May 1988 Issue 19
U.S.A. $3.50
Canada $4.75

ST LOG

THE ATARI ST MONTHLY MAGAZINE

SPECIAL ADVENTURE ISSUE

The Perfect Game by

ORSON SCOTT CARD

Crin's Castle
Master Switch

A SPECIAL INCLUSION
Strathello

0 73361 64736 8 05

BREACH

"I was moving a squad of four of my best through the *kehst*-beridden jungles of Kiskismok, when suddenly Darrow's detector picks up what looks like a couple of life forms 50 meters out."

"'Nothin' but a couple of those brachiators,' he says to me. Just then Darrow takes it through the chest."

"We all drop into the mud, flipping through our helmet displays trying to find out where the shot came from when I hear this rumbling. More like I *feel* this rumbling. An' then there it is. A battle robot."

"Hsiang shoots the thing in its sensory grid with a bolt, but it doesn't do much good 'cause he's a psionic talent and they don't give 'em half-way decent guns. It starts tracking him with its dual guns and suddenly he takes some hot plasma too. Now it's just me and the kid, Yamaguchi."

"'Guchi,' I said, 'Direct your fire into the lifters so it can't move. These things are pretty lousy about protecting their undersides.' So like he was a vet of 20 drops, he rises to one knee and hits the thing right in the lifters. And then he does it twice more."

"'One battle robot: out of action,' he says. I toss an energy grenade at the hulk just to make sure and then we start the long job of carrying the boys north, where we know the landing boat'll be."

The Serayachi Campaign—Sgt. Robert Sherwood, FWSF *Ret.*

Breach is a single-player tactical-level combat game for one person. It features:

- Smooth animated movement and combat.
- Macintosh™ version includes digitized sound.
- The ability to lead squads of up to 20 marines.
- A campaign of several scenarios included with the game.
- Up to 40 opponents per scenario—ranging from vicious beasts to marines as intelligent and well-equipped as your own. Six different classes of opponents in all: marine, alien, beast, overlord, autogun, and battle robot.
- Equip your marines with 20 different types of objects—including rocket launchers, demolition charges, first aid kits, and cracking units to break into enemy computer systems.
- Four different classes of marine: marauder, infiltrator, scout, and psionic talent.
- Your squad leader is independent of any scenario—play **Breach** just like a role-playing game *or* as a single session wargame.
- Build-up your squad leader for special advanced training—all of his combat experience is saved!
- **Breach** includes a **Scenario Builder**—create your own scenarios or modify existing ones!
- Scenarios can have several different victory conditions which can be mixed together to form extremely complex battles.
- Additional scenario disks available soon!

Breach is available for the Atari ST, Macintosh, IBM, and Amiga. Photos are for the Atari ST version.

To order, visit your software dealer. For direct orders (VISA/MasterCard/COD), phone (203) 658-6917. To purchase by mail, send check, money order, or credit card information to Omnitrend Software, Inc., PO Box 733, West Simsbury, CT 06092. Cost is $39.95 plus $3.00 for shipping and handling.

OMNITREND SOFTWARE
P.O. BOX 733 • WEST SIMSBURY, CT 06092

Circle #101 on reader service card.

ST-LOG

THE ATARI ST MONTHLY MAGAZINE

ST-LOG CONTENTS MAY 1988

FEATURES

REVIEWS

COLUMNS

ST-LOG (ISSN 0890-9601) is published monthly by L.F.P., Inc., 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

U.S. subscription $28 for one year (12 issues), $52 for two years (24 issues, $76 for three years (36 issues). Foreign subscriptions: add $7 per subscription per year. Single copy $3.50 (add $1 for postage). Change of address: six weeks advance notice, and both old and new addresses are needed. POSTMASTER, send change of address to: **ST-LOG** Magazine, P.O. Box 16928, North Hollywood, CA 91615. Second-class postage paid at Beverly Hills, CA, and additional mailing offices.

Editorial

We all need to escape.

Isn't it true? That's why the movie industry can coax us out of millions of dollars a year, why every home in the U.S. holds a television set, and why people like James Clavell, Tom Clancy and Stephen King provide us with glimpses from their imaginations in trade for seven-figure checks from their publishers.

It's also why one of the most popular forms of computer entertainment is the adventure game.

When you think about it, computer adventure games have a lot in common with books and movies. They can transport us from the hum-drum staleness of our everyday lives into wild and exotic surroundings, where anything can happen, where the unexpected is commonplace and excitement is guaranteed. Whether it be exploring the deepest jungles of Africa or battling creatures from another planet, the adventure game takes us on a journey into an unpredictable and exhilarating unknown. We go willingly into danger, knowing escape is always possible—aware that, no matter what predicament we find ourselves in, we can always return to the safety of reality.

Of course, computer adventure games—especially the text-only type—won't delight all who attempt them. They require an inordinate amount of patience and can be as frustrating as trying to lose weight on a cheesecake diet. In fact, some people I know would rather chew glass than be forced to complete an adventure. It takes a very different sort of person to explore the labyrinth of Zork than it does to guide an airborne Ground Attack Vehicle through the battlefield of Starglider.

Or does it?

Now, with more powerful graphics capabilities, adventures are becoming—at least in the visual sense—more action oriented, more like movies. So much so, that confirmed arcade fanatics are starting to climb aboard the adventure bandwagon, are discovering that it is, after all, *fun* to untangle the Gordian knot of perplexities adventure creators are so fond of weaving; satisfying to *think* about their next move, rather than blast blindly forward, twisting the handle of a joystick into oblivion.

Hey, I think that's great. We're all adventurers at heart.

This issue is packed with people who have their hearts in the right place. Award-winning science fiction author Orson Scott Card gives us an intriguing look at adventure games of the future, while Michael Banks, creator of Pryority Software's Gateway (and a science fiction author, as well), gives programmers advice on writing and marketing adventure games—a must-read for any software designer interested in entering this lucrative marketplace.

Want to know where you can find the solutions to those stumpers? Andy Eddy presents some valuable words for frustrated gamers, advice culled from years of adventure experience. And topping off our adventure features is Brad Mott's "Crin's Castle," a complete, type-in text adventure written in ST BASIC, along with West Coast Editor Charles F. Johnson's lesson in schizophrenic programming.

So read on, friend. The **ST-Log** adventure awaits.

Clayton Walnum
Technical Editor
ST-Log

ST-Log Staff

Publisher: Lee H. Pappas. *Executive Editor:* Clayton Walnum. *Art Director:* Kathy Wiesner. *Managing Editor:* Dean Brierly. *East Coast Editor:* Arthur Leyenberger. *Midwest Editor:* Matthew J.W. Ratcliff. *West Coast Editor:* Charles F. Johnson. *Contributing Editors:* Michael Banks, Ian Chadwick, Andy Eddy, Arnie Katz, Bill Kunkel, Maurice Molyneaux, Steve Panak, Douglas Weir, Joyce Worley. *Cover Art:* Gary Lippincott. *Copy Chief:* Katrina Veit. *Copy Editors:* Anne Denbok, Sara Bellum. *Typographers:* David Buchanan, Klarissa Curtis, Judy Villanueva. *Contributors:* Orson Scott Card, Betty D. DeMunn, A.N. Kensington, Todd Kepus, Brad Mott, David Plotkin, D.F. Scott. *Production Director:* Donna Hahner. *Production Assistant:* Steve Hopkins. *Advertising Production Director:* Janice Rosenblum. *Subscriptions Director:* Irene Gradstein. *Vice-President-Sales:* James Gustafson.

U.S. newsstand distribution by Eastern News Distributors, Inc., 1130 Cleveland Rd., Sandusky, OH 44870.

Where to write

Correspondence, letters and press releases should be sent to: Editor, **ST-LOG**, 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

Correspondence regarding subscriptions, including problems and changes of address, should be sent to: **ST-LOG** P.O. Box 16928, North Hollywood, CA 91615. Or call (818) 760-8983.

Correspondence concerning a regular column should be sent to our editorial address, with the name of the column included in the address. We cannot reply to all letters in these pages, so if you would like an answer, please enclose a self-addressed, stamped envelope.

Advertising Sales

J.E. Publishers Representatives — Los Angeles: (213) 467-2266.
San Francisco: (415) 864-3252. Chicago: (312) 445-2489.
Denver: (303) 595-4331.
6855 Santa Monica Blvd., Suite 200, Los Angeles, CA 90038.
New York: (212) 724-7767.

Address all advertising materials to: Janice Rosenblum — Advertising Production, **ST-LOG**, 9171 Wilshire Blvd., Suite 300, Beverly Hills, CA 90210.

Permissions

Subscriptions

ST-LOG, P.O. Box 16928, North Hollywood, CA 91615; or call (818) 760-8983. Payable in U.S. funds only. U.S.: $28.00-1 year; $52.00-2 years; $76.00-3 years. Foreign: add $7.00 per year per subscription. For disk subscriptions, see the cards at the back of this issue.

Authors

When submitting articles and programs, both program listings and text should be provided in printed *and* magnetic form, if possible. Typed or printed text copy is mandatory, and should be in upper- and lowercase, with double spacing. If a submission is to be returned, please send a self-addressed, stamped envelope.

Easter Super Printer Package Sale

Atari ST

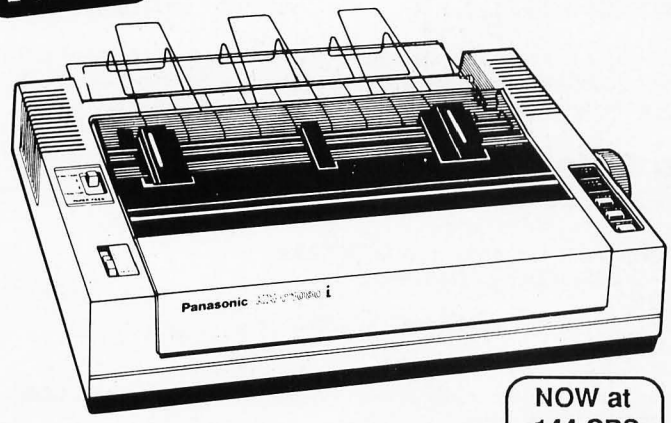
Panasonic Office Automation OA

8 Bit Atari

&

ComputAbility

NEW



NOW at 144 CPS

Panasonic Office Automation OA **KX-P1080i-II**

Super Printer Package
***for 8 Bit Atari computers**

with Xetec Graphic AT **$215**

with Supra 1150 interface **$225**

*Package price includes Delivery in continental U.S.A.

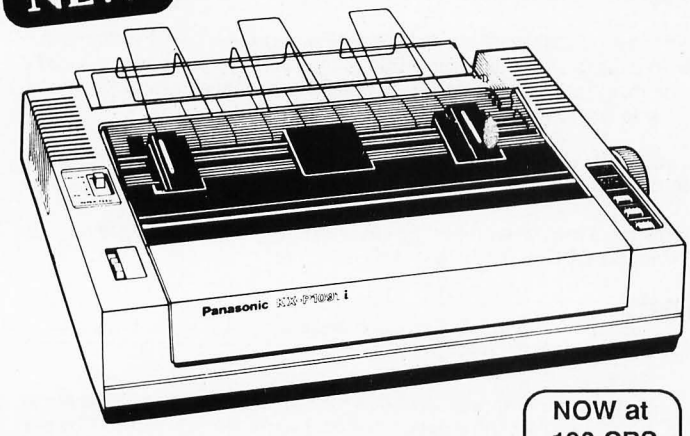
Panasonic Office Automation OA **KX-P1080i-II**

Super Printer Package
***for All Atari ST computers**

with ST Printer cable **$199**

*Package price includes Delivery in continental U.S.A.

NEW



NOW at 192 CPS

Panasonic Office Automation OA **KX-P1091i-II**

Super Printer Package
***for 8 Bit Atari computers**

with Supra 1150 interface **$245**

with Xetec Graphic AT **$235**

*Package price includes Delivery in continental U.S.A.

Panasonic Office Automation OA **KX-P1091i-II**

Super Printer Package
***for All Atari ST computers**

with ST Printer cable **$219**

*Package price includes Delivery in continental U.S.A.

* Price for APO & Non Continental U.S.A. orders. See Special order information in our 2 page spread

Mon-Fri 9am-9pm CST
Sat 11am-5pm

order Call Toll Free
800-558-0003

SINCE 1982

ComputAbility Consumer Electronics

No surcharge for Mastercard or Visa

Inquiries, or for Wisc. Order
414-357-8181

Circle #115 on reader service card.

Reader comment

Organizing against piracy

I think we all agree piracy is wrong. The time has come to do something about it. I don't want my 520ST to be abandoned by most software companies because of actions by irresponsible computer owners.

As a system operator of a 100-percent clean bulletin board, I am fed up with new callers assuming my board supports piracy. It's gotten to the point where if a BBS does *not* support piracy, it is in the minority, at least in my area. Almost daily I find someone offering to upload the latest game in exchange for "special access" to the pirate section.

The time has come to form an organization opposed to piracy and to make ourselves known to all pirate groups and bulletin boards. Individual screams seem to fall upon deaf ears. A nationwide organization should be able to. . . be heard and force action to be taken!

I have some ideas and want to hear from others interested in forming this organization. Any readers with ideas, comments, and/or suggestions should write to me at the address below. We've sat around long enough; it's time to make our stand.

Tom Bellucco
52 Hamlin Street
Rochester, NY 14615

CPs for STs

I would like to make a comment on the Atari ST line. Perhaps this would be best phrased as a question: "Why, oh why can't the ST machines have a 'command processor' DOS, such as MS-DOS, or DOS XL (which I use on my 800)?" I prefer the power and speed of a command processor (using a few keystrokes) over the hassle of messing with mice. Drag. . .click . . .point. . .double-click. . .Ugh!

Call me old fashioned, or a power user, but I'll take a CP anytime.

Do you think that Atari, or perhaps a third-party company, can offer an alternative to messing with mice?

Also, as a staunch Atari supporter, I'd like to see them take a much more aggressive marketing tack with the ST line. It seems to me that a three-pronged offensive would be best, those areas being: (1) ST MIDI music; (2) ST desktop publishing; and (3) ST CD ROM.

I feel these three areas, along with perhaps CAD, offer the best potential growth for Atari in the marketplace.

Bill Somrak
Columbus, OH

Why can't the ST machines have a command processor? But they do! Many of them, in fact. The Atari Developer's Kit has always included COMMAND.TOS, which disables the GEM environment and lets you do things the "old-fashioned" way. There are also several command interpreters available from third-party developers, including MichTron's DOS Shell (576 S. Telegraph, Pontiac, MI 48053, 313-334-5700) and Beckemeyer's Micro C-Shell (592 Jean Street #304, Oakland, CA 94610, 415-658-5318) which was reviewed in issue 11 of ***ST-Log****. If your pocketbook is a bit tight, there are also several command interpreters available in the public domain, including ASH070 and PCOMMAND, both of which are available for downloading on the ANALOG Publishing Atari SIG on Delphi.*

As for Atari's marketing, they've never been overly aggressive (at least, not since Jack Tramiel took over), but you'll be happy to know that the areas you mentioned are already a big part of the Atari plan. The latest polls show the ST rapidly becoming the #1 choice of musicians, so much so that there's been talk of selling STs not only in computer outlets, but in music stores too. Atari STs may soon see increased demand in the desktop publishing field, also—as soon as the new MEGA STs and the Atari SLM laser printers are released (see "Status report" in issue 16).

When it comes to CD ROM, it's anybody's guess as to when Atari will actively pursue this exciting new technology. They have hinted at plans of developing CD ROM systems for the entire Atari line, but have delayed the project due to high prices. When Atari manages to find a way to produce CD ROM players for what they consider to be a reasonable price, there's no doubt that these will be available for your ST. Personally, we can't wait!

Crin's Castle

GAME

LOW RESOLUTION

The quest for the Flame Sword.

by Brad Mott

The king has called you to the castle for a great cause. Crin has stolen the magical Flame Sword and without it the entire kingdom is in danger. You must go to **Crin's Castle** and find the sword before the kingdom is lost. "I will reward you well if you succeed," the king said. Being a knight of great honor, you've accepted the challenge. You know no one has ever returned from **Crin's Castle**, but you're going anyway, hoping for the best.

Typing it in.

Type the program exactly as shown in Listing 1. Make sure to type it correctly, or you may end up with an adventure that's impossible to finish. Also, make sure to save a copy before running the program, because GEM is cut off when the screen is being drawn—if you get an error, the system will lock up. Use "ST-Check" to be sure there are no typos.

Playing *Crin's Castle.*

In **Crin's Castle**, you use two-word commands. They should be in verb/noun format (such as *GET BOOK, DROP BOOK*). To move from room to room, use the commands *N, S, E* and *W.*

There are some special commands you should also know. They are: *SAVE GAME, LOAD GAME, QUIT* and *INVENTORY.* Use the *SAVE* command to save the game to disk. *LOAD GAME* will restore the last game you saved, and *QUIT* returns you to BASIC. The *INVENTORY* command will show what you're carrying. //

Brad Mott is 17 years old and attends Union High School in Clinton, North Carolina. He's owned a 1200XL for almost four years and 1040ST for six months. He likes to program in BASIC and assembly and plans to learn C very soon.

Listing 1.
ST BASIC listing.

```
5 ' CRIN'S CASTLE  By: Brad Mott  8-17
-86  ver 0.2
10 clear:option base 1:fullw 2:clearw
2:color 1
15 if peek(systab)<>4 then gotoxy 2,2:
? "Low resolution only!!!":end
25 ni=32:in=1:gosub START:gosub SCREEN
30 room=23:goto PARSER
500 ROOMDATA:
505 gotoxy 2,3:? space$(20):gotoxy 2,3
:a$=room$(room):gosub DECODE
510 gotoxy 13,4:? space$(8):gotoxy 13,
4:if en(room)<>0 then ? "N ";
515 if es(room)<>0 then ? "S ";
520 if ew(room)<>0 then ? "W ";
525 if ee(room)<>0 then ? "E ";
530 for t=0 to 3:gotoxy 2,6+t:? space$
(34):next
535 x=2:y=0:for t=1 to ni
540 if abs(iloc(t))=room then gotoxy x
,6+y:a$=item$(t):gosub DECODE:y=y+1
545 if y=3 then y=0:x=19
550 next:gotoxy 3,12:? space$(33)
555 return
600 DECODE:
605 for z=1 to len(a$):? chr$(asc(mid$
(a$,z,1))-1);
610 next:?
615 return
1000 PARSER:
1005 gotoxy 2,15
1010 if can=1 then ? "You can't do tha
t!!!"
1015 if can=0 then ? "Okay..."
1020 can=0:on error goto 7500
1025 gosub ROOMDATA:gotoxy 2,12:? chr$
(7);:input cm$
1030 for t=1 to len(cm$):b$=mid$(cm$,t
,1)
1035 if b$>="a" and b$<="z" then mid$(
cm$,t,1)=chr$(asc(b$)-32)
1040 next
1045 for t=0 to 2:gotoxy 2,14+t:? spac
e$(34):next
1060 if len(cm$)=1 then 2000
1065 if cm$="QUIT" then color 1,0,0,8,
2:end
1070 verb$=left$(cm$,3):a=instr(cm$,"
")
1075 noun$=mid$(cm$,a+1,3)
1080 v=instr(vtab$,verb$):v=int(v/3)
1085 n=instr(ntab$,noun$):n=int(n/3)
1090 if v=7 then 1125
1100 if n=0 then gotoxy 2,14:? "Don't
understand that Noun!!!":can=2:goto PA
RSER
1105 if v=0 then gotoxy 2,14:? "Don't
understand that Verb!!!":can=2:goto PA
RSER
1110 if room=19 and iloc(27)<>100 then
 ti=ti+1:if ti=2 then a$="Uif!hvbse!bu
ubdlfe!zpv///":goto DEAD
1125 on v goto 3000,3000,3200,3300,340
0,3400,3600,3800,3900,4000
1130 v=v-10
1135 on v goto 4100,4200,4300,4400,450
0,4600,4700,3400
1195 goto PARSER
2000 gotoxy 2,14
2005 if cm$="N" then 2100
2010 if cm$="S" then 2200
2015 if cm$="E" then 2300
2020 if cm$="W" then 2400
2025 can=1:goto PARSER
2050 gotoxy 2,14
2055 ? "You can't go that way!!!"
2060 can=2:goto PARSER
```

