LIFE *Line*

by
Carl Helmers
Editor, BYTE

Games played with computer equipment are applications of value above and beyond the momentary "hack" value of putting together an interesting program. The creation of a game is one of the best ways to learn about the art and technique of programming with real hardware and software systems. LIFE Line concerns a game – the Game of LIFE, originated by Charles Conway and first publicized by Martin Gardner in *Scientific American.* The Game of LIFE serves as the central theme of LIFE Line – a well defined application of the type of hardware and software which is within the reach of BYTE readers. The description of the LIFE application is the "down to earth" goal of LIFE Line. However, I have an ulterior motive as well – LIFE Line is a very convenient and practical vehicle for teaching ideas about program and system design which you can apply for your own use. Even if you never implement a graphics output device and interactive input keyboards, you can gain knowledge and improve your skills by reading and reflecting upon the points to be made in LIFE Line. The LIFE application also has the side benefit of illustrating some techniques of interactive visual graphics which can be used much more generally.

The Starting Point

In developing a system, it always helps to know *what you want to do!* The ability to pin down a goal for a programming effort – indeed, any effort you make – is one of the most important tools of thought you have available (or can develop) in your personal "bag of tricks." Goal setting does not necessarily mean a complete and detailed description of the result – the feedback from the process of reaching the goal can often modify the details. Goal setting means the setting of a standard in your mind – and on paper – of what you want to accomplish. This standard is used to evaluate and choose among alternatives in a methodical approach to a system which meets that standard.

How to Get From Here to There

The goal of LIFE Line is a hardware/software system which enables the home brew computer builder such as you or me (the "byter") to automate the game of LIFE using relatively inexpensive equipment. It's appropriate here to give a preliminary road map of the course LIFE Line will take, as an illustration of the first steps in the development of a complicated system . . .

1. *The facts of LIFE.* Defining the rules of the game and its logical requirements always helps – after all, I would not want to confuse it with chess, poker or space war!

2. *What do I need to implement LIFE?* Once I know the rules, my next problem is to sketch the hardware and software requirements for a reasonable implementation.

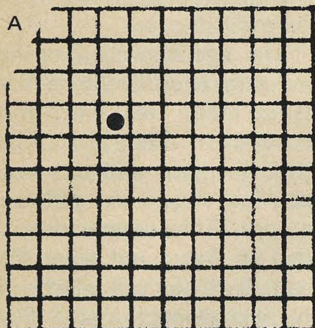
3. *Programming.* Given the necessary hardware, the biggest lump of effort is the process of programming the application. Some parts of this lump include . . .

   –*Control flow:* Outlining the major pieces of the program and their relationships.

   –*Partitioning:* A well designed system is simple! But how can the desired simplicity be reconciled with "doing a lot." One way is to partition the system into pieces. Within each piece, a further partition provides a set of sub-pieces and so on. Each piece of the program is thus kept at a level of relative simplicity, yet the whole system adds up to a quite sophisticated set of functions.

   –*Coding:* With the application design laid out in some detail, the program must be coded and debugged for a particular computer. The result could be a series of octal or hexadecimal numbers for your own computer, or a high level language program which can be translated by an appropriate compiler.

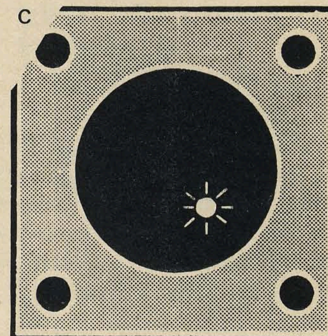
Fig. 1. Three views of LIFE: (a) on paper; (b) in memory; (c) on a display.



A live "cell" is a dot on paper.

```
B    0000000000000
    00000000000000
   000000000000000
  0000000000000000
  0000000000010000
  0000000000000000
  0000000000000000
  0000000000000000
  0000000000000000
```

A live "cell" is a "1" bit in memory.



A live "cell" is a point of light on a graphics display.

What Are The Facts of LIFE?

Ask a biologist the question "What are the facts of life?" and you will get one answer; ask a "byter" and you'll get the "real" answer – an evolution algorithm used to generate the placement and "cell" content of a square grid given the previous state of cells in the grid. The inspiration of the game is a combination of modern biology, the concept of "cellular automata" in computer science and the pure fun of mathematical abstractions. In making a computer version of the game, the simplest approach is to think of a group of individual "bits" in the computer memory – with your thoughts assigning one memory bit to each "square" of the grid. (The hand operated form of the game algorithm uses graph paper for the squares in question.) If I have a place in memory which can store one bit, it can have a value of logical "zero" or logical "one".

