How to Build a Maze

David Matuszek
Department of Computer Science
8 Ayres Hall
University of Tennessee
Knoxville TN 37916

Mazes are fun to solve. With a little imagination, mazes can be incorporated into many different computer games. If you know how, it's a simple matter to use the computer to generate random mazes.

A traditional maze has one starting point and one finishing point. In addition, all locations inside the maze are reachable from the start, and there is one and only one path from start to finish. While it is easy to place doorways and barriers randomly inside a maze, it is more difficult to satisfy the two latter constraints. This article describes a fairly simple method that efficiently produces a random traditional maze.

The General Approach

We begin with a rectangular array. Each cell of the array is initially completely "walled in," isolated from its neighbors (see figure 1).

Secondly, we judiciously erase walls inside the array until we arrive at a structure with the following property: for *any* two cells of the array, there is only one path between them. Thus, any cell can be reached from any cell, but only by a single unique path (see figure 2). Computer-science jargon refers to such a structure as a *spanning tree*, and it is the creation of this spanning tree that is the tricky part of building a maze.

Finally, the border of the maze is broken in two places to provide a start and a finish position. Since there is a unique path between *any* two cells of the maze, there will be a unique path from start to finish. Hence, start and finish can be chosen in any convenient manner, say, at random locations on opposite sides of the maze (see figure 3).

Building the Spanning Tree

Starting with a fully "walled-up" array (see figure 1), pick a single cell in the array and call this cell the

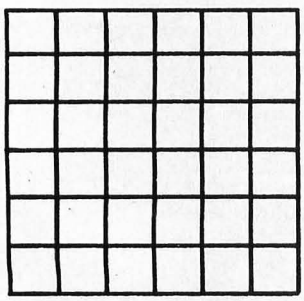


Figure 1: *The initial array from which the maze will be constructed.*

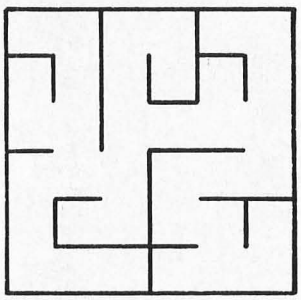


Figure 2: *One possible spanning tree for the array in figure 1.*

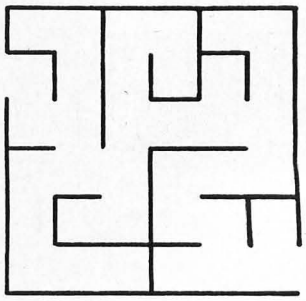


Figure 3: *The spanning tree from figure 2 with possible entry and exit points added.*

Circle 20 on inquiry card. →

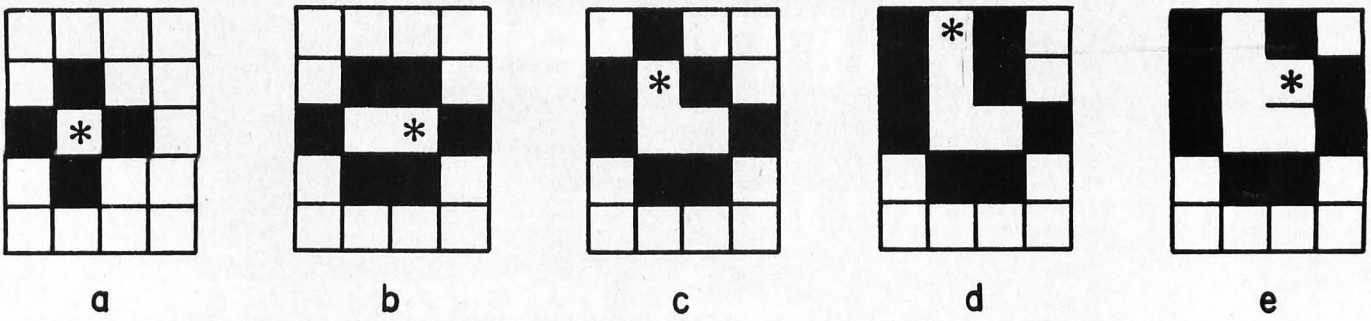


Figure 4: *Initial steps involved in building a maze. The cell added at each step is marked with an asterisk. The next cell to be added to the maze will be selected from the shaded frontier cells.*

spanning tree. Then adds cells one at a time to the spanning tree until it *fills* the entire array.