Crin's Castle *continued*

```
2075 gotoxy 2,14:a$="Uif!epps!jt!dmptf
e///":gosub DECODE:can=2:goto PARSER
2100 if en(room)=0 then 2050
2105 if room=23 and do3=0 then 2075
2110 if room=6 and do1=0 then 2075
2115 if room=11 and do2=0 then 2075
2120 if room=9 and iloc(3)=0 then a$="
Uif!xj(bse!mfbwft!xifo!zpv!foufs/":gos
ub DECODE
2125 if room=9 and iloc(3)<>0 then a$=
"Uif!xj(bse!buubdlfe!zpv///":goto DEAD
2195 can=0:room=en(room):goto PARSER
2200 if es(room)=0 then 2050
2205 if room=1 and do3=0 then 2075
2210 if room=11 and do1=0 then 2075
2295 can=0:room=es(room):goto PARSER
2300 if ee(room)=0 then 2050
2305 if room=21 and in<>1 then a$="Uif
!gmpps!gbmmt!pvu!gspn!voefs!zpv///":go
to DEAD
2310 if room=21 and pow=0 then a$="Dsj
o!buubdlfe!zpv///":goto DEAD
2315 if room=21 and pow=1 then a$="Dsj
o!wbojtift!xifo!zpv!foufs/":gosub DECO
DE
2320 if room=19 and ti=1 and iloc(27)=
-19 then a$="Uif!hvbse!buubdlfe!zpv///
":goto DEAD
2395 can=0:room=ee(room):goto PARSER
2400 if ew(room)=0 then 2050
2405 if room=18 and iloc(4)<>0 then a$
="B!bsspx!tipu!zpv///":goto DEAD
2410 if room=18 and iloc(27)<>100 then
 ti=0
2495 can=0:room=ew(room):goto PARSER
3000 a=in:if in>4 then can=2:gotoxy 2,
14:a$="Zpv!dbo(u!dbssz!boz!npsf///":go
sub DECODE:goto PARSER
3005 if iloc(n)<0 then can=2:gotoxy 2,
14:a$="Zpv!dbo(u!hfu!uibu///":gosub DE
CODE:goto PARSER
3010 if iloc(n)=0 then can=2:gotoxy 2,
14:a$="Zpv!bmsfbez!ibwf!ju///":gosub D
ECODE:goto PARSER
3015 if n=10 and brive=0 and room=12 t
hen a$="Uif!dppl!buubdlfe!zpv///":goto
 DEAD
3020 if n=13 and di=1 then a$="Uif!dpp
l!buubdlfe!zpv///":goto DEAD
3025 if n=11 and room=21 then a$="B!mb
shf!tupof!gfmm!po!zpv///":goto DEAD
3030 if n=16 and room=22 then goto WIN
NER
3035 if iloc(n)<>room then gotoxy 2,14
:can=2:? "It's not here!!!":goto PARSE
R
3150 if iloc(n)=room then iloc(n)=0:in
v$(in)=item$(n):in=in+1
3155 goto PARSER
3200 for t=1 to in
3205 if inv$(t)=item$(n) then a=t:goto
 3220
3210 next:gotoxy 2,14:? "You don't hav
e it!!!":can=2
3215 goto PARSER
3220 s=1:for t=1 to ni:if abs(iloc(t))
=room then s=s+1
3225 next:if s>5 then gotoxy 2,14:? "N
o more space here...":can=2:goto PARSE
R
3230 iloc(n)=room:for t=a+1 to in-1
3235 inv$(t-1)=inv$(t)
3240 next:in=in-1:inv$(in)=" "
3245 goto PARSER
3300 if iloc(18)<>0 then gotoxy 2,14:c
an=2:a$="Zpv!offe!tpnfuijoh!up!dvu!xju
i///":gosub DECODE:goto PARSER
3305 if abs(iloc(n))<>room then gotoxy
 2,14:can=2:? "It's not here!!!":goto
PARSER
3310 if room=23 and n=21 and st=1 then
 3350
3315 gotoxy 2,14:a$="Zpv!dbo(u!dvu!uib
u///":gosub DECODE:can=2:goto PARSER
3350 iloc(21)=100:stc=1:goto PARSER
3400 gotoxy 2,14
3405 if n=1 and room=3 and ma=0 then i
loc(3)=3:a$="Zpv!tff!b!bnvmfu!po!ijn//
/":gosub DECODE:ma=1:goto PARSER
3410 if n=6 and room=6 and do1=0 then
a$="Uif!epps!jt!mpdlfe///":gosub DECOD
E:goto PARSER
3415 if n=7 and room=8 and bo=0 then i
loc(20)=8:a$="Zpv!tff!b!mbcfm!jo!pof//
/":gosub DECODE:bo=1:goto PARSER
3420 if n=8 and iloc(8)=0 and boe=0 th
en a$="Uifsf!jt!b!mjrvje!jo!ju///":gos
ub DECODE:goto PARSER
3425 if n=9 and iloc(9)=0 and fr=0 the
n fr=1:a$="Zpv!gjoe!b!gjsf!tqfmm///":g
osub DECODE:iloc(31)=room:goto PARSER
3430 if n=11 and iloc(11)=room then a$
="Ju!jt!gvmm!pg!ejbnpoet!boe!hpme/":go
sub DECODE:goto PARSER
3435 if n=15 and room=19 then a$="Uifs
f!jt!b!hmpxjoh!cbmm!po!ju///":gosub DE
CODE:goto PARSER
3440 if n=19 and room=24 and ro=0 then
 iloc(18)=24:ro=1:a$="Zpv!tff!b!ebhhfs
///":gosub DECODE:goto PARSER
3445 if n=6 and room=23 and st=0 then
iloc(21)=-23:st=1:a$="Zpv!tff!b!tusjoh
///":gosub DECODE:goto PARSER
3450 if n=24 and room=11 and de=0 then
 de=1:iloc(25)=11:a$="Zpv!tff!b!opuf//
/":gosub DECODE:goto PARSER
3455 if n=6 and room=11 and do2=0 then
 a$="Uif!epps!jt!mpdlfe///":gosub DECO
DE:goto PARSER
3460 if n=5 and room=5 and co=1 and ke
=0 then iloc(12)=5:a$="Zpv!tff!b!lfz!j
o!ju///":gosub DECODE:ke=1:goto PARSER
3465 if n=20 and iloc(20)=0 then a$="J
u!tbzt!BDJE":gosub DECODE:goto PARSER
3470 if n=1 and room=17 and lett=0 the
n iloc(26)=17:a$="Zpv!tff!b!mfuufs///"
:gosub DECODE:lett=1:goto PARSER
3475 if n=17 and room=20 and pap=0 the
n iloc(23)=20:a$="Zpv!tff!b!qjdf!pg!qb
qfs///":gosub DECODE:pap=1:goto PARSER
3595 can=2:? "You don't see anything s
pecial!!!":goto PARSER
3600 clearw 2
3605 gotoxy 14,1:? "Inventory":?
3610 for t=1 to in-1:a$="!!!!"+inv$(t)
:gosub DECODE:next
```

```
3615 ?:?:? "   Press any key...":t=inp
(2)
3620 can=2:gosub SCREEN:goto PARSER
3800 gotoxy 2,14:? "Saving..."
3805 open "O",#1,"CRIN.DAT"
3810 for t=1 to ni:write #1,iloc(t):ne
xt
3815 write #1,in:for t=1 to in:write #
1,inv$(t):next
3820 write #1,room,ti,st,stc,ma,do1,do
2,do3,bo,fr,ro,de,st,boe,ba
3825 write #1,co,brive,di,ke,lett,pap,
pow
3830 close:goto PARSER
3900 gotoxy 2,14:? "Loading..."
3905 open "I",#1,"CRIN.DAT":close:open
 "I",#1,"CRIN.DAT"
3910 for t=1 to ni:input #1,iloc(t):ne
xt
3915 input #1,in:for t=1 to in:input #
1,inv$(t):next
3920 input #1,room,ti,st,stc,ma,do1,do
2,do3,bo,fr,ro,de,st,boe,ba
3925 input #1,co,brive,di,ke,lett,pap,
pow
3930 close:goto PARSER
4000 gotoxy 2,14
4005 if n=8 and room=11 and iloc(8)=0
and boe=0 then a$="Uif!epps!nfmut!bxbz
///":gosub DECODE:do2=1:boe=1:goto PAR
SER
4010 if n=8 and boe=0 and iloc(8)=0 th
en can=2:a$="Zpv!dbo(u!ep!uibu!ifsf///
":gosub DECODE:goto PARSER
4015 if n=8 and boe=1 and iloc(8)=0 th
en can=2:a$="Ju(t!fnquz///":gosub DECO
DE:goto PARSER
4020 can=1:goto PARSER
4100 gotoxy 2,14
4105 if n=6 and room=6 and iloc(12)=0
then do1=1:goto PARSER
4110 if n=6 and room=6 and iloc(12)<>0
 then a$="Ju(t!mpdlfe///":gosub DECODE
:goto PARSER
4115 if n=6 and room=23 and stc=0 then
 a$="B!ebsu!tipu!zpv///":goto DEAD
4120 if n=6 and room=23 and stc=1 then
 do3=1:goto PARSER
4125 if n=5 and room=5 and iloc(10)<>0
 and ba=0 then 4150
4130 if n=5 and room=5 and iloc(10)=0
and co=0 and ba=0 then goto 4160
4135 if n=5 and ba=1 and room=5 then c
o=1:goto PARSER
4145 can=1:goto PARSER
4150 a$="B!wbnqjsf!buubdlfe!zpv///":go
to DEAD
4160 a$="B!cbu!gmjft!pvu-!boe!mfbwft//
/":gosub DECODE:co=1:ba=1:goto PARSER
4200 gotoxy 2,14
4205 if n=6 and room=6 and do1=1 then
do1=0:goto PARSER
4210 if n=6 and room=23 and do3=1 then
 do3=0:goto PARSER
4215 if n=5 and room=5 and co=1 then c
o=0:goto PARSER
4230 can=1:goto PARSER
4300 gotoxy 2,14
4305 if n=31 and fr=1 and room=19 then
 iloc(27)=100:iloc(22)=-19
4310 if n=31 and fr=1 then a$="Gjsf!gm
jft!gspn!zpvs!gjohfst///":gosub DECODE
:goto PARSER
4315 can=1:goto PARSER
4400 gotoxy 2,14
4405 if n=14 and room=19 and iloc(3)=0
 then 4450
4410 if n<>14 then can=2:? "Nothing ha
ppens!!!":goto PARSER
4415 a$="Ju!wbqpsj(fe!zpv///":goto DEA
D
4420 if n=8 and iloc(8)=0 and boe=0 th
en a$="Uifsf!jt!b!mjrvje!jo!ju///":gos
ub DECODE:goto PARSER
4450 a$="Zpv!gffm!b!tvshf!pg!qpxfs///"
:gosub DECODE:pow=1:can=2:goto PARSER
4500 gotoxy 2,14
4505 if n=8 and iloc(8)=0 and boe=0 th
en a$="Zpv!nfmu!bxbz///":goto DEAD
4510 if iloc(n)<>0 then can=2:? "You d
on't have it!!!":goto PARSER
4515 can=2:? "Yuck!":goto PARSER
4600 if n=25 and iloc(25)=0 then goto
NOTE
4610 if n=26 and iloc(26)=0 then goto
LETTER
4615 if n=23 and iloc(23)=0 then goto
PAPER
4620 if n=32 and room=18 then gotoxy 2
,14:a$="Hp!cbdl/////":gosub DECODE:can
=2:goto PARSER
4625 can=1:goto PARSER
4700 gotoxy 2,14
4705 if n=13 and room=12 and iloc(13)=
0 then goto 4720
4710 can=1:goto PARSER
4720 can=2:brive=1:a$="If!ublft!uif!ej
bnpoe///":gosub DECODE
4725 di=1:goto 3200
5000 DEAD:
5005 clearw 2
5010 gotoxy (38-len(a$))/2,2:gosub DEC
ODE
5015 gotoxy 6,4:? "This adventure is o
ver!!!":gotoxy 6,6
5016 ? "Crin enjoyed your stay..."
5020 gotoxy 2,11:input "Play again??";
 a$
5025 if a$="y" or a$="Y" then goto 10
5030 if a$="n" or a$="N" then clearw 2
:end
5035 goto 5020
6000 NOTE:
6005 poke systab+24,1:clearw 2
6010 color 1,6,1,8,2:linef 50,40,250,4
0:linef 50,100,250,100
6015 linef 50,40,50,100:linef 250,40,2
50,100
6020 fill 0,0
6025 te=4:gosub TEFFECT
6030 gotoxy 7,6:a$="Uiptf!xip!ibwf!opu
ijoh":gosub DECODE
6035 gotoxy 7,8:a$="ibwf!opuijoh!up!mp
tf//":gosub DECODE
6040 goto 6500
6100 LETTER:
```

Crin's Castle *continued*

```
6105 poke systab+24,1:clearw 2
6110 color 1,6,1,8,2
6115 linef 20,10,290,10:linef 20,150,2
90,150
6120 linef 20,10,20,150:linef 290,10,2
90,150
6125 fill 0,0
6130 te=4:gosub TEFFECT
6135 gotoxy 3,4:a$="J!ibwf!gpvoe!pvu!u
ibu!uif!bnvmfu":gosub DECODE
6140 gotoxy 3,5:a$="tipvme!qspufdu!zpv
!gspn!uif":gosub DECODE
6145 gotoxy 3,6:a$="xj{bse!boe!ju!tipv
me!lffq!uif":gosub DECODE
6150 gotoxy 3,7:a$="hmpxjoh!cbmm!gspn!
ljmmjoh":gosub DECODE
6155 gotoxy 3,8:a$="zpv!jg!zpv!upvdi!j
u/!!J!bn!po":gosub DECODE
6160 gotoxy 3,9:a$="nz!xbz!up!gjoe!ju/
//":gosub DECODE
6165 gotoxy 28,11:a$="NBY":gosub DECOD
E
6170 goto 6500
6200 PAPER:
6205 poke systab+24,1:clearw 2:color 1
,6,1,8,2
6210 te=4:gosub TEFFECT
6215 linef 70,25,225,25:linef 70,25,70
,100
6220 linef 70,100,225,100:linef 225,10
0,225,80
6225 linef 225,80,195,60:linef 225,25,
225,32
6230 linef 195,60,225,32:fill 0,0
6235 gotoxy 10,4:a$="Uvso!cbdl!jg!zp":
gosub DECODE
6240 gotoxy 10,5:a$="ibwf!opu!upvdi":g
osub DECODE
6245 gotoxy 10,6:a$="uif!hmpxjoh!C":go
sub DECODE
6250 gotoxy 10,9:a$="!!!!!!Csbe!Npuu":
gosub DECODE
6255 goto 6500
6500 poke systab+24,0
6505 gotoxy 10,17:color 0:wm=2:gosub W
MODE
6510 ? "Press any key...";:color 1
6515 x=inp(2):wm=0:gosub WMODE:te=0:go
sub TEFFECT
6520 gosub SCREEN:goto PARSER
7000 WINNER:
7005 color 2,1,0,8,2:clearw 2
7010 wm=2:gosub WMODE
7015 te=16:gosub TEFFECT:th=25:gosub T
HEIGHT
7020 gotoxy 3,3:? "C O N G R A T U L A
 T I O N S":th=6:gosub THEIGHT
7025 te=0:gosub TEFFECT:color 6:gotoxy
 9,5:? "You have completed"
7030 gotoxy 10,7:? "Crin's Castle!!!"
7035 gotoxy 6,9:? "Quest for the Flame
 Sword"
7040 goto 5020
7500 gotoxy 2,14:? "Error ";err;" at l
ine ";erl:resume PARSER
8000 START:
8005 poke systab+24,1
8010 fullw 2:clearw 2:wm=2:gosub WMODE
8015 color 2,1,6,8,2:gotoxy 0,0:? " "
8020 fill 0,86
8025 te=16:gosub TEFFECT
8030 th=13:gosub THEIGHT
8035 gotoxy 5,3:? "C R I N ' S   C A S
 T L E"
8040 th=6:gosub THEIGHT:color 7:te=4:g
osub TEFFECT
8045 gotoxy 5,5:? "Quest for the Flame
 Sword"
8050 color 4:te=1:gosub TEFFECT
8055 gotoxy 11,7:? "By Brad Mott"
8100 color 1,0,8
8105 linef 151,100,151,140:linef 151,1
00,148,105
8110 linef 151,100,154,105:linef 148,1
05,148,140
8115 linef 154,105,154,140
8120 color 1,0,6
8125 linef 143,141,159,141:linef 142,1
42,160,142
8130 linef 143,143,159,143:linef 148,1
44,154,144
8135 for t=149 to 153:linef t,145,t,15
0:next
8140 linef 148,151,154,151
8200 dim en(25),es(25),ee(25),ew(37)
8205 dim room$(25),item$(ni),iloc(ni)
8210 dim vtab$(1),ntab$(4),inv$(6)
8215 room=23
8220 ntab$="***MAN***AMUSHICOFDOOBOXBO
TBOOGARCHEKEYDIABALALTSWOTAB"
8225 ntab$=ntab$+"DAGROCLABSTRASHPAPDE
SNOTLETGUACOOTABVAMSPESIGGAM"
8230 vtab$="***GETTAKDROCUTLOOSEAINVSA
VLOAPOUOPECLOCASTOUDRIREAGIV"
8235 vtab$=vtab$+"EXA"
8240 restore 8300:for t=1 to 24
8245 read en(t),es(t),ee(t),ew(t):next
8300 data 0,23,16,2,0,3,1,4,2,0,0,0,6,
5,2,0,4,0,0,0
8305 data 11,4,7,0,0,0,0,6,9,0,0,0,10,
8,13,0,0,9,0,0
8310 data 20,6,0,0,0,13,0,0,12,0,14,9,
17,15,0,13,14,0,0,16
8315 data 0,0,15,1,18,14,0,0,0,17,0,19
,0,0,18,0,0,11,21,0
8320 data 0,0,22,20,0,0,0,21,1,24,0,0,
23,0,0,0
8400 restore 8420
8405 for t=1 to 24:read room$(t):next
8420 data b!mbshf!sppn,b!ibmm,b!nffujo
h!sppn,b!ibmm,b!dszqu,b!ibmm
8425 data b!cfesppn,b!dmptfu,b!mbcpsbu
psz,uif!xj{bse(t!cfesppn
8430 data uif!nbtufs!cfesppn,uif!ljudi
fo,b!ibmm,b!mbshf!sppn,uif!ejojoh!sppn
8435 data b!mbshf!sppn,b!upsuvsf!dibnc
fs,b!fousbodf!sppn,b!mbshf!sppn
8440 data b!mbshf!sppn,Dsjo(t!usfbtvsf
!sppn,b!sppn
8445 data gspou!epps,tpnf!spdlt
8500 restore 8520
8505 for t=1 to ni:read item$(t),iloc(
t):next
8520 data efbe!nbo,-17,efbe!nbo,-3,hpm
e!bnvmfu,100,tijfme,5,dpggjo,-5
8525 data epps,100,cpyft,-8,cpuumf,9,n
bhjd!cppl,10,hbsmjd,12,diftu,21,lfz,10
0
```

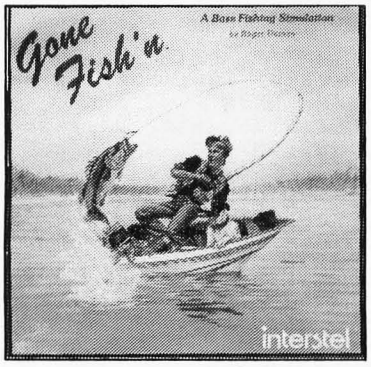
```
8530 data mbshf!ejbnpoe,16,hmpxjoh!cbm
m,-19,bmubs,-19,GMBNF!TXPSE,22,ubcmf,-
15
8535 data ebhhfs,100,spdlt,-24,cpuumf!
mbcfm,100,tusjoh,100,btift,100
8540 data qjfdf!pg!qbqfs,100,eftl,-11,
opuf,100,mfuufs,100,hvbse,-19,dppl,-12
8545 data ubcmf,-20,wbnqjsf,100,gjsf!t
qfmm,100,tjho,-18
8600 for t=1 to 3500:next
8605 color 1,2,2
8610 for t=1 to 85:ellipse 151,83,151,
t:next
8615 color 1,1,1
8620 for t=1 to 85:ellipse 151,83,151,
t
8625 if t=11 then gosub 8700
8630 next
8635 poke systab+24,0:return
8700 color 3:te=4:gosub TEFFECT
8705 gotoxy 8,8:? "Get ready to enter
the"
8710 gotoxy 9,9:? "world of adventure!
!"
8715 return
9000 SCREEN:
9005 poke systab+24,1:fullw 2:clearw 2
9010 wm=2:gosub WMODE
9015 color 1,8,6,8,2
9020 linef 6,10,296,10:linef 6,10,6,15
7
9025 linef 296,10,296,157:linef 6,157,
296,157
9030 linef 50,0,50,10:linef 253,0,253,
10
9035 linef 6,37,296,37:linef 6,46,296,
46:linef 6,99,296,99
9040 fill 0,0:fill 55,0:color 2,1,6,9,
2:fill 0,0
9075 te=4:gosub TEFFECT:gotoxy 11,0:?
"Crin's Castle"
9080 te=1:gosub TEFFECT:color 1:gotoxy
 2,2:? "Place:"
9085 gotoxy 2,4:? "Exits are:"
9090 gotoxy 2,5:? "Visible objects:"
9095 gotoxy 2,11:? "What do you wish t
o do???"
9100 te=0:gosub TEFFECT
9105 poke systab+24,0:return
10000 TEFFECT:
10005 poke contrl,106:'OPCODE
10010 poke contrl+2,0
10015 poke contrl+4,1
10020 poke intin,te:'Text effect
10025 vdisys(1):return
10100 THEIGHT:
10105 poke contrl,12:'OPCODE
10110 poke contrl+2,1
10115 poke contrl+6,0
10120 poke ptsin,0
10125 poke ptsin+2,th:'Character heigh
t
10130 vdisys(1)
10135 return
10300 WMODE:
10305 poke contrl,32:'OPCODE
10310 poke contrl+2,0
10315 poke contrl+6,1
10320 poke intin,wm:'Wrinting mode
10325 vdisys(1):return
```

ST CHECKSUM DATA.

```
 5 data  697, 156, 846, 106, 996, 707,
 196, 138, 664, 674, 5180
 525 data  625, 891, 980, 611, 847, 16
, 358, 388, 749, 282, 5747
 615 data  348, 463, 914, 457, 563, 47
7, 478, 859, 114, 50, 4723
 1045 data  942, 180, 85, 448, 915, 51
7, 507, 260, 387, 364, 4605
 1110 data  371, 467, 241, 518, 79, 90
4, 712, 721, 692, 740, 5445
 2025 data  835, 909, 927, 831, 569, 1
93, 928, 972, 921, 432, 7517
 2125 data  709, 801, 206, 973, 911, 8
09, 181, 519, 894, 473, 6476
 2320 data  629, 798, 220, 483, 18, 81
9, 622, 354, 883, 842, 5668
 3020 data  936, 243, 82, 277, 607, 79
, 154, 324, 981, 78, 3761
 3220 data  483, 674, 410, 511, 456, 8
1, 302, 648, 898, 145, 4608
 3350 data  372, 918, 485, 295, 295, 5
27, 116, 150, 942, 700, 4800
 3445 data  968, 357, 187, 97, 850, 50
6, 170, 358, 552, 705, 4750
 3610 data  762, 491, 665, 908, 19, 77
, 456, 41, 162, 939, 4520
 3900 data  12, 634, 66, 475, 74, 195,
 942, 908, 879, 707, 4892
 4015 data  635, 829, 911, 295, 517, 4
76, 130, 587, 177, 561, 5118
 4145 data  844, 634, 878, 914, 846, 4
7, 582, 836, 917, 356, 6854
 4310 data  881, 847, 920, 405, 111, 7
78, 529, 873, 923, 412, 6679
 4510 data  963, 102, 978, 358, 111, 4
78, 857, 929, 226, 849, 5851
 4720 data  497, 208, 75, 548, 961, 37
7, 248, 537, 299, 979, 4729
 5035 data  578, 165, 643, 490, 689, 3
62, 46, 374, 429, 575, 4351
 6100 data  487, 646, 155, 718, 701, 3
75, 40, 428, 573, 726, 4849
 6150 data  110, 149, 590, 869, 581, 3
55, 460, 41, 529, 815, 4499
 6225 data  789, 605, 20, 148, 861, 37
2, 592, 519, 514, 699, 5119
 6515 data  461, 902, 481, 313, 697, 8
29, 220, 554, 969, 41, 5467
 7040 data  573, 444, 366, 519, 962, 4
5, 544, 960, 976, 868, 6257
 8040 data  839, 38, 998, 457, 728, 85
, 110, 742, 726, 118, 4841
 8130 data  139, 523, 761, 963, 466, 8
32, 390, 935, 231, 337, 5577
 8235 data  570, 635, 811, 961, 64, 66
5, 534, 441, 62, 632, 5375
 8420 data  539, 477, 100, 188, 815, 1
01, 66, 705, 439, 609, 4039
 8530 data  939, 446, 211, 680, 2, 747
, 418, 743, 572, 312, 5070
 8630 data  81, 631, 22, 143, 83, 483,
 460, 779, 701, 184, 3567
 9020 data  156, 851, 238, 311, 471, 2
80, 138, 119, 464, 601, 3629
 9100 data  39, 615, 763, 653, 528, 54
8, 150, 906, 789, 541, 5532
 10110 data  530, 552, 179, 23, 802, 5
33, 450, 547, 531, 555, 4702
 10320 data  543, 909, 0, 1452
```

Test your knowledge, start a war, or just go fish – courtesy of

interstel



Grab your rod and reel and head for the lake with **GONE FISH'N™**, the first (and only!) bass fishing simulation. Outstanding color graphics and sound effects give you the feel of actually being out there.

Use the weather and fishing reports to decide when the fish are biting. Then head for one of eight different lakes, and start fishing!

GONE FISH'N provides everything you need for a day of fishing: a boat, electric and outboard motors, a depthfinder, maps, rod and reel, 16 lures – everything but the cooler!

Cast your line, and the action begins. With **GONE FISH'N**, you control rod and reel action. Land a lunker (that's a real big fish, ya'll) and you could get into the Bass Fishing Hall of Fame. There are even three-day tournaments for you competitive types. **GONE FISH'N** is as close as you can get to real fishing without getting wet!

For the Atari ST: $45.00.

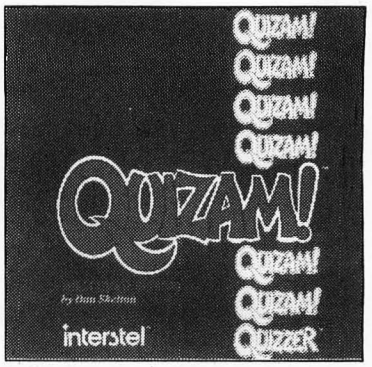
The ultimate strategic wargame! **EMPIRE – Wargame of the Century™** puts you in control of land, air, and sea forces in this simulation of global conflict, conquest, and empire building.

It's non-stop action as you venture into unknown territory, capture cities, and set them to producing the additional forces needed to achieve final victory. But be careful! Your enemies are lurking, and when you meet, it's a fight to the death.

Once you get a taste for command, it's hard to stop. So for all you closet Generals with nothing better to do (like eat, sleep, or go to work), **EMPIRE** provides millions of worlds to conquer.

WARNING: EMPIRE is highly addictive!

For the Atari ST: $49.95.



Finally, a computer trivia game where the computer isn't trivial! **QUIZAM!™** is more than just a trivia quiz – it's a real strategy game for one to eight players.

QUIZAM! includes eight different game boards, eight levels of difficulty, and nearly 2000 multiple choice questions in two question sets: SCHOOL DAYS and FUN FACTS.

And only **QUIZAM!** has QUIZZER, a unique, easy to use program that lets you create your own question disks on any subject. Great for parties or family reunions. And teachers – use QUIZZER to turn **QUIZAM!** into a custom-made educational game for your students.

QUIZAM!'s color graphics and musical effects add to the fun. So throw away your dice and turn on **QUIZAM!** – the first trivia game that belongs on a computer!

For the Atari ST: $35.00. Also for the C64/128, Amiga, and Apple II.

interstel corporation

P.O. Box 57825, Webster, TX 77598
(713) 486-4163

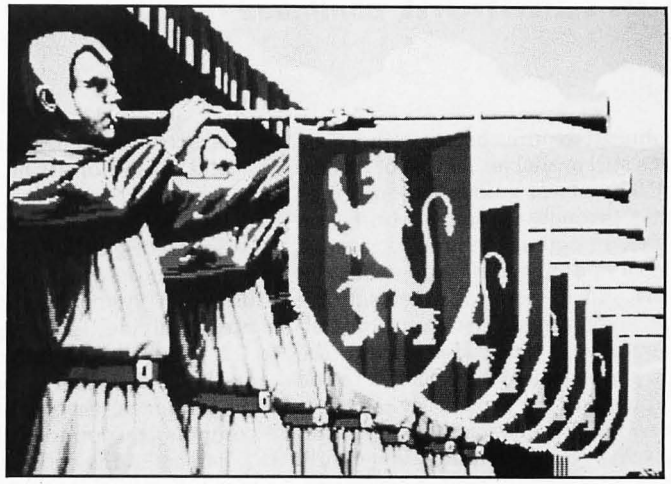
To experience **interstel** games for yourself, visit your retailer or call 800-245-4525 (in CA call 800-562-1112) for VISA and MC orders. Or write to **Electronic Arts**, P.O. Box 7530, San Mateo, CA 94403. Add $5 shipping. CA residents add sales tax. Atari is a registered trademark of Atari Corporation.
Write to **interstel** *today for a free catalog.*

Distributed by

ELECTRONIC ARTS®

Circle #105 on reader service card.

