The Influence of the UNIX® [1] Operating System on the Development of Two Video Games.

Peter S. Langston

Bell Communications Research[2]

ABSTRACT

The Lucasfilm Games Group was established to explore ways of applying the technological and methodological expertise developed at Lucasfilm (principally for film production) in a new entertainment medium — video games. One of the major tools on which that expertise is based is the UNIX operating system. It is an interesting coincidence that this operating system's early development was heavily influenced by games and the interests of game designers.[3]

This paper describes the development of the two video games ''*BALLBLAZER*''™ and ''Rescue on Fractalus!''™ and the ways in which UNIX software aided and influenced their design. The initial decision to use a UNIX system was based largely on my personal preference as leader (and sole member at the time) of the games group, but as the project progressed we discovered more and more reasons to believe it had been the right choice.

HISTORY

''Months ago in a deserted warehouse in Marin County, George Lucas met with a lone programmer...'' begins one of the many descriptions of the formation of the Lucasfilm Games Group. Needless to say, some journalistic license is evident in such accounts of the group's formation but when the purple prose is stripped away what remains is the important part of the group's charter — to look at the booming industry of video games and see in what ways the ''magic'' of Lucasfilm could be brought to bear. That ''magic'' consists of two parts: an uncompromising attention to detail (leading to a sense of involvement and realism even in the most unrealistic of situations) and the use of high-tech tools to make possible the creation of formerly impossible sequences and images. An important part of those tools was the UNIX operating system introduced within Lucasfilm by the Lucasfilm Computer Division, of which the Games Group was a part. When we formed the games group at Lucasfilm the benefits of using a UNIX system for large-scale graphics software development was already well appreciated; what remained to be seen was whether those benefits would also apply to small-scale software development for ''toy'' machines.

The first project we (the games group) undertook was a survey of the games industry to learn how games development was being carried out. We were amazed. Home video games were being produced in one of two ways. Either a single programmer would spend a year or two of spare time creating the game concept, graphics, sound, play mechanics, and writing the game program in assembler (typically in a basement on a system with too little memory, too few floppy disks, no reasonable way to make a backup, and few if any debugging tools). Or a team consisting of a programmer and one or two helpers would be allotted two to three months to carry a game from concept to debugged implementation. Programmers in either case often had no prior experience in computer programming and were plagued by flakey equipment and the lack of many aids that we considered basic: high-level languages, support tools, libraries of software, input/output redirection, hierarchical file systems, large storage devices, and an understanding of software design methodologies.

---

[1] See the last page for a list of trademarks.

[2] The work described here was done at Lucasfilm Ltd., San Rafael, Calif.

[3] c.f. ''The influence of games on the development of the UNIX operating system'' (unpublished).

When we looked at arcade (‘‘coin-op’’) video games the situation was different, but no better. The typical development cycle appeared to be: 1) come up with a game concept and get approval for it, 2) design the hardware, 3) build the hardware, 4) test the hardware, 5) write the software, 6) try the game out, 7) decide whether to continue, and 8) debug and polish. Although steps 3, 4, and 5 could be overlapped it still took 6 to 18 months to get to step 6. At that point, it was too late to make major changes or to drop the project, which would entail a big loss. As a result many uninteresting coin-op games were produced. We learned that the reason general purpose hardware was not used was that special purpose hardware is less expensive to manufacture than general purpose equipment and they wanted to sell 10,000 to 100,000 of these games. Thus a manufacturing savings of $1,000 on each unit could mean a saving of $10,000,000 to $100,000,000 overall. This combination of optimism and greed apparently blinded the industry to any thoughts of rational planning.

Media interest in Lucasfilm has always been high and so it was not long before we found ourselves being quoted in the press, pompously telling the games industry How It Should Be Done. In the midst of all this media attention our group realized that since none of us had ever taken a video game through all the steps of production we might be setting ourselves up for some embarrassing surprises. ‘‘**Lucasfilm Computer Scientists Eat Words**’’ and ‘‘**Ivory Tower Talk is Cheap**’’ screamed the banner headlines in our nightmares.