The LIFE game treats each location of the grid (its "squares") as a place where a "cell" might live. If the place is empty, a logical "0" value will be used in the computer memory; if the place is occupied, the "cell" will be indicated by a logical "1" value. The rules of the LIFE algorithm are defined in terms of this idea of a "cell" (logic 1) or "no cell" (logic 0) at every point in the universe of the grid. Fig. 1(a) illustrates a single live cell on a section of graph paper as I might record it when I work out the LIFE process by hand. Fig. 1(b) shows a similar section of the computer memory in which bits ("0" mostly, but "1" for the cell) stand for the content or lack of content of a square on the grid. Fig. 1(c) shows a third view – the output of a program which puts the computer memory bits of the grid onto a graphics display.

Look again at Fig. 1(a). The "cell" on the graph paper grid is a black dot placed in some location. Count the number of graph paper squares which directly surround the live "cell" location. There are 8 possible places which are "nearest neighbors" to the place held by the live cell. Similarly, if you pick an arbitrary square on the graph paper, you can count up its nearest neighbors and find 8 of them also. The rules of the LIFE algorithm concern how to determine whether to place a "cell" in a particular square of the grid for the "next generation", given the present content of that square and its 8 nearest neighbors.

What are the properties of a specific grid location of the game? I've already mentioned its binary valued nature (it has a "cell" or it doesn't) and its neighbors. One more property which is crucial to the game of LIFE is that of the "state" of its 8 nearest neighbor squares. For LIFE, the "state" of the neighbors of a grid location is defined as "the number of occupied neighbors." In the examples of Fig. 1, the "state" of the grid location with the live cell is thus "0" (no neighboring cells), and the state of any cell location which touches the single live cell's location is "1". If I were to fill the entire graph paper or its memory equivalent with live cells, the state of any grid location in the middle would be "8".

Stated in words, the rules of the LIFE algorithm determine the content of each grid location in the "next generation" in terms of its present content and the state of its nearest neighbor grid locations. The rules divide into two groups depending upon the present content of the grid location whose "next generation" value is to be calculated:

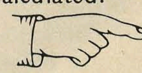
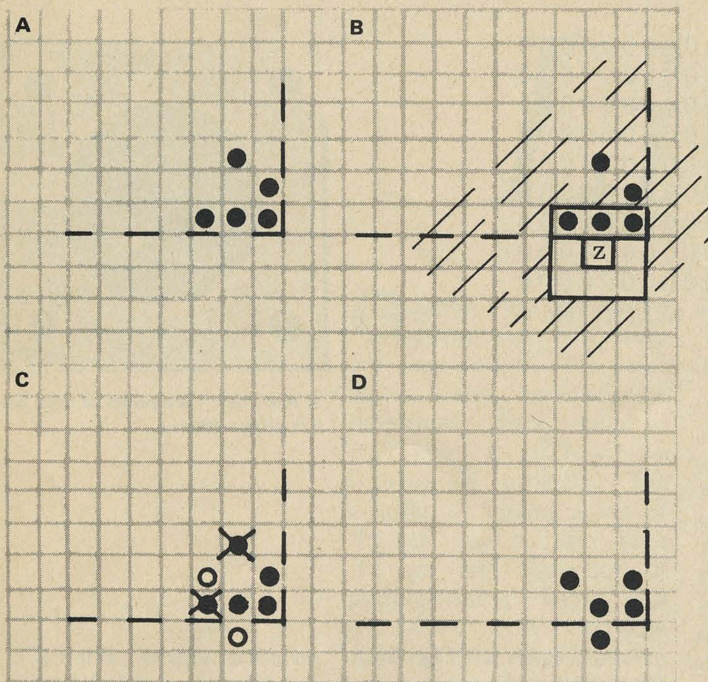


Fig. 2. (a) A "glider" generation #n. (b) Examining location "Z" and its nearest neighbors. (c) What has to change for generation #n+1. (d) The second phase of the glider (generation #n+1).



Rule 1. *LIVE CELL LOCATIONS. If* the location to be evolved has a live "cell" at present ("this generation") then,

1.1 *Starving for Affection. If* the location to be evolved has a state of 0 or 1, there will be no cell at the location in the next generation. Metaphorically, if the cell has only one or no nearest neighbors it will die out for lack of interaction with other members of its species.

1.2 *Status Quo. If* the location to be evolved has a state of 2 or 3, the present live cell will live into the tomorrow of the next generation.

1.3 *Overpopulation. If* the location to be evolved has a state of 4 thru 8, there will be no cell at the location in the next generation. Metaphorically, the cell has been crowded out by overpopulation on a local basis.

Rule 2. *EMPTY LOCATIONS. If* the location to be evolved has no live "cell" at present ("this generation") then,

2.1 *The Sex Life of Cells. If* the location to be evolved has a state of 3, a new cell will be "born" in the formerly empty location for the "next generation." Metaphorically, the three neighboring "parent" cells have decided it is time to have a child.