ENTERTAINMENT



Defender of the Crown

Worlds to conquer

The history of adventure gaming

by Arnie Katz, Bill Kunkel and Joyce Worley

Cooperative storytelling is as old as the communal campfire. The unstructured folk-art process was transformed into a game with rules and limitations by two military simulation buffs, E. Gary Gygax and David Arneson.

In the mid-1970s, both men developed an interest in fighting tabletop medieval battles with miniature soldiers. As visionaries might, the two began to wonder why their armies were meeting on the battlefield, what the troops themselves hoped to gain—or desired not to lose—in the confrontation, and the names of the commanders and heroes who led them into battle.

Each participant in Blackmoor, as Arneson called his campaign, devised background and created characters to explain one or more of the fighting forces in the game-world. As these histories became richer and more detailed, both in Blackmoor and Gygax's Greyhawk, the creation took a turn toward the fantastic. Before long, magic, monsters and treasures became part of the scene. Much of fantasy literature has a medieval setting, so it seemed natural to embellish the campaign with these elements.

Gradually, emphasis shifted from full-scale battles to smaller encounters involving individualized characters. Gygax codified the structure into a three-book set of rules which his then-new company, TSR, published as *Dungeons & Dragons*, now fondly known as D&D.

Many changes and improvements have been made in the decade since its inception, but D&D has set the pattern for all nonelectronic role-playing games (we'll refer to them as RPGs here).

An RPG session consists of give-and-take between a game master (GM) and a variable number of players. The GM creates the world in which the players' characters have their adventures. Every RPG employs a system, often a set of random die-rolls, to establish the attributes of new characters. A player guides one or more characters through exploits which increase its experience level. The more experience a character gets, the more powerful it becomes. Play sessions are generally linked together in an ongoing campaign, so characters can be built up over months or even years of participation.

Interactivity has sold well over 10 million RPGs in the last decade, but a flaw in the concept opened the way for the booming computer adventure category. The stumbling block: the person who's most interested in the game never gets to play. The GM consults charts and referees action impartially, while everyone else gets to actually operate the characters.

Another drawback is that a GM can work on an episode for a week, only to have four players demolish it in a couple of hours. A prepared module eases the GM's burden, but it still requires hours of study and fine-tuning to fit a "canned" scenario into an ongoing RPG campaign.

Adventures transfer the GM's duties to the machine. While no computer game can compare to a skilled human GM, adventure software requires little or no preparation and provides an outlet for the suppressed heroic urges of erstwhile GMs.

Crowther and Wood's Adventure appeared on computer mainframes in the mid-1970s. It and other "puzzle dungeons" consist of a string of all-text logical puzzles.

Adventureland (Scott Adams International), authored by Scott Adams, was the first puzzle dungeon for the microcomputer. The home versions were, if anything, even more terse than the mainframe programs, because of the severe memory limitations of 8-bit systems.

The illustrated adventure, which added drawings to the verbiage, made its debut in 1981 with Ken and Roberta Williams' Mystery Funhouse (Sierra On-Line). An illustrated adventure presents each location in the game as a full-screen picture accompanied by a written description.

Until the introduction of Ultima I (Origin Systems) in 1981, all adventures relied on a parser. This part of the program analyzes the player's typed commands and picks an appropriate response. Ultima (and Sir-Tech's Wizardry for the Apple II system) substitutes single-keystroke commands for the laborious typing.

Graphic adventures began with a classic that's still available, Temple of Apshai (Epyx). The player guides the on-screen character through the game-world, and combines an action button and joystick movement to drop and pick up items, fire weapons and to perform other actions. Graphic adventures differ from illustrated adventures in one major respect: the graphic player's character operates inside the graphics on the screen. By contrast, there's no way for a character to jump into the drawing of an illustrated adventure.

The 16-bit systems like the Atari ST unshackled adventure designers. Previously, creating an adventure meant deciding what to leave out. There wasn't enough memory to allow full development of graphics, sound and content.

Improved hardware has encouraged designers to experiment with new ways of structuring adventures. Sierra's graphic adventures for the ST, using a revision of the King's Quest system, incorporate a small but powerful parser which allows the user to execute actions too complex for the joystick.

CinemaWare has invented a new format called the arcade adventure. Titles such as Defender of the Crown and Sinbad (both from Mindscape), utilize real-time interactive action contests to dramatize key portions of the story line. Menus, decision screens, animated sequences, and extensive music and sound effects flesh out the story and characters.

An adventure anatomy lesson

Some readers, especially newer ST owners, may be unfamiliar with this entertainment software category. Although adventures demand more from players than do some other types of computer games, knowing a few basics can spare novices a world of frustration.

Role-playing and interaction are key qualities of the adventure game. The user *plays the role* of the protagonist and *interacts* with the people, places and things portrayed in the game. Every adventure incorporates role-playing and interaction, though there are many possible ways to express these elements in game terms.

Two forms of role-playing are commonly found in computer adventures. Adventures are subdivided into two categories:

Interactive Fiction. In this type of adventure, the player becomes a specific character within the framework of the game.

Role-Playing Games. Instead of assuming the identity of a specific individual, the computer user constructs a new character from scratch and, based on its attributes and its earned experiences, creates an original, fictitious personality.

Separating titles which employ a parser from those which use an alternative order entry system is another meaningful way to divide the category.

A parser dissects orders the player types on the keyboard. It isolates key words in the sentence, then determines how the computer responds to the command.

Parsers in the earliest adventures have vocabularies of less than 200 words and only handle orders in a rigid "verb-noun" shorthand. These limitations narrow the plot possibilities to simple puzzles.

Zork (Infocom) introduced a much more sophisticated parser in 1981. It "reads" complete sentences, accepts multiple commands in the same sentence, and has a vocabulary of 1,000 to 1,200 words. Combined with other features of the Infocom system, this allows designers to present more complex situations. The larger vocabulary also reduces the "guess the right word" aspect of text adventures, which so many gamesmen dislike.

Nonparser systems cut the difficulty of interacting with the game, to heighten the impact of the play experience. Most players find it hard to fully enter the world of the adventure if they have to devote so much energy to typing commands. The continued evolution in computer adventures is the story of attempts to give gamers maximum output for minimum input, while increasing the ability of the player to interact with the game.

Ultima I by Lord British (Origin Systems) introduced the first alternative to the parser. The computer user directs movement and orders a wide range of action with single keystrokes. Obviously, it's easier to hit *O* than to key in *Open Door.*

Another alternative assigns all functions to the mouse or joystick. Players direct on-screen characters, with the joystick, through a picture of the location, and combine button and mouse/joystick movements to select other activities.

This system promotes ease of play, but can't do much with conversation between characters. Most controller activated adventures require the player to collect items while fighting adversaries.

Pull-down menus, as in Temple of Apshai (Epyx), increase the flexibility of controller-activated games. The player picks actions from menus and confirms selections by clicking a button. (ST owners can enjoy an updated version of Apshai and the other two titles in this fantasy set, the Apshai Trilogy.)

Rogue (Epyx) combines the controller with the ability to click on objects. The player guides the on-screen hero over an object to pick it up, then clicks on the corresponding picture in the inventory box to activate it.

Many titles use multifaced control systems. Kings Quest (Sierra) depends heavily on the mouse or joystick, as *do* Apshai and Rogue, but adds a parser for more complex activities. Deja Vu (Mindscape) uses the controller to click on objects, but also incorporates icons and menus to allow such complicated activities as using one object to affect another.

Six hints for novices

Each adventure game is unique, so it's impossible to provide sweeping suggestions about specifics. Here are some general hints to make things easier.

(1) Make sure the game is not too difficult. Many companies, including Infocom, rate the difficulty of their programs on the packages. Put aside ego; you can move up to the real stumpers later.

(2) Read the documentation. Sure, you can figure things out as the game progresses, but why accept the extra frustration that invariably accompanies ignorance of the rules.

(3) Make sure you understand the goals of the game. This will keep you from getting sidetracked and expending great energy to do unnecessary things.

(4) Make a map. This isn't always necessary, but becoming disoriented is an automatic loss in many adventure games.

(5) Determine the amount and form of available help. Many games have a built-in "help" feature, while others are the subject of hint books. A few publishers, like Sierra, even have hot lines and bulletin boards to lend assistance.

(6) Save frequently. Repeating long stretches of a game just because there's one trouble spot is annoying. Conserve time and energy by saving the game position before trying anything too risky.

Home computer adventures have progressed from semiconnected sets of written brainteasers to rich, multisensory experiences in less than fifteen years. The next five should see similarly dramatic growth and improvement. Current technology is still underutilized by designers, and exciting innovations like CD-ROM are poised to enter the market. We can only eagerly anticipate improvements in adventure game software. ///

Circle #116 on reader service card.

REVIEWS

by Clayton Walnum

If there's one thing ST owners have no trouble finding, it's games. So many new entertainment products show up each month at **ST-Log**'s offices, that it's quite literally impossible to cover them all—unless we start dedicating entire issues to them, and that, of course, is not an acceptable alternative.

The other day I was looking over our software shelf, wondering what we were going to do with this flood of new products. I decided an article containing brief reviews of as many games as I could cover over the course of a month might be a good way to catch up a little. The publishers agreed, and so I loaded up a crate of the most exciting looking games on the shelf and carted them home. After an initial weeding out period (based on the amount of time I would have to dedicate to a game to give it a fair review), I began the longest game playing session of my life—a session that was sometimes rewarding, sometimes frustrating, but never boring.

And now, friends, the results are in, the scores are tallied, and it's time to see where that entertainment dollar can best be spent.

Empire

INTERSTEL CORPORATION
Distributed by Electronic Arts
P.O. Box 57825
Webster, TX 77598
(713) 486-4163
Color or monochrome $49.95

If you look on the back of **Empire**'s box, you'll see the following:

"WARNING: This program is highly addictive. Considerable otherwise productive time might be lost. Play only during vacations."

And folks, they're not kidding. I know people who have become so wrapped up in this game that, out of desperation, they finally had to start budgeting their time, allowing only a limited number of hours per day to the playing of **Empire**. This is a game you have to *tear* yourself away from. I've even heard rumors that there's going to be a new branch of AA called "EA" (Empireolics Anonymous).

The scenario: Alliance space is under attack by the dreaded Krellan Empire. The United Galactic Alliance is getting nervous as, one by one, planets are falling prey to the invading forces. Your job as Captain William P. Brown is to begin a counterattack, to land on each planet where the Krellans have begun to build an empire and to defeat them by building your own dominating empire.

Basically, what we're talking about here is a war game, not unlike the famous Risk, where two or three opposing players try to take over an entire planet. But unlike Risk, **Empire** is very realistic. You cannot see what your enemy is up to unless you have pieces keeping you up to date on the other players' movements. In fact, at the beginning of a game, the entire board is a mystery—a blank slate that can be filled in only by constant exploration.

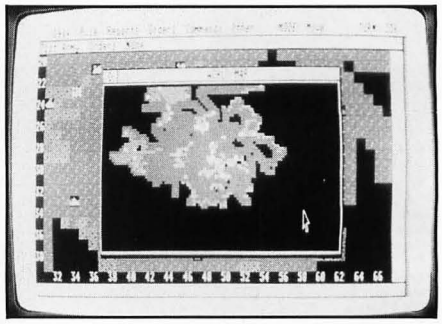
At the start, you've got one city and can "see" only those squares surrounding that city. Each time you move a piece, the area surrounding it is filled in. In this way, little by little, you begin to discover new cities to conquer, and eventually run into your enemy's forces (or they into you).

The playing board is a 100x60 square grid, the entire map being over four times the size of the screen. Since the map is drawn in a standard GEM window, you can scroll over the different areas using the window's scroll bars and arrows. In case you haven't guessed, it'll take you a long time to explore and conquer an entire map. Thank goodness you can save games in progress!

As I mentioned, you start out with one city and, since the cities are where all your production is done, it's imperative to quickly locate and conquer new ones. Once a city comes under your control, you decide what type of piece it should start producing: armies, fighters (jets), troop transports, destroyers, submarines, cruisers, battleships or aircraft carriers. Each piece takes a certain amount of time to produce—from a low of five turns for an army to a whopping 50 turns for a battleship—and each has its own attack, defense and move characteristics. For instance, armies may move only one square per turn, whereas fighters may move five, making them excellent tools for exploring the map.

It'll take many turns before your empire has grown into a force to be reckoned with, and during those beginning stages (typically 60 to 100 turns), you'll see little or no evidence of your enemy's burgeoning power. But he's there, somewhere on the map, becoming stronger with each passing turn—just as you are.

Finally, either you'll stumble upon your opponent's shore or he upon yours, and the battle will begin. This is where the game gets *really* addicting, because, just as in a real battle, the pieces on the board are in a continual flux. Just because your fighter spotted an enemy troop transport a couple of squares offshore is no reason to believe that it'll still be there when you sail up with your battleship. You can see your enemy only in those areas surrounding your own pieces. (Although, once an enemy piece is spotted, its icon will remain in that square after the piece itself has been moved, indicating the location it was last spotted in.) This means planning your strategies under realistic restrictions.



Empire

This is one game you absolutely must have in your ST library. The programmers have covered every detail, including such turn-accelerating features as allowing multi-turn commands. (For example, you may place an army into a random movement mode so that it'll move automatically each turn, thus continually patrolling a particular area with no additional input from you. There's also a lot of keystroke commands to speed things up, such as pressing *H* to send a fighter back to its nearest friendly city.)

The graphics, though not as visually stunning as some ST games, are perfectly suited to the application. And the best news is that it'll run in both monochrome and color (unusual for a graphically oriented game), so nobody has to be left out of the fun. Also, the disk is not copy protected, so you can transfer all the files to a hard disk, should you own one.

To completely describe this game would take too much space (the manual is 72 pages), so I'll say no more and leave the rest to your discovery.

Recommendation: Buy it.

The Sentry

FIREBIRD, INC.
Box No. 49
Ramsey, NJ 07446
Low resolution $44.95

In the last seven or eight years, ever

since home computers became popular, thousands of games have been released. If you took all those games and sorted them by type—putting all the Pac-man clones in one pile, all the Space Invaders clones in another pile, etc.—you'd find that the actual number of *different* games is much lower. These days, it's a real treat to come across a truly original piece of game software, one that you'd have a hard time finding a pile for. **The Sentry** is one such game.

Because the game is so original, it's a little tough to describe, but I suppose you could call it a sort of magical mountain climbing game. The object is to sneak your way up a series of plateaus without being seen by the ever-watchful Sentry and her buddies, the Landgazers. To move around the landscape, you must get energy. Energy is obtained by "absorbing" trees, boulders and Synthoids, a kind of temporary body that you can move in and out of. Unfortunately, you can only absorb an item when you can see the square on which it is resting. And, of course, the higher you are in the landscape, the more you can see.

To move, you must complete a series of actions. First, you place your mouse cursor on a square, then press the *R* key. This creates a Synthoid to which you can move. Once you've created your Synthoid (at the expense of a goodly amount of energy), you transfer yourself into it by pressing the righthand mouse button. The screen turns blue, you hear a mysterious melody, and you're in your new Synthoid, looking back at the old one. To get a higher view, you can create boulders on which to stand (they cost energy too, of course). The higher you stand, the more you can see, and so, the more trees you can absorb to build up your energy.

To finish a level, you must manage to get to the square on which the Sentry stands (the highest in the landscape) and absorb her, and you must do this without being seen. To complicate matters, the Sentry (and the Landgazers, if they are present) are constantly turning. It's no easy trick to stay behind them.

The game employs effective three-dimensional graphics, the screen displaying what your Synthoid can see. You can scan the landscape in any direction, including up and down, and can even get yourself a bird's-eye view by using the mouse to point to a spot in the sky—a spot which is then used as a reference point for looking back at the landscape. Though the boulders and trees are somewhat lacking in detail, the overall 3-D effect can be stunning.

The manual claims there are 10,000 different landscapes to conquer, so it'll be a long time before you get to them all (*I* sure haven't yet), making this game a long-term investment, something that you'll never wear out.

The only criticism I have is that the manual does only a minimally acceptable job of explaining the game. You'll have to invest a couple of hours learning how to move around the landscape and figuring out what to do. But the time spent will be well worth the investment. This is a definite winner.

Recommendation: Buy it.

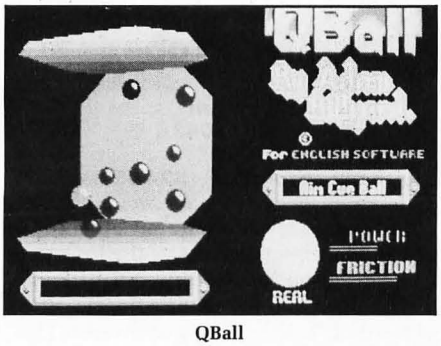
QBall

by Adam Billyard
MINDSCAPE, INC.
3444 Dundee Road
Northbrook, IL 60062
Low resolution $29.95

Yet another entry into the 3-D graphics game foray, **QBall** is an interesting variation of pool. Imagine, if you will, a pool table constructed as a cube within which the law of gravity has been negated. In other words, rather than playing on the surface of the table, the balls move around in the air contained within the cube. There are eight pockets, one in each corner of the cube, and if you think learning to play conventional pool is tough, wait until you get a look at this.

To make matters even more complicated, you may set the amount of spin on the cue ball, the amount of force with which you'll strike the ball, and the friction of the air. You may also take your shot in the "real" or "Planer" modes of play. The former is the 3-D mode where the balls can move in any direction within the cube. The latter forces the balls to "roll" within a single plane.

You may view the cube from any angle by adjusting its position with the arrow keys. Even with this ability, however, setting up a shot is a frustrating and confusing bit of work. You use the aiming keys (the numeric keypad) to position a marker that shows the point at which your shot will strike another ball or the walls of the cube. To say that this perspective is confusing is a gross understatement. It would have been so much better if the aiming was done by rotating the cube from a single viewpoint—that of the cue ball—thus allowing you to see the table from the cue ball's position.



QBall

The two-page manual isn't going to give you much help either, since it does a poor job of describing the game's mechanics. For instance, it states that to begin the game you must choose a one- or two-player game. I spent the first 10 minutes with the game trying to figure out why the 1 and 2 keys on the regular keyboard didn't respond. I thought I had a bad copy of the game until I discovered—purely by guesswork—that you have to press the number keys on the *numeric* keypad.

The graphics are simple, but well done, giving the player believable 3-D movement of the balls. The frustration level of the game, however, is a bit higher than many people may be able to tolerate. It seems most shots are made purely by luck, especially considering the game's timer doesn't allow you to spend as much time as you might like setting up your shot.

Recommendation: Get a demonstration before you buy.

Video Vegas

BAUDVILLE
1001 Medical Park Drive, S.E.
Grand Rapids, MI 49506
(616) 957-3036
Low resolution $34.95

Cancel those plane tickets and put your wallet away. Now you can gamble Vegas-style without risking a nickel. Baudville's **Video Vegas** is about as good a simulation of those infamous Las Vegas gambling machines as you're ever going to see. Available for your risk-taking pleasures are the Lucky 7 slot machine, draw poker, Keno and Blackjack, all of which operate in the same manner as the Vegas originals.

Video Vegas

For those of you who have never had a chance to see those Las Vegas machines, here's a quick rundown of how the games work.

Everyone, whether they have actually played or not, is familiar with slot machines or "One-Armed Bandits" as they're sometimes called. The **Video Vegas** version, called "Lucky 7," is a realistic simulation of the 3-reel type of machine that squeezes so many dollars from unsuspecting tourists each year. The graphics are superb, but just as important—especially

in the case of a slot machine—the sounds have been carefully programmed. This is some of the best use of sound I've heard on the ST. All the way from the sound of the crank being pulled to the clinking of coins falling into the machine's tray when you hit a payoff, the simulation is about as close to the real thing as you could expect from a computer game.

In the video version of draw poker, you place your bet and then are dealt five cards. After choosing which cards you wish to keep, you draw new ones to replace those discarded. If your hand contains a pair (jacks or better), two pair, three of a kind, a straight, a flush, a full house, four of a kind, a straight flush or a royal flush, a payout is in order, based on your hand and the amount originally bet.

In Keno, a Bingo type game, you mark from 1 to 15 of the 80 numbers on the board, after which the computer randomly selects 20 numbers. Payoffs are based on the number of "hits," numbers that both you and the computer marked.

Finally, in Blackjack, you draw cards in an effort to get as close to 21 as possible, without busting (going over). If you do a better job than the dealer, you're awarded a payoff based on the amount of your bet. There are a lot of details to the game that I won't go into here. Most of you are familiar with this game, anyway. Rest assured that **Video Vegas**'s version of Blackjack is a complete simulation, including such player options as doubling, splitting your hand and insurance. For those who need a refresher course, the manual provides all the details.

The graphics in all four simulations are outstanding.

Recommendation: If you like Vegas-style action, buy it.

Plutos

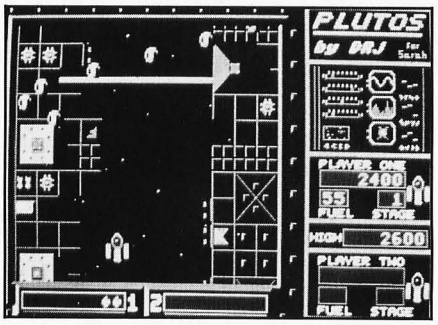
MINDSCAPE, INC.
3444 Dundee Road
Northbrook, IL 60062
Low resolution $29.95

Ever since the Space Invaders craze swept through the nation back in the early 80s, more space shoot-'em-up games have been released than just about any other kind of computer entertainment. Strangely enough, not too many have been released for the ST—until recently, that is.

First came MichTron's Goldrunner, and now, following the same tradition, we have **Plutos** from Mindscape. In **Plutos**—which is very similar to Goldrunner—you pilot your spacecraft over the surface of an alien fortress, destroying as much as you can, while at the same time dodging and battling the fortress's airborne defenses. There's really not much to say about a game like this: You shoot and shoot and shoot, until your joystick hand is so numb you're forced to pause the game for a breather—or your ship has taken too many hits, bringing on the end of the game.

Plutos reminds me of the first game I ever purchased for my Atari 800, way back in the Stone Age: Caverns of Mars. You remember that one? In **Plutos**, just like Caverns of Mars, shooting fuel dumps refuels your ship. (I've never been able to figure out how destroying one of these things manages to fill a ship's fuel tanks; a case of artistic license, I guess.) There's also the waves of enemy spacecraft that you must guide your ship through, either by clever dodging or skillful blasting.

Other targets in this nicely animated game include flip-flopping discs, twirling balls (they actually look a lot like soccer balls) and spinning spacecraft. Some extra surface targets include question marks that, when shot out (you need multiple hits), may or may not award you with an extra life, rotating disks that fire missiles, and various sizes of buildings.



Plutos

The object of the game is to blast your way to the end of each fortress, then shoot out the eyes of the cybernetic sentry guarding the entry to the next level. If you manage to blind the sentry, you'll be moved to the next level. If you don't, you'll have to repeat the current level. Each succeeding level offers more action than the one before, making this one of those games that joysticks dread.

Though the graphics offered here aren't as nicely done as those of its predecessor, Goldrunner, the game is, I think, more exciting to play. All in all, not a bad job.

Recommendation: For shoot-'em-up fans.

Jupiter Probe

MICRODEAL
576 S. Telegraph
Pontiac, MI 48053
(313) 334-8729
Low resolution $24.95

Yet another space shoot-'em-up is **Jupiter Probe**, brought to you by the fine people who gave us Goldrunner (though MichTron is now using the MicroDeal label for their entertainment software). This game is not as graphically exciting as Goldrunner, but due to digitized voices, a rousing musical score (or at least as rousing as you can expect from a three-voice computer), player-controllable shields, and Ultra-Sonics (which, with one exception, destroy everything on the screen) it is a little spicier than the previously reviewed Phobos.

However, like all software, there is a toss-up. The animation here, while acceptable, isn't as impressive as Phobos's. Also lacking are ground targets. (Well, there are these little stars that, when you go over them, give you extra shields or Ultra-Sonics, but you can't shoot them, and wouldn't want to anyway.)

On board your ship is a battle computer that warns you, in a digitized voice, of an advancing enemy, as well as telling you when you've earned an extra shield or Ultra-Sonic. Your enemy may appear in one of three forms: Enemy Fighters that do not fly in formations, though they do appear in groups; Mutations, which are the Fighters' mother ships, take several hits to knock out and are unaffected by Ultra-Sonics; and Formations, which attack in a closely knit group. All enemy ships are capable of firing back at you, so you'd better plan on some fancy maneuvering to make it through to the next level. And, of course, as the levels go up, so does the difficulty. This quickly becomes one tough contest.

Like most arcade games of this type, you can't win against **Jupiter Probe**. Going for the high score is the objective here. A minimum score of 50,000 is required to make it onto the high scoreboard, so you better plan some late-night zapping if you want to impress the neighbors.

Even though this is a step down graphically from the impressive Goldrunner, it's a fun game that'll keep your trigger finger active for many an hour.

Recommendation: For shoot-'em-up fans only.