To avoid such possibilities we decided to design and implement one or two ‘‘throw-away’’ games as a combination rite-of-passage and reality check. These games would also serve as a way of identifying the areas that would profit most from the creation of some software tools. It’s a credit both to the members of the Games Group and the ideas we championed that the two ‘‘throw-away’’ games produced have gotten such enthusiastic reviews[4] — ranging from ‘‘George Lucas does it again!’’ to ‘‘setting a new standard in the industry’’.

Of course, the games never did get ‘‘thrown-away’’ and will appear in stores in the near future. As to whether they will show the world How It Should Be Done — we think so, but soon you will be able to decide for yourself...

THE HARDWARE

The Atari 800 computer contains a 6502 microprocessor chip, 32K to 64K of RAM, and several special purpose chips designed by Atari. The ‘‘ANTIC’’ chip handles DMA, some interrupts, vertical and horizontal scrolling, and the vertical line counter. The ‘‘GTIA’’ chip handles the graphics generation, graphics objects, collision detection, and miscellaneous switches and triggers. The ‘‘POKEY’’ chip handles the keyboard, serial communications port, timers, hardware random number generator, potentiometers, and the sound generators. The ‘‘PIA’’ chip handles the joystick controllers and interrupt lines.

An Atari 800 computer, complete with keyboard, power supply, and cabling that allows use of a conventional television set as the screen, costs about $100 in discount stores today. The 6502 microprocessor chip used in the Atari machines (also used in the Apple II and other home computers) costs about $1.50 (retail) in single unit quantities. All in all, the Atari 800 and the 5200 are *small* computers. Nevertheless, we found that writing good programs for a small computer often requires using tools that are very large. Such a discovery rules out using the Atari machines to run the tools. A much better arrangement is a hybrid that uses a large machine for editing and assembly and a slave Atari machine for testing.

THE TOOLS

Many pieces of software were created by our group specifically for the game development projects. Typical of these were:

1) a cross assembler

2) a library of macros for the cross assembler

3) a pair of programs that download an Atari 800 home computer over a serial line from a DEC VAX 11/750

---

[4] OMNI magazine named one of the games in its ‘‘Top Ten Video Games of the Year’’ (tied for second) even though the game has never been released.

4) a similar pair of programs that download an Atari 800 floppy disk over a serial line

5) a simulation of flight dynamics and graphics on an Evans & Sutherland Picture System I

6) a program that turns an Atari 800 into a drum rhythm machine

7) a music scoring system using *pic* and *troff*

8) a storyboard editor that runs on a Sun Workstation

9) a 6502 disassembler

It was not surprising to us (although it was to some people in the industry) that the majority of our software ended up written in high-level languages. The cross assembler and its macro library were written in Lisp. The Atari end of the download programs and the drum machine program were written in our Lisp-like cross assembler. The rest of the tool software was written in C.

Lisp was chosen for the cross assembler because it only took two weeks to implement a cross assembler that allowed both assembler macro definitions and arbitrary Lisp expressions to be included in the assembly task. This allowed us to extend or reconfigure the assembler as we discovered unforeseen needs and deficiencies.

The macro library had two important effects. First, it allowed us to write in a pseudo-structured assembler that included ''*if-then-else*'', ''*for*'', ''*while*'', and other familiar constructs. Second, it acted as a first step toward a subroutine library of commonly used routines. We found that many of the routines we included in the initial set never were used at all, while others that we added as we went along were used frequently.

The download programs made it easy to maintain the source on larger machines running UNIX where we could assemble the code and debug syntactic problems using many different tools before sending executables to the small Atari machines for debugging. Unfortunately, debugging on the Atari was not always convenient, so we designed a dual-ported memory card that would allow us to debug the running Atari program using software running on the larger machines. We discarded the idea of simulating the Atari on the larger machines after discovering an incredible number of important undocumented eccentricities in the Atari hardware.

The simulation on the E & S Picture System allowed us to start fine tuning flight dynamics and make some general decisions about ways to achieve the graphic results we wanted while the Atari graphics rendering code was still being designed.