At any point during this procedure, there will be three types of cells in the array:

- those that are already in the spanning tree
- those that are not in the spanning tree, but are immediately adjacent (horizontally or vertically) to some cell in the spanning tree (we call these cells *frontier cells*)
- all the other cells

The algorithm follows:

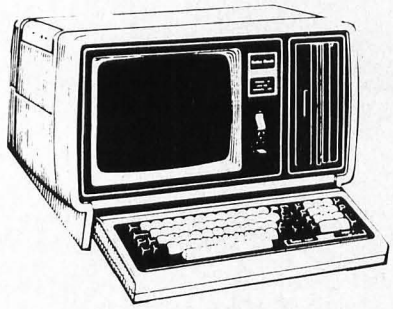
1. Choose any cell of the array and call it the spanning tree. The four cells immediately adjacent to it (fewer if it is on an edge or in a corner) thus become frontier cells.
2. Randomly choose a frontier cell and connect it to *one* cell of the current spanning tree by erasing *one* barrier. If it is adjacent to more than one cell of the spanning tree (and it could be adjacent to as many as four!), randomly choose one of them to connect it to, and erase the appropriate barrier.
3. Check the cells adjacent to the cell just added to the spanning tree. Any such cells that are not part of the spanning tree and have not previously been marked as frontier cells must now be marked as frontier cells.
4. If any frontier cells remain, go back to step 2.
5. Choose start and finish cells.

Figure 4 shows the first few steps in building a maze. In each case the array is shown as it would be just before execution of step 2 of the algorithm. Note that the newly added cell (marked by an asterisk) in figure 4e was adjacent to *two* cells in the spanning tree, yet it was connected to only one of these (the one to its left) by randomly choosing and erasing *one* barrier.

If you're mathematically inclined, it is easy to show by induction that this process results in a spanning tree. When the tree consists of a single cell, there is (vacuously) only one path between any pair of cells. As each new cell is added, it forms no new paths between cells already on the tree (since the tree is a dead end), and there is exactly one path from the new cell to any other cell (you can get out via only one cell, and from that cell there is only one path). Finally, the process ends when there are no more frontier cells (cells adjacent to the spanning tree but not in it), and this can happen only when all cells have been absorbed into the spanning tree.

Implementation Details

You should store in each cell of the



96K CP/M®
(For your TRS-80* Model II)

Multiple Job Executive

Add a whole new dimension to your TRS-80 Model II. Let it work while you work!!

ATON's unique JobStream™ CP/M 2.2, along with additional RAM memory; allows you to **simultaneously** compile, assemble, or link in one 64K background partition (62K TPA) while you edit files, **and** spool to the printer, **and** communicate with another computer in up to four 32K foreground partitions (28K TPA).

As you expand memory beyond 64K, you also enter the amazing world of TrackMode BIOS™ which not only **multiplies diskette speed** up to **five times**, but also automatically performs read after write checks for the ultimate in data reliability.

- **Gain hard disk performance** for a fraction of the cost—**and no backup problems**!!!
- Works in 32K, add RAM memory to 256K using standard Radio Shack memory boards.
- **Supports two sided expansion disk drives** (1.2 megabytes per diskette).

JobStream CP/M 2.2 (with Z-80 Debugger)**	**$235**
Omni Writer™ Video Text Editor	**$130**
Z-80 Debugger Source Code	**$ 50**
Package of above (a $415 value)	**$295**

"Software with Service"

Prepaid, Visa, MasterCard or COD.
Shipping and handling extra.
California residents add 6% sales tax.