Pinball Wizard

ACCOLADE
20813 Stevens Creek Blvd.
Cupertino, CA 95014
(408) 446-5757
Low resolution $34.95

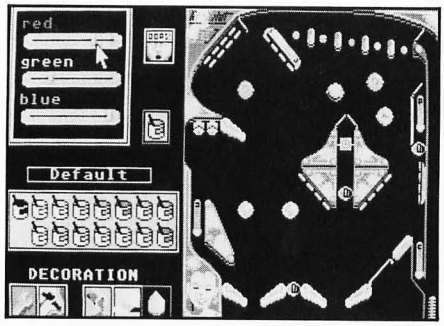
Ever since Bill Budge came out with his Pinball Construction Set way back in the early 80s (the Atari 800 version was distributed by Electronic Arts), these do-it-yourself pinball games have been quite the rage. Just about every computer system in the market has some type of pinball construction program available, and lucky owners of the Atari ST actually have two to choose from. Although, in my mind, there's really no choice. **Pinball Wizard** from Accolade is the clear winner.

For those of you who have never seen

one of these remarkable programs, or never had the chance to use one, a pinball construction program gives you a set of pinball parts—bumpers, flippers, spinners, etc.—that you may arrange any way you wish, allowing you to create your own version of the ultimate pinball game. Of course, if you're the lazy type, there are several pre-made games you can load from the disk to play immediately.

As far as parts go, **Pinball Wizard** has them all, and each one can be set to any score value you want. You can even link several targets together, giving the player bonuses when he hits them all, including not only extra score, but extra balls and even free games. Parts are placed on the play area by picking them up with the mouse pointer and moving them to the chosen location. It's so easy, you could put together a rough game in only a few minutes.

To create your own solid objects, or to beautify the board, **Pinball Wizard**'s toolbox has a "Decoration and Obstacle" mode, a sort of stripped-down DEGAS where you can paint directly on the pinball layout in two modes. The first mode, used for decorating the play area, is "invisible" to the ball, which rolls over whatever was painted in this mode as if it weren't there. In the second mode, you can actually add, via "paintbrush," non-



Pinball Wizard

scoring parts the ball cannot pass through. This lets you create all manner of barriers, including shoots, slides and any other type of construction you may need.

Once you have all your painting done, have placed all the parts and set their score values, it's time to set the game's "physics," including slope (the angle of the table), the sensitivity to tilting, the ball's speed, the elasticity (the amount of bounce) of the bumpers, and the number of balls allowed to the player per game. You can also set a feature called "stroboscope," where the ball flips between visible and invisible states at a predetermined rate. That's one sure way to make even the simplest pinball layout an excruciating challenge.

The game can be played from either the keyboard or using the mouse. If you choose the keyboard mode, you may, if you wish, redefine which keys control the game.

Unfortunately, **Pinball Wizard** has one minor problem (or major problem depending on how clever you are). The manual (all six pages of it) is painfully brief, leaving a lot of the program's workings up to your own discovery. It describes the program in only a general manner, not even bothering to give you a rundown on what all the parts are and what they do. You're going to need to do a little experimentation, and spend some time playing the sample games, to get a clear idea of what does what.

Still, it's a nice piece of software that uses exceptional graphics and sound, and that can provide the player with many hours of pinball fun.

Recommendation: Buy it if you're used to dealing with incomplete manuals.

Now, if it's okay with you, I'm going to take a *long* break from game playing. My head is spinning, my hands are blistered, and all my joysticks are broken. You have my recommendations; all you need now is a software dealer and your wallet. As for me, I'm going to give those joysticks a decent burial—then take a nap. ///

Goldrunner

by Steve Bak and Pete Lyon
for MICRODEAL
Distributed by MICHTRON
576 S. Telegraph
Pontiac, MI 48053
(313) 334-5700
Low resolution $29.95

by D.F. Scott

It's about time someone wrote this game. The ST game player withstood nearly two years of not having one decent Galaga-style game. No one could explain how coin-op games could scroll so smoothly and have such fluidly-moving targets using 8-bit "engines," when we ST-owners possess superior hardware and our scrolling looks like riding a stagecoach over Death Valley.

Finally, there's **Goldrunner**. The scrolling of its background terrain is so fluid (as I said in **ANALOG Computing**'s "ST notes") that you'll be taking apart your monitor in search of axle grease. Authors Steve Bak and Pete Lyon of Microdeal (their previous achievement was Karate Kid II) receive my "Gold-plated Blitter Chip Award for Finally Coming Through with Something We Needed after Waiting Forever." This may also be the first game I've played whose music I'm happy to leave *on*.

You really don't need docs for this game; you shoot things and they blow up. You can fly with either the joystick or mouse. Don't let other things shoot you, and by all means, don't run into any tall buildings with long shadows. Your ship is a brilliant gold and begins life fully equipped with five deflector shields, four cannon turrets, and "turbo thrusters" which behave more like trans-warp drive. Your shields protect your ship, but not your faculties—you lose turbos after two shots and two of your cannons after four. After six hits, it's pixels to pixels, dust to dust. You have four lives to complete your mission, with no bonus lives.

The terrain levels are called "rings" because if you travel far enough in any one direction, you'll end up where you started. People who see this game instantly say, "Wow, this is like Xevious!" They later learn **Goldrunner** has a few features of interest that Xevious didn't pursue: speed control and direction reversal. This is where the mouse works wonders—it can be as responsive as the accelerator pedal on a Lamborghini.

The scrolling doesn't just flip between slow and fast gears; it changes speed *gradually*. If you slow down enough, you don't stop—you go into *reverse*, giving you full freedom of motion. Your gold ship performs a dazzling 180-degree backflip that would make Greg Louganis green with envy.

The objective is to destroy a minimum number of ground targets in each ring, here represented as "energy processing centers." Your progress is recorded by a pink atom icon on the right side of the screen. As the ring's energy is depleted, pixels drop from that icon like light bulbs from a Las Vegas marquis during a hailstorm. When each pixel is gone, you're free to find the exit into the next ring.

The ground features are assembled components rearranged in successive rings. They include tall buildings which would be relatively easy to avoid if you weren't being attacked by airborne enemy ships every second. There are also several harmless shrines and ground murals apparently constructed in worship of various gods and squid.

Certain regions of each ring contain tight packages of about 100 ground targets, none of which will fire back at you. The casual player might approach this region as a bounty handed to him on a gold platter. Don't be trapped (the Tao reminds us) by illusions; your bullets destroy the first target they touch. When flying around the "energy cell sandwich," the air defenses consider *you* an easy target. Unless you have a clear path between yourself and the airborne target, you're virtually defenseless, since your bullets end up embedded in ground targets.

Thankfully, the airborne "ships" cannot collide with you. Instead, they glide over or under you. If collision were the order of the day, the average game would last four seconds. Ships usually travel in fleets of three, whether they fly into view from off-screen or are dispensed by a mothership. There may be twenty different types of enemy ships, of which only one—a yellow/violet diamond-shaped guy I like to call "The Yellow/Violet Diamond-shaped Guy"—can *multiply*. If you shoot one of the three "Guys," another multiplies into three, leaving you faced with five. Shoot one of those, *and yet* another violates the laws of physics to make seven.

If you die, the ring you've worked so diligently to pulverize rebuilds itself. This presents a peculiar predicament for one's scoring strategy: as the player learns to master one ring, after dying, she may destroy the same targets *again*, and really rack up some points. Yet, when she finally masters that ring, then heads to another (ring 4 is hell; it's even called "Styx"), her fate may be sealed by enough enemy mines to dwarf all those planted in the harbors of Nicaragua. With mastery may come lower scores. Enemy mines look like those in Defender, although each may have peculiar characteristics. Some just float on by; others are magnetized to your ship; and some seem to be tethered to you—turn and run, and they whiplash into your bare backside like a mosquito who's spotted a varicose vein.

The avid Galaga or Gyruss player will welcome the "challenge waves" in between rings, where the object is to blow up everything for a 10,000-point bonus. Your ship is nonvolatile during challenge waves, though, since this stage has no speed control, I find it convenient to drop the mouse and pick up my joystick.

After exhausting all possible praises of the authors' permutations on popular game premises, we arrive at the fact that the spectacular fine-scrolling routine Bak and Lyon have conceived is what makes **Goldrunner** work—to be precise, what makes it *fly*. If they use it in another game, it too will succeed. This is yet another instance where two programmers have created code that outperforms DRI's, and they are to be commended. ///

D.F. Scott is an artist, writer, educator and programmer living in Oklahoma City. He is currently engaged in the study of quantum physics, computing and other ways in which elementary particles interact with each other. Otherwise, he fills infinite pieces of paper.

REVIEWS

Keys to Solving Computer Adventure Games

by M.K. Simon
PRENTICE-HALL, INC.
Englewood Cliffs, NJ 07632
(201) 767-5054
286 pages (softcover)
$19.95

by Clayton Walnum

Adventure games make up a huge part of the entertainment software market, so large a part, in fact, that they've spawned a sub-market that's filling many a company's coffer with cool, green cash. We're talking, of course, about the business of selling adventure clues.

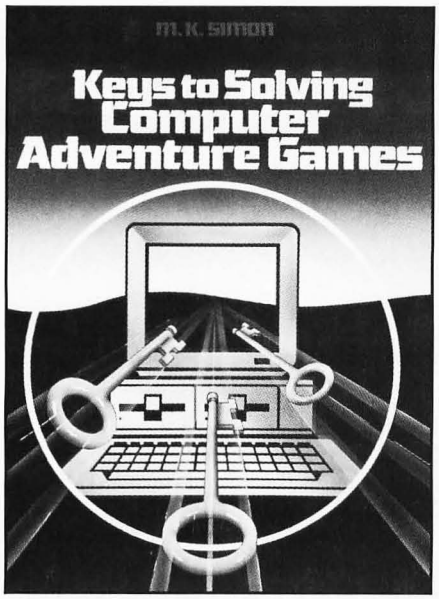
Why is this market so lucrative? If you don't know, you've probably never played one of these delightful, yet frustrating contests. Nothing is worse (in an ironic sort of way) than being stuck on a single puzzle for hours, days, even *weeks*, reducing your brain to mush as you attempt—in an effort to get *any* helpful response from the game—every word combination possible in the English language. Anyone who's struggled with the Babel fish puzzle in Infocom's excellent Hitchhiker's Guide to the Galaxy knows exactly what I mean. (Go ahead, ask them.)

It didn't take game developers long to realize that customers would pay goodly sums to obtain relief from the "brick wall syndrome"; so the manufacturers themselves became the first to sell hints and solutions to trembling, weary-eyed adventurers.

Several years ago a company known as Arrays, Inc. released a book titled *The Book of Adventure Games*. It was an immediate hit and can now be found on adventurers' bookshelves across the nation. The people at Arrays snatched up part of the hint-pushers market by supplying clues and solutions to dozens of games within the pages of a single book, at a price less than that of three Infocom single-adventure hint books. They were so successful that it wasn't long before *The Book of Adventure Games II* made its way into the public's eye.

And wherever you find success and money changing hands, you'll find imitators. Prentice Hall quickly scooped up a manuscript called ***Keys to Solving Computer Adventure Games*** (hereafter referred to as ***Keys***), and, by providing clues to new games not covered in previous books, gave Arrays, Inc. some competition.

M.K. Simon, the author of ***Keys***, is so bold as to mention both volumes of *The Book of Adventure Games* in his introduction, admitting that his book resembles those earlier ones. (In his defense, I must mention that he does tell the reader that if he or she is interested in the solutions to games not covered in his book, they should purchase *The Book of Adventure Games I and II*. This may have been



Keys to Solving Computer Adventure Games.

just enough to stop the people at Arrays from leaping from high windows with the manuscript of Book III tucked under their arm. This, of course, is purely speculation.)

Keys was written with Apple computers in mind, and so, not surprisingly, all twenty-five games represented are available on that machine, while only twelve are also ST titles: Black Cauldron, Borrowed Time, Brimstone, Crimson Crown, King's Quest II, Leather Goddesses of Phobos, Nine Prices in Amber, Oo-Topos, Spellbreaker, Tass Times in Tonetown, Transylvania and Trinity (8-bit users will find only five games of interest to them). For ST people, that comes to $1.66 per game, still a good value.

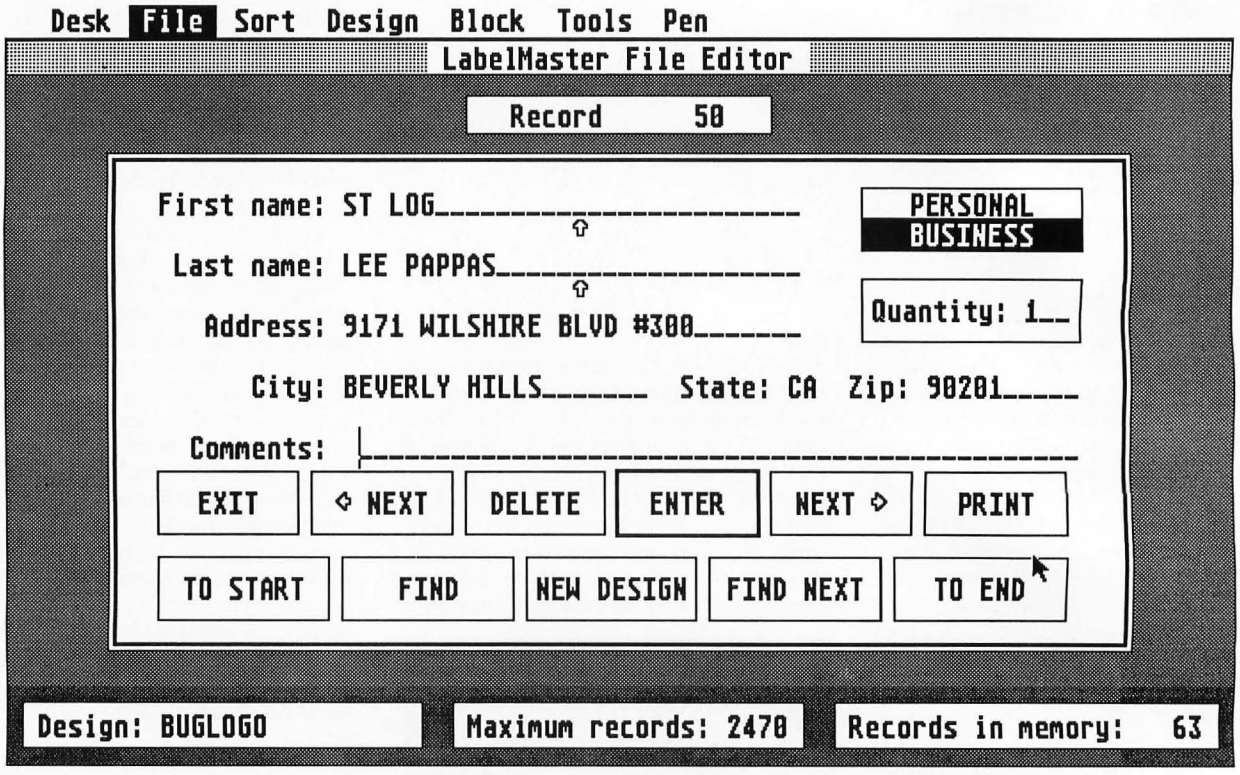
The book is divided into three sections. The first gives the reader complete adventure maps, including exits and room descriptions. Help is organized by placing a hint number in the room with the puzzle. This number is used to look up a clue in the list provided in this section. This clue is another number used to find the actual hint in the master clue list, section two of the book. There are 911 clues in the master list. Wow!

The third section supplies the stymied adventurer with complete solutions (or "walk-thrus" as they're frequently called), leading the player step by step through a game. Not all the games are represented in this section, since those included are, in the author's words, "games where it is possible to define unique solutions and which . . . are sufficiently complex as to warrant its inclusion." ST users will find walk-thrus for all games except Brimstone and King's Quest II.

Also included in the book is a chart showing which games are available on which computers (Apple II, IBM PC, Macintosh, Commodore 64/128, Amiga, Atari XL/XE, Atari ST, TRS 80 and Tandy 1000). That'll help those with other systems (in *addition* to their ST, of course!) decide whether the book will fit their needs. Also, there's a short user's guide that explains how to use the book and a "Tips on Playing Adventure Games" section that'll help novices get up to speed with their adventuring.

Those of you with an interest will find ***Keys*** a good example of what can be accomplished with a desktop publishing system. The entire book (except for the table of contents and copyright page) was obviously "typeset" using one of those marvelous new software packages. And, although the quality of the text is less than one would expect from a regular typeset book, it is, nonetheless, an impressive job, especially on the maps. There's no clue as to what software the author used (if indeed it *was* the author) to set up the book, though it was undoubtedly done on an Apple II, since that's the author's chosen machine.

Like its predecessors, this book is a fine value for all adventure fanatics, providing a cure for frayed nerves, short tempers and suicidal tendencies. At $19.95, it's cheaper than punching out your monitor and more sensible than screaming at loved ones. Pick it up, find out how to get that Babel fish, then you too can stop kicking the dog. ///



LabelMaster Elite

by Norm Richards and Walt Knott
MIGRAPH, INC.
720 S. 333rd (201)
Federal Way, WA 98003
(800) 223-3729 or (206) 838-4677
High and medium resolution $44.95
Supports most popular 9- and 24-pin dot-matrix printers

by Betty D. DeMunn

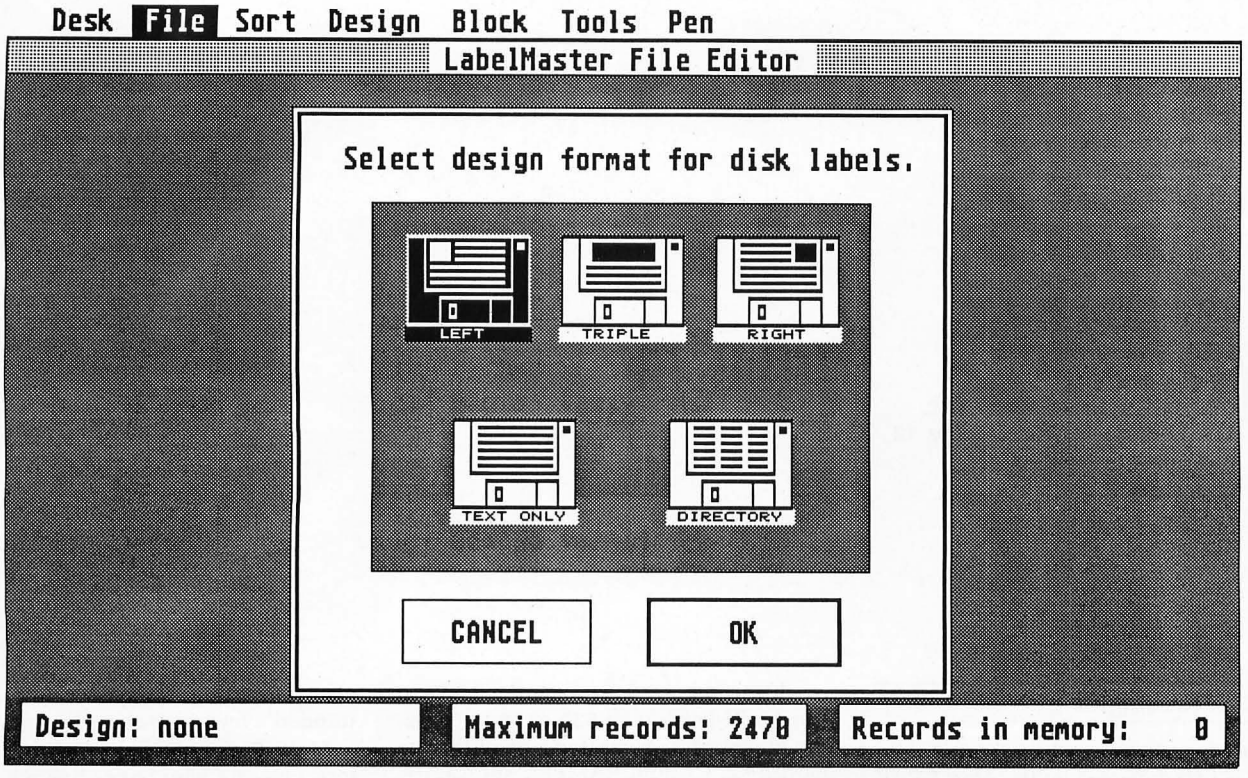
In 1986, Migraph, Inc. introduced the first LabelMaster, which was an effective, graphics-oriented, label-printing program—but limited to 1x3½-inch labels, severely restricting its use. Heeding popular demand, Migraph has replaced LabelMaster with **LabelMaster Elite**, which not only accommodates 1-, 2-, 3- and 4-inch continuous stock forms, but 3x5 and 4x6-inch index cards as well. The graphics editor has also been beefed up with more sophisticated drawing potential to inspire the least creative of us. "Blah" labels can now become "Ahh-h!" labels. (More on the graphics editor later.)

The term *user-friendly* is painfully overworked, so I'm calling **LabelMaster Elite** tail-waggin', hand-lickin', roll-over user-affectionate! Within minutes of booting, I was merrily spinning change-of-address labels for my daughter, hilarious (to me) mailing labels for friends, and a couple of tidy disk labels to replace the messy ones I've been trying to decipher. Credit is largely due to the straightforward, "A-B-C" manual which "tells it like it is"—a rare advantage for those of us who are not techies, progies or hackies.

Now, let's get up close and personal. **LabelMaster Elite** is GEM based, with pull-down menus, mouse or one-stroke keyboard commands. The mailing list manager allows you to define business or personal records. Personal records have three lines of print; business records have four, allowing you an "Att:Name" on top if you wish. Both have an additional forty-character comment line which can be printed or not. It's nice for recording salient facts. Or rude remarks. Or clever slogans. There is room on the disk for about 1400 records, and, as **LabelMaster Elite** is not copy protected, you're not limited in storage capacity. If you run out of record room, just copy a new disk or trash some unnecessary files.

Some of the word-processor functions are: search, delete, modify, and sort on first or last name, address, city, state or zip. There is a feature that allows you to print "freestyle," meaning you can mix type styles and pitch on the same line for emphasis or aesthetic satisfaction. And you can print up to 999 labels at a whack. Print entire files of "personal" or "business," or both. With or without a design. **LabelMaster Elite** keeps track of the number of records in a file as you go, and lets you know how many blanks are left.

Indeed, the authors have paid meticulous attention to detail. Careless or lazy typists will appreciate the automatic comma between city and state, the capitalization of the two-initial state abbreviation and the caret-defined parameters of the working label. This latter feature keeps the printing from oozing off the right side of the label, and saves counting letters. The cursor arrows (or mouse) move you around the record quickly, and a double row of buttons give you options such as forward or back in the file, exit, change design, delete, etc. There's even a print button for one label, just for practice or preview.



The creative challenge of **LabelMaster Elite** is in its graphics editor. Supplied with the program are about eighty ready-to-go designs covering a multitude of subjects, most of which are extremely well done. And there's space to stow plenty of your own creations. When you click on a specific design, you can innovate to suit your wildest urges. Pull down the "Pen" menu. Now you can "draw" in black, white, or gray (checks). Or choose "Point," to plunk one dot at a time on the screen. When you edit your design, you can move it up, down, left or right. It wraps. Erase what you don't want. Vertical flip, horizontal flip, and invert are other options under edit design. Wait, there's more! Under the "Tool" menu: frame, circle, square, box, line and mirror. What more could you ask?

Oh, so you want a block menu? Okay. **LabelMaster Elite** gives you a clipboard, so cut-and-paste, copy and move blocks to your heart's desire—in replace or transparent modes.

And, if that's not enough, there's a modern capital font called Broadway, for elegant initials. An unexpected bonus: **LabelMaster Elite** is compatible with PrintMaster graphics.

One of my favorite features is the 3½-inch disk label setup. Choose normal, wide or condensed print. Place your icon or drawing right, left or center of label. You can select triple-width for the graphic, then print up to seven lines of text, including the title, on the upper edge of the disk. And, if you want a disk directory, **LabelMaster Elite** will print that too.

The applications of this program are virtually endless. Try running the graphic through in one color, and the text in another. Make gift stickers, ID labels for practically anything and everything: loaned articles, name tags, packages, books, canning labels, spice jars, records, albums. And what fun for holiday card mailing labels! Or print a message on one side of 4x6-inch cards (up to ten lines of text without design), then turn 'em over to print your mailing list for special announcements or reminders.

My peerless 520ST is new, so I'm being very selective about the software I buy for it (for a change). As you may have guessed, I'm quite pleased with **LabelMaster Elite**, its authors, Migraph—and myself, for choosing it. This is one time you can read the plentiful information on the package and be assured that it will do what it says it will. In addition, Migraph is that rare breed of software company that won't ignore you once the warranty card is in. From personal experience, I've found the folks there friendly, helpful and quick to respond.

You don't often find a workhorse and a toy in one package. Migraph listened to its customers, acted upon constructive criticism, and came up with a powerful program that takes plain old boring labels and transforms them into eye-catching, attention getters. Although I have stressed home use, this would answer the needs of small business very nicely.

Important! Registered owners of the original LabelMaster (you *did* remember to send in that warranty card, didn't you?) may upgrade to **LabelMaster Elite** by sending the original master disk, plus $5 to Migraph. Your data files can be easily converted to the new version.