2.2 *Emptiness. If* the location to be evolved does not have three cells in neighboring locations, it will remain empty.

This is the simplest set of rules for the LIFE algorithm, a version which will allow you to begin experimenting with patterns and the evolution of patterns. More complicated extensions can be made to provide an actual interactive (two people) competitive game version; an interesting variation I once implemented is a LIFE game with "genetics." In the genetics variation, each grid location (graph paper square) is represented in the computer as a "character" — an 8 bit byte — of memory. The character in the square is the "gene" pattern of that cell. Then, when rule 2.1 is implemented, LIFE with genetics uses a set of genetic evolution rules to determine which character will be put in the newborn cell based upon the "genes" of the parents. (This genetic evolution program for LIFE was written for my associates at Intermetrics, Inc., as a test program to try out a new compiler's output.)

How Do You Use The Facts of Life?

To illustrate the facts of LIFE, a hand-worked example is a valuable tool of understanding. Consider a "typical" pattern of LIFE as shown in Fig. 2(a). Fig. 2(a) shows what LIFE addicts call a "glider" for reasons which will become clear a little bit later in this article. The glider pattern of Fig. 2(a) consists of the five cells indicated by black dots, and their positions relative to one another. I have also indicated a dotted line in all the illustrations of Figs. 2 and 3 as a fixed reference point in the grid.

The algorithm for evolving one generation to the next is illustrated for one grid location in Fig. 2(b). The LIFE program will examine each location in the grid one by one. This examination is used to figure out what the content of the cell will be in the next generation according to the facts of LIFE. Since these facts only require knowledge of the given grid location Z and its 8 nearest neighbor locations, Fig. 2(b) depicts a box of 9 squares including Z. The rest of the universe is shown shaded. To determine what grid-space location Z will be like in the next generation, the LIFE program first counts up the live cells in all the nearest-neighbor positions. The count is the "state" of Z. In this case there are 3 live cells on the top edge of the box containing Z. Then, the program chooses which rule to use depending upon whether or not location Z has a cell. In this case, Z is empty so the "empty location" set of rules (numbers 2.1 or 2.2) is used. Since the state of Z is 3, rule 2.1 applies and a cell will be born in location Z for the next generation.

Now if I had a true "cellular automaton" to implement the LIFE program, all grid locations would be evolved "simultaneously" — and very quickly — in the computation of the next generation. In point of fact, however, I have

a computer which can only handle 8 (or 16) bits at a time which are stored in words of memory. For small microcomputers, these bits for the LIFE grid will be stored as "packed" bit strings and will be accessed by a series of subroutines which will be described in LIFE Line when the time comes. I have to sequentially look at every bit of the internal LIFE grid of the program and examine its *old* nearest neighbors in order to calculate its new value. I emphasize *old* for the following reason: if I store the new value of the grid location just evolved back into that location with no provision to recall its old value, I'll end up with a mixture of old and new data when I look at the next grid location in the row. That mixture is not part of the rules and constitutes a "faulty" program for evolution. It turns out to be sufficient to remember all the data in one previous row before it was changed in order to calculate the next row after the change. Similar problems of keeping track of partially updated data often occur in computer programming, to be solved by the identical technique of temporarily remembering a copy of the un-updated data.

In Fig. 2(c), the result of examining all the grid locations in the vicinity of the glider of Fig. 2(a) is illustrated. The changes are indicated by three notations for cells:

○ — this indicates a new cell generated by rule 2.1

⊗ — this indicates an old cell which dies by rules 1.1 or 1.3

● — this indicates an old cell which is retained by rule 1.2

Generation "n+1" of the grid of LIFE is illustrated in Fig. 2(d), which was obtained by "executing" the changes noted in Fig. 2(c). When the LIFE program is run, all this is done automatically for each point in the grid — resulting in a new generation as soon as the computer can complete all the calculations. The patterns will be seen to "evolve" in real time as new generations are calculated and sent to the scope output. One "dot" on the scope display corresponds to each live cell of the grid pattern. Fig. 3, (a), (b) and (c), continue the pattern evolution illustrated in Fig. 2 for the "glider". In Fig. 3(a), changes to generation n+1 are indicated with the same notation as was used in Fig. 2(c). The resulting generation n+2 pattern is shown at the right. Fig. 3(b) shows the changes from generation n+2 to generation n+3, and 3(c) shows the change going to generation n+4.

