## LEAP, THE FASTEST-KNOWN CURSOR MOVING METHOD

LEAP at first seems like nothing more than a variation on a standard string-search or FIND facility. In practice it is very different. All experienced Cat users that I have spoken with miss this one feature more than any other. The extraordinary facility of cursor motion it confers is not apparent from a written description; nonetheless, a paper presentation is what we will have to make do with.

In the Sywft (and the other products) there were two LEAP keys, positioned accessible to the thumbs either under or on both sides of the space bar on an otherwise standard keyboard layout. Leap was created in response to the extreme sloth of having to use a mouse or other GID to move the cursor, especially in text where the targets are small. As Fitts's law tells us quantitatively [Card et al 1983] the time it takes us to move a cursor to a target increases with the distance to the target and also increases as the target gets smaller. Letters on a display are small.

Additionally, a lot of cursor motion takes place during writing and editing text, which is primarily keyboard-based (using LEAP in graphic contexts will be discussed separately). It is both tiring and time-consuming to be moving the hands from keyboard to GID and back repeatedly. Research [Card et. al. 1983 pg. 264] tells us that while it typically takes an average typist .2 seconds to type a character, it takes .4 seconds to move the hand to the GID, 1.1 seconds to point to a character on the display, .2 seconds for the mouse click there, and .4 seconds to move the hand back to the keyboard--ignoring the time taken in cognitive functions for this process. This totals 2.1 seconds. Studies with the Cat showed that on the average it took fewer than four characters of typing to accomplish a LEAP. Combined with the stroke of the LEAP key (.2 seconds again) we have a total time of under 1 second for LEAP. In other words, LEAP is typically about twice as fast as using a mouse or other GID.

For off-page moves, where the user has to use scroll bars, page number selectors, or other mechanisms, LEAP had an even greater time advantage, as well as the cognitive advantage of there being only one mechanism for cursor moves of any length instead of having different strategies depending on where the target is. LEAP is used for many kinds of information retrieval, including looking up things in help files.

LEAPs efficiency makes it ideal for voice-controlled systems and for use by the blind and those with motor impairments.

## HOW TO LEAP

You could LEAP forward from your position in the text by using the right LEAP key, and backwards by using the left LEAP key (and the reverse in languages that are written right-to-left). To quote the manual [Alzofon, 1989], "Hold down the appropriate Leap key with your thumb, and while holding it down, type what you want to find, then let go."

The target of a LEAP is always a particular character in the text. Unlike some editors (EMACS comes to mind) you always type the target character first, followed by other characters to create

the search pattern. For LEAP to be effective, it must be fast, preferably finding the next instance of the building pattern in under 250 msec. (approx. human reaction time). This can be achieved even in very large texts with a single-processor machine by a number of techniques including using the Boyer-Moore fast string search algorithm, by pre-building search tables, and by searching ahead (and backwards) for instances of the search pattern while the user is typing or thinking.

## RETHINKING CURSOR DESIGN

In testing the earliest prototypes of our Cat software on naive users we discovered a problem that we had not suspected: the usual cursor design, [illustration of the doubly bifurcated cursor], invented at PARC [check this] and used by nearly everybody since is confusing.

This came as a surprise because we had all become so acclimated to the usual cursor that it seemed fine (one exception: I had observed that some people, including myself, had to locate it lower with respect to a line of type than expected, and thus sometimes selected text one line higher than intended.)

Here is the problem: the usual vertical bar cursor has two loci of action. It refers to the character to its left if you use the DELETE (also called BACKSPACE) key, but refers to the character location (not to the character) to its right when you type. [reference, Raskin, Jef "Systemic Implications of an Improved Two-Part Cursor," Proceedings of the Computer Human Interface Conference, Austin, 1989]. If you intend to delete a character you must remember to move the cursor to its right, if you intend to insert, you move the cursor to between the characters. It is only in the latter case that naive users find that the graphical representation matches the action.

If you use LEAP, then the target of your cursor motion is a single character. Should the cursor land to the right or to the left of this character? If it lands to the left, this is correct if you intend to insert at that character locationl. If it lands to the right, this is correct if you intend to delete that character. The computer, it would seem, had to know your intention in order to display the cursor correctly. This was a difficult time at Information Appliance, with the company divided in a holy war as to where the cursor should land. It was an uncomfortable time, and many had become disgusted with my insistence that there had to be a solution. But one day in a discussion with Ren Curry, Jim Winter, and me (I think we were the only ones who hadn't given up on the problem), the solution hit us simultaneously: change the cursor to the kind that forms a rectangle about (or an underline beneath) the letter; when you LEAP to the letter, the cursor lands ON the letter itself. Now you can insert there or delete there without ambiguity.