Betty D. DeMunn is a professional actress and free lance writer who lives in Buffalo, New York. She's been addicted to Ataris since 1982, when a 400 followed her home one day, and grew up to be a 520ST. Other hobbies include: one husband, five children, seven grandchildren, and one great-grandson, Nick. Wow.

Multi-Forth 1.1 for the Atari ST

by Greg Guerin
CREATIVE SOLUTIONS, INC.
4701 Randolph Road, Suite 12
Rockville, MD 20852
(301) 984-0262
520/1040ST $89.00
Requires TOS in ROM

by A.N. Kensington

More than two years have passed since FORTH first appeared on the Atari ST. Indeed, FORTH has been with us since the very beginning, when the Dragon Group's 4xForth became the ST's first commercially available language. At least a half-dozen other FORTHs have been offered since then. Some evolved, while others have already faded into obscurity.

Creative Solutions' **Multi-Forth** has, indeed, evolved. The original 1.0 release, published last summer at $149.95, won immediate recognition for its exceptional versatility and wealth of features. Release 1.1 offers even more power at a substantially lower price.

Multi-Forth is based on CSI's highly regarded MacForth system, which is in turn derived from the Forth-79 standard. Old-timers may raise an eyebrow here, and wonder why **Multi-Forth** doesn't conform to the newer Forth-83 standard. If the differences between Forth-79 and Forth-83 matter to you, then you probably know enough about FORTH to work around them. If you don't know—or care—about FORTH standards, relax. **Multi-Forth** runs virtually all of the examples found in the major reference books with no changes whatsoever.

The package consists of two single-density disks and a fat, gray binder containing nearly 400 pages of documentation. Happily, there are no extra "levels" of support to buy, no licensing fees, no royalties, and no copy protection. You can run **Multi-Forth** on a 520 or 1040ST, using one or two drives, with or without a hard disk. Custom RAMdisk software is provided with the system. The only hardware requirement is TOS in ROM.

Multi-Forth's multitasking interpreter employs a direct token-threaded code, which is position independent and fully relocatable. Its 32-bit stacks and variables give you direct access to the entire addressing range of the ST's 68000 microprocessor.

CSI claims 100-percent compatibility with Motorola's 68010 and 68020 processors as well, so programs you write with **Multi-Forth** may be transportable to future Atari systems using those chips.

All the features you'd expect in a good FORTH system are here. A debugger, tracer and decompiler are all built in. There's an in-line 68000 macro assembler, which will be extended to include 68020 opcodes in a later release. You also get floating point math capability, provided by a Motorola's 68343 Fast Floating Point package. Extensive support of both TOS and GEM, plus a nice assortment of graphics utilities for windows, rotation and scaling, give you easy access to the ST's full capabilities.

Multi-Forth lets you include local variables in your definitions. I've never used a FORTH system with this capability before, but it looks like a great way to improve the readability of code. The support mechanism for arrays and structures is also interesting, though I wouldn't have minded a few more string manipulators.

Multi-Forth is supplied with two editors. One is a bare bones line editor that treats FORTH code in nostalgic 1024-byte "blocks," as was done in the 1970s. This may be useful if you're transferring code from some other ST version of FORTH that also uses blocks, or if you actually prefer blocks (ugh). I'll stick with the newfangled "stream" files, thank you, produced by a normal text editor like MicroEMACS.

The version of MicroEMACS included on the **Multi-Forth** disk is adequate, but not as good as other text editors I've seen in the public domain. The cursor keys work, but the function keys and other special keys don't, and the mouse just sort of sits there. You can use almost any text editor with **Multi-Forth**, as long as it doesn't use any weird escape codes. Still, it'd be nice if CSI *included a slick*, GEM-based text editor written entirely in **Multi-Forth**, both as an alternative to MicroEMACS and as a programming example.

My other gripe with **Multi-Forth** is the documentation. It's certainly handsome enough, with nice typesetting, brisk writing, cute drawings and a minimum of obvious errors. What information it presents is clear and very usable. But there seems to be a lot of other interesting stuff on those two disks, in the form of source code and examples. It's nice to know that information is there. But *what* is it, and *where*?

The dozens (and I mean *dozens*) of files are not indexed or summarized anywhere in the manual. The manual itself isn't indexed either, though it does contain a useful glossary of **Multi-Forth** words—but are they *all* the words?

At least CSI makes it easy to ask questions. They maintain a telephone hot line that offers free support to registered owners of **Multi-Forth**. If you're a CompuServe subscriber, just type *GO FORTH* at any system prompt and you'll find yourself in CSI's MacForth/Multi-Forth SIG, with dozens of experienced users to buttonhole and data libraries full of free code.

Two other **Multi-Forth** features deserve special attention. Snapshot lets you capture an image of your current work environment on disk, so that you can easily customize your FORTH system to include all the features you want, and load it all in at once.

The second feature, Turnkey, turns your completed FORTH programs into stand-alone applications that are self-loading, self-relocating and completely independent of **Multi-Forth**.

Turnkey makes **Multi-Forth** potentially useful as a development system for commercial software. The ST version of **Multi-Forth** uses the same 68000 kernel as CSI's MacForth and Amiga Multi-Forth systems. In theory, you should be able to port your ST **Multi-Forth** applications to the Mac and/or Amiga with relatively few changes. Just be careful to isolate the machine-dependent portions of your programs.

I was curious to see precisely how compatible Atari **Multi-Forth** was with its cousins. I obtained a copy of the Amiga Multi-Forth and typed in several example programs from various FORTH reference books. They yielded identical results on both machines. Then I tried a standard version of the "Sieve of Erastothenes" prime number benchmark, to compare the speed of the ST and Amiga interpreters. My 520ST executed ten iterations of the Sieve in 12.5 seconds. The Amiga (with 512K) ran exactly the same test in 14 seconds. I also compiled the Sieve into F83 2.1, a public domain ST FORTH available on CompuServe. It came in at just a hair under 14 seconds.

Creative Solutions' **Multi-Forth** feels good. Although much of its basic functionality may be found in other less expensive FORTH systems, the little touches here and there make it unmistakably professional. This was an intriguing package when it cost $149.95. At $89.95, serious FORTH programmers may well find it irresistable. ///

A.N. Kensington has worked with Atari computers for over six years. Once employed as the Technical Editor of a major computer magazine, he has since discovered that there's more money to be made in selling software than in criticizing it.

ST Sprite Factory

FUTURE SOFTWARE SYSTEMS
21125 Chatsworth Street
Chatsworth, CA 91311
(818) 341-8681
Low resolution $39.95

by David Plotkin

ST Sprite Factory (**SSF** for short) is a utility for creating your own sprites and animating them. It includes the actual program for creating multicolored sprites and some sample programs showing how to use sprites in GFA BASIC, C and Personal Pascal.

Sprites are blocks of data which form a picture for display on the screen. Atari 8-bit users are familiar with this concept called "Player-Missile Graphics." A good example of sprites are the characters in MichTron's Time Bandits.

If done properly, a small, colorful sprite can move across the screen without affecting the background. But defining the sprites themselves can be a difficult and time-consuming procedure. This is especially true if you want multiple versions of the sprite—each one slightly different from the one before—to use in an animation sequence. **SSF** solves this problem by providing a whole host of tools for "point-and-click" design of up to 60 sprites, each 32 lines high and 32 bits wide, using all 16 available colors. Once you've designed some sprites, **SSF** lets you animate any sequence of the 60 frames, in any order, at your chosen speed.

When you boot **SSF**, you'll see the main screen where you design your sprites. To the left is a large 32x32 grid area. To "turn on" a pixel in your sprite, just click in this area. As the sprite takes shape, you can view it at its actual size in a small area on the right of the screen. Across the top of the screen is a color bar where you can use the mouse pointer to select the current color for drawing. On the right side of the screen are boxes which you can click to activate the tools used to draw your sprite. These include several styles of fill; mirroring horizontally and vertically (or both); and drawing lines, circles, boxes and filled shapes.

You can rotate the sprite in 1-degree increments or quickly in 90-degree jumps; flip the image; shift it up, down, left and right; and shrink the image in pixel increments. You can also swap colors in the palette, as well as set the palette to the colors you want.

SSF also supports cutting, copying and pasting to another file—it can hold two

in memory at once—and to a special "paste" buffer where you can work on the image using the tools mentioned above. You can view all 60 frames (or however many you have defined) on a separate screen, and select which frame to edit next from this screen. Of course, you can load and save the frames to disk, in standard DEGAS format files. **SSF** will also automatically create a "mask" file to prepare the background to receive a sprite, without distorting the colors in certain modes.

Once you've defined some frames, you can animate them. To do this, you switch to the screen showing all the frames, then select with the mouse pointer the frames in the order you want to use them. You can also set the animation speed. The order and speed can be saved in a special animation file, which can be reloaded and used in future animations with the joystick demo included. An animation sequence can consist of up to 1,000 frames.

If **SSF** only consisted of the utility to create sprites, it would still be a good program; it just wouldn't be particularly useful to programmers who aren't familiar with the ins and outs of bit-blitting. But there *is* much more included on the disk. Three archived files contain demos showing how to use the sprites you've created. The carefully written manual also provides tutorials on screen memory, using GEM, the various bit-blitting modes and what they mean, and performing sprite collision detection.

The most thorough demo is called "Joystick." With this program, you can load a sprite file and set up animation sequences corresponding to the 16 joystick directions (eight with button up, eight with button down). These sequence files can be saved, and you can plug in a joystick and move your sprite over the screen.

Your control of each frame is extensive. You can set the speed of changing frames, the speed at which the sprite moves across the screen and even the blitting mode (OR, NOT, XOR, AND, etc.) and mask mode. It's somewhat cumbersome to set all the variables for each frame of the animation, but the high degree of control can produce some outstanding effects.

By studying the source code (Personal Pascal) included on the disk, you can see how to use sprite files and animation files in your own programs. Most of this information is also detailed in the manual, although no demonstration code is included, nor is the exact code to include in your program shown. However, the file structure for animation files *is* shown in the manual, as is a good section on animation theory, and which GEM calls you will need to make. You can even load a DEGAS format "playfield" (background) to move your sprite on top of.

The second demo is called "Mapmaker." In Mapmaker, you can load up to three sprite files and build maps that are not only bigger than the screen, but can be many levels "deep," with passages leading between levels. By defining your sprites to look like passages, stairs and parts of buildings, you can build mazes, caverns, or anything else you can imagine.

Three sprite files are included on the disk for those (like me) lacking in artistic ability. By pressing a function key, you can switch to screens that show the contents of the three sprite files. You select the sprite you want to use with the mouse, and return to your map to place a copy wherever you choose.

The noteworthy thing about Mapmaker is that it uses so little memory. The included map, which is 500 screens, takes up only about 60K of data. This is because the large sprites which fill the screen take up only one byte each. Atari 8-bit programmers will recognize this: It's similar to using redefined characters instead of graphics modes to set up a playfield. The Mapmaker program even lets you "coarse scroll" over the playfield you've built. Again, you will have to study the source code of the Mapmaker program to see how the scrolling is achieved in such a small amount of memory, but the effort is worth it.

The final demo shows fine scrolling using only half the screen. Studying the listing here also reveals some interesting programming tricks only loosely related to building sprites.

The **ST Sprite Factory** is an excellent programmer's tool. Because the Joystick demo can be used by non-programmers, **SSF** can produce custom animations that are joystick sensitive with nothing more than an artistic eye, a little patience and a copy of DEGAS. The manual gets a little technical in sections (especially the part about sprite collision detection), but then the subject of "bit-blitting" *is* quite technical. The portion of the manual dealing with the actual use of **SSF** is not hard to follow and, overall, the documentation is well written and complete. If you're serious about learning the tricks of your ST, I highly recommend **ST Sprite Factory.** //

David Plotkin, a chemical engineer by trade, currently works as a human resources analyst for Chevron Corp. in San Francisco. His first computer was an Atari 400, purchased in 1980. Currently he uses a 520ST (two monitors) with 1 meg memory and hard disk. His interests include writing and games.

TUTORIAL

Designing an adventure game

Tips from the author of Gateway.

by Michael A. Banks

As a game programmer, you've probably considered developing a text or graphic adventure. Adventure games are the most sophisticated type of computer entertainment and, not incidentally, the game category with the longest sales life. While arcade games come and go quickly (one Broderbund executive pegged the average sales life of an arcade game at nine months), adventures sell and sell and sell, year after year. Adventures such as Zork, Mystery House and Adventureland have been around for over five years—not even Pac-Man lasted that long!

If you have any doubts about the commercial viability of adventures, consider the many software publishers, small and large, that have based lucrative businesses solely on adventure games—among them Sierra On-Line and Infocom. Further, look around at the software publishers who, three years ago, were publishing no adventure games, but who now are jumping on the adventure bandwagon.

Maybe you're aware of all this, and feel capable of creating an adventure system, but have some doubts about your creative ability. The market is, after all, a very demanding one. Adventurers today are too sophisticated to accept freewheeling worlds offering only vague treasures, magic and dragons, and software publishers are aware of this. Gamers want interesting worlds and intriguing puzzles, to be sure, but contemporary adventures must have story value—which means plot, character development and interaction, and more.

As Dallas Snell, author of Penguin Software's The Quest and Ringquest, puts it, "The story, characters, and other 'soft' elements of an adventure are more important than the programming. But this doesn't mean that you can get away with sloppy programming. That is what killed a number of early good stories."

> *"...contemporary adventures must have story value—which means plot, character development and interaction, and more."*

Thus, the reason for this article—I would like to share my viewpoints on adventure game design and construction, as a fiction writer and software designer. I hope to provide information you can use to give your adventures an edge in the marketplace.

Programming.

As a programmer, you should already be aware of the requirements of creating a parsing and database system for adventures, so I won't go into the subject of programming techniques here. However, we can take a general look at how an adventure system is structured.

An adventure program is nothing more than a specialized system for handling words, and changing text displays and the values of variables in response to player input and the passing of time. The system is complex, however. A special kind of database, which contains text descriptions and responses, instructions on how to handle the commands and variables it is given, and how to track moves, time and scoring, is the "heart" of an adventure. The adventure program obeys the instructions in the database regarding changing variables and text descriptions. The program's parser, which is the "front end" of the system and the most visible part of the program, takes player input and processes it into information that the program and database can use. (The parser is the most visible part of the program to the player, and it should understand all parts of speech—including objects, indirect objects, prepositions and modifiers.)

Languages and Computer Systems.

Although a number of software publishers use their own proprietary systems (such as Infocom's ZIL—Zork Implementation Language—and Adventure Interna-

tional's AIL—Adventure Implementation Language), machine language has been the most popular language for adventure developers, due mainly to its speed. Recently, however, quite a bit of adventure development has been done with Pascal and C.

Which machine(s) you will implement an adventure on is something that you'll have to decide for yourself, unless you've developed a machine-independent code such as ZIL or AIL. The machine(s) and language(s) that you work with will be determined in large part by your own skills, but it is worth noting that the Apple II family of computers seem to offer the biggest markets for the adventures, followed closely by the Atari ST, IBM PC, Macintosh and Commodore. A well-designed and -written adventure for any system will find a market, however.

Designing an adventure.

In developing an adventure, you will go through the same creative processes as a scriptwriter or novelist—you are, after all, creating a work of fiction. When I wrote my text adventure, Gateway, for Pryority Software, I found that the entire process, from beginning to end, was remarkably similar to that of writing a novel.

> *"Think of yourself as creating a universe, one in which you make the rules."*

As you put an adventure together, you will find that the process is evolutionary—that is, your adventure will grow of its own accord and, in some instances, even write parts of itself as you develop it.

The Story.

The first step in creating your adventure is to decide upon the type of adventure you are going to write—fantasy, mystery, science fiction, adventure, romance, or whatever. This is important—no matter what the source of your original idea, you should establish some ground rules for the adventure. You cannot allow events to take place in a random and inconsistent manner, as was the case with early adventures; logic and consistency are the major requirements for believability, no matter how strange the settings and situations in your story.

Think of yourself as creating a universe, one in which you make the rules. You can make your setting as weird—or as normal—as you wish. You can have magic, or dragons, or anti-gravity devices, whatever it takes to tell your story. No matter what your story elements are, make sure that they are consistent—don't break your own rules. The genre that you choose will, in part, dictate what can or cannot exist in your story universe (magic has no place in a high-tech science fiction story, for instance), but only to the extent of providing general guidelines. You make the rules, and you must abide by them.

This established, your next move will be to create the storyline, which is basically a chronicling of the major events in the story, in much the same way a writer plans a novel. (It may help you to visualize a storyline as a running account of the story; like telling a friend about a movie you saw or a book that you read.) Or you may wish to create a storyboard, showing scene-by-scene what can happen in your story, as well as what the major goal will be. Once you have the storyline firmly in mind, you can plan actions and their consequences for each scene. (Try to maintain control by allowing the player to take only those actions which might logically be tried, in a real situation. You can waste a lot of memory trying to anticipate every humorous or outré command that a player might input.)

Don't make things too easy for your main character (the player). In well-plotted films, novels and short stories, the story is often made by the limits that are placed on characters. Thus, avoid giving the player overly-simple "outs" for problems, such as magic words. Seasoned adventurers recognize this ploy as a plot device, for the convenience of the author.

Rooms.

Planning your story will allow you to plan your locations, or rooms, intelligently. The genre and the story itself will suggest most of the locations. If you are writing a science fiction adventure, you will probably have an alien world or two, and a spacecraft. For a romance, your setting might be a small town.

Write the descriptions for the opening scene and the final scene first. After this, determine just what locations are required by story events. A romance story, for example, might begin at the home of the heroine, and end at a wedding chapel. In between, you could have such events as the initial meeting between the hero and heroine, a date or two, and an emotional encounter at a train station. These events imply a meeting place location (perhaps a department store), date locations (a theater and a restaurant), and the train station. Other activities imply other locations.

Mapping.

Needless to say, you should map your adventure. As your adventure grows, new locations will suggest themselves (a spaceship control room may imply an engine room, for instance), and you'll be hard-pressed to keep track of all locations without a map, even though you are the creator!

Objects.

Objects are very important—they allow the player to actually do things in your adventure world, to take control of his circumstances. But you should have your locations established before creating many objects—after all, you have to have someplace to put the objects! The objects required will be determined in part by the genre and setting. If you are writing a fantasy quest adventure, it is almost axiomatic that there will be certain valuable treasures such as jewels or rings to find. A jungle adventure will most likely require that your hero have a rifle and perhaps a bush knife.

Major story events will also have a bearing on the objects required. The romance story described earlier could, for example, require an engagement ring and wedding dress.

Depending upon how detailed and realistic you want to make your adventure, you could include all manner of objects. For instance, a department store location could be filled with items—most of them useless to the plot—that the player can pick up or buy. The armory of a medieval castle might offer swords, knives and more exotic weaponry. The number and kinds of objects present depend upon the level of detail you wish to include.

> *"Complicate problems whenever possible—adventurers love complex problems."*

You might also include some objects in descriptions, for the sake of realism, that are not "getable."

Then there are the "red herrings"—objects that seem valuable, but offer no real aid to the adventurer. For example, you might include an empty chest, which

the adventurer will assume will be required to haul some object, but which, in reality, will be used for nothing. This type of element makes for a feeling of uncertainty in your adventure, just as in real life!

Problems and Plotting.

By the time you work out the storyline, locations and objects, you will be ready to start setting up obstacles—the complications that make up the plot. These are the logic problems and puzzles that adventures are famous for. Some problems will be direct, simple obstacles, like the need to find a key to open a door. Others will be tricky challenges, involving not only the player's knowledge of your adventure "universe," but also his knowledge of general or special topics.

Complicate problems whenever possible—adventurers love complex problems. If the player must get a key to open a door, put some obstacles between the player and the key—perhaps the player will have to defeat a ghost to get at the key, and this will require that he get some water from a well to throw on the ghost, but first he must find a bucket. . .or add a double-whammy by putting in two keys—one that is easy to find (the wrong one), and one that is dangerous to acquire. And don't pass up the opportunity to spice up problems by making the wrong solution a fatal one.

> *"Like most works of art, an adventure is flexible and self-perpetuating."*

You may be surprised to find just how much these story elements will interact. Your storyline, and the objects, locations and problems in the story will begin to affect one another very early on, and changes in one element will almost always "feed back" to other elements. As a result, new locations, objects, problems and solutions will suggest themselves. Like most works of art, an adventure is flexible and self-perpetuating.

Descriptions.

While you are going through the processes just described, you should be making notes about what is in each room, as well as the room's general appearance. You will also want to develop one-line descriptions for getable and non-getable objects that you will allow the player to examine.

Room descriptions should give the player an immediate feel for the location—that is, let the player know right away if he is outside or inside, and the general type of location (kitchen, meadow, etc.) he is in. Next, list any "static" (non-getable) objects present, along with any important details such as exits, unusual features (great for giving clues), and the like. The final items in a room description should be anything that has changed since the player last visited the room, and responses to actions that result from the player's actions.

An object name should give the player a good mental image of the object (usually, just a single noun, such as *Flashlight* or *Rifle,* is enough). Use detail in "look" or "examine" descriptions to provide clues: *The end of the rod has a socket in it,* or to explain unusual detail: *This is an antique jewelry case, with a secret compartment in the bottom.*

Text descriptions of any type should be as brief as possible (remember that you are limited by disk space, and the player is not planning on reading an entire novel), but should use effective prose. When writing descriptions, strive to use the most specific, image-producing nouns and verbs possible. For instance, use *German Shepherd* instead of just *Dog,* and *Sauntered* or *Strolled,* rather than *Walked slowly.* Avoid verbosity by cutting out adverbs, long clauses in sentences, and the like. Always write in the active voice, which means that your sentences should be direct in structure—Subject/Verb/Object.

Characters.

Adventures are more interesting (and far more realistic!) when active characters are present. These are known as non-player characters, or NPCs. There are four types of NPCs in adventures—for convenience, we'll call them the Companion, the Room Inhabitant, the Random Event, and the Occasionally Present. The Companion is a character who is always with you (unless you are able to do something to drive him away, or kill him). The Companion may respond to your actions or words, but usually with a fixed set of responses. Normally, a Companion is used to give you a source of information, and perhaps to perform one action in a particular location.

The Room Inhabitant is a character whose existence is limited to one room. Characters of this type are usually guardians of a treasure who must be banished or killed. An example might be a suit of armor that attacks the player whenever he enters a certain room.

The Random Event character is one who shows up at random to do something to the player or steal an object—or, perhaps, to do nothing but enhance the story's realism. (Usually, these characters are not as random as they seem, since they show up only when the player is carrying certain objects or in certain rooms.) A troll who pops out of nowhere and steals a sack of gold is a Random Event character, as is a ghost who is sometimes present to block the player's way when he tries to enter a particular room.

An Occasionally Present character is one who may show up in a particular room at any time. This type of character is usually harmless, and may be a source of help or clues.

Character Interaction.

Allowing the player to interact with NPCs is an excellent way to add interest and heighten realism. Interaction may be as simple as a character responding favorably when the player bows to him, or as complex as the character asking questions. It is also possible to establish variable states in which a character will refuse to help the player if the player was not courteous to that character early in the game. The player may also be able to trade objects for information, other useful objects, or safe passage.

When other characters are not obviously evil, the player will probably want to talk with them. Depending on your system, you may have the command TALK (Name of Character) elicit a response, or you might give the player the ability to ASK a specific question.

Extras.

Mazes.

A maze is a set of rooms which lead nowhere, or from which one can only escape through a certain combination of directions.

To create an inescapable maze, connect the maze room to itself in *all* directions. (The net result will be that the same room description comes up no matter which way the player moves.)

Making an escapable maze is as simple as creating a variable which, when present, will alter the player's location. The variable, of course, is brought in only by the player having fulfilled the requirements you've stipulated. This is far easier—and less memory intensive—than

actually creating a series of rooms with the same description.

Suspense.

If you wish to add polish to your adventure, don't leave out time limits. You can establish suspense by placing a time limit on how long a player has to solve a puzzle or reach a goal. Time limits should be touched off by a player action—the player might pick up a bomb for disposal, and have only so much time or so many moves before it explodes. The actual limit can be keyed to number of moves, or to real time, if your system lends itself to that.

Final words.

Like any artistic endeavor, the process of writing an adventure does not readily lend itself to a step-by-step approach. You will find that what I've had to say here can vary from one project to another, but I hope that I've given you some guidelines to developing your own adventure methodology.

I'll leave you with one last bit of advice—perhaps the most important of all. Do not attempt to write an adventure without knowing the field. Look at the current best-sellers as well as the "classics," and listen to what adventurers are saying about the games. Then, armed with knowledge of what the gamer wants, create a story that meets the needs of the market. ///

Michael A. Banks is the author/designer of Gateway, a text and graphics adventure for the ST, published by Pryority Software and Action Software.