One of the most interesting features of the LIFE game is the evolution of patterns which "move" across a graphics display device. With a fast enough processor, a glider such as the one used in this example will "glide" to the lower right of the screen at a breakneck speed, going off into limbo at the edge — or if the program is sufficiently "smart", reappearing elsewhere on the screen due to a "wrap-around". The reason that the glider gets its name is because of its motion attributes. Note now the fourth generation ("n+4") in the sequence repeats the original glider pattern, *but* has moved one unit along a diagonal of the LIFE grid toward the lower right. (The reference line shows this movement.) It took four generations for the glider pattern to regenerate its original form, which defines the "period" of this pattern. When you get your graphics interface up and running, you will find numerous other classes of patterns, some of which have periods which run into hundreds of generations. There are also other forms of moving patterns similar to the glider.

What Do I Need to Implement LIFE?

The fun part of LIFE is to experiment with patterns of cells and observe how the evolution from generation to generation changes with patterns and classes of patterns. In the lexicon of LIFE lovers, there are whole classes of "gliders", "space ships", "blocks", the "blinkers", "beehives", the "PI" and other patterns. You'll be able to set up initial configurations of these and other patterns, and observe the course of evolution using the hardware/software system

Fig. 3. (a) Third phase of the glider. (b) Fourth phase of the glider. (c) Back to the first phase, but displaced!

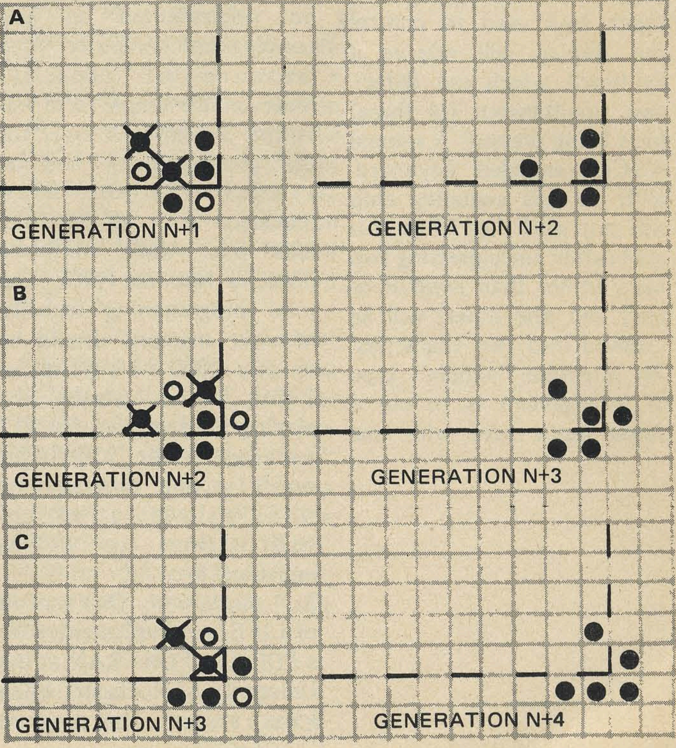
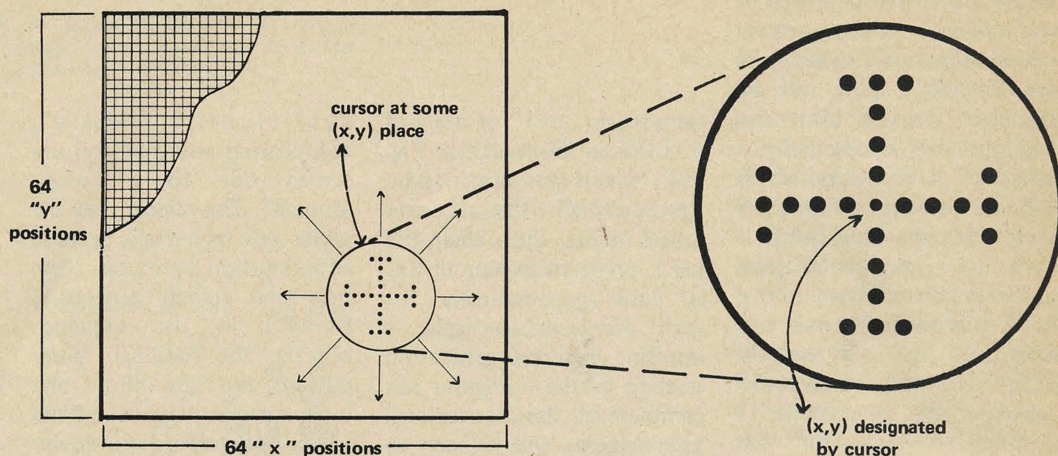


Fig. 4. The LIFE grid display with cursor detail (showing suggested pattern).



concepts of LIFE Line. The hardware requirements of this application's first simple form are three:

1. *An input method.* The best all around input you can get for your computer is an ASCII encoded typewriter keyboard. This hardware will be assumed, with 7-bit ASCII codes used in the examples of programs. If you feel like embellishing the program with special hardware, a "paddle" with several keys can be wired in parallel with your main keyboard to control the special functions of the LIFE program. The input keys used to control the display will require a keyboard which can detect two simultaneous (or three) keys being pressed. A normal ASCII encoded keyboard with an LSI encoding chip will not work "as is" in this application since pressing two keys (other than *control* or *shift* and one other) will be resolved into two characters. An alternate "paddle" type of arrangement is to use a single input port with one switch key switch for each bit of the port, debounced by software. A keyboard which is encoded by a diode matrix can be used since the diode matrix will give a new code (logical sum) based upon which keys were depressed.