CP/M® Digital Research, Inc.
*™Tandy Corp. **™Zilog Corp.
JobStream, TrackMode BIOS™ ATON Intl.
Omni Writer™ Omnigraphics

Aton International, Inc.

260 Brooklyn Avenue, San Jose, CA 95128
(408) 286-4078

Circle 37 on inquiry card.

Circle 303 on inquiry card

Introducing Pascal Sourcebooks™

from the NATI Pascal Software Library

```
84 | PROCEDURE wand_input;
85 |   BEGIN {wand_inp
86 |     read_cod
87 |     IF
88 |
89 |
90 |
```

Complete Annotated Program Listings

Cross References and Block Summaries

Block Indices by Page

This new library of UCSD Pascal* compatible programs provides practical applications and system augmentations available nowhere else. Each Pascal Sourcebook ™ is a treasure trove of ideas and proven program techniques. These Pascal programs will run on every major personal computer system currently manufactured which has the UCSD p-System* installed.

Pascal Sourcebooks™ contain completely commented and annotated source listings and machine readable bar code texts**** of well designed, clearly written UCSD Pascal* programs. Use the programs "as is", or modify and customize them to your own personal tastes. Either way, you gain a practical working library available nowhere else.

File System: Interrogate directories from application programs in order to select existing files or create new files interactively...
FS.01 Pascal Sourcebook™ alone** $69.95**
FS.02 Pascal Sourcebook™ and Apple Pascal media .. $99.95**

Incremental Backup System: Use these daily rituals to save recently used files to prevent loss of floppy or hard disk data.
IBS.01 Pascal Sourcebook™ alone** $69.95**
IBS.02 Pascal Sourcebook™ and Apple Pascal media . $99.95**

Report Generator: Use this report formatting program to create professional word processing quality documentation with the UCSD Pascal* system's Screen Editor. Uses Diablo, NEC, Ricoh, Ahearn&Soper, etc. printers with Diablo 1620 compatible serial interface.
RGI.01 Pascal Sourcebook™ alone** $79.95**
RGI.02 Pascal Sourcebook™ & Apple Pascal media .. $109.95**

Telephone Utilities: Use these Hayes Smartmodem*** programs to dial originate or answer mode data calls. Save session images on disk. Transfer files to the line. Use the autodialer for voice calls.
TU.01 Pascal Sourcebook™ alone** $59.95**
TU.02 Pascal Sourcebook™ and Apple Pascal** media .. $89.95

Graphic Applications-I: Practical examples of Pascal programs driving application oriented graphics: examples of hydro power profiles, spherical lens ray tracing, sun-earth-moon orbital model, numerical integration, etc.
GAI.01 Pascal Sourcebook™ alone** $19.95**
GAI.02 Pascal Sourcebook™ and Apple Pascal media . $49.95**

Typewriter Simulators: Use a word processing quality printer with your terminal as an eclectic electric typewriter. Automatic address accumulation, envelope addressing, line by line correction. Uses Diablo, NEC, Ricoh, Ahearn&Soper, etc. printers with Diablo 1620 compatible serial interface.
TSI.01 Pascal Sourcebook™ alone** $49.95**
TSI.02 Pascal Sourcebook™ and Apple Pascal media . $79.95**

* UCSD Pascal and UCSD p-System are trademarks of the Regents of the State of California
** Apple Pascal is a trademark of the Apple Computer Company
*** Smartmodem is a trademark of Hayes Microcomputer Products
**** All Pascal Sourcebooks™ include NATI Text Format™ bar code images of compressed machine readable Pascal language source texts. Bar code readers and bootstrap software are available for Apple Pascal machines.