The drum machine program provided a workbench on which we could experiment with rhythmic patterns to be used in the games. We were able to include the features that make commercial microprocessor-based rhythm units so popular and add new features to suit our particular composition styles. Further, the program had the advantage of providing us with the exact sounds that we would be able to include in the games.

Although a music scoring system using *pic* is unavoidably unwieldy it still allowed us to use familiar text editors, the Sun Workstation graphics screen, and the Imagen printer for manipulation and distribution of musical ideas and final compositions.

A computerized storyboard editor seemed like a particularly good idea because the pixel sizes and graphics modes of the Atari graphics system are quite eccentric. Unfortunately, the Atari graphics are *so* eccentric that even the Sun Workstation's much higher resolution screen was unable to capture all the ''nuances''. A further problem was the need to display the colors that appear on the Atari screen. Because of artifacting effects[5] this would have been difficult to simulate even if we had had color graphics on the Sun Workstations. As a result we found the existing, primitive graphics editors available for the Atari more useful than the more sophisticated but inappropriate editor we had written for the Sun Workstation.

In our survey of game software, we found one or two programs with unusually sophisticated graphics and arranged to speak with their authors. We were disappointed to discover that few game programmers are willing to discuss the technical details of their work or share insights they may have gained.[6] With a

[5] The term ''artifacting'' refers to the interaction of colors in adjacent pixels on the screen.

[6] Not wishing to follow this clandestine approach we went so far as to write an article called *Ten Tips from the Programming Pros*, for the Spring 1984 issue of Atari Connection Magazine in which we described many of the

little work, we were able to write a disassembler that made the difficult task of deciphering a 6502 executable feasible.

Our intention had been to learn the steps necessary to produce a video game. As such we did not want to try any radically new approaches; instead, we wanted to develop a list of ways to improve on the state-of-the-art. On the other hand, some aspects of the ‘‘state-of-the-art’’ were just too primitive for us to bear. As a result we compromised by making only those improvements that were quick to implement and required minimal long-range planning. The need for an improved macro assembler was obvious and it was relatively quick to implement whereas the dual-ported memory cartridge turned out to be a larger project than we had expected, and its construction was postponed.

UNIX AIDS

The ways in which the UNIX system aided the games design process can be grouped into two broad categories, software available on UNIX systems, and capabilities of the UNIX operating system itself. Available software included program languages, editors, communications programs, and other miscellaneous tools. The capabilities of the system included a hierarchical file system that made file sharing convenient, input/output redirection, and a general ‘‘permissiveness’’ that kept the system from getting in our way when we needed to do something strange.

Languages

Having languages like C, awk, the shell, and Portable Standard Lisp (PSL) readily available allowed us to choose a language to fit the requirements of any particular task. Some programs had to execute a specific, well-defined task as quickly as possible and were written in C (e.g. the down-load programs). Other programs were more experimental and needed flexibility in the way they manipulated symbolic entities; these were written in Lisp. Still other programs had to be written quickly and would only be used once or twice; these were written as shell command files or as awk scripts. Had we been using the typical games industry systems we would have written all these programs in assembler (or perhaps Fortran if we were lucky) and we would still be debugging some of them today.

Editors

Although every class of Computer Science graduates seems to produce another editor-to-end-all-editors, none of these editors seems to have made it into common use in the video games industry. We, on the other hand, made much use of *emacs*, *vi*, and *ed*, enjoying the luxury of fitting the choice of editor to the requirements of the task at hand with few, if any, religious debates over the virtues of modelessness or the sins of meta-cokebottle-shift.

Communications

Our group put in many 20 hour days, sometimes spending 16 hours at work and then going home to say ‘‘remember me?’’ to the family before logging in for another few hours of work. After a few months of this, it became hard to predict when any particular person would be in the office or even be awake. The UNIX mail system allowed us to keep in touch with each other and share ideas and status reports even when schedules didn’t overlap. The ability to access files and programs through dial-in lines allowed occasional family visits and the speedy incorporation of middle-of-the-night inspirations.

