of much more. It is a two-dimensional matrix of cells where the value of each cell can be made to depend on any other cell or group of cells. It is surprising how much of the mathematical structure of the world can be coaxed into such a format. Indeed, it turns out that the spreadsheet represents a quite general context for describing mathematical and logical relations.

A simple example can give a better grasp of how a spreadsheet is manipulated and suggest what its potential is. Each cell is specified by its coordinates in a grid; in most of the programs the columns are identified by letter and the rows by number, starting at the upper left. Suppose cells A1 and A2 are each assigned a numeric value of 1. In cell A3 a formula is then entered: the value at position A3 is set equal to the sum of the value in the cell immediately above it and the value in the cell above that one. In other words, A3 is equal to the contents of A2 plus the contents of A1, and it displays the value 2.

What has been accomplished so far is trivial: it is a highly elaborate scheme for expressing the relation $1 + 1 = 2$. It is now possible, however, to copy the formula in cell A3 into many other cells. (The exact procedure for making copies varies from one program to another, but all the programs include such a facility.) Suppose the formula in A3 is copied into cells A4 through A10. Each of those cells will then hold a value equal to the sum of the values in the two cells above it. Note that the formulas are all identical, but because they are applied to different values the results are not. The numbers displayed, reading from top to bottom, are 1, 1, 2, 3, 5, 8, 13, 21, 34 and 55.

There are many ways of generating the Fibonacci series with a computer, and most of them make far more efficient use of the machine's resources than this one does. There is something quite distinctive, however, about the spreadsheet strategy: it is nonalgorithmic. In almost all programming languages a task or the solution to a problem is defined in terms of an algorithm, that is, a sequence of explicit instructions to be executed one after another. An algorithm is like a recipe: it might begin, "First mix flour, yeast and water, then let rise and finally bake." Doing the same operations in another sequence would have a quite different result. The spreadsheet does not have this characteristic temporal ordering. What goes into the cells is not a sequence of steps that leads from the problem to the solution but a static structure that attempts

to encompass the entire procedure all at once. It is a description rather than a recipe: it states that bread consists of flour, yeast and water that have been mixed, allowed to rise and baked.

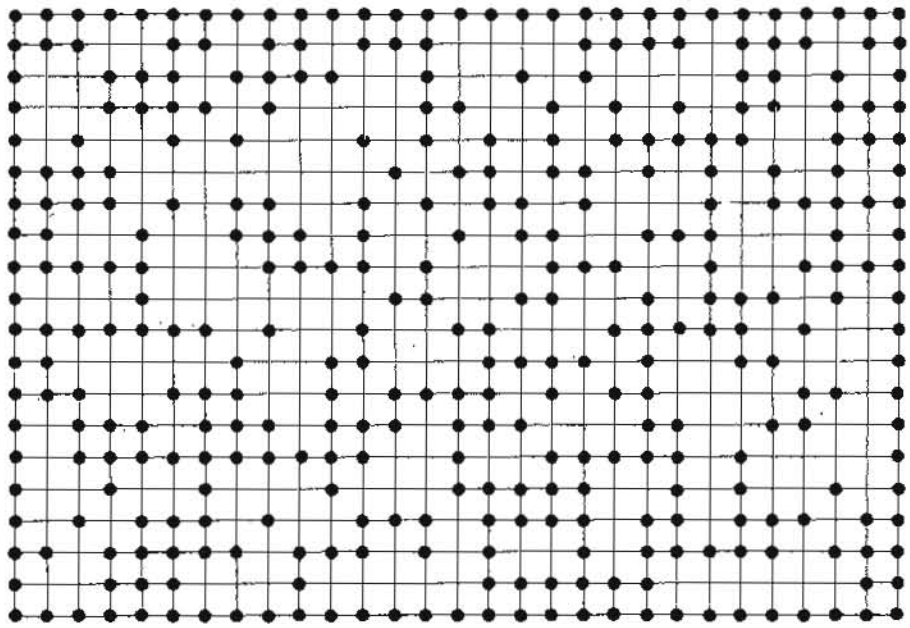
The distinction between an algorithm and a static description can be made clearer by another example. Consider the procedure for multiplying two matrixes of numbers when each matrix has three columns and three rows. The standard algorithm begins with instructions to multiply each element in the first column of the first matrix by each element in the first row of the second matrix, to add the three results and to store the sum as the first element of the product matrix. The same instructions are then repeated for the other eight combinations of rows and columns. The statement of the problem in a spreadsheet takes another form, and it can exploit the structural similarity of a mathematical matrix and an array of cells. Instead of writing a sequence of instructions, one simply defines the product matrix setting each cell equal to a formula that represents the appropriate combination of columns and rows. When the formulas have been entered, they are evaluated "all at once" and the entire product matrix appears.