Dealer Inquiries Invited

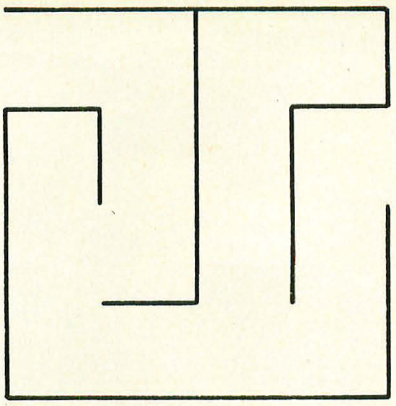
FREE BONUS! With each order, we'll send customers a personal copy of the booklet "Alternative Software Delivery Media" describing our NATI Text Format™ printed bar code technology

NORTH AMERICAN TECHNOLOGY, INC.
Strand Building
174 Concord St.
Peterborough, NH 03458
(603) 924-7136

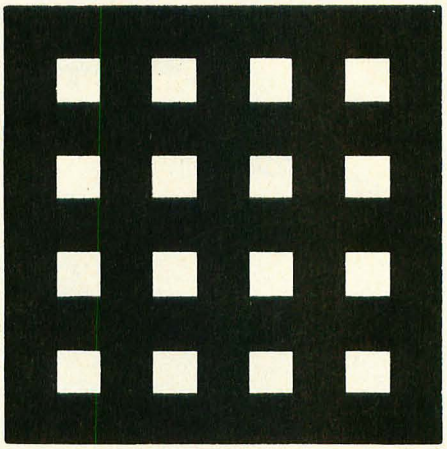
Order your Pascal Sourcebooks™ and media copies now. Call our 24 hour credit card phone order service with your Master Card, or Visa billing information:

800-854-0561 operator 860

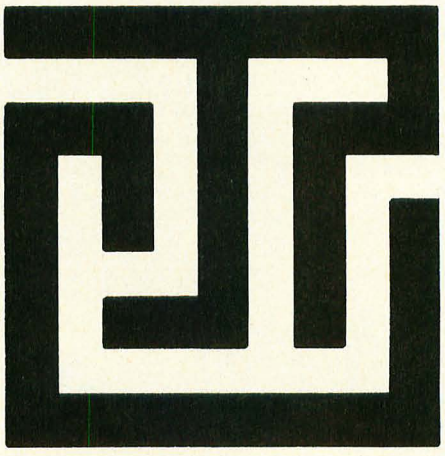
(or in California, 800-432-7257 operator 860)



5a



5b



5c

Figure 5: *For an* m *by* n *maze to be displayed on a computer graphics system, a resolution of at least* 2m+1 *by* 2n+1 *must be available. The 4- by 4-maze array of figure 5a requires a graphics array of 9 by 9. The initial cells of the 4 by 4 array are shown displayed using the 9 by 9 resolution in figure 5b. The finished maze, with openings between the cells where paths exist, is shown in figure 5c.*

array a number indicating: 1. whether it is in the spanning tree, in the frontier, or in neither; and 2. if it is in the spanning tree, which of the cell's barriers have been erased. One possibility is to use—1 for frontier cells, positive numbers for cells, positive numbers for cells in the spanning tree, and 0 for all other cells.

any cell of the spanning tree is open to at least one other cell, I suggest the following encoding: start with 0 in each cell, add 1 if the barrier on the right is erased; add 2 if the barrier below is erased; add 4 if the barrier on the left is erased, and add 8 if the barrier above is erased. The result will be a number from 1 to 15 that specifies exactly which combination of barriers has been erased. (Decoding this number shouldn't be too hard if you work with binary numbers.) Note that when you erase a barrier between two cells you will have to add the appropriate numbers to each of them.

The minor exception mentioned above is the initial cell of the spanning tree, immediately after step 1 of the algorithm (see figure 4a). Since it is the first, it is not yet open to any other cell. Give it the value 16 (or 100, or 1984, if you prefer) so that it will be positive, and subtract this number out again in step 5.

Now that the array representation has been settled, let's discuss efficient implementation of the algorithm. In step 2 a frontier cell was randomly chosen. To prevent bumbling around in the array, you must keep a list of those cells. This can be simply accomplished by storing the indices of the n cells of the frontier (each of which is specified by a row number and a column number) in the first n locations of two arrays, R (row numbers) and C (column numbers). A frontier cell can be quickly chosen by randomly choosing a k less than or equal to n, and using the cell whose indices are given by R(k), C(k).