This engendered another problem: Say you had the word "BIRD" with the cursor on the "I" and you type the letter A. You get "BAIRD". But where is the cursor? Again you seem to need to know the intent of the user: if the cursor is on the "I" does it indicate that that letter will be the insertion point of the deletion point. Deletion should be the inverse of insertion, so that DELETE should delete the "A", but if you type another letter, it should displace the "I".

Similarly, if the cursor landed on the "R", then pressing DELETE should leave "BID", which is fine, but where should the cursor be after the delete? We had the same problem: it depends on what you intend to do next.

The solution to this problem seems straightforward in retrospect: as soon as you type or delete, the cursor splits into two parts that sit on two consecutive characters: the first character has the delete cursor on it, the second has the insert cursor on it. The cursors are designed so that when they coalesce (as they must after a LEAP or other cursor movement) it is graphically clear that they both reside on the same character.

I said "consecutive" rather than "adjacent" for while consecutive characters are usually adjacent, they are not in cases such as the first being at the end of a line. For left-to-right languages the delete cursor is typically to the left of the insert character; for right-to-left languages it is the other way around. For vertically-printed languages the insert cursor would be below the delete cursor and they'd alternate linewise for boustrophedonic scripts.

When using a GID with the two-part cursor, target acquistion is easier as the targets, namely characters, are larger than the spaces between them and the hot region is easier to visualize (the hot region for the PARC cursor extends about halfway into each character on both sides of the space between them, but nothing visually delineates the boundary, effectivly making the target smaller it the user is to feel secure in their aim.)

We started with the observation that naive users sometimes had trouble (albeit briefly) in understanding how a cursor works, a phenomenon that is so transient involving an interface detail that is so familiar that it is rarely, if ever, questioned. When we went to the two-part cursor that condensed on a single character when doing a cursor move, we found that the initial confusion as to which character a cursor points out disappears for nearly all users, solving our original problem, as well as a few others in passing.

## REDUCTION

In physics some of the greatest advances have come from a reduction of what were considered separate phenomenon into one framework. Newton simplified our understanding of the universe by showing that the fall of an apple and the paths of the planets were both consequences of the same, simple law of gravity. In interface design the user's learning burden (and the programmer's!) is eased when what were separate concepts can be seen as examples of a more fundamental one.

This is not the same as just putting every command you can think of into a portmanteau application, or building disparate applications into a suite of programs. For example, a spreadsheet consists of rows and columns of text. The only difference between a spreadsheet and a word processor that can create tables is that the text (often in the form of numbers) in the boxes can have formulae behind it, where the variables in those formulae can be derived from other places in the table.

We can reduce the two concepts to one by creating a word processor where any object in the text can have an underlying expression (which can be numerical, a string operation, or any programmable construct) including assignment to a name which can be used in other expressions. If one has such a word processor, then any table in that word processor is a spreadsheet. This is a lot less to learn than a separate spreadsheet and word processor, and it offers further rewards: the results of a computation are not limited to a spreadsheet format. A result can appear anywhere in text, and anything mentioned in text can be used as an input to a table. This goes far beyond embedding a piece of a spreadsheet in a word processor or a block of text in a spreadsheet.

If any text can have an underlying structure, that program can be a referece to a net address (URL) or other hypertext link.

If, to name one, a sort operator can be applied to columns and lists in text, then many data base operations can be reduced to word processing. An address book is just a sorted list of names and addresses in a word processor (or spreadsheet; they are all the same). If a process can be associated with any text, then phone numbers, wherever they appear, can be dialable. One does not need a special "address book" program to accomplish this.

Nor do you need a special fax program or email facility: is there any difference between selecting a section of text and deleting it, selecting a section of text and italicizing it, selecting a section of text and emailing it, or selecting a section of text and faxing it? In each case all that is required is the correct operator to apply to the selection. We do not need separate environments, separate applications for all of these.

ASIMOV'S THREE LAWS OF ROBOTICS, REVISED

1. A computer should never harm a user, nor through inaction allow such harm to occur.

*"Harm" includes psychological harm, such as frustration, annoyance, or wasting the user's time or requiring undue effort..*

2. The software should obey the instructions of the user, unless following such instructions would violate the first law.

*A computer is a servant to the human, the human's role should never be that of serving the computer's needs.*

3. A computer should never allow its software or the user's data to become destroyed, damaged, or misplaced unless such loss is necessary to preserve the first and second laws.

*A user's data and intentions are sacred, the computer can never presume that it knows better.*

UNDO / REDO

Ultimately UNDO should work back to the empty, new state of the machine, REDO should predict your next work...