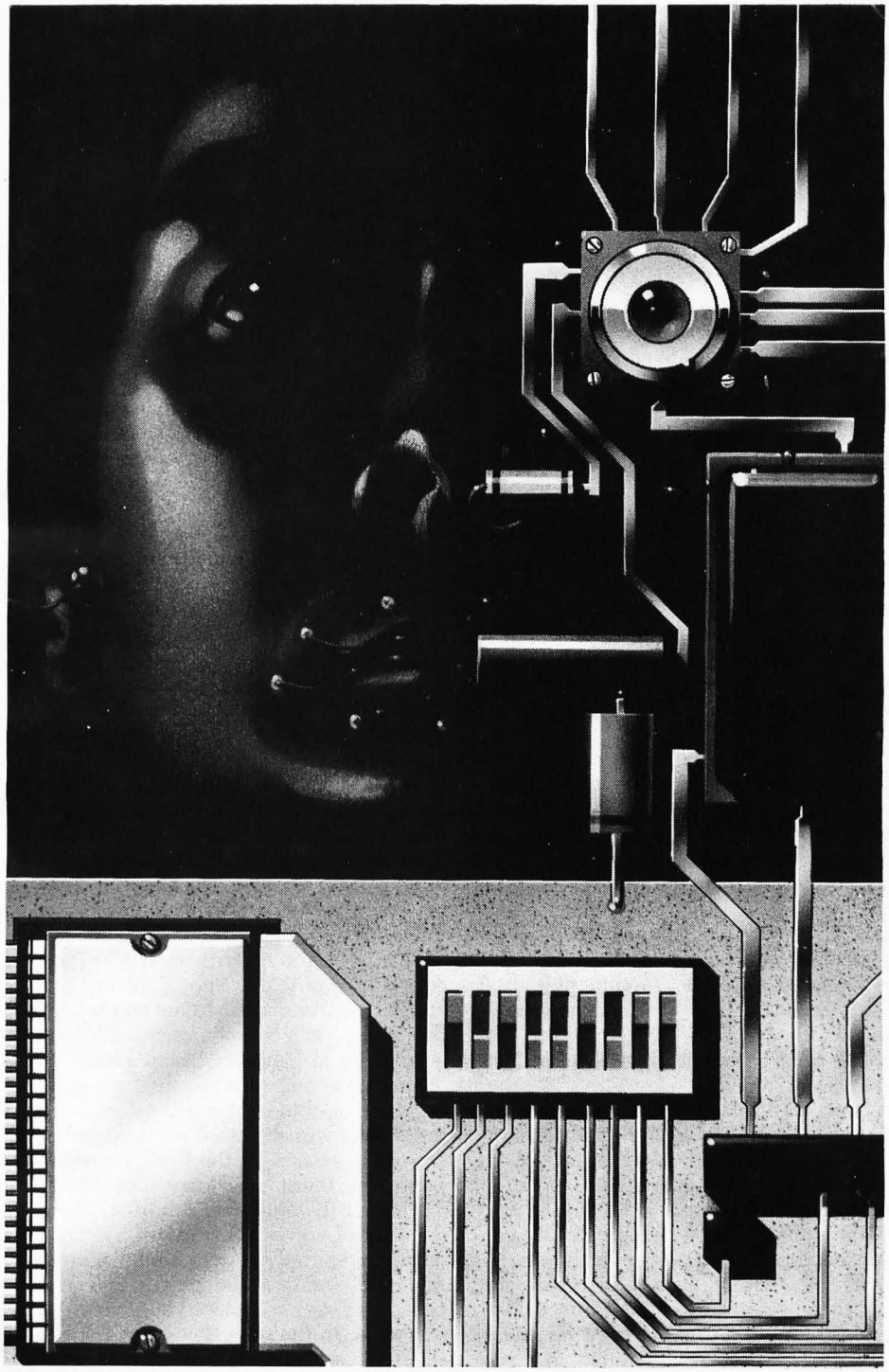
Banks also writes science fiction novels (among which is The Odysseus Solution, *from Baen Books), and non-fiction books (*The Official Guide to Delphi, *Brady Books, and* Second Stage: Advanced Model Rocketry, *Kalmbach Books).*

He currently has three novels, one juvenile book, and eight non-fiction books in print. A full-time writer for four years and a computer user for six, Banks resides in Ohio, with his wife, daughter, son, and no cats.

SPECIAL FEATURE

The perfect computer game

From games of old to speculation on games yet to unfold



by Orson Scott Card

You remember Pong, don't you? This was back in the mid-70s, so some of you were still ankle-biters—but you old people, now in your late twenties and thirties, *you* remember.

You put a quarter in, you and a friend each grab a dial, and you control your paddle as you volley a little square "ball" back and forth across a screen.

Hard to imagine now, of course, but at the time it was an extraordinarily captivating game. A quick, exciting contest you could play with a twist of the wrist. No chasing a runaway ball. Ping-Pong for couch potatoes.

It was just the beginning. Breakout was a quantum leap forward—though it achieved "color" with cellophane strips across the screen! There were race-car games. Falling dominos. A bunch of extraterrestrial morons marching downward to destroy us in Space Invaders. Then Asteroids, which for the first time gave us complete freedom of motion, as we blasted and dodged our way through a devilishly crowded outer space.

Ritual

What was their magic, that they sucked us in? They were electronic pinball machines: we didn't so much play *them* as they played *us*. It's like we were paying a quarter for the privilege of having a machine teach us how to catch little cathode-ray dots, just the way pinball players get

trained to flip the paddles without stopping to think. Thinking is too slow for games like that. You have to train your reflexes to play the game without your mind hooked in.

Don't get me wrong. I'm not criticizing those early games at all—or pinball, for that matter. I'm an avid pinball player precisely *because* of that rhythmic, powerful, mindless involvement. Ritual of that sort is a vital part of all games, and the early video games were a hypnotic new way of losing yourself in the game. I remember playing Breakout so long that the angular movements of the "ball" would continue to replay themselves inside my head whenever I wasn't paying attention to anything else. And swarms of Asteroids kept homing in on me when I closed my eyes after an hours-long marathon.

Story

Not everyone likes to end up in a hypnotic trance, however, and the second generation of video games introduced a whole new angle. Pac-Man's idiotic chomping, pursued by colorful ghosts, opened up whole new worlds of cuteness. Donkey Kong was even cuter, but then more serious games like Venture and the marvelous Joust proved that video games might actually grow up into a serious kind of storytelling.

It wasn't just a matter of getting better graphics, but better graphics made it possible. Video games could now tell stories whose flow wasn't absolutely predetermined by the design of the game.

In all the previous video games, you took whatever the game threw at you and dealt with it as best you could. But in Venture, you could choose which dungeon chamber you'd enter, and the open floor space allowed many routes to the treasures. In Dig-Dug, you made underground tunnels that determined the path your opponents would take.

It was no longer just a matter of letting the game train you. You had some freedom of choice.

The game played differently with different players. You could play with *style*. You became, to a small degree, both an actor in the story and co-author of the script.

Another important result of improved graphics was that gamewrights were creating interesting worlds. In Joust, you could actually ride a flying ostrich over and under islands floating in the air. In Crystal Castles, it seemed there was no end to the deep array of three-dimensional mazes. It was worth playing a game just to explore, to see what came next.

The ritual element of game-playing was still strong in the best of these games—you had to be quick and smart (and, I began to think, it didn't hurt if you also happened to be 12 years old) to find out what was around the next corner. But with more realistic animation, more freedom of action, and increasingly deep and interesting worlds to explore, video games were becoming more like stories.

Playing a video game was getting to be just a little bit like improvising a part in a play. There was a glimmer of a possibility that this stuff might actually become creative. An art.

Bringing the games home

But there remained—and remains—one barrier that arcade games just can't get past: they have to make money. And the way they make money is to take your quarter and kill you off as quickly as possible, while still enticing you to play again.

If you could play for hours on a single quarter, the arcade would lose money. If, to play for hours, you had to keep pumping in the quarters, the arcade would get rich.

One strategy, of course, is to get you to put in quarters in mid-game and continue playing from where you were. That was fine in Dig-Dug and Ghosts and Goblins, but immediate death still loomed over you.

Fortunately, there's a place where you can stay for hours and hours without putting quarters into machines: home. (You have to pay rent, but that doesn't count.) And more and more homes began to have machines in them that could also be used to play games.

In fact, many home computers were bought *in order* to play games. I know that's why my first Atari 400 found its way into my house, back when it cost $600 just to set up the system with a cassette player and 48K of RAM. I bought every cartridge, then upgraded to an 800 with a disk drive—at a cost of more than $2,000—in order to play disk-based games.

And I couldn't even pretend I was buying these machines for work. All my word processing was on a Z-80 Altos. My Atari was there because of the worlds it promised to me, the stories I knew it could tell.

I wasn't the only one. Admit it. There are a lot of you out there with STs right now, who started out with 8-bit machines like I did, and even though you told your parents or your spouse or your friends or the dealer in the store that you were getting the machine to do your taxes or write letters or keep your mailing list or balance your checkbook, nobody was fooled.

Heck, maybe you actually *did* those things. Maybe you keep the grandma of all databases on your ST. Maybe you've got the Great American Novel asleep in the megabits of RAM. But you know and I know that your ST isn't really alive unless it's playing a game.

And at home, you can play forever. Time is no longer a limit on the game. Gamewrights don't have to try to kill you off as quickly as possible. They can create games that are thoughtful, games that have limitless possibilities, games in which you stay alive for hours, days, weeks. Games that you don't just play—you live them.

Puzzles

One kind of game has never been in the arcade: the text adventure. Since the early days of Adventure, as thousands of players figured out that you have to bring the bird and turn it loose to frighten away the huge serpent, text games have posed us with puzzles that took many frustrating, wonderful hours to solve.

The puzzle is the third aspect of game-playing, after ritual and story, and while it has had a minor role in arcade games, the puzzle has always been at the heart of text adventures.

Text adventures rarely attempted to use computer graphics at all. By using words, far more complex and detailed worlds could be created in limited memory.

Where Scott Adams's pioneering BASIC adventures tended to focus almost entirely on puzzles, which had to be solved in a particular order, Infocom virtually reinvented the text adventure, not just because their parser could handle something closer to English, but because their worlds could be explored in almost any order. The player was in charge of far more than in most text adventures.

Infocom has brought us, step by step, to amazingly deep games that can be explored almost at will.

Is the game a story?

But when they call their games "interactive fiction," is it true? Are these games the real computer storytellers?

I don't think so. Since every possible outcome of every possible situation must be expressed in language that must be programmed in the computer from the beginning, there is a limit to the number of possible endings to the story.

Now, this is hardly a drawback to calling it fiction, is it? After all, the average

novel has only *one* outcome, and nobody complains about *that*. (We may complain about what the ending was, but not that it had just one.)

But the computer game and printed fiction are two different arts, with different strengths and weaknesses. While the reader can't choose what should happen next in traditional fiction, can't even decide the order in which things happen, there is a feeling of truth.

Yes, truth. Even though fiction is designed to be lies, we still think of what "really" happened in our favorite books, and resent it when the movie version changes them. There's one true story. It isn't negotiable.

The strength of the computer text adventure is that the story is negotiable. The weakness is that it isn't as real. There's no "true" version.

To put it another way, while it is very important that Frodo threw the One Ring into the cracks of doom in *Lord of the Rings*, it is not all that important that Fred Bliss in Piskatoxee, Arkansas finally got the last of the clues to allow him to get into the hidden final room in Dog Star Adventure.

And that's fine. Computer games aren't books, and books aren't computer games. Computer games don't get "better" the more closely they resemble books. They get better the more fully they exploit the possibilities of computers.

The perfect computer game

Text adventures are terrific, but they're a dead end. They can't get any realer than they are. There's a limit to what words can do.

In the days when 48K and a single disk with less than 100K of storage and four colors on the screen at a time meant you had the *best* home computer, text adventures could make much better and realer worlds than graphics games.

But that just ain't so anymore.

Computers are finally getting so much speed, so much RAM, so much fast disk space, so many on-screen colors, that realistic animation and vivid settings are not only possible, they're almost commonplace.

And you're reading this magazine because you own—or wish you owned—one of the best graphics machines, period.

There are brilliant games in your ST just waiting to get out.

I'm not going to point to games on the market right now. Few games have yet appeared that really use the power of the ST—most are designed first for other, lesser machines, then ported over, almost guaranteeing that the ST's power will go largely unused. Of those actually designed for the ST, those I've seen are still pretty much in the stage of trying to dazzle us with the machine's tricks instead of coming up with any great depth of gamemaking. Besides, by the time this issue comes out, there'll be better ones available than any I could name right now.

What I'll do is point in the direction that games can, should and *must* go. Judge for yourself which games come closest to measuring up.

A world you can live in

The perfect computer game will have a *real world*. Like the best stories, the hero—the player—won't come out of nowhere, riding on his charger, to have a couple of adventures and then split. Instead the hero of the perfect game is closely involved in the world around him. He has jobs to do. He has limitations.

The perfect computer game will have a *wide world*. You can explore for hours, just going here and there, and still not discover the limits. There are towns or planets or castles or caves that you will not discover until the tenth time, the hundredth hour that you play.

In fact, the perfect game will be capable of generating new worlds, so that it will be impossible ever to run out of new places to explore.

The perfect computer game will have a *deep world*. The hero won't be the only human being in it. There will be other characters that he has to talk to, query, help, fight, or simply enjoy. And they won't be cardboard characters who have only one thing to say, who always stay in the same place; they will be living their own lives, going from place to place on their own schedules, and responding to the hero according to their own needs. In short, each character will be living out his own story, which will be different every time the game is played.

A moral dimension

The hero of the perfect computer game will have a *soul*: the soul of the player. While there will certainly be built-in puzzles and challenges, and the player will have to learn the rituals of play, the story itself will be an honest reflection of the player's own character.

Do you play violently? Then you'll find yourself surrounded by violent characters, both friends and foes. Do you want to settle down somewhere and simply live? Some adventures will come to you, yes, but you'll also have a chance to build something that lasts, have some achievements that don't involve killing people or getting treasures.

I'm not saying that find-the-treasure and kill-the-aliens games are somehow bad. Not at all. What I'm saying is that with virtually limitless memory and speed and graphics and sound, the perfect computer game should allow you to play it in many different ways, and respond to you according to the type of game that your own play determines.

You should be able to have a completely different playing experience than anyone else. Like the best stories, the perfect computer game will help you better understand who you are. That's how the computer game will become 'true,' as the best fiction is true.

It takes a machine

Now that I've described such a game, is it really possible?

Yes, I think so. And it won't take a whole new kind of computer. The perfect computer game wasn't possible on an 8-bit Atari. It *is* possible, right now, today, on the ST.

Now, I'm not calling for perfect animation, Disney-quality figures moving around. Though I can envision the possibility of feature-length computer-generated animation that will be as realistic as real actors, that isn't what the home computer is for—that's the kind of thing somebody else will make and sell you on videotape.

The ST already has good enough animation to give you a player-figure and a marvelous world, not photographically realistic, but close enough that it makes no difference. It will be marvelous to look at, and that's all that's really required.

What the memory and ever-growing disk capacity should provide is not overdone graphics, but something more for the graphics to be *about*. More things to do. More people to meet. More places to go. Broader, deeper, realer worlds.

And above all, more possible responses to your character as you play.

There is one piece of hardware that no game has yet used that *will* make a difference, though: the compact laser disc. If all the graphics images for each part of the world, if all the basic data for hundreds of characters, if all the facts and puzzles and possibilities were dumped willy-nilly onto a compact disc, there'd still be room left over for even more imagination. The gamewright would have, at last, no limit to the depth and breadth of

his world. (And, with the perfect game to sell it, the computer-laser interface would be commercially worth producing —cheaply.)

Then your regular read-write disk would be used for storing only the information *you* generate in the course of a game—a complete history of all your actions, for instance.

Imagine that: playing a magnificent game, and then giving the computer a command that plays the whole thing back, like a movie, like a story, so you can watch the adventure that you helped the gamewright create.

What took you hundreds of hours to play might take only a few hours to replay —a full-length feature film.

At last the collaboration between gamewright and player would be complete. The gamewright would truly have prepared a vast movie set, with other actors and thousands of extras, hundreds of places you can go; and then, using the world he has created for you, you will tell your own story.

And if you think you have a terrific game, you could make copies of your game disk and pass them around. Upload them onto Delphi or other on-line networks. Since a laser-disc-based game has built-in copy protection, there'd be no limit to the copies you could make of the record of your *own* play. Instead of playing once and having nobody else see what you achieved, you could have an audience. You could be a true artist, a performer; the Olivier or Hepburn of the perfect game.

The serpent in paradise

Yeah, that all sounds great. But the fact is, once the perfect game is made, we're quickly going to discover that not everybody is a perfect player.

The way games are now, either you achieve the goal or you don't. Either you win or you die.

But in the perfect game, there'll be many, many ways to win. Many lives to live. While the game you play back when you're through may be rich with fascinating, important adventures, it may also be unspeakably repetitive or dull.

All these years you've been playing computer games and reading reviews of them. And you've loved it when some incompetent game designer got sliced to ribbons by a clever, merciless critic.

But now, as you take your place as a player of the perfect computer game, offering your game for others to see, the reviewers wouldn't just review the gamewright's work.

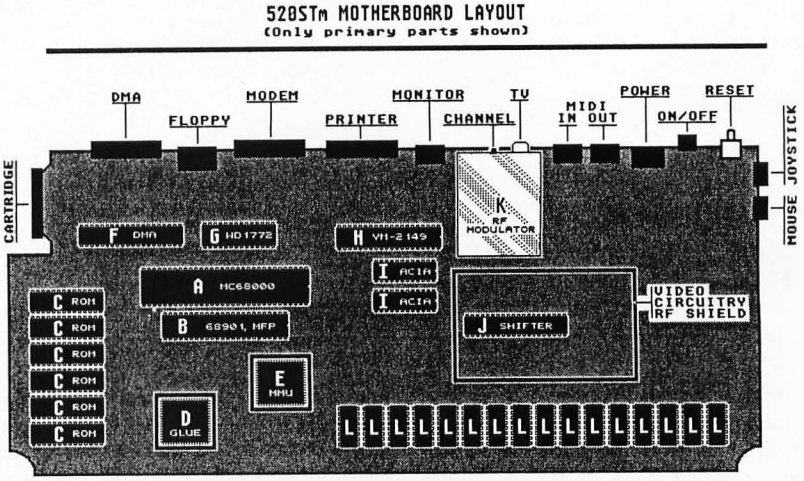
In fact, *I'll* almost certainly be one of the critics. I just can't wait to watch you play. ///

Orson Scott Card is the author of Ender's Game, *which won the 1985 Nebula and 1986 Hugo awards. It was followed by* Speaker for the Dead, *which won the 1986 Nebula and 1987 Hugo awards. Card's interest in computers and computer games is very evident in these books. Card is currently working on a series of novels called* The Tales of Alvin Maker. *The first book in the series,* Seventh Son, *was released last July. The second,* Red Prophet, *will appear this spring. Card lives in Greensboro, North Carolina, with his wife and three children.*

FEATURE TUTORIAL

Step 1
THE HARD FACTS

A look under the hood—part 1



by *Maurice Molyneaux*

Thus far, **Step 1** has been geared to explaining the ST systems in a manner intended solely to give the beginning user some idea of what the system is capable of doing, and how to use it. I certainly intend to stay in that vein, but this month we're going to take a detour from the norm and briefly tour the ST hardware. . . and I don't mean a description of the ports, etc., as appeared in the very first **Step 1**. This time we're going to go deeper. . . into the very bowels of the beast. Next month, when we continue this discussion, we'll go into further detail.

Before you beginners cast this aside, feeling I've betrayed you and am pandering to techies, hang on. There are reasons for covering this subject. First, many ST owners ask why their machines can't do this or that, not knowing a lot of the reasons are in the hardware. Second, things *can* go wrong with a system. If you know something about the hardware and how it works, it's easier to fix simple problems on your own, or tell a technician exactly what's wrong. For those of you interested in a "How-NOT-To" lesson on computer repair, read Matthew Ratcliff's "ST Nightmare Repair" (**ST-Log** issue 12), and learn from *his* mistakes. Third, it's not a bad idea to understand just *how* a computer works, because then you can better appreciate its strengths and limitations.

Finally, I don't want a bunch of letters from techies castigating me for my brief—and rather nontechnical—descriptions. This article is meant to give the average user some idea of what goes on inside his or her computer. It's *not* a detailed analysis of ST architecture.

The grand tour

Now, we're going to pop the hood of a 520ST, and take a look around. 1040ST and 520STfm owners can rest assured that everything described here is the same in their machines, though the exact placement of parts may differ. For this tour, there's no need to open up your ST, as that will void your warranty (if it's still in effect). Photographs are provided to show you what I'm talking about.

Under the ST's pretty gray top is the keyboard. The keyboard is not attached to the top part of the system case, but sits on supports in the lower half. On the bottom of the keyboard are a cluster of electronic components and a chip ("chip" is the nickname for an "Integrated Circuit" or "IC"). The chip is a small processor IKBD or Intelligent KeyBoarD controller. It provides the main processor (the brain) of the ST with information related to the state of the keyboard, real-time clock, mouse and/or joysticks.

One feature of this chip is that it contains the real-time—or time of day—clock (set from the control panel accessory). The time and date set in this clock are "stamped" on a file entry when you save it to disk. If you always display directories on the desktop using icons, select "Show As Text" under the view menu, then open a directory window. If you can't see the date and time, resize the window horizontally or click on the "full screen" button in the window's upper right corner. Now you should see that each filename has next to it its size in bytes, in addition to the date and time it was saved.

The clock in the IKBD, and every other component, loses all power when you turn off your ST, so the time and date settings are lost—unless you have a Mega ST, or have added a battery backup clock to your system. On system power-up, the clock is set to a default time and date, which can be changed from the control panel. If you aren't in the habit of setting the clock, don't expect the correct time and date to be stamped on your files.

The IKBD's main functions are monitoring the keyboard and the mouse/joystick ports. Whenever you use the mouse, fiddle with a joystick, or press a key on the keyboard, the action is noted by the IKBD, which passes the information deeper down into the system.

The IKBD is a chip manufactured only for Atari ST systems. Such specialized parts are called "custom" chips because they're custom built for a given system. Other chips are considered "off-the-shelf" because they can usually be purchased at an electronics store. If you need to replace a custom chip, you'll have to contact Atari or an ST service center. In this article, all custom chips will be indicated as such. All others are off-the-shelf.

The keyboard is connected to the depths of the ST by a tangle of colored wires, which descend through an opening in the metal shield below the keyboard, and are fitted into a slim connector. The connector must be pulled up and out in order to go deeper into the system.

Now that we've done that, we're faced with a big sheet of dull silver metal. This is the ST's RF shielding. RF stands for Radio Frequency, and shielding means what the name implies, only twofold. This shielding prevents outside radio signals (radio, TV, etc.) from interfering with your computer's operation, and vice versa. This shielding isn't a complimentary feature—the FCC (Federal Communications Commission) is *very* serious about radio interference.

The shield consists of two halves, top and bottom. To look inside, we must remove a few screws, then untwist a bunch of metal tabs—one or more of which may be soldered down. Once this is taken care of, we can lift the top portion of the shield and. . .*Behold!* Here is the heart of the ST. Not much to look at, but this is what makes it work.

Under all the pretty gray plastic and white keys we find a printed circuit board: A big flat panel, covered with electronic gizmos, sprinkled with black chips, and rimmed with port connectors. (There's also a power supply and floppy drive in the 1040ST, 520STfm and Mega ST, as well as a fan in the Mega). This board is the center of the entire computer system. Because it's the main component, it's called a "motherboard." Consequently, add-on boards are called "daughterboards," since they spring from the motherboard.

Here's where we stop and look around. I'm going to briefly introduce you to the parts of the system that make your ST an ST, and not a C-64 (shudder).

All parts are labeled with letters, so the following descriptions will correspond with the marked components.

Before we begin the tour, let me explain the difference between *serial* and *parallel* interfaces, as some parts will feature these. A serial interface is one in which data can travel in only one direction at a time, one bit at a time. A parallel interface is one where data is transferred a *byte* at a time, by sending all 8-bits simultaneously, in *parallel*, a number at a time—side by side, so to speak. The MIDI and modem ports of your ST are serial interfaces, while the printer port is a high-speed paralle interface.

The big brain

A — This is the biggest and most important chip in the ST. It's the brain and heart of the entire system: the microprocessor. A microprocessor receives data, processes it, then outputs data and instructions for other parts of the computer system. The ST's microprocessor is a Motorola MC68000. The 68000 is a fast and powerful 16-bit microprocessor. When we refer to bits, we're referring to the number of data bits (on/off signals) the processor can handle at any given instant. The 68000 can handle *16* data bits at a time (cycle). The 6502 processor used in Atari's 400, 800, XL and XE models can only handle *8* bits of data each cycle. This means that with each processor cycle, the 68000 can process *twice* the number of bits a 6502 or other 8-bit processor can.

Different microprocessors, and various versions of the same models, have different "clock speeds," which refers to the number of cycles per second. The Atari 8-bits' 6502 has a clock speed of 1.79 MHz (1.79 million cycles per second), while the ST's 68000 runs at a dazzling 8 MHz (8 million cycles per second!). This means the 68000 runs nearly 4.5 times faster than the 6502, and can address twice the number of data bits per cycle.