At a deeper level, of course, a computer running under the direction of a spreadsheet program is indeed executing an algorithm. A computer that has only one central processing unit can do only one thing at a time, and so the cells are necessarily evaluated in some sequence. The user of the program, however, ordinarily has no need to take the sequence into account, and indeed he is often unaware of it. Thus the user need not think in terms of algorithms.

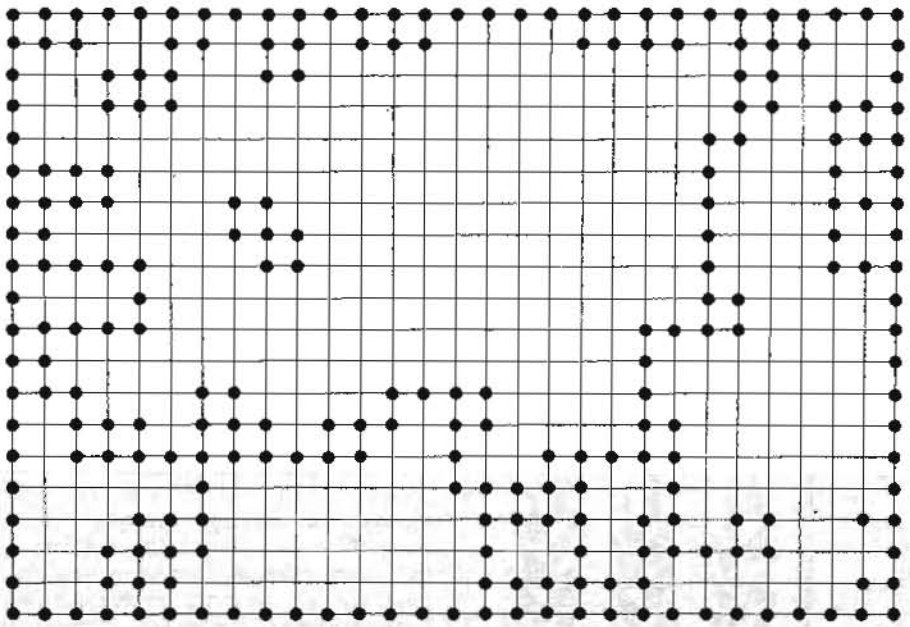
It is certainly not my intention to suggest that the nonalgorithmic mode of thought is somehow better than the algorithmic one. Some people may well prefer it, but that is largely a matter of taste. When a procedure becomes very complex, there is much to recommend the algorithm, which is more readily broken down into manageable pieces. Solving the problem all at once calls for understanding it all at once. It does seem likely, however, that there are certain problems or classes of problems that lend themselves naturally to a nonalgorithmic formulation.

One field where the spreadsheet's two-dimensional rectilinear array provides an appropriate medium is in constructing systems of cellular automata. The study of such systems was initiated in the 1950's by John von Neumann and Stanislaw Ulam, whose main concern at the time was with self-reproducing patterns. The rules they set themselves called for a "uniform cellular space" in which each cell represents an automaton, or machine, that has only a finite number of possible states. The space is uniform in the sense that the laws governing the state of the automata are the same for all the cells. A further constraint is that the state of a cell can be influenced only by its own history and by its close neighbors.

The conditions defining a system of cellular automata can readily be met by a spreadsheet program. In principle the number of states available to a cell is extremely large (perhaps 10^{100}), but it is clearly finite, and it can be reduced to a small number if that seems best; for example, a cell can be assigned a formula that can yield only two possible values, such as 0 and 1. The requirement of uniformity adds an interesting constraint. It implies that every cell in which a formula has been written must hold precisely the same formula. (There is more than one way of deciding whether two formulas are the same. Suppose a formula in cell A1 refers to cell B1, immediately below it. A formula in A2 might be considered identical if it also refers to B1, where the "absolute address" is the same, or if it refers to B2, where the geometric relation is preserved. The latter interpretation is generally more useful and seems more in keeping with von Neumann's and Ulam's ideas, but either scheme is acceptable if it is applied consistently. The spreadsheet routines for copying the contents of cells provide an easy operational test of uniformity. An array of cells can be considered uniform if a formula can be entered into one of



A lattice generated by a spreadsheet program simulates percolation



"Thinning" the lattice makes continuous paths easier to recognize

HAVE YOU HEARD?