2. *A processor.* The game can be implemented on any conventional computer. As a measure of capacity, however, the simple form will assume a 64x64 bit array for the playing field, and an available home brew processor such as an Intel 8080 (i.e.: Altair), Motorola 6800, or National PACE. The total programming capacity of your memory should be roughly 4000 8-bit words, or 2000 16-bit words; the playing field will require 512 8-bit words, or 256 16-bit words – and programming will include a set of subroutines to access individual bits.

3. *A display.* My first version of LIFE was implemented on a PDP-6 in FORTRAN at the University of Rochester when I was a student. That program used a direct link out to a DEC Scope controlled by a PDP-8 – with a teletype for input. I have since implemented life programs using character-oriented terminal output and line printers.

The display to be used for LIFE Line purposes I'll leave undefined in detail, but with the following characteristics: It should have an X-Y selection of coordinates for display elements (LIFE grid locations), which can be individually controlled. Its size will be assumed 64x64.

A Note Regarding Speed

The LIFE algorithm to be illustrated in LIFE Line is optimized fairly well for speed – a requirement which will become obvious in the context of your own system if you use a typical microprocessor. With a fairly large pattern of cells, it may take as much as a minute or more to compute the next generation. Trading off against speed is memory size – use of a packed bit structure is necessary if the matrix and programs are to fit in a micro computer which is inexpensive. But the packed bit structure requires time to access bits (eg: the shift/rotate instructions several times might be used in the access process). I predict that the program will be "dreadfully slow" if run on an 8008, and perhaps passably quick if you use a 6800 or 8080. ("Passably quick" means under 10 seconds per generation.) A used third-generation mini (high speed TTL) would be ideal.

User Features

No application is complete without taking into consideration the *user* of the system. The interface which controls the system is an important section of the design. There is a temptation on the part of individuals such as you or I to say words to the effect: "Since I am making it for me, who the heck cares about the user interface." But! Removing the system from the working product realm to the purely personal realm does not eliminate the need to design a

usable system. You have at least one user to think of — yourself! In point of fact, however, I doubt that any reader who builds a scope or TV graphics interface will be able to resist the temptation to show it off to his or her family and friends; so, even for "fun" systems, consideration of users is still a major input to the design.

The user interface for the LIFE program will provide the following functions to enable a pattern to be drawn on the screen and initiated:

1. *Cursor.* The display output should provide a "cursor" which is maintained all the time by a subroutine in the software at a given "X" and "Y" position of the matrix. Fig. 4 illustrates the point matrix of the screen (here assumed 64x64) and the cursor pattern. The cursor is a visual feedback through the display to the user of the LIFE program, illustrating where the program will place or erase information. Fig. 4 shows a blow-up of one possible cursor pattern.

Two additional features are required for a useful cursor output of the program for LIFE. These are:

— A blinking feature. Suppose you have filled the screen with a complicated pattern drawn with the cursor controls described below. A significant number of the screen points are now filled with dots — and there will be a strong tendency to confuse the cursor pattern of Fig. 4 with the actual data pattern you have entered. A "blink" feature can be built into the programs which create the cursor so that you will always be able to distinguish it by its flashes.

— A blanking feature. For the LIFE game, a necessary attribute of cursor control is the ability to blank out the cursor during the actual evolution of patterns. I consider this necessary due to observation of a demonstration LIFE program for one desk top programmable CRT terminal: its cursor is always present and mildly annoying when the LIFE game is in operation.

A basic way to make the cursor disappear from view at certain times is to require active control by cursor display routines when the program is in its input mode. If the LIFE program leaves the input mode to go evolve some patterns, the cursor will die a natural death until the active maintenance is resumed on return to the input mode.

2. *Cursor Control.* The whole purpose of the cursor is to provide a means of feeding back to you — the user — the current grid location the LIFE program is pondering. Movement of the cursor provides the opportunity for three types of data entry to the program:

— Positioning of the Cursor. By simply moving the cursor under control of the keyboard (see below) you can direct the LIFE program's attention to different parts of the screen.

— Sowing Seeds of LIFE. By moving the cursor while indicating a "birth" function, the cursor will leave a trail of "cells" indicated in the display by illuminated points. (One keyboard key is required for this function.)