So, in raw processing terms, the 68000 can handle close to 9 times more data per second than the 6502! Of course, this *doesn't* mean an ST is 9 times as fast as an Atari 8-bit, because there are a lot of other factors that contribute to overall performance. For example, the 68000 is a 16-bit chip, but many of its peripheral ICs are 8-bit, and cannot move data as quickly as the 68000. Further, some chips "hog" the address and data buses (buses are accesses to RAM addressing and data lines) and keep the 68000 from using them. This is called "cycle stealing," and chips that do this can

slow down overall speed when they are in use. Therefore, clock speed alone doesn't mean a whole heck of a lot. If *nothing else*, you should now have a better idea of how the brain compares to that of the older 8-bit systems. (I'm not knocking them, honest!)

While the 68000 has a 16-bit data bus, its address bus is 24 bits wide. Internally (inside the chip), its own data and address registers are 32 bits wide. (Registers are places where the chip stores values.)

There are eight data and nine address registers in the 68000. I won't go into what this means, but I can tell you that this is the reason the 68000 is often referred to as a 16/32-bit microprocessor. And, if you've ever wondered just what "ST" stands for, it refers to the Sixteen/Thirty-two-bit processor. Because of this design, the 68000 can directly address up to 16 megabytes (16,384K!) of memory. Conversely, the 8-bit 6502 can directly address only 64K! To use more than this, a 6502 must "switch" between different banks of memory (as does the 130XE).

Finally, note that the 68000 is soldered *directly* to the ST's motherboard. This is called "surface mounting" and is usually carried out only on parts with high reliability and very little chance of failure. Sometimes, though, parts are surface mounted to keep costs down, in addition to improving reliability, because a soldered down part can't come loose.

Many other chips are set in sockets, which means their pins (legs) are pushed into a special receptacle. Socketed chips can be popped out with little effort. Although more expensive than surface mounting, socketed chips are much easier to replace if damaged, or if an upgraded chip becomes available. I will denote the status of each chip in its description, but please keep in mind that variations do occur, and I cannot speak for parts in the Mega, as I haven't yet seen the guts of a production model. The IKBD chip, discussed earlier, is socketed.

Pardon the interruption . . .

B — The handling of interrupts is a crucial part of any computer system based on a 68000. In the Atari, this is accomplished through the use of a Motorola 68901, also known as the Multi-Function Peripheral (MFP) chip. It's called "multi-function" for good reason: it handles quite a number of tasks. Its primary job, however, is handling system interrupts. For example, in some systems, the processor monitors things like the keyboard and joyports a number of times each second. This uses valuable processor cycles, which could be better utilized in doing calculations, etc.

In a 68000-based computer, the MFP stands in "front" of the 68000, intercepts all interruptions from other parts of the system, carries out various interrupt control measures, then passes them on to the 68000. In this way, the 68000 merrily goes about its business, not having to monitor other parts of the system. When something requires the 68000's attention, the MFP sort of "taps the processor's shoulder" and makes it aware of the event. In other words, an *interruption*.

The MFP's job entails more than that. Its features include an 8-bit parallel port, a built-in serial interface and the ability to handle 16 possible interrupt sources. Further, the chip contains four timers. One of these, the 200-Hz clock, is used for all of GEM's timing. The chip itself runs at a 4-MHz clock speed. The MFP chip is surface mounted, and not easily replaced.

ROM wasn't built in a day

C — This bank of six chips is the ST's ROM, containing all the code for TOS and GEM. These chips, more than anything else, are responsible for making your ST work the way it does. These chips are socketed, and can be easily replaced if need be. The total ROM of all the chips combined adds up to 192K. Because they contain the ST's OS (Operating System), they are, of course, custom chips.

Sticky stuff

D — This is the system's electronic "adhesive." The chip serves to interface various components and works the peripheral chips. Due to its importance in holding everything together, this chip is aptly named Glue (also GLU, for General Logic Unit). Glue communicates with just about every component in the ST, and keeps watch over all operations.

In many other systems, a complex array of smaller parts handle the functions of Glue. Because of this, it's difficult to briefly describe just *what* Glue does. Since Atari consolidated all these functions into a single chip, the cost of manufacturing the computer is kept down, which helps make it as affordable as it is. Glue is seated in a 64-pin socket, and often held in place by a metal clamp. It's a custom IC.

I remember!

E — This is the ST's Memory Management Unit, or MMU.

The MMU's primary job is to interface the processor with the system's RAM (memory). It also works with the system's video chip to produce the video signal for your monitor—giving it access to the RAM which contains the data for the screen. Further, the MMU works in conjunction with the system's DMA chip, for passing data between RAM and the DMA/hard disk and floppy disk ports.

This chip handles "who gets what RAM and when," thus keeping your desk accessories and Neo-Chrome from trying to simultaneously occupy the same RAM. It is primarily the MMU that limits current ST systems to addressing only 4 megabytes of RAM, although the 68000 is capable of addressing up to 16 megabytes. MMU runs at a blinding 16-MHz clock speed. Like Glue, it is a custom chip, is seated in a 64-pin socket, and usually held in place by a small metal clamp.

Quick and direct

F — This is the ST's DMA, or Direct Memory Access chip. Its job is to oversee the floppy disk controller chip and the hard disk—or anything else plugged in through that port, like a CD-ROM or IBM emulation box. This chip is tied into the processor's data bus, allowing very fast data handling. DMA can eliminate the need for data to be moved through the 68000 when it's being transferred between a peripheral device and the ST's memory.

When dealing with the floppy drives, the data transfer rate is so slow that the 68000 has no trouble dealing with direct transfers. However, because the DMA/hard disk port has the *potential* to move data at a rate of 8 megabits per second (no device, at this time, uses this port at that speed), this could cause severe problems for the 68000! In fact, when dealing with a hard disk or other high-speed device, the 68000 uses the DMA chip to send "pause" commands to the device, so it will wait while the 68000 processes the data it has.

The DMA chip—because it works with both floppy and hard disk interfaces—is a likely source of trouble if you begin having problems with one or both. I recently had hard disk and floppy access problems, which were partially due to a bad DMA chip. Fortunately, this chip is socketed, and easily replaced. DMA is a custom chip.

The wild, wild west

G — This chip is a Western Digital 1772, a component with one task: control the system's floppy drives. This chip features built-in drive motor controls, and supports single- and double-density formats. Atari chose this chip, despite its limitation of being able to use only *two* floppy drives, because it offered a *complete* floppy controller in a single chip. The WD1772 is socketed.

Sounds complicated

H — This chip is a prime example of Atari's attempt to maximize the effectiveness of each component of the ST system and keep the cost down. The chip, a Yamaha YM-2149 PSG (Programmable Sound Generator), not only serves the task of adding audio to your ST's video, but it also features two bi-directional 8-bit parallel ports.

Atari chose the YM-2149, not only because it was an inexpensive and reliable sound chip, but because they also got those parallel interfaces in the bargain. This reduces the number of overall parts, as this chip handles both sound and the printer port. The same component is also made by General Instruments, and called the AY-3-8910.

The YM-2149 can generate sounds on three separate sound channels, over a range of between 30 Hz and 125 KHz. The chip also features a noise generator for distortion. While not as capable as a custom sound chip would be, the YM-2149 does provide decent sound. Atari has hinted of a super sound chip called AMY, but whether she will ever emerge from the vapor is doubtful.

In many STs, the YM-2149 is surface mounted, but on the Revision H motherboard I recently obtained, it was socketed. If you have problems with your printer port, this chip could be to blame.

Keys and music

I — Here, we have a rare case of two identical components, a pair of 6850 ACIAs. ACIA stands for "Asynchronous Communications Interface Adapter." The ACIAs are serial interface operators and, while identical, each of the two chips has a special job. One chip is interfaced to the IKBD and, therefore, communicates with the keyboard, mouse/joystick ports, and runs the real-time clock. The second one controls the (serial) MIDI interface.

Many people do not realize that, while the MIDI ports are primarily intended for use with electronic musical devices, they are serial data ports. Thus, they can also be used to network other types of hardware together. The MIDI-Maze game by Hybrid Arts uses the MIDI interface to connect multiple STs together for a round of game action. These

two chips are surface mounted.

You look mah-velous

J — This chip is concealed under a small RF shielding box on the motherboard, and is surrounded by video circuitry. This is the video chip, called Shifter, which converts screen information in RAM to signals necessary for a monitor. The 520STm, 520STfm and 1040STfm models (which feature TV output) have additional circuitry for an RF modulator (which converts the video signals into those used by a television). When such a modulator is present, additional pins are connected on the ST's monitor jack, providing a *composite* video signal—which is what's utilized by monitors used with Atari 8-bits, etc. The composite signal is not available on STs *without* an RF modulator.

Shifter contains 16 "palette registers," which hold the values for the colors you use when in color mode. Only four of the registers are used when in medium resolution, but all 16 are used in low resolution. If you wonder why your ST is normally limited to only 16 colors at a time, look no farther than Shifter.

Note that Shifter is a socketed chip. In many cases, RAM expansion boards for the ST will plug into the Shifter socket in order to gain access to RAM address lines. A bad or poorly seated Shifter chip is often the root of problems relating to screen display. Distortion or garbled images are often signs of a problematic or loose Shifter. Shifter is a custom IC.

K — This is the RF modulator, present in all 520STs manufactured after 1985, and a tiny handful of 1040STfms released last summer. The shielded box here contains the RF signal converter, and a channel select switch for choosing which TV station you'll be using. To add such a unit to an ST can be complex, because usually the associated video circuitry for RF is missing.

Pretty chips all in a row

L — This row is *very* important. The 16 chips here (32 in a 1040ST) are the system's RAM. This is the memory where all the important information (like text, graphics, etc.) are stored and manipulated. These chips are known as 256x1K chips, and 16 of them provide 512K of RAM, while 32 provide 1024K (1 megabyte). In a Mega ST4, there are 16 1-*megabit* RAM chips, providing 4 megabytes of memory. These chips are surface mounted, and there's no simple interface for plugging in additional RAM (though the processor bus in the Megas might change this).

Fast mover

No identifying letters here. You won't find this part in your 520 or 1040, but you Mega owners will. In the Mega ST there's an IC that we've been waiting on for a *long* time: the Blitter chip. The Blitter is a custom microprocessor whose primary task is moving blocks of memory at high speed. STs without a Blitter chip accomplish their blitting (a method of moving blocks of graphics data) entirely in software. The Blitter chip takes over these tasks in many cases.

Since the Blitter is designed specifically for the kinds of memory transfers required for blitting, it's usually much faster at the job than software alone. Moreover, since the Blitter is a block memory mover—and not a dedicated graphics IC—it can also speed up other memory transfer intensive functions.

The Blitter is not always used and can, in fact, be disabled from the Mega ST desktop (options menu). The chip is reportedly a 64-pin IC identical in appearance to MMU or Glue and seated in a similar socket.

Drop the hood

I hope you never have cause to refer to such information for repair reasons, but if so, this might be of some help in diagnosing the problem. Now we've covered the hardware details, but we're not quite finished. Next month I'll tell you how all of this works together (in a general way) and explain *why* the ST can and can't do various things (why it can't run Atari 8-bit software, etc.).

Oh, and before I go, let me again urge those of you with questions or comments to write to me in care of this magazine. If there are topics you'd like to see covered in **Step 1** that haven't been mentioned, or barely touched upon, let me know. I'm always anxious to find out what *you*, the end-user, need help with. *Ciao!* //

Allergic to all things Commodore, Maurice Molyneaux is an author and artist, who—when not writing articles for ***ST-Log****—continues to struggle with a recalcitrant 8-year-old science fiction novel, paints, illustrates and also uses his ST for "every conceivable task." His interests include classic cel animation as well as the computer variety, and he draws the meanest* Star Trek *pictures on microcomputers. His Delphi username is MAURICEM.*

COMPUTE!'s Technical Reference Guide ATARI ST Volume One: VDI

by Sheldon Leemon
COMPUTE! PUBLICATIONS, INC.
P.O. Box 5406
Greensboro, NC 27403
(919) 275-9809
343 Pages $18.95

by Charles F. Johnson

Sheldon Leemon (DRX on Delphi) has been an active supporter of Atari computers for years. He has published articles and programs in just about every Atari-specific magazine, and caused quite a stir in the Atari community a while back when he wrote an article entitled "The Myth of ST Superiority."

With the intention of "debunking" claims made by some ST supporters, he compared the ST to the Commodore Amiga—not always in the ST's favor—and faced a lot of criticism, even outright animosity from some quarters. One of the points he made at the very beginning of the article, however, was that he liked and used *both* machines, and preferred the Amiga by only a slight margin.

With this in mind, we shouldn't be surprised to find that Mr. Leemon is the author of a book—indeed, a series of books—about programming the ST. COMPUTE! Books is publishing the series that can now be found on the stands (I found mine at Walden Books in Los Angeles). In the typical long-winded fashion of COMPUTE! Books, the title is (take a deep breath), ***COMPUTE!'s Technical Reference Guide, ATARI ST, Volume One: VDI.*** Gasp.

Since I don't want to type this every time I mention the book, I'm going to call it ***COMPUTE!'s VDI*** for purposes of this review. For $18.95, you get 343 pages of documentation on the VDI graphics functions built into the ST's operating system. So how is it? In a word, excellent.

This book is so good that I now use it instead of the Developer's Kit docs or the Abacus GEM Programmer's Reference when I need to know something about the VDI.

For the beginners among us, VDI stands for "Virtual Device Interface." The VDI part of the ST's operating system is supposed to provide a uniform way to access all possible graphics peripherals. Using the VDI, a programmer can employ the same graphics commands to produce output on any hardware device (printer, plotter, monitor screen) without necessarily knowing the down-and-dirty details of each device's operation.

The first part of ***COMPUTE!'s VDI*** (before the appendices, which take up almost half of the book) contains chapters covering "Setting Up the Graphics Environment," "Drawing Points and Lines," "Color," "Filled Shapes," "Drawing and Manipulating Image Blocks," "Text," and "Input Functions." Each chapter contains several short demo programs to illustrate the points covered. To minimize typing errors, the examples are reproduced from printouts, not typeset.

The basics of programming the VDI are covered in terms of BASIC (ST BASIC—ugh), C and assembly language. For C and assembly programmers, a program "shell" is listed, taking care of all GEM initialization procedures; you simply key in the demo programs, then link them to this shell, which saves a lot of tedious retyping. There are complete and easy to follow instructions for compiling and linking the example programs, using the Alcyon C compiler and AS68 assembler from the Atari Developer's Kit. The C examples will translate easily (for the most part) to Megamax or Mark Williams C. To use the assembly examples with another assembler, you'll probably have to modify them somewhat.

These sections of the book are handled very well, except for one *huge* mistake in the discussion of the ST's screen memory organization (pages 69-70). Mr. Leemon's description of the monochrome screen's organization is correct; however, the descriptions of low and medium resolution are seriously flawed.

Leemon says that the first byte in low-resolution screen memory contains the pixel data for the first 2 pixels (horizontally) on the screen. Since 4 bits can hold a value from 0 to 15, each 4 bits of that byte contains a value from 0 to 15, which represents the color that appears at that location. For example, if the first byte of screen memory was hexadecimal $4E, the first 2 pixels would be colored according to the contents of color registers 4 (hex 4) and 14 (hex E). Sounds logical, and indeed, the Atari 8-bit's screen memory is organized in exactly this way. The only problem is that, for the ST, it's dead wrong.

The ST's screen memory is more complicated than this in reality. The color data

for the first pixel in a low-resolution display is spread among the first four *words* (a word is defined as 2 8-bit bytes) of screen memory, in a scheme sometimes referred to as "bit-planes." The first 4 bits of color data are contained in the leftmost bits (bit 15) of the first four words. The next pixel's color value is contained in the next bit of the first four words (bit 14), etc. The Abacus book *Atari ST Internals* has a fairly good description of the ST's screen memory. Consult Abacus and forget about pages 69-70 in ***COMPUTE!'s VDI***.

Appendix A of ***COMPUTE!'s VDI*** provides a complete list of every documented VDI function, in a clear, well written manner. This is the section of the book you'll probably find yourself using the most, once you've digested the introductory material in the beginning pages.

Each VDI call is listed by its title (e.g., Inquire Current Polyline Attributes) and its name according to the C bindings (e.g., vql_attributes), followed by a description of the call's purpose, the devices it's required for (e.g., screen, printer or metafile), an example of its usage and C syntax, and the proper parameters for the VDI arrays. Even those VDI calls that require the use of GDOS are listed here. In fact, throughout the book, GDOS is referred to as an integral part of the VDI system. Atari's current policy regarding GDOS states that developers must pay a one-time $500 fee to license GDOS for commercial software; it's unclear where this leaves the hobbyist programmer who writes software primarily for public domain distribution.

Appendix B lists the extended codes for every key on the ST's keyboard, including all ALT, SHIFT, and CONTROL key combinations. This comes in handy when you want to check for input from the function keys, the numeric keypad, or the cursor key group. Unfortunately, we find *another* error in Appendix B. . .the extended key codes for the right and left arrow keys are reversed.

Appendix C is a description of the GDOS/VDI font file format. All disk-based fonts to be used with the VDI text font calls (such as vst_load_fonts and vst_font) must be stored in this format. Appendix D shows the entire default ST ASCII character set. (Is there a law stating that every reference book for the ST must include an ASCII table?)

Aside from the errors mentioned above, and a small mistake on page 37 (SETBLOCK is incorrectly described as an XBIOS function—it's a GEMDOS function), I can find very few nits to pick with ***COMPUTE!'s VDI***. Along with a plethora of programming examples, it contains three indexes—a conventional index by keywords, and indexes to the VDI functions, both alphabetically and by function number. I could find no errors at all in Appendix A, the VDI Function Reference. The language of the book is clear and, given the technical nature of the subject, relatively free of computer buzzwords. It's soft cover opens flat for easy access during long programming sessions. If you're programming on the ST (for fun or professionally), ***COMPUTE!'s VDI*** should be on your necessary list.

Well done, Sheldon! I'm anxious to see a volume devoted to the Application Environment Services (AES) section of GEM; to date, all sources of AES documentation repeat the same errors *ad nauseam*. An AES reference book of the same quality as ***COMPUTE!'s VDI*** would be greatly appreciated. And please, correct that screen memory mistake in any second printing! //

Charles—and our readers—will be glad to hear that Volume Two is on its way. Its topic, naturally, is GEM's AES.

The Master Switch

Getting the most out of your printer.

by Matthew J.W. Ratcliff

There are probably many readers out there who own both 8-bit and 16-bit Atari computer systems. And—if you're like me—your printer tends to serve double-duty between both machines. Once while juggling my Gemini 10X between the two computers (with power on), I zapped the printer and the ST! That *nightmare* was detailed in issue 12. Suffice it to say, I learned from my mistakes—and built **The Master Switch** to prevent a reoccurrence.

For two computers to share the same printer, we must switch the eight data lines and the STROBE line going to the printer. Normally, this would require a pretty hefty mechanical switch, which is what most expensive printer switch boxes use. These boxes are universal in that they'll work with just about anything using Centronics connectors. With one of these switch boxes you can hook many computers to the same printer, or many printers to one computer. The switch we need is a bit simpler, and can probably be built for less than $30.

Looking at the schematic (next page), you can see that the heart of **The Master Switch** is a pair of 74LS244s. These are 8-bit latches with tri-state outputs. The DPDT switch allows only one chip, and thus, computer, to talk to the printer at a time. When a chip is off, its output lines go to a "high Z" state, where they don't affect the outgoing data lines.

The DPDT switch toggles an enable line on the chips. It also switches the STROBE line coming from the appropriate computer. There weren't enough lines in the 74LS244 chips to handle this one.

Notice that the busy signal coming back is "wire ORed" to both computers. Two computers may look at the same input line, without affecting it. The outputs must be separated with the switch and chips, however. Otherwise, they would hopelessly confuse the printer—and possibly the computers. (Note that we can "wire OR" the outputs of the 74LS244 chips together because of their tri-state outputs, which esentially disconnects them from the control of the printer when disabled.)

I've also included an LED with a 1K current limiting resistor on each side of the DPDT switch. This just serves as a reminder that the printer's power is on. The switch position indicates which computer has control of the printer, but the chips in the switch require power. It gets that power from pin 18 on the Gemini printer connector. If you want to build this box for a different printer, check your manual to see what pin +5 volts is available on. On some Epson printers it's either pin 18, or pin 35.

If your printer doesn't bring +5 volts out to its Centronics connector, you have several options. You can find +5 volts on a chip inside the printer and run it to an unused pin on the connector. You could also take +5 volts off pin 9 on your 850 parallel interface. (This, however, would require that you have the 850 interface on whenever you want to use the printer, even if using the ST.)

Your final option may be to get a small power pack—as long as it puts out +5 volts DC at a minimal current rating. (Nine-volt DC packs are common for calculators. You can regulate that down to 5 volts with the proper zener diode and current limiting resistor.) These LS (Low power Schottky) chips require very little power and shouldn't draw much current from the source you use.

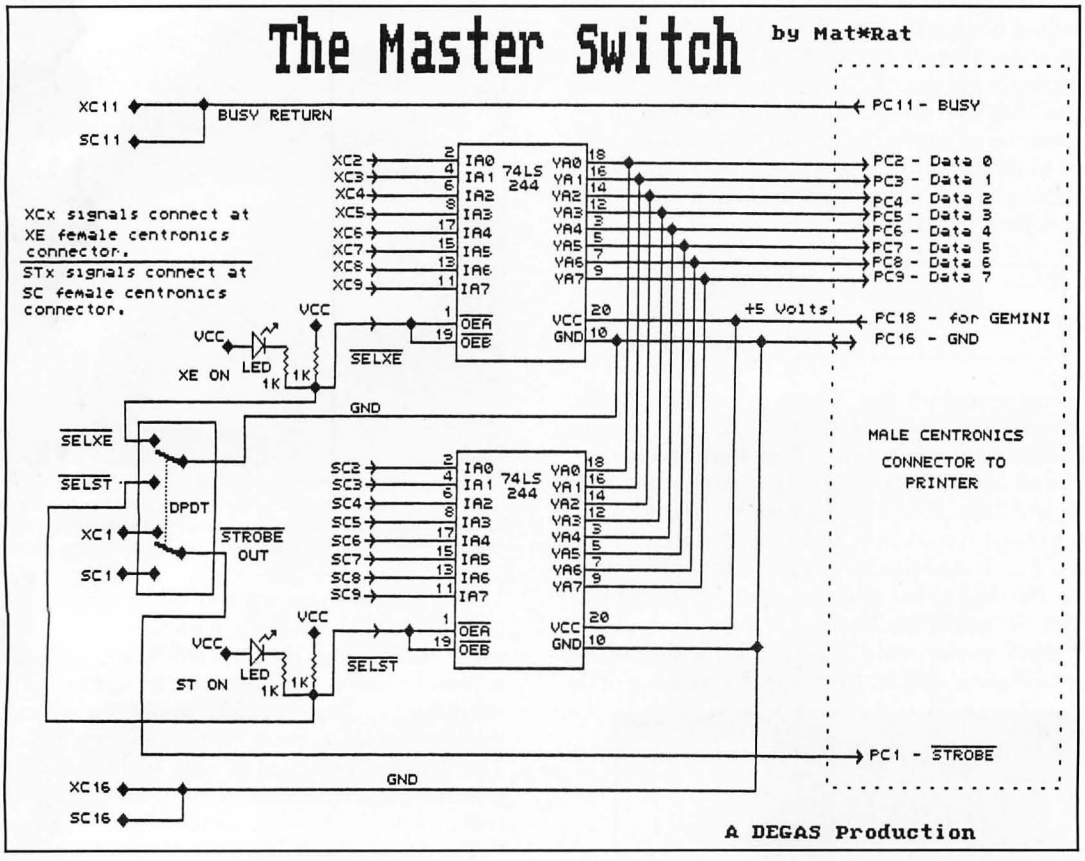
This switch will work with two STs sharing the same printer as well. Two 8-bit Ataris will also work with **The Master Switch**, as shown here.

To start, you will need the following items:

A small (3x6x2″) plastic project box.

Two Centronics female connectors and mounting hardware.

One male Centronics connector, and 3 feet of 12-wire cable.



Two 74LS244 chips.

An electronic project board.

Two 1K, ¼-watt resistors.

Two LEDs and two more 1K, ¼-watt resistors (optional).

One DPDT switch.

A low wattage soldering iron (25 watts) and solder.

Two 20-pin sockets (wire wrap type if you're planning to wire wrap the project).

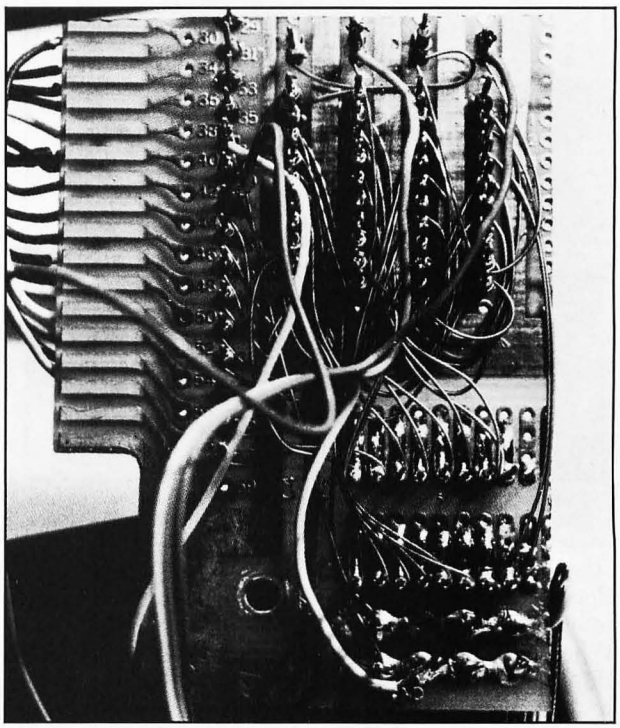
Some standoffs and screws for mounting the board.