The sound of hunger is louder than the rumble of an empty belly or the cry of a mother with nothing to feed her child. Hunger thunders through the generations and echoes against the dead end of abandoned dreams.

In the end, hunger can be heard in the scream of protest, in revolution and in rifle fire.

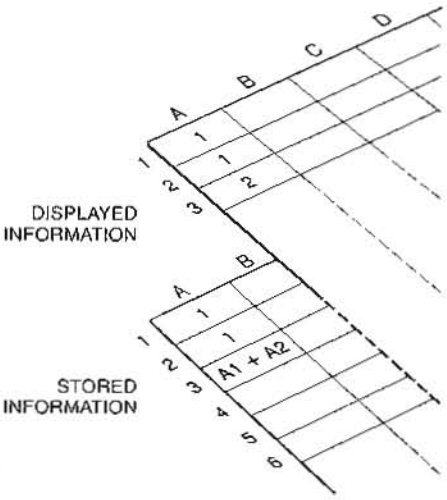
Oxfam America knows a better way. We work with people who are developing their own food and economic resources in 33 countries in Asia, Africa and Latin America. We put people to work in our own country, too, learning about the causes of hunger and what we, as responsible world citizens, can do about it.

We need your help to change the roar of hunger into a whisper of hope.

Oxfam America

Box N200
115 Broadway
Boston, Ma. 02116
617 482-1211

A full, audited financial report is available from Oxfam America, or, for New York residents, from the Department of State, Office of Charities Registration, Albany, NY 12231.



Structure of an electronic spreadsheet

them and then copied by the program into all the others.)

Von Neumann was able to prove that a self-replicating configuration of cells does exist. He did it the hard way, by showing there is a universal constructor that can create any pattern and therefore must be able to create its own pattern. The proof calls for some 200,000 cells having 29 possible states. As far as I know, the pattern has never actually been constructed, either manually or with the aid of a computer. Conceivably it might be within the capabilities of the larger spreadsheet programs.

A far simpler system capable of self-replication was devised in 1960 by Edward Fredkin of the Massachusetts Institute of Technology. Each cell has just two possible states, living and dead, which can be represented by the numbers 1 and 0. The state a cell will occupy in the next generation is determined by the present state of its four orthogonally adjacent neighbors, that is, the four cells immediately to the north, east, south and west. If the number of living neighbors is even (0, 2 or 4), the cell dies or remains dead. A cell with an odd number of living neighbors (1 or 3) lives.

It is a straightforward matter to express this rule in a spreadsheet formula, particularly with programs that include a function for modular arithmetic. In the case of cell B2 the formula is (B1 + C2 + B3 + A2) modulo 2. The effect of the formula is to add the values in the four neighboring cells, divide by 2 and retain only the remainder, which is necessarily either 0 or 1. It then remains only to copy the formula (in such a way that cell references preserve the same geometric relations) into all the cells throughout a region of the spreadsheet. Actually there is one further subtlety in the construction of the system: two copies of the cellular space are needed. One copy represents the present generation and one preserves the state of the preceding generation. The present state of a cell is based on the number of neighbors it had in the earlier generation.

When the spreadsheet is set up according to Fredkin's rules and an initial pattern is supplied, each cycle of recalculation yields a new pattern. After a few cycles four copies of the original configuration appear. Later the copies themselves are copied four times, so that the initial pattern has been reproduced 16 times. The number of cycles needed for reproduction depends on the complexity of the initial pattern; in the simplest case (a single live cell) the four offspring appear immediately.

Watching the progress of a growing colony can be fascinating. Fourfold symmetry is maintained at all times, and some of the patterns have a striking, stellate form. There is a rhythm to the process: the perimeter of the occupied area expands continuously, but the interior periodically becomes filled with a dense thicket of cells and then empties again.

Surely the best-known system of cellular automata is the game of life, invented by John Horton Conway of the University of Cambridge and introduced to the world at large in 1970 by Martin Gardner in his *Scientific American* column "Mathematical Games." The game, like its own pullulating automata, has by now spread to virtually every kind of computer system and programming language. There is a good reason: the game rewards the attention given it, whether the attention takes the form of casual spectatorship or close analysis.

In Conway's game of life the rules are defined not to ensure the replication of a pattern but rather to maximize variety or minimize predictability. Each cell again has two possible states, but the surrounding neighborhood is made up of the eight nearest cells, including the diagonally adjacent ones. If a cell is living, it will continue to live in the next generation only if it has either two or three living neighbors. With fewer neighbors it is said to die of loneliness and with more it dies of overcrowding. For a nonliving cell a birth is possible only if there are exactly three living neighbors.