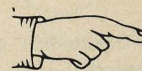
— The Grim Reaper. By moving the cursor while indicating a "death" function, any cells in the path of the cursor will be eliminated, by turning off the corresponding display point. (One keyboard key is required for this function.)

Motion control is also used to enter data. By picking a data key and at the same time depressing one or two of the cursor direction keys, a "trail" will be left. A timing loop in the input program will be used to set a reasonable motion rate in the X (horizontal) and Y (vertical) directions, so that the data entry will be performed automatically as long as the keys are depressed. The motion control keys and useful combinations are illustrated in Fig. 5.

3. *Program Control Commands.* This is the section of the LIFE program design which is the software analog of the "backplane" data bus concept in a hardware system. LIFE Line concerns a modular LIFE program which will be subject to many variations and improvements.

Birth — the cursor leaves a path of "cells," illuminated points.

Death — cells in the cursor's path are eliminated.



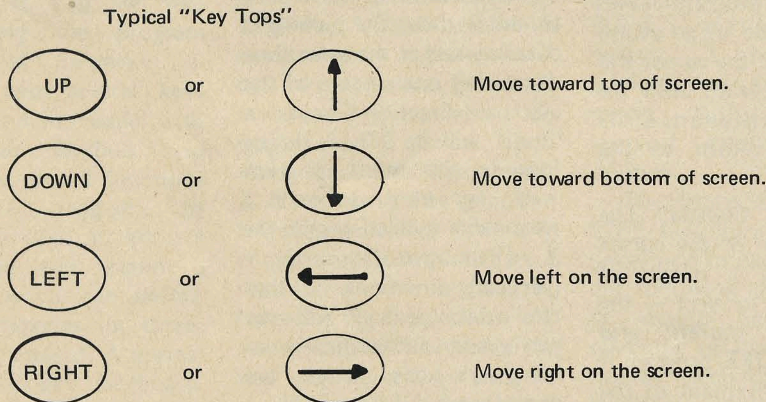
KILLING TWO BIRDS WITH ONE STONE, or "HOW I DESIGNED A GENERAL INTERACTIVE GRAPHICS SOFTWARE INITIALIZATION PACKAGE IN THE GUISE OF A SPECIFIC APPLICATION.

The ideas contained in this article are by no means limited to control of the graphics display type of device in the LIFE context used for this application. The only necessary connection between the LIFE program proper and the display "drawing" and updating functions is in the existence of several subroutines needed to turn on/turn off selected points, and the ability of the display input ("drawing") routines to call the LIFE program. One logical extension of the program control mechanisms to be included in LIFE Line is to allow the invocation (ie: activation, calling, etc.) of other programs and games which use the display.

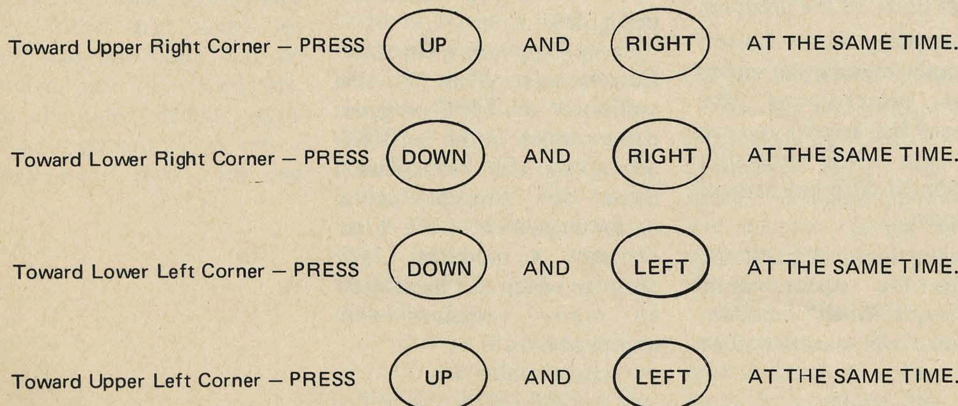
When the "drawing" routines are up and running, even before you hook up the LIFE algorithm proper, you'll be able to manipulate the contents of the scope under software control and draw pictures on the screen.

Fig. 5. Cursor motion control commands.

The following commands (one key on your keyboard for each) are used to simply move the cursor in one of the grid directions at a rate set by the cursor control software:



The following combinations can be used to achieve motion in diagonal directions:



Remember that all eight of these possibilities can be used to "sow the seeds" or erase data if the appropriate data key is pressed simultaneously.

The first demonstration of LIFE in these pages is just the bare bones of a LIFE program. When it is fully described you will see the input display routines, the evolution algorithm, the program control *mechanism* and little else. The program control *mechanism*, however, is quite general and will be used to integrate additional commands, variations on LIFE, etc. The means of achieving this modularity is a set of "hooks" which enable you to add commands beyond the bare minimum by coordinating new modules with the program. The following is a minimum set of program control commands for the first version:

RUN – a key assigned to this function will terminate the input ("drawing") mode, and begin the "run mode."