Seriously, without REDO, UNDO should only be single level (Undo undoes the last Undo if it was the most recent user action). Granularity: one user event.

Use of GOMS modeling in development

MARKTING-DRIVEN DESIGN ERRORS

There were a number of mistakes in the design that were dictated by Canon's marketing. One that can be given no more generous a label than "stupid" was the interchange of the ERASE (backspace or delete) key and the UNDO key. Our design had the backspace key in the standard, upper-right location. It was a relatively large button since its use is so frequent. Directly under it was the UNDO key, with its related function. Canon--without consultation--interchanged the position of those two keys, on the grounds that UNDO was a wonderful new feature (it is, of course) and, therefore, should be on the larger key. This made the keyboard frustrating to any touch-typist, since you hit "UNDO" when you wanted to backspace. Users of the CAT cannot readily move back and forth between the CAT and standard keyboards. I sent an enormous number of trans-Pacific messages on this score, to no avail.

One of the features of the CAT, as designed, was that it would go into a low-power state if it had been unused for a few minutes. This did not hamper use as it came on in a fraction of a second when any key (including SHIFT keys, unlike some current products) was pressed. The greatest saving was realized by turning off the power-hungry CRT circuits. This was years before this kind of design became common with the "green" stickered products of the 90's. Marketing decided that a blank screen was a waste of a marketing opportunity, and they changed the system to have the words "Canon Cat" bounce around on the screen when it went into its sleep state. Of course, this meant that the power consumption did not significantly drop when it was sleeping, destroying the whole rationale. Nonetheless, their literature still touted the energy-efficient "sleep" state as originally designed.

A handle was designed into the palm rest in front of the keyboard (the palm rest was also ahead of its time). We tested the design to make sure that you could carry the product one-handed without it banging against your legs as you walked. Canon suprised us by moving the handle to the top, placing it such that you could not close your hand around the handle and so that the Cat would slip out of your grasp if you used the handle. It required using your other hand to carry it safely. They never explained why they rejected our carefully-tested handle, except to say that they didn't like the appearance of our design.

When creating the Macintosh I insisted on using the Sony-type floppy drive because it could sense when a disk was inserted and could automatically eject a disk under program control. The Cat was designed with such a drive, and it made disk operation nearly foolproof. Canon rejected the design because it used a competitor

Jef Raskin
8 Gypsy Hill
Pacifica CA 94044
415-359-8588 415-359-9767 Fax
KE6IGI—jefraskin@aol.com—AMA L-88

February 15, 1997

# THE SWYFT AND THE CAT

note: all company, feature, and product names are trademarks or registered trademarks.

A number of interface directions that are only now coming into prominence were foreshadowed by a series of products from Information Appliance Inc. (IAI) None of the products achieved wide recognition at the time, but interest in their revolutionary interfaces has steadily grown. This article was written in response to many requests for information about this work. The best known of the products was the Cat, a desktop machine designed for the well-known Japanese firm, Canon Inc. Information Appliance's SwyftWare for the Apple II series and the Swyft portable computer were the other major designs that came from IAI. The chief architect of these products was the company founder. He also originated the Macintosh project at Apple computer. The aim of this essay is not so much to present a retrospective as to inspire.

[Photo of the Cat]

In many ways subsequent work, such as Gelertner's LifeStream chronological and linear file model is not very different from the methods incorporated in the effective and simple SwyftCard product for the Apple II, introduced in 1983. The flavor of document-centered paradigms are currently hinted at in software such as Apple's OpenDoc and Microsoft's OLE, but a much fuller expression of docucentric computing permeated all of Information Appliance's products.

According to information from Canon, Inc., about 20,000 Cats were sold. While the Cat was in fact a bit-mapped computer with a Motorola 68000 microprocessor, with the same screen size and inherent power as contemporary Macintosh computers from Apple, Canon chose to sell the Cat though their electronic typewriter division, and limited the use of graphics to those that could be reproduced by the daisy-wheel printer that they sold with the it--namely none. Nor did they realize the importance of third-party software though Information Appliance had begun to get third-party development started (especially in the file-conversion area). Though most customers never knew of or benefitted from the Cats full capabilities, the reaction of users and reviewers alike to the product's ease of learning and use was overwhelmingly positive. [look up and quote some reviews].

## THE SWYFT

The Swyft was a small portable computer with built-in software for many common computer-

based tasks, such as word processing, graphics, data-base operations, spreadsheet, and telecommunications. The user could easily create macros to automate often-repeated tasks. To encourage software development, it was its own programming environment, though the key to enable programming was not widely disseminated as most users have neither the desire nor the competence to do software development. Standard telephone-style jacks were provided for telephone communications, there were both serial and parallel ports, the serial port being Mac-compatible, the parallel IBM-compatible.