Miscellaneous Tools

We used everything from *adb* to *yacc* at one time or another in this project. We got recalcitrant programs to work with *adb*, kept track of important deadlines using *calendar*, found the changes you make by falling asleep *on* your keyboard with *diff*, kept track of immense numbers of interdependent modules with *make*, printed cartridge dumps for the copyright office with *od*, scored music with *pic*, made global name changes with *sed*, prepared press releases using *spell*, *tbl*, and *troff*, borrowed tools from other sites with *uucp*, disassembled other designer’s programs with *yacc*, and so forth. Because we had access to these tools we were able to participate in parts of the production that normally are either left to specialists who

techniques we used to write our games.

have no other connection with the project or are left undone altogether. As a result, the games display an unusual coherence, with a common thread of design style unifying all the parts.

We also found that other parts of the Lucasfilm Computer Division had created tools useful to us. The Pixar (graphics processor) project had written a program to load PROMs that we found useful. The Digital Audio project had written a program (LUDS, pronounced "Lewds") to aid in circuit design and the creation of schematics, wire lists, etc. for their design of the ASP digital sound processor. We used LUDS for our dual-ported memory design.

Finally, we used programs written for, or as a result of, other game projects. These ranged from programs that helped maintain large numbers of related source files to programs written to manipulate the masses of data produced by games like ‘‘Empire’’.

Although none of these programs was written with our specific needs in mind, each, like much of the software associated with UNIX systems, was written to make as few limiting assumptions as possible and were perfectly adequate for our needs.

File Sharing

Crucial to a team approach to software development is the ability to break programs up into separate modules and share access to them among the entire team. The UNIX operating system made this easy although a little more concurrency control in the various editors would have avoided one or two minidisasters. Often three people would be working on related parts of a game, passing files back and forth for advice or criticism while a fourth was compiling and testing the results of the others’ changes.

Input/Output Redirection

Many of the tools that we used could not have existed on a system where the author of the software has to know what kind of device the data is coming from and what kind of device it will go to. Because we had to know less about the intended uses of a tool at the time we wrote it we were often able to use an existing tool to solve a problem instead of having to create a new one.

UNIX INFLUENCES

The ways in which the UNIX system influenced our games development are subtle, as influences often are. While some of the members of the games group had not been exposed to the UNIX operating system before joining, the majority of the people in the group were longtime UNIX aficionados and their enthusiasm for the system was soon adopted by all.

Many of the influences we felt were philosophical. Although no one of them is exclusively associated with UNIX software, the group, taken as a whole, implies UNIX influence strongly. It would be impossible to consider the influence of the UNIX system without including the influence of people like Brian Kernighan and John Mashey who have been eloquent spokesmen for many of the ideas and philosophies that permeate the system.

The concept often called ‘‘fail fast’’ seemed particularly relevant to games development. Much of the decision making in game design comes down to making a yes-or-no decision about a potential approach. Such choices vary from ‘‘is this idea interesting enough?’’ to ‘‘can it be done at all?’’ We found it immensely effective to try solving the hardest parts first in deciding any of these questions, allowing us to discard doomed approaches quickly and devote our energies to potentially successful ones.

We always tried to make individual *tools* that performed a single general task well and then use combinations of such tools to perform complex tasks rather than making a new program to solve each specific complex task. As a result, we had to write relatively few tools and ended up using each in many contexts — often in concert with tools written by others.

The programs we wrote generated little or no gratuitous output. Thus, when someone wanted to make that program a part of some larger construct, there was little or no effort spent trying to get rid of useless verbosity. Further, when diagnostics were called for, they really stood out rather than being lost in a sequence of ‘‘program starting...’’, ‘‘Pass 1 completed’’, ‘‘pass 2 loading...’’, and so forth.

We found simplicity to be an even better policy than honesty. Every time we were tempted to elaborate on some program the result was an unending job of fine tuning the elaborations — probably made more difficult because they never really belonged in the first place. When we stayed simple, it was easy to see what needed to be done, and it was easy to tell when we were finished and could go on to the next task.