DRAW – a key assigned to this function will be tested during the "run" mode to cause a return to the "draw" mode.

CLEAR – a key assigned to this function will be used to clear the screen in the "drawing" mode, leaving only the cursor and a blank screen.

The above features are only a minimum set of user controls for LIFE. Additional program control commands which will prove invaluable when added include:

SAVE/RESTORE – commands to write and read LIFE patterns on cassette tape or other mass storage device in your home brew system.

INITIALIZATION – functional key entries for the generation of various "standard" LIFE patterns placed at the current cursor location.

Next month, LIFE Line will enter into the realm of software design to describe the LIFE program software in more detail.

LIFE Line Glossary.

Communication of meaning requires definition of terms. The following is a listing of selected terms used in LIFE Line with short explanations. The terms which are marked "L" are primarily significant only in the LIFE application – all others are fairly general terms.

"Active Control" – in the LIFE example, a desired requirement for the cursor is that it disappear automatically if not continually refreshed. This can be accomplished in software by instituting a "garbage sweeper" for the screen which clears the screen memory periodically and updates from the latest non-cursor sources of data. Normally, the cursor control/display subroutine would be called after the screen is updated – but if the cursor control routine is not called, the cursor will be absent after garbage sweeping. The cursor is thus said to require "active control" because it must be explicitly posted on the screen following the garbage sweeping operation if it is to appear at all. (L)

"Algorithm" – this term has a formal mathematical origin as the generalized methodology for arriving at some result. In the computer science area, it retains this definition: an algorithm is the most general processing required to achieve some result. "Algorithm" is a term which includes the term "program" in the following sense: a program is an algorithm (general) as written and coded for a specific system.

"Application" – an application is a specific system designed to accomplish some goal. In the computer systems area, applications are generally composed of hardware and software components which must "play together" to accomplish the desired functions. The LIFE Line's target – a working game of LIFE – is an example of an application.

"Backplane Bus" – the hardware concept of a set of wired connections between identical terminals of multiple sockets. In modular systems, the common wiring makes each socket identical to every other socket. Hardware modules can then be inserted without regard to position in the cabinet containing the equipment.

"Cellular Automata" – conventional computers employ a serial or sequential method of processing. One instruction, then the next, is executed in a time-ordered sequence. The "cellular automata" concept is one way of visualizing large and complicated parallel computing elements. Hypothetically, the LIFE game could be played by such a cellular computer, one which calculates each matrix element simultaneously. In the present state of computer technology, this is not possible, so you have to settle for a simulation of the parallel computation's result, using a serially executing program.(L)

"Coding" – the process of translating a functional specification of a program or routine into a set of machine readable elements for actual use in a computer. Coding can mean writing FORTRAN statements, writing PL/1 statements, writing assembly language statements, or . . . if you have no compiler, coding is the writing of machine codes directly onto a sheet of paper using tables of op codes, an eraser and patience.

"Cursor" – a mark on a display screen used to identify a particular place. This interpretation is an electronic adaptation of the standard definition in Webster.

"Evolution" – patterns in the game of LIFE change from generation to generation according to the rules. The sequence of such changes can loosely be called the evolution of the pattern.(L)

"Feedback" – in the context of system development, feedback is the use of observed system behavior to modify and improve the design of the system.

"Functional Specification" – a functional specification of a system is one which describes "what" the system must do, more or less independent of any technology which is required to make the "what" work. It is easy to come up with loose functional specifications – the hard part is to refine the specification and pin it down to something which is "do-able" in a given context of technology. I have a functional specification in my mind, for instance, of a useful interplanetary travel method – but whether or not I ever see such a system depends upon advances in physics, engineering and economic understanding. *BYTE* often concerns itself with functional specifications of much more "do-able" systems which readers can and will implement on home computers.

"Generation" – this term in the LIFE context means the present "state" of all the locations in the "universe of the grid" at some point in time.(L)

"Implement" – technical jargon verb for the creation of a system or element of a system. A hardware designer might implement a controller or a CPU; a software programmer implements a system of programs; a systems designer implements a hardware/software combination which achieves a desired functional end.

"Indexing" – the technique of referencing data in collection of similar items by means of numerical "indices." In the LIFE Line example, the collection is that of the 64x64 array of bits in the computer representation of "grid space." Indexing by row and by column is used to pick a particular bit within this array when the program requires the data.

"Interact" – when a system "interacts" with "something/person" it is operating under an algorithm which allows conditional behavior dependent upon data. The data is obtained from the "something/person" and may in fact be influenced by previous interactions as well as new inputs. In many computer contexts "interact" has the additional implication of "quick" response in "real time." Thus when you think of an "interactive" terminal or game, you think of a computer programmed so that it keeps up with the inputs from the human operator.