[Photo of the Swyft]

The specifications for the Model III Sywft are: weight 4.0 lbs (1.8 kg), 11.8" X 10.6" X 1.3" (300 mm X 270 mm X 32 mm), 8 AA Nicad batteries were good for 6 hours of use (longer with Alkaline batteries), 640 X 200 supertwist Liquid Crystal Display, 13 function keys, a 2400 baud modem, and 512K of memory. The software handled file transfers to and from both Macs and IBM compatible computers. The floppy drive was external and the retail price of the Swyft Model III was to be $999. The Model I with less memory and no modem was $799. 2400 baud was fast at the time, and the memory size was very large for a portable.

One of the fundamental principles of common-sense interface design is that simple things should be simple. A single button turned Swyft on, there was no delay. If you wanted to write something, you just started typing. You needed no commands, you just start typing. All text was inserted, you could not wipe out text by typing over it.

The principle that a user's work is sacred, and should not be changed without a user command or as a side effect of the action of some other command, is an important one. Its violation can be costly. Cut-and-paste paradigms are problematical in this regard and deserve some comment, especially as we did not learn the lesson well enough when designing an analogous feature in the Cat.

## CUT-AND-PASTE AND OTHER DESIGN ERRORS

In almost all of today's word processors you can replace text by highlighting it and then typing. The highlighted text disappears and your typing replaces it. This design feature was intended to save the step of having to explicitly delete the highlighted text, that is instead of select-delete-type you need only select-type. Deletion is a byproduct of typing when there is a selection. When I gave a talk at the Bay Area Chapter of the Association for Computing Machinery's Special Interest Group in Computer-Human Interaction (Palo Alto Ca 11 Feb. 1997) I asked how many people had never lost any text due to this design. Out of the some 300 people there, only one person raised her hand.

What makes this design flaw especially grotesque is that the selection may not even be visible on the display, and it may be of any size. You can be looking at a display, not noticing that you have forgotten to or failed to move the cursor (which is not always particularly conspicuous) to the current screen and start typing. The display suddenly shifts and you can see that you are typing on some other screenful, but you do not see that you have just deleted an arbitrary amount of

text.

When designing the Cat, we made a similar design error. Originally, to move text, we had the user select the text to be moved, place the cursor where it was to be moved to, and then tap a special key to invoke the MOVE command. This way there was never that uncomfortable interim period found in CUT-and-PASTE-based moves where you have just deleted the text but have yet to place it anywhere, and a subsequent CUT costs you the text you cut while intending the first move. This is another design flaw that occasionally causes lost text for most users.

Noted author and designer Scott Kim, then working for Information Appliance, pointed out that we could eliminate the MOVE command by having the rule that if the cursor was moved (via the LEAP command, which will be discussed later), any highlighted text would automatically be moved. The sequence of actions became select-and-then-just-move-the-cursor. The text followed the cursor and stayed selected so that you could move it again if the move had been to the wrong place. The UNDO provided additional safety. Moving text this way was a safe shortcut as in the worst case, we argued, since while you might accidentally end up with some text where you didn't want it, at least you'd never lose anything. You could always move it back. This seemed wise counsel, and we adopted this design for the Cat.

In practice it was not too bad, most of the time. But making moving text a side-effect of cursor motion turned out to not be entirely successful, precisely because it was a side-effect that you might not be paying attention to when your attention was on the act of moving the cursor. I can remember vividly a few particularly annoying cases that happened to me. In one case I had selected nearly half of a fifty-page essay, for some reason. All of a sudden I remembered a phrase that I had wanted to further explain. I moved the cursor to that phrase and typed away at my idea with full speed. Of course, a huge chunk of my essay got pulled into the wrong place and my subsequent typing and editing, made without observing that I had moved a mass of text, eliminated all possibility of using the single-level UNDO.

I continued writing and editing, and only much later noticed that something was wrong. It is amazing how hard and frustrating it was to figure out what had happened! I felt that the text had been scrambled. And once I had figured out the problem, it took a lot of searching and reading to further discover just what had been moved and where, exactly, it had been moved from. Another occasion was when a few digits got accidentally moved in some data. It was impossible to figure out what had been moved as there was no redundancy.

Other users reported the same occasional difficulty, but due to a psychological phenomenon that Don Norman has so often pointed out [reference] they blamed themselves for misuse of the product rather than the design flaw that I had approved.