UNIX ENVIRONMENT

The UNIX environment was perfectly suited to our task. When we needed a program, it was either already there or it was easy to write. Perhaps that's because the UNIX system was designed by (game) programmers to make the job of program development as easy as possible. In any case, it made *our* job easy. We didn't have to fight with programs that *almost* did what we needed but had made some limiting assumption. Nor did we have to trick the operating system into letting us do something that it thought we shouldn't do.

We were heavily influenced by the software, philosophies, and concepts associated with the UNIX operating system. It could be argued that the time was right for these developments and that the UNIX system was just the vehicle for ideas that would have appeared anyway; but it's clear that without these, our games projects would have ended very differently. We would have spent more time chasing down blind alleys and would have examined fewer potentially profitable approaches. We would not have been able to participate in as many areas of the production as we did, thereby leaving many crucial design decisions to the same people who crank out the standard, bland products that flood the market. Worst of all, we probably would have had to throw away the ''throw-away'' games not only because we would not have had time to complete them but they would not have been as interesting as they are.

THE GAMES

In a paraphrase of the reviewer from *Video Game Update* ''These games simply must be experienced to be appreciated.''

CREDITS

The work described in this paper was done by the members of the Lucasfilm Games Group, a part of the Lucasfilm Computer Division which, in turn, was a part of Lucasfilm Ltd. Although I headed and often spoke for the group it would be a mistake to think that I contributed more than any of the others in the group. Every member performed an irreplaceable function in the development of these games, and without any one of these people the games would not have been as they are, indeed, they might not have been at all.

The cast of characters, in alphabetical order, is:
Loren Carpenter
David Fox
Charlie Kellner
Peter Langston
David Levine
Gary Winnick

Others made contributions to our effort that should not be overlooked: Steve Arnold, Eric Benson, Steve Cantor, Ed Catmull, Terry Chostner, Mike Cross, Marty Cutler, Bob Doris, Gary Hare, Charlie Keagle, George Lucas, Lyle Mays, Pat Metheny, Lorne Peterson, Rob Poor, David Riordan, Richie Shulberg, Steven Spielberg

REVIEWS

''And the games, some enthusiasts say, are among the most innovative on the market.''
-- Businessweek

''Its [*BALLBLAZER*] high-speed action and split screen ... may leave players a bit dizzy.''
-- the San Francisco Chronicle

''All of which promises to do for the game industry what Lucasfilm has done for the film industry: rewrite the standards.''
-- Paul Cohen in Atari Connection Magazine

“... both games include sharp, sophisticated graphics and challenging game strategy.”
-- the San Francisco Examiner

“... cartridges that materially advance the state-of-the-art.”
-- Arnie Katz in Electronic Games Magazine

“... the 3-D look makes a game that is chock full of sensory excitement.”
-- San Jose Mercury News

“It sounds like John Coltrane!”
-- Pat Metheny

“*BALLBLAZER* ... meets the standards that Lucasfilm set in motion pictures”
-- Alan Michaels in Atari Connection Magazine

“*BALLBLAZER* ... is by far one of the most interesting two-player competition games to date.”
-- Newsweek Access Magazine

“... the latest breakthroughs in the art of video game design.”
-- Howard Pearlmutter, the Knoware Institute

“A breathtaking technical innovation in games design”
-- Paula Polley & Marina Hirsch in Atari Connection Magazine

“... sophisticated animation and graphics technology heretofore only seen in more advanced computer simulations.”
-- the Hollywood Reporter

“We were very impressed with this game”
-- The Video Game Update

“[*BALLBLAZER* & Rescue on Fractalus!] ... use more sophisticated computer graphics and animation techniques and have more of a three-dimensional appearance than most existing games.”
-- Wall Street Journal

TRADEMARKS

UNIX is a trademark of AT&T Bell Laboratories; Apple II is a trademark of Apple Computer Inc.; Atari is probably a trademark of Atari Inc.; DEC and VAX are trademarks of Digital Equipment Corporation; Picture System I is a trademark of Evans & Sutherland, Sun Workstation is a trademark of Sun Microsystems Inc.; Rescue on Fractalus, Ballblazer, Pixar, LUDS, and ASP are trademarks of Lucasfilm Ltd.