An algorithmic specification of this procedure tends to be highly repetitious. It calls for examining a given cell, counting its neighbors, deciding whether the cell is to live or die, then going on to the next cell and the next until all of them have been checked. The repetition is generally embodied in the program structure called a loop, which is executed once for each cell. When the game of life is encoded in a spreadsheet, the repetition is still there, but it is spatial rather than temporal: the same formula is entered into each cell in a large array.

Teach yourself speed reading in 10 hours or your money back!

1-800-228-5655

Most executives must read a million words each week if they are to do their job.

1,000,000 words! And that's reading related to immediate business. It does not include other business related reading—you know the reading that helps you stay ahead of the game.

That's why most executives spend almost half of every work day reading. About 3 hours a day is average.

The good news is that now you can cut this load in half—at the very least.

You can use Speed Reading Self-Taught (SRST) to teach yourself speed reading. In just 10 hours or so you can double or triple your reading and increase your comprehension. Xerox Learning Systems guarantees this or your money back.

Speed Reading Self-Taught was originally created especially for our clients. (We help thousands of companies help their people do a better job, including 357 of the Fortune 500).

Thousands of executives in sales, data processing, engineering, marketing and management have proved out the SRST learning system, reporting significant gains in reading speed as well as impressive lifts in comprehension. SRST works for slow readers...for fast readers. It will work for you. The materials are mailed to you complete. All you need is a cassette recorder and a watch with a second hand.

SRST costs $95. Your organization will probably pay or reimburse you. We guarantee SRST will teach you lasting speed reading skills or your money back.

Most executives read at a seventh grade pace. Isn't it time you broke out of that pack?

You can teach yourself speed-reading in 10 hours with SRST. You can double or triple your reading speed and increase your comprehension. Xerox Learning Systems guarantees this or your money back.

To order Speed Reading Self-Taught, simply call us, toll free, at 1-800-228-5655. In Nebraska, 1-800-642-8777. Major credit cards accepted or send check for $95.00 plus $4.95 shipping and handling to Xerox Learning Systems, 6 Commercial Street, P.O. Box 944, Department SRST-S9ASA, Hicksville, N.Y. 11802.

XEROX

Xerox Learning Systems

There are many ways of writing a formula to evaluate the state of a cell in the game of life. The best I have seen (meaning the one that runs the fastest) was devised by Ezra Gottheil of Lotus. The basic procedure is to multiply the value in the cell under examination by 9, yielding a result of either 0 or 9, then add the values in the eight surrounding cells. That result is looked up in a small table that gives the new state of the cell for all possible values of the sum (namely those between 0 and 17).

Ideally the game of life would be played on a cellular matrix of infinite extent. One of the fascinations of the game is that certain small initial patterns grow into magnificent symmetrical blooms after just a few generations; other patterns emit compact projectiles that glide off into the indefinite distance. The evolution of the pattern is changed whenever an organism falls off the edge of the world. An infinite matrix is impossible under any circumstances, and when one is working with a spreadsheet, the practical limits are indeed rather small. They are set by the capacity of the program itself, by the memory capacity of the computer and by one's own patience. (The time needed for creating a new generation is roughly proportional to the number of cells included.) One strategy for creating an array that has no boundaries even though its area is finite is to define the cells along opposite edges as being adjacent; the effect is to change the topology of the spreadsheet. Joining two edges in this way creates a cylinder or, if the sheet is twisted, a Möbius band. Joining all four edges in pairs forms a torus.