Since the order of the n frontier cell locations in arrays R and C is not important, the following code suffices to remove the chosen cell k:

$$R(k) = R(n)$$
$$C(k) = C(n)$$
$$n = n-1$$

```
6a  XXXXXXXXX
        X   X
    XXX X XXX
    X X X X
    X X X X X
    X   X X X
    X XXX X X
    X       X
    XXXXXXXXX
```

```
6b  XXXXXXXXXXXXXXXXX
                XX      XX
    XXXXXX      XX      XXXXXX
    XX  XX      XX      XX
    XX  XX      XX      XX  XX
    XX          XX      XX  XX
    XX  XXXXXX  XX      XX
    XX                  XX
    XXXXXXXXXXXXXXXXX
```

Figure 6: *The maze of figure 5 as it might appear as printed output, with each maze-array element represented by space characters or X characters. One space or X is used in 6a; two spaces or Xs are used in 6b.*

THE NEW OMR 500 SEES THE LIGHT

An Optical Version of our MR 500 Makes it Even Easier to Enter Data into Your Microcomputer

No Special Pencils Needed
Now you can read punched holes, preprinted data, or pencil marks on standard OMR cards. All with the incredibly compact OMR 500 optical card reader.

Using state-of-the-art fiber optics to "read" each card, a single long-lasting bulb does the job. Reliably and accurately.

Simple, Fast, and Low-Cost
The OMR 500 is a low-cost alternate to keyboard data entry. And at less than 1/2 second per hand-fed card, you won't be sacrificing speed.

Compact and lightweight, our new optic reader is a mere 4-lb, 4-½ inch cube. Automatic turn-on is standard.

Wide Variety of Interfaces
The reader is available with intelligent interfaces to Apple, TRS-80, PET and Atari that simplify user software requirements. Also available are RS-232 and S100 interfaces.

Lighting the Way
At $1095, including the intelligent interface, the OMR 500 truly adds an affordable new dimension to card reader flexibility. Its uses are virtually unlimited. Small business, the entire educational field, personal computers — wherever data entry is required.

And remember, we still offer the industry's largest selection of card readers. So whatever your needs, we've got the right card reader for you.

Write or phone for complete details. Better yet, put in your order today.

CHATSWORTH DATA CORPORATION

20710 Lassen Street Chatsworth, California 91311 Phone: (213) 341-9200

When this frontier cell is added to the spanning tree, some of the cells adjacent to it (those having a zero value) become new frontier cells, and their locations must be inserted into the R and C arrays. Adjacent cells with value −1 are already frontier cells and already have their locations recorded in the R and C arrays; they must not be inserted again.

Finally, how large should arrays R and C be? For an m by n array, analysis shows that in the worst case $(2/3)\ mn$ locations will be required, but practical experience shows that $3(m+n)$ is almost always enough. However, if you use the latter figure there is a slight probability that the program will fail.

Concluding Remarks

While we have discussed building a maze, nothing has been said regarding how to display it. That depends entirely on your particular hardware and software; the answers are different for the display screen of a Commodore PET than for that of an Apple II, and different again for a character printer.

To display a maze on a screen with graphics capabilities, the following scheme is appropriate. For an m by n maze, you need to be able to display at least $2m+1$ points vertically and $2n+1$ points horizontally—the "cells" will be those points at the intersection of even-numbered rows with even-numbered columns (see figures 5a through 5c). Maze building on the screen proceeds exactly as in figures 1 through 3, except that the walls are necessarily thicker.

To print a maze out, the same general scheme is used with, say, "X" characters for walls and blanks for paths (see figure 6). Of course, you can't erase an X once it is printed, so it will be necessary to build the entire maze internally before printing it. Then you can decipher and print the maze one row at a time.

As a final note, if you are an aficionado of hexagonal grids, the maze algorithm is easily modified for other than rectangular grids. Implementation may be a bit messy—but then, implementation is always messy. ■

Circle 447 on inquiry card.