## TWO CURES FOR CUT-AND-PASTE PROBLEMS

In the Swyft we returned to the Cut-and-Paste model, but added a new feature where all cuts were stored sequentially in a system-created document (a separate "permanent delete" command allowed the permanent erasure of text from that document). This prevented accidental loss of text

(due to the causes discussed in this section) and the collection of cut material often proved a valuable resource. The first item in the document is what would be pasted when the PASTE command was executed. Deleting something from the Cut Document merely moved it to the beginning.

It would be a useful study to compare cut-and-paste-with-store as we did for the Swyft with the original MOVE command, with the alternative of having each document have its own associated cut document, and possibly some other alternatives, to see which causes the fewest disasters. It would not be an easy study, as the frequency with which the problem occurs is low. But low-frequency high-impact design errors are just the kind of thing that makes people edgy about and dislike a product. As various experiments have shown [cite some], an occasional, seemingly random punishment for what is normally a permissible or even desirable act can cause neurotic behavior in animals.

We should design so that this kind of punishment is never dealt us by software.

Neither the Cat nor the Swyft, fortunately, allowed deletion of text to be a byproduct of typing as do almost all current word processors, one of the reasons they were singularly unfrustrating.

ABSTRACT
Mode errors can be reduced if the user action that sets the mode has to be actively maintained by the user. The difference is exemplified in the use of a modal caps lock key versus holding a shift key to the same effect. Since the way the state is maintained has different psychological consequences for the user, it is proposed that a new term, pseudomode, be used for the case where a mode is user-maintained rather than system-maintained. Etymological and grammatical support for this proposal is presented.

KEYWORDS: Pseudomode, mode, modal, interface design, usability engineering, safety, error reduction, terminology.

## PSEUDOMODES

A pseudomode occurs when a user is pressing and holding a switch and while continuing to hold the switch proceeds to do other operations, the semantics of which are modified by the fact that the switch is being held pressed. An example is the shift key on most typewriter and computer keyboar ds.

Modes are a set of system states in which the semantics of particular user actions differ. It has been long realized that users have difficulty remembering system state and that having such state can cause operator errors. [1] However, if the action that establishes the mode is not momentary or transient but is physically maintained by the user for the duration of the state, those errors usually called mode errors are diminished or eliminated. [2] Even though user-maintained modes have been used as the basis of the interface design for commercial products [3] there is no single term in general use to describe them. It is therefore proposed that the term pseudomode be used to distinguish modes which are maintained kinesthetically from those that are maintained by a change in mechanical or electronic state of the system being used.

It is perhaps partially due to the freedom from mode errors granted by pseudomodes that many systems designed by and for computer practitioners make extensive use of them though this can degenerate to unmemorable and nearly untypable commands, e.g. shift-alt-ctrl-option-W.

## PSEUDOMODE TERMINOLOGY

The literature sometimes refers to pseudomodes as spring-loaded modes and spring-locked modes. The author has previously used the term quasimode and its adjectival form, quasimodal (I have a hunch this term might ring a bell with some readers).

The terminology spring-locked mode is inappropriate since (i) the term should not be tied to mechanics; no physical spring may be involved, (ii) the term is misleading since the essence of a pseudomode is that it is not locked (which is its great advantage), and in particular it is not locked by a spring, and (iii) it does not cause the errors modes do and should therefore not be called a mode. Objections (i) and (iii) also apply to the term spring-loaded mode. User-maintained mode as used in [2] is accurate but lengthy.

The term pseudomode has a supportable etymology, since while it shares one attribute of modes in that its use moves the interface from one region of its state diagram to another, it does not have the adverse psychological effects of true modes. Thus it is a false mode and it is appropriate to use the Greek Yeudo as a prefix: it means falsehood. Purists may object to pseudomode as the root is from the Latin modus , meaning method, but such mixed-root-language words are now wide #ly accepted

.

Lastly, there is no simple adjectival form for user-maintained mode, spring-locked mode, spring-loaded mode, whereas one can readily use pseudomodal to describe an interface feature.

REFERENCES

1. Norman, D.A. Categorization of action slips. Psychology Review, 1981, 88 (1). pp. 1-15

2. Sellen, A., Kurtenbach, G. & Buxton, W. The prevention of mode errors through sensory feedback. Human Computer Interaction, 1992. 7(2), 141-164.

3. Raskin, J. Systemic implications of an improved two-part cursor, in Proc. CHI 89 Human Factors in Computing Systems (Austin, 30 April 1989), ACM Press, pp 167-170.

ACKNOWLEDGEMENTS

I would like to thank Michael S. Miller, of the AT&T Human Interface Technical Center for suggesting we use the term pseudomode in place of quasimode.