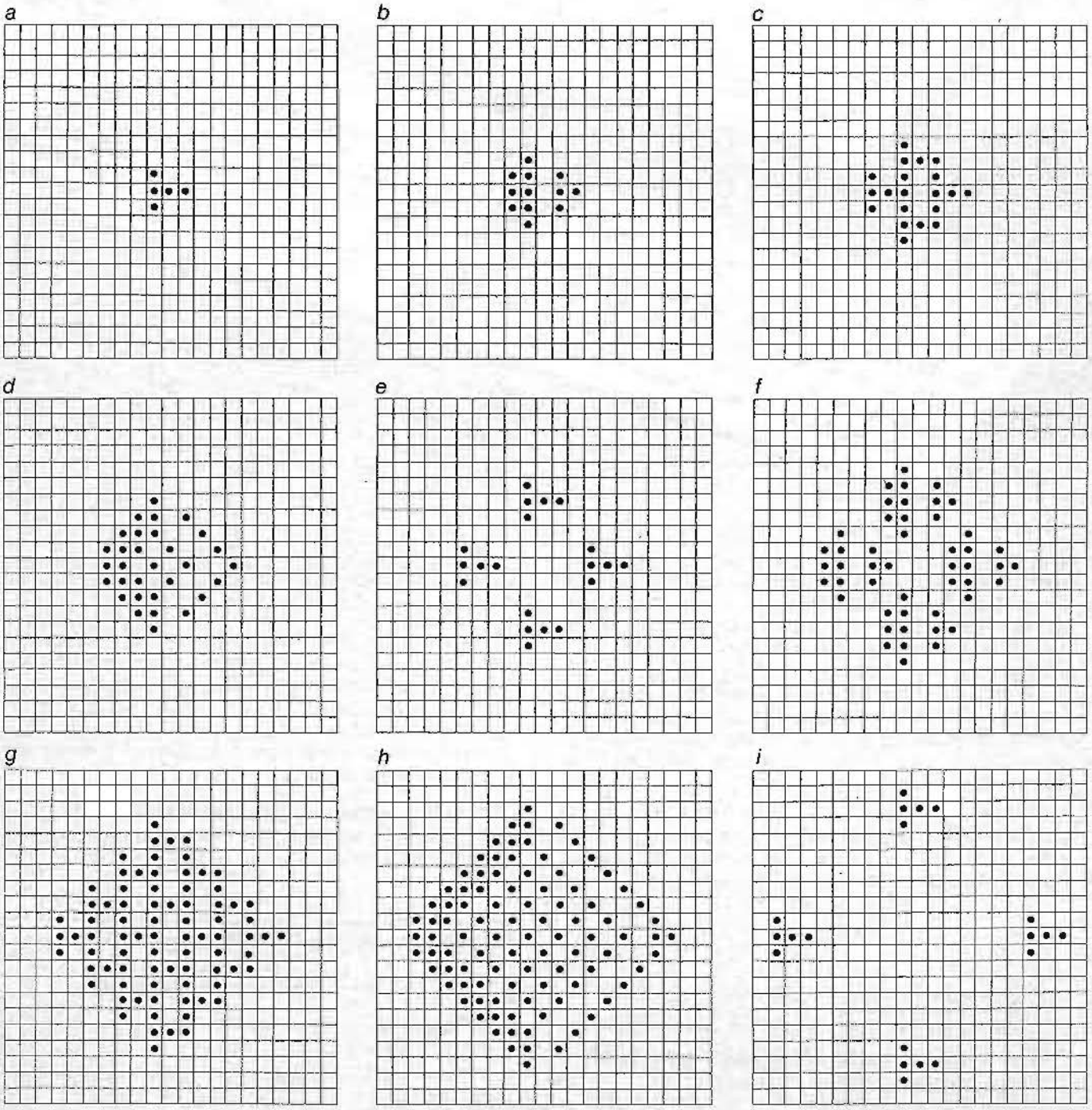
The Ising model is a simulated physical system that superficially resembles some cellular automata, although its interpretation is quite different. The model, which was introduced in the 1920's by the German physicists Wilhelm Lenz and Ernest Ising, can represent a number of physical phenomena, but it is most commonly applied to the description of ferromagnetic materials. Each site in a lattice represents the spin angular momentum, and hence the magnetic moment, of an atom. Each spin has a fixed magnitude, but the spin axis can point either up or down. When all the spins point in the same direction, the material is fully magnetized; when the spins are random, the magnetization is zero.

A spreadsheet realization of the Ising model is somewhat more complicated than one for Fredkin's replicating automata or Conway's game of life. The state of a given spin is again influenced by the nearest-neighbor cells, in this case the four orthogonal ones. In the Ising model, however, there is an element of randomness, which represents the effect of nonzero temperature. If a cell's neigh-