INTEL 1K 2102 RAM

Factory prime, tested units. Factory selected for much faster speed than units sold by others. 650 NS. These are *static* memories that are TTL compatible and operate off + 5 VDC. The real workhorse of solid state memories because they are so easy to use. Perfect for memories because they are so easy to use. Perfect for TV typewriters, mini-computers, etc. With specs.

$3.95 ea. or 8 for $30

SIGNETICS 1K P-ROM

82S129. 256 x 4. Bipolar, much faster than MOS devices. 50NS. Tri-state outputs. TTL compatible. Field programmable, and features on chip address decoding. Perfect for microprogramming applications. 16 pin DIP. With spec. $2.95 ea.

8T97B

By Signetics.

Tri-State Hex Buffer

MOS and TTL Interface to Tri-State Logic.

Special $1.49

DO YOU NEED A LARGE COMMON ANODE READOUT AT A FANTASTIC PRICE?

S.D. presents the MAN-64 by Monsanto - 40 inch character. All LED construction - not reflective bar type, fits 14 pin DIP. Brand new and factory prime. Left D.P.

$1.59 ea. 6 for $7.50

MOTOROLA POWER DARLINGTON - $1.99

MJ3001 - NPN - 80 Volts - 10 Amps - HFE 6000 typ. To-3 Case. Ideal for power supplies, etc. We include a free 723 regulator w/schematic for power supply with purchase of the MJ3001. You get the two key parts for a DC supply for only $1.99. Regular catalog price for the MJ3001 is $3.82.

LARGE SIZE LED LAMPS

Similar to MV5024. Prime factory tested units. We include plastic mounting clips which are very hard to come by.

Special 4 for $1

48 HOUR SERVICE

You deserve, and will get prompt shipment. On orders not shipped in 48 HRS' a 20% cash refund will be sent. We do not sell junk. Money back guarantee on every item. WE PAY POSTAGE. Orders under $10 add 75¢ handling. No C.O.D. Texas Res. add 5% tax.

S. D. SALES CO.

P. O. BOX 28810 DALLAS, TEXAS 75228

"Lexicon" – the list of buzzwords in any given field. This glossary is a subset of a lexicon coupled with explanations. In compiler and language design, "lexical analysis" is a derivative of this term concerned with language keywords and their relation to a grammar.

"n", "n+1", "n+2"... – when it is useful to specify a sequence of things, where no particular number is intended, a "relative" notation of the sequence is useful. "n" is some arbitrary number; "n+1" is one number greater than an arbitrary number, and so on. When I say "generation n+1" of LIFE, I mean the next generation after generation "n" where "n" is arbitrary.

A suitable LIFE display peripheral is an oscilloscope graphics interface such as the Digital Graphic Display Oscilloscope Interface designed by James Hogenson and printed in the May 1975 issue of ECS Magazine, *the predecessor to* BYTE. *The graphics interface article will be expanded and published in* BYTE *No. 2, October 1975. Until supplies are exhausted, back issues of May ECS (and earlier articles) can be ordered at $2 each. Orders and inquiries regarding ECS back issues should be sent to M. P. Publishing, Box 378, Belmont MA 02178.*

"Partitioning" – the technique of "divide and conquer." Rather than view a complicated system as a monolithic blob of "function," an extremely useful design method is to partition the system into little "bloblets" of function which are easy to understand. Hardware designers of CPUs thus think of MSI chips as sub-elements in partitioning; hardware systems designers think of CPUs and peripherals and memories as sub-elements of partitioning, and software designers consider divisions of complicated programs and program libraries as their sub-elements.

"State" – the present condition of some system, or elements of the system. This term applies to any system which has "memory" to distinguish one possible "state" from another. The term applies equally well to small sub-elements of a system such as the bits of a memory: in the LIFE Line context, the "state" of a single grid location is a number from 0 to 8 counting how many "neighbor cells" are present.

"System" – the most general of all general purpose terms. A system is a collection of component elements (technological, hardware, software, human-interface) selected to play together according to some design or purpose. A system is a human-invented way of doing things.

"Undefined in Detail" – I know what is needed, can specify its interface, but am not at present supplying the detail design. This is a useful attitude since it allows for "plug compatible" designs differing widely in their internal principles of operation. A similar expression would be to call the subsystem in question (the graphic display mentioned in this LIFE Line example) a "black box" and leave it at that. (Software always seems to reference hardware in this way, and hardware does the same for software.) A synonym for the attitude is the mathematician's way of saying "in principle there exists a solution!" without telling you what it is.

"Universe of the Grid" – this is the set of all possible places in which a LIFE cell could be placed. These places are called "grid locations".(L)