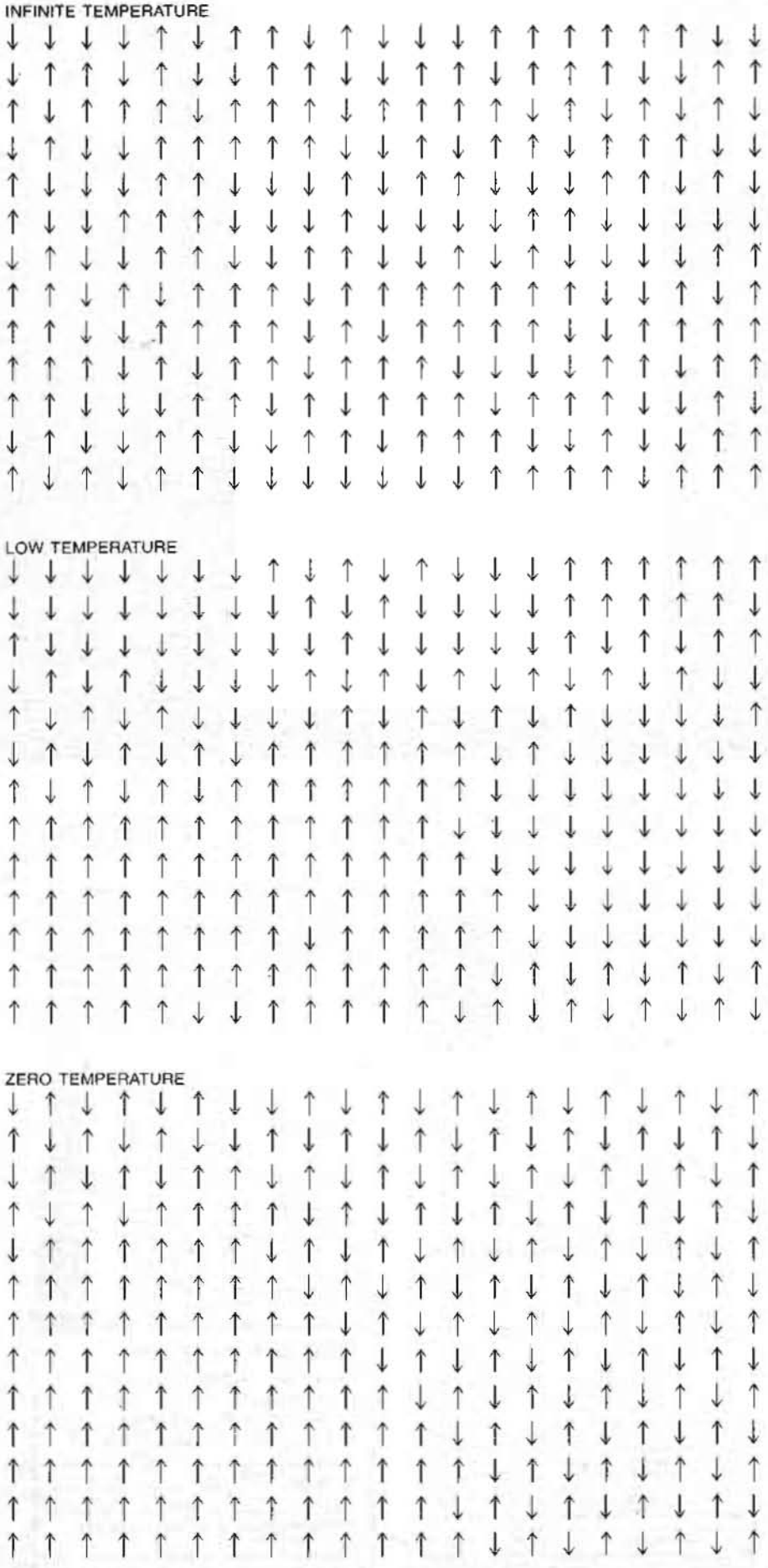
bors are all pointing up, the cell also has a tendency to point up, but it is not certain to do so; the probability is inversely proportional to the temperature.

A few experiments I have undertaken with a spreadsheet Ising model have yielded mixed results. The expected properties of the two-dimensional model have been known exactly since 1944, when Lars Onsager of Yale University solved the system analytically (rather than by numerical simulation). As the temperature is lowered through a threshold value (the Curie temperature) the spins should begin to fluctuate wildly and then should become fully magnetized. I have not seen such a phase transition, but I am not really surprised. The Ising model makes outrageous demands on computational resources. Getting accurate results requires a large lattice and a consideration of all possible configurations of the spins, which can take many hours even with an efficient program and a high-speed processor. At the Institute for Theoretical Physics of the University of California at Santa Barbara a special-purpose Ising-model computer has been built that calculates 25 million spins per second. The corresponding figure for the spreadsheet version of the model is about 25.

Even though the interesting events near the Curie temperature cannot be observed, the spreadsheet Ising model does seem to embody some other properties of magnetic materials. At high temperature the array of spins is without apparent pattern, as one would expect. At low temperature the magnetization of the lattice is obvious. Large, irregular blocks of aligned spins develop spontaneously, and blocks of opposite polarity seem to contend along their boundaries. One surprise (at least it surprised me) was the appearance of an antiferromagnetic phase, in which every other spin points in the opposite direc-



System of cellular automata devised by Edward Fredkin in which any pattern reproduces itself



A spreadsheet version of the two-dimensional Ising model of ferromagnetism

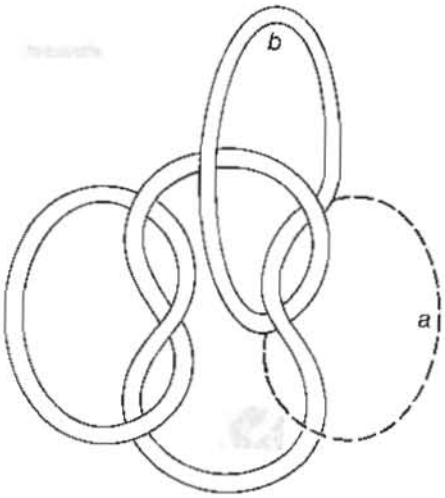
tion. At a temperature of absolute zero the antiferromagnetic phase seemed to be the stablest configuration, but this may reflect a deficiency or an error in my implementation of the model.

Other physical systems can also be conveniently dealt with on a rectilinear lattice. One phenomenon that interests me is percolation, which describes not only the brewing of coffee but also the structure of some polymers, the conductivity of alloys, the efficiency of telephone networks and the propagation of forest fires and infectious diseases. In one simple percolation model the cells of a lattice might represent possible sites of copper atoms in an insulating matrix. As a first approximation, the probability that a site is occupied is proportional to the bulk concentration of copper. The quantity of interest is the probability of forming an unbroken chain of copper atoms across some domain of the lattice; it is this probability that determines the conductivity of the material.

A percolation model in a spreadsheet again demands some form of random function. The easiest approach employs a uniform cellular space, where all the sites have the same probability of being occupied, and they are all independent of one another. The result is a random array of filled and vacant cells. It is then necessary to determine whether or not there is a continuous path across the space. A proper solution would be to mount a systematic search of each potential path, but it is not clear to me how that can be done without resorting to algorithmic methods. A cruder but still helpful technique is to thin out the underbrush by including in the formula for each cell a function that eliminates any atom with fewer than two neighbors. Because any atom that forms part of a chain must have at least two neighbors, the chain is not affected by this procedure, but many dead ends and isolated clusters wither away with each recalculation of the spreadsheet.

A spreadsheet program is surely not the ideal medium in which to represent any of these mathematical or physical models. For serious work each of them would have to be embodied in its own special-purpose program. In the case of the game of life I have compared various spreadsheet versions with a program (an algorithm!) written in the native language of a microprocessor. For arrays of the same size the machine-language program is almost 100 times faster than the fastest spreadsheet. The reason is not hard to find: even though a cell can never have a value other than 0 or 1, the spreadsheet program calculates the value to 15 decimal places.

If the spreadsheet cannot claim efficiency, however, it has the compensating virtue of versatility. Writing a machine-language program for playing the



Proof that a square knot is alternating

game of life is more than an evening's entertainment. Moreover, the program can do nothing else, whereas the simple matrix of linked cells in a spreadsheet constitutes a problem-solving device of impressive generality. There is much more, beyond the ideas sketched above, that is clearly within the capabilities of the programs. It appears that any series of numbers in which the terms are defined by algebraic or trigonometric functions can be generated. A prime-number sieve can be constructed out of one short formula, repeated some hundreds of times. A physical field can be represented by allowing the address of each cell to serve as its coordinates in two-dimensional space. As the sales brochure invariably says: "The only limit is your own imagination."

Is it true? Can a matrix of interdependent formulas without an algorithmic structure be made to compute anything that is computable? Is the mechanism not merely general but universal? In the case of an infinite matrix the question has been settled. Conway has proved that the cellular world of the game of life has sufficient resources for the construction of a Turing machine, the conceptual model of a universal computer. Since an infinite spreadsheet could be employed to play the game of life, it could also be employed to create the Turing machine.

Such a result is certainly worth knowing, but even if the requirement of infinite area could be relaxed, the demonstration would be of no practical significance. Life is too short, and the game of life too long. There is a less formal approach to measuring the scope of the spreadsheet programs that I find more promising. It is the hit-or-miss method of applying the programs to various problems and exercises from the stock of old favorites in computer science. The interesting test cases are likely to be the ones with a highly efficient algorithmic solution. One example is the Tower of Hanoi puzzle, in which several rings are stacked in order of decreasing size on one of three pegs; the aim is to move the rings one at a time, without ever allowing a larger ring to rest on a smaller one, until they are stacked in the same order on another peg. The standard solution employs a recursive algorithm, one that states the final stage in the procedure explicitly and then calls on itself to define the earlier stages.

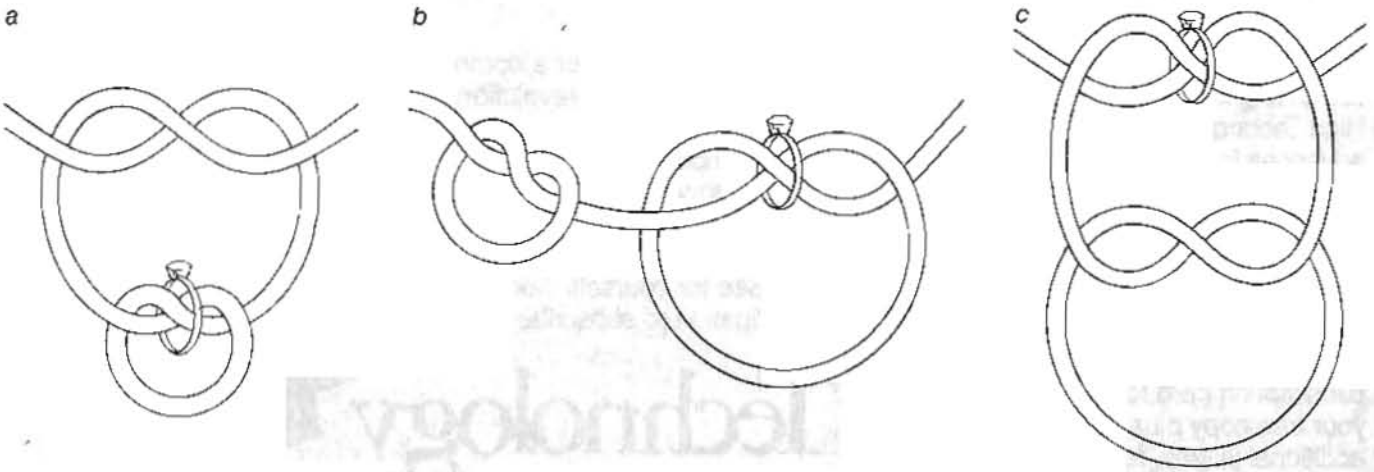
Can the Tower of Hanoi be solved by completely nonalgorithmic methods? Can it be done with a spreadsheet? Exhibiting such a solution would certainly not be a proof that a spreadsheet can do anything an algorithm can, but it would considerably enlarge the spreadsheet's range of action. Note that there is a trivial method of solution that must be declared out of bounds. One can solve the problem by hand, noting the configuration of the disks at each stage, and then write a series of formulas specifying the transitions from one configuration to the next. It is characteristic of such forced methods that with any slight change in the initial conditions, such as the addition of another disk, one must essentially start over. A robust solution would be able to handle any size stack without change or perhaps with a change only in the dimensions of an array. Note also that at least one spreadsheet program, Lotus 1–2–3, includes a simple algorithmic language; clearly all recourse to this facility must also be ruled out.

Another interesting case is the eight-queens problem, where the task is to place eight queens on a standard chessboard so that no queen is attacked by any other. Here the format of the problem—the finite array of squares—is tantalizing. There is certainly no difficulty representing a chessboard with a spreadsheet program. It is also easy to write a formula that reports whether a cell is currently attacked by a queen elsewhere on the board. (The formula merely checks for a nonzero value along all columns, rows and diagonals for a distance of eight cells.) If this were all one needed to solve the problem, however, it would not have attracted the attention of Carl Friedrich Gauss, who investigated it in 1850 but did not solve it. It seems each cell needs information not only on the current configuration of the board but also on the record of configurations that have already been tried. The difficulty of supplying this information in a static representation of the problem suggests that algorithms have a secure future.

The two knot problems posed last month by Martin Gardner are answered as follows:

The top illustration on this page shows how a square knot can be changed to an alternating knot of six crossings. Simply flip dotted arc *a* over to make arc *b*.

The illustration below shows one way to solve the ring-and-granny-knot puzzle. First make the lower knot small, then slide it (carrying the ring with it) up and through the higher knot (*a*). Open it. Two trefoil knots are now side by side (*b*). Make the ringless knot small, then slide it through and down the other knot. Open it up and you have finished (*c*).



Solution to the ring-and-granny-knot puzzle