Wire wrap wire and tools, if preferred.

Small hookup wire, if soldering method is desired.

If you have any electronic kit-building experience this project shouldn't be too tough. I spent more time mounting all the stuff in the plastic case than actually wiring it up.

Mount your Centronics female connectors in the case and label them for reference. I called one "XC" for the 130XE connection and the other "SC" for the ST hookup. The output connector to the printer is referred to as "PC." Connect pins 2-9 of the XC Centronics connector to the appropriate pins on one of the 74LS244 chips (called the XB chip, for XE Buffer). Wire the STROBE line from XC pin 1 to one side of the DPDT switch. On the same end of the switch, at the opposite terminal, connect pins 1 and 19 of the XB chip (the SELECT signal). Pins 1 and 19 of the XB should also be tied to +5 volts through a 1K pullup resistor. You will also attach your LED and current limit resistor to this same point.



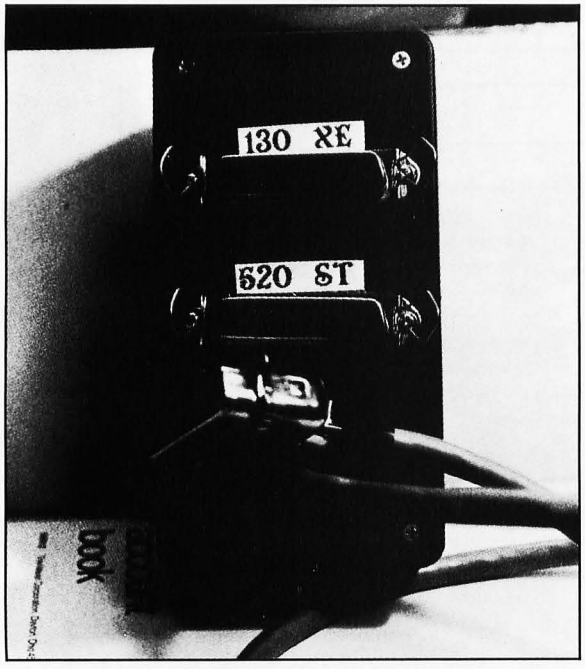
The bottom of The Master Switch circuit board, showing detail of the wire wrap job.

Follow the same procedure for the other chip, SB, and the ST Centronics connector. Remember to wire the ST STROBE line exactly opposite the XE STROBE on the DPDT switch, and similarly for the SELECT line. The two center terminals of the DPDT will be the switched control lines. Connect the center terminal on the SELECT side to ground. The other goes to the STROBE line to the male Centronics connector for the printer, PC pin 1. Your switch should be wired like the following example:

```
SELXE ----  □              □  ----------STROBE XE
GND -------  □    DPDT      □  ---STROBE to printer
SELST-----  □              □  ----------STROBE ST
```

When the switch enables the XE to drive the printer, SELXE will be tied to ground and STROBE XE will be connected to STROBE out to the printer. Pins 1 and 19 will be switched to ground, enabling the XB chip to drive the printer from the XE data lines. It will also provide a ground reference for the LED—if wired in—and it will light. At the same time, pins 1 and 19 of the SB chip will be pulled high to 5 volts, thus disabling that chip's output lines, regardless of what the ST might try to send it.

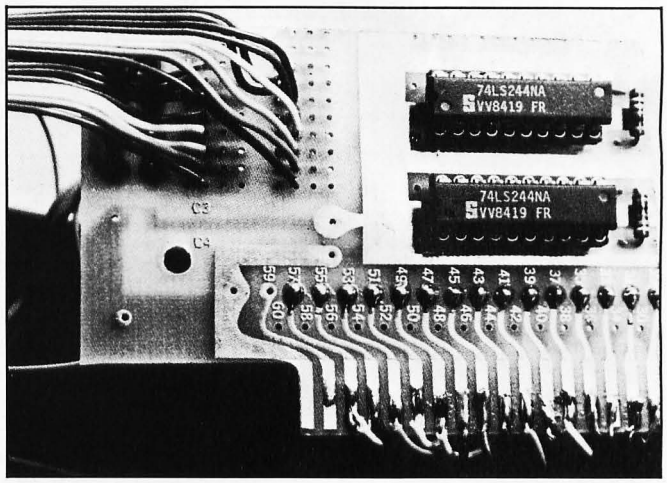
The output lines of the two 74LS244 chips should be wired together as shown, and to the printer Centronics male



The rear of The Master Switch case, showing the two flush mounted female connectors and the cable for the printer. The labels were made with Print Master.

connector. I wired them to a 3-foot cable to connect the printer.

Whether you wire wrap or solder this project, be sure to double check all connections before hooking it up. Make sure there are no solder bridges between pins on the chips or connectors. Look over the photos here to get a better idea of how I put mine together.



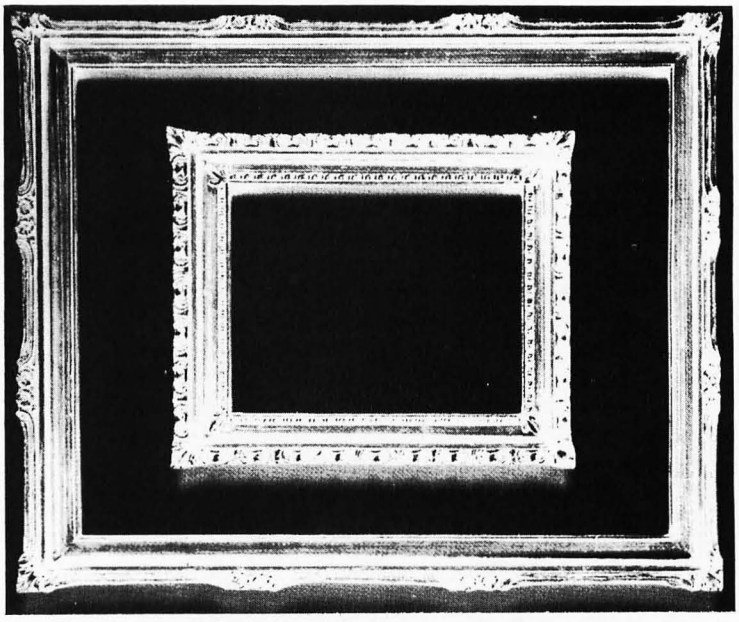
The top side of The Master Switch board, showing the 74LS244 chips. The two vertical rows of wires to the left are the wires going to the 8-bit and ST Centronics printer connectors. The wires soldered to the bottom of the board are going to the male Centronics connector for the printer.

If you want to save a few bucks, you can replace the XC connector with a 15-pin male connector on a cable, hooking it directly to the 850 interface. If you plan to switch a printer between two 8-bit systems, you could do the same with the SC. You'll have to figure out which pins to use to make your connections. Just remember, if you do it this way, **The Master Switch** will only work with 8-bits. By using Centronics connectors, it will work with almost any two computers sharing the same printer.

The Master Switch will save lots of wear and tear on your cables, connectors and printer. At the flick of a switch, either machine can use the printer—without the risk of damaging your valuable equipment. ///

Matthew Ratcliff is an electrical engineer in St. Louis, Missouri. When not using his spare time to write articles, he's president of ACE St. Louis and a remote SYSOP on Gateway City BBS, (314) 647-3290.

ALL RESOLUTIONS



Art Gallery

A schizophrenic picture viewer

by Charles F. Johnson

The first demonstration I ever saw on an Atari 520ST was a slideshow of pictures created with the Neo-Chrome drawing program. I was fascinated by the sharp 16-color graphics of the ST's low-resolution mode—I still am. I now have a collection of over 500 pictures drawn by ST artists from all over the world; and I suspect I'm not alone, judging from the number of pictures in the download sections of most BBSs and information services.

Art Gallery is a GEM desk accessory written in 68000 assembly language, using the AS68 assembler and "AS68 Helper" (**ST-Log** issue 12). I wrote **Art Gallery** because all the picture display utilities I had come across were in slideshow form. Now, slideshows are very nice for impressing the friends and neighbors, but I found that many times I wanted to look at just one picture; it got tedious to wait through all the other pictures on a disk to see the one I wanted.

I decided to write a program to show individual pictures saved in the formats used by the popular ST drawing programs, DEGAS and Neo-Chrome. I also chose to make it a desk accessory, so I could easily view pictures without leaving any GEM program (such as 1st Word, DEGAS Elite, or Flash). This turned out to be a fairly simple task, since both DEGAS and Neo-Chrome save pictures in a straightforward manner—they just copy the screen memory area to a disk file, along with color information and (in the case of Neo-Chrome) some other data. I then threw caution to the winds and also decided to show pictures saved in the popular public domain Tiny compression format (written by David Mumper), since many bulletin boards and services like Delphi, CompuServe, and GEnie feature pictures in this format. Here, I ran into a slight snag.

The only documentation I had for the Tiny compression format was an inaccurate and mistake-riddled text file captured from a local BBS. After attempting for some time (with little success) to translate this into workable code, I ran across a sample of C code that Tom Hudson had uploaded to CompuServe, for loading Tiny pictures. This code helped to fill in the gaps in my understanding of the Tiny format, and I was able to finish the decompression routine quickly after that. I haven't used Tom's code directly in this program, but I thought it only fair to mention it, since I had one wheel in the sand before coming across it. Thanks, Tom!

Using the program

Load ST BASIC and type in Listing 1, then check your typing carefully with "ST-Check". When you run the BASIC program, it will create a file on drive A called ARTGALRY.ACC. You can change the drive the program gets written to by changing *filename$* in the first line of the BASIC program.

To install the **Art Gallery** accessory, just copy it to the main directory of your boot disk, or drive C for hard disk users. The next time you boot the ST, **Art Gallery** will load and be available from within any GEM application that uses drop-down menus. Just click on its name in the "Desk" drop-down menu. The screen will clear, and then you'll see the **Art Gallery** dialog box in its center. There are four buttons in the box, labeled in what I hope is a self-explanatory manner. Click on the appropriate buttons to display DEGAS or DEGAS Elite pictures (compressed or uncompressed), Neo-Chrome pictures (disabled in high resolution), or compressed Tiny picture files. The button marked EXIT will, of course, send you back to whatever you were doing.

Clicking on any of the buttons except EXIT will cause a GEM "File Selector" box to appear. The operation of the box is the same as in any other GEM program. You can change the drive being accessed by editing the directory line in the File Selector. Click on the filename of the picture you wish to load and click the "OK" box (or just double-

click on the filename), and you'll be looking at that picture before you can say Vincent Van Gogh!

If you load a DEGAS Elite, Neo-Chrome, or Tiny picture that has color rotation information, **Art Gallery** will display the animation; if it's a DEGAS Elite picture, **Art Gallery** will display all four of its animation channels. When you wish to load another picture, just press the left mouse button once, and the File Selector will reappear. Click on "Cancel" to return to the main dialog box.

When you leave **Art Gallery**, the program that was running will redraw its screen area (assuming it's a properly written GEM application, of course) and you can continue from wherever you were. The **Gallery** is handy with a GEM-based terminal program; you can download pictures from your favorite BBS and have a look at them without ever leaving the terminal program. Instant gratification—isn't that the American Way?

The schizophrenic program

That's right, **Art Gallery** is a program with a severe identity crisis; it can't decide if it wants to be a program or a desk accessory. Instead, it seems to have come to terms with its psychosis and solved the problem by being both! If you name it with an extension of .ACC, as explained above, it will load and run as a desk accessory. But, if you give it a .PRG extension, the same file will work quite happily as a program you can run from the desktop. The subject seems well adjusted, despite its obvious schizophrenia.

To understand how this trick works, you need to know a little about the differences between accessories and programs. When GEMDOS (the part of TOS that deals with disk file manipulations and memory allocation, among other things) loads a program, it allocates the entire available RAM space for the use of the program.

The first thing the program must do is release back to GEMDOS the memory it doesn't need. Otherwise, since your program is greedily hogging all the RAM, operating system functions that allocate RAM for their own purposes will fail. The GEMDOS Mshrink() function is used to release back to GEMDOS the unused memory (the Abacus books call this function SETBLOCK). However, in the case of an accessory, TOS takes care of little details like this for you. All your accessory needs to do to be properly initialized is set up some stack space of its own and put that address in the 68000's stack pointer.

Another difference: a program usually exits by calling one of the GEMDOS terminate functions, which normally pops you back to the desktop. An accessory, on the other hand, *never* exits. When inactive, an accessory is sitting in an AES event call, waiting for you to select it from the "Desk" drop-down. When the accessory is closed (exited), it just goes back into that event call again; the GEMDOS terminate functions are never used.

With this info in hand, we can see how a program could run as either .ACC or .PRG. . . just check and see which extension it has, and branch around the unnecessary code. But another problem comes up when you try this: you could search the current disk directory for the filename of your program, but what if the user has renamed it? You won't find the name you're looking for!

The solution to this dilemma lies in a little-documented AES call, shel_read. This call will return the name of the currently active process; in other words, the name of your program. So, if the user has renamed ARTGALRY.ACC to ARTSTUFF.ACC, you'll still be able to tell whether it has an extension of .ACC or .PRG. Take a look at the source code for **Art Gallery** to see how this is done.

This technique can really help during the development process, since accessories can be quite difficult to debug. No debugger for the ST (that I know of) lets you run and trace through accessory code, so being able to test and debug a file as a program until you're ready to make it an accessory is a definite asset.

I hope you enjoy **Art Gallery**. If you're a programmer, take a look at the source code for some valuable hints on using the vertical blank interrupt, object trees, and other GEM structures, directly in 68000 assembly language. I included all the object trees in the source code instead of using an external .RSC file, because of a GEM bug that isn't very widely known. If an accessory loads a resource file created with the Resource Construction Set, it will keep stealing memory every time the screen resolution is changed with the desktop's "Set Preferences" option. This is because GEM doesn't release the memory allocated for an accessory's resource file on a resolution change. Therefore, the best practice is to always include object trees directly in an accessory program. ///

Charles F. Johnson is a professional musician and, now, a semi-professional computer programmer/reviewer/author. He lives in Los Angeles with his wife Patty and Spike, the world's most intelligent cat. Charles is a SYSOP on the ANALOG Publishing Atari SIG on Delphi; his user name is CFJ.

Listing 1
ST BASIC listing

```
100 filename$="a:\ARTGALRY.ACC"
110 fullw 2:clearw 2:gotoxy 0,0:print
"creating file..."
120 option base 0
125 dim a%(16000):def seg=1:v$=""
130 p=varptr(a%(0)):bptr=p+1
140 for i%=1 to 4697
150 read v$:code%=val("&H"+v$)
160 poke p, code%:print ".";
170 p=p+1
180 next
190 bsave filename$,bptr,4697
200 print "file written":end
1000 data 60,1A,00,00,0D,50,00,00,03,9
2,00,00,B9,02,00,00
1010 data 00,00,00,00,00,00,00,00,00,0
0,00,00,2A,4F,2E,7C
1020 data 00,00,C9,CC,42,79,00,00,11,0
6,23,FC,00,00,10,CC
1030 data 00,00,0D,50,23,FC,00,00,48,1
C,00,00,13,BC,23,FC
1040 data 00,00,49,1C,00,00,13,C0,61,0
0,0D,12,20,7C,00,00
1050 data 48,1C,70,08,0C,18,00,2E,67,0
4,51,C8,FF,F8,22,7C
1060 data 00,00,10,15,70,02,B3,08,66,0
C,51,C8,FF,FA,33,FC
1070 data 00,01,00,00,11,06,4A,79,00,0
0,11,06,66,2A,2A,6D
1080 data 00,04,20,2D,00,0C,D0,AD,00,1
4,D0,AD,00,1C,D0,BC
1090 data 00,00,01,00,2F,00,2F,0D,42,6
7,3F,3C,00,4A,4E,41
1100 data DF,FC,00,00,00,0C,60,52,42,B
9,00,00,13,AC,42,B9
1110 data 00,00,13,B0,42,B9,00,00,13,B
4,42,B9,00,00,13,B8
1120 data 23,FC,00,00,10,18,00,00,0D,5
0,61,00,0C,90,23,FC
1130 data 00,00,10,36,00,00,0D,50,33,F
9,00,00,12,9E,00,00
1140 data 11,9E,23,FC,00,00,0F,3A,00,0
```

```
0,13,BC,61,00,0C,6E
1150 data 33,F9,00,00,12,9E,00,00,11,1
C,3F,3C,00,04,4E,4E
1160 data 54,8F,33,C0,00,00,11,1A,3F,3
C,00,19,4E,41,54,8F
1170 data D0,3C,00,41,26,7C,00,00,16,4
4,28,7C,00,00,16,84
1180 data 2A,7C,00,00,16,C4,16,C0,18,C
0,1A,C0,20,7C,00,00
1190 data 0F,FD,22,7C,00,00,10,05,24,7
C,00,00,10,0D,3A,3C
1200 data 00,07,16,D8,18,D9,1A,DA,51,C
D,FF,F8,0C,79,00,02
1210 data 00,00,11,1A,66,12,13,FC,00,3
3,00,00,16,4B,33,FC
1220 data 00,08,00,00,0E,86,60,08,13,F
C,00,3F,00,00,16,4B
1230 data 2A,7C,00,00,A0,00,28,7C,00,0
0,0F,2C,7A,0C,10,15
1240 data B0,14,67,04,52,8D,60,F6,20,4
D,B9,0D,66,E8,51,CD
1250 data FF,FA,0C,28,00,46,FF,FF,66,D
C,20,08,23,C8,00,00
1260 data 10,EA,22,08,C2,BC,FF,FF,FF,F
E,20,41,55,88,22,10
1270 data B0,81,66,F8,23,C8,00,00,10,E
6,33,E8,00,08,00,00
1280 data 11,14,23,FC,00,00,10,C2,00,0
0,0D,50,23,FC,00,00
1290 data 0D,BC,00,00,13,BC,3A,3C,00,0
A,33,C5,00,00,11,9E
1300 data 61,00,0B,7A,51,CD,FF,F4,4A,7
9,00,00,11,06,67,66
1310 data 23,FC,00,00,10,22,00,00,0D,5
0,23,FC,00,00,16,04
1320 data 00,00,13,BC,61,00,0B,56,0C,7
9,00,28,00,00,16,04
1330 data 66,DE,30,39,00,00,16,0C,B0,7
9,00,00,11,1C,66,D0
1340 data 23,FC,00,00,10,B8,00,00,0D,5
0,33,FC,00,01,00,00
1350 data 11,9E,61,00,0B,28,3F,3C,00,1
9,4E,41,54,8F,33,C0
1360 data 00,00,11,16,3F,3C,00,00,2F,3
C,00,00,17,44,3F,3C
1370 data 00,47,4E,41,50,8F,42,45,38,3
C,00,0F,2A,7C,00,00
1380 data 15,BC,3F,3C,FF,FF,3F,05,3F,3
C,00,07,4E,4E,5C,8F
1390 data 3A,C0,52,85,51,CC,FF,EC,3F,3
C,00,02,4E,4E,54,8F
1400 data 23,C0,00,00,10,E2,4A,79,00,0
0,11,06,66,08,61,00
1410 data 09,A8,60,00,00,E4,23,FC,00,0
0,10,AE,00,00,0D,50
1420 data 2A,7C,00,00,11,9E,42,55,3B,7
C,00,04,00,02,61,00
1430 data 0A,AC,33,F9,00,00,12,A0,00,0
0,11,20,33,F9,00,00
1440 data 12,A2,00,00,11,22,33,F9,00,0
0,12,A4,00,00,11,24
1450 data 33,F9,00,00,12,A6,00,00,11,2
6,23,FC,00,00,10,86
1460 data 00,00,0D,50,42,55,3B,79,00,0
0,11,20,00,02,3B,79
1470 data 00,00,11,22,00,04,3B,79,00,0
0,11,24,00,06,3B,79
1480 data 00,00,11,26,00,08,61,00,0A,5
4,4A,79,00,00,12,9E
1490 data 6B,00,02,8E,33,F9,00,00,12,9
E,00,00,11,1E,23,FC
1500 data 00,00,10,90,00,00,0D,50,3A,B
9,00,00,11,1E,3B,79
1510 data 00,00,11,20,00,02,3B,79,00,0
0,11,22,00,04,3B,79
1520 data 00,00,11,24,00,06,3B,79,00,0
0,11,26,00,08,61,00
1530 data 0A,0C,0C,79,00,02,00,00,11,1
A,66,06,3A,3C,01,7B
1540 data 60,04,3A,3C,01,B7,28,79,00,0
0,10,E2,2A,7C,00,00
1550 data 17,84,2A,DC,51,CD,FF,FC,61,0
0,08,A2,61,00,08,CE
1560 data 42,80,30,39,00,00,11,28,C0,F
C,00,18,D0,BC,00,00
1570 data 0D,BC,20,40,42,68,00,0A,0C,7
9,00,0A,00,00,11,28
1580 data 67,00,01,A8,61,00,08,76,0C,7
9,00,07,00,00,11,28
1590 data 66,28,23,FC,00,00,16,44,00,0
0,10,F2,23,FC,00,00
1600 data 16,14,00,00,10,F6,23,FC,00,0
0,0F,B5,00,00,10,EE
1610 data 33,FC,00,B0,00,00,11,12,60,5
8,0C,79,00,08,00,00
1620 data 11,28,66,28,23,FC,00,00,16,8
4,00,00,10,F2,23,FC
1630 data 00,00,16,24,00,00,10,F6,23,F
C,00,00,0F,CC,00,00
1640 data 10,EE,33,FC,00,D0,00,00,11,1
2,60,26,23,FC,00,00
1650 data 16,C4,00,00,10,F2,23,FC,00,0
0,16,34,00,00,10,F6
1660 data 23,FC,00,00,0F,E7,00,00,10,E
E,33,FC,00,A8,00,00
1670 data 11,12,20,79,00,00,10,E6,20,B
9,00,00,10,EE,31,79
1680 data 00,00,11,12,00,08,00,68,00,0
1,FF,FE,61,00,07,CE
1690 data 23,FC,00,00,10,7C,00,00,0D,5
0,23,F9,00,00,10,F2
1700 data 00,00,13,BC,23,F9,00,00,10,F
6,00,00,13,C0,61,00
1710 data 08,EC,61,00,07,A8,20,79,00,0
0,10,E6,20,B9,00,00
1720 data 10,EA,02,68,00,FE,FF,FE,31,7
9,00,00,11,14,00,08
1730 data 20,79,00,00,10,F6,4A,10,67,0
0,FE,DE,0C,79,00,01
1740 data 00,00,12,A0,66,00,FE,D2,42,4
0,20,79,00,00,10,F2
1750 data 10,10,90,3C,00,41,3F,00,3F,3
C,00,0E,4E,41,58,8F
1760 data 20,79,00,00,10,F2,54,88,22,7
C,00,00,17,04,7A,3F
1770 data 4A,10,67,06,12,D8,51,CD,FF,F
8,7A,3F,0C,21,00,5C
1780 data 67,04,51,CD,FF,F8,52,89,42,1
1,48,79,00,00,17,04
1790 data 3F,3C,00,3B,4E,41,5C,8F,42,6
7,2F,39,00,00,10,F6
1800 data 3F,3C,00,3D,4E,41,50,8F,4A,4
0,6B,00,FE,6C,33,C0
1810 data 00,00,11,18,A0,0A,61,00,07,0
C,42,79,00,00,11,08
1820 data 0C,79,00,07,00,00,11,28,66,0
6,61,00,00,AC,60,14
1830 data 0C,79,00,08,00,00,11,28,66,0
6,61,00,02,4C,60,04
1840 data 61,00,02,A8,A0,09,60,00,FE,E
A,4A,79,00,00,11,06
1850 data 67,00,00,82,0C,79,00,02,00,0
0,11,1A,66,06,3A,3C
1860 data 01,7B,60,04,3A,3C,01,B7,28,7
9,00,00,10,E2,2A,7C
1870 data 00,00,17,84,28,DD,51,CD,FF,F
C,23,FC,00,00,10,9A
1880 data 00,00,0D,50,33,F9,00,00,11,1
E,00,00,11,9E,61,00
1890 data 07,CC,23,FC,00,00,10,A4,00,0
0,0D,50,61,00,07,BE
1900 data 3F,39,00,00,11,16,3F,3C,00,0
E,4E,41,58,8F,2F,3C
1910 data 00,00,17,44,3F,3C,00,3B,4E,4
1,5C,8F,23,FC,00,00
1920 data 10,B8,00,00,0D,50,42,79,00,0
```

```
0,11,9E,61,00,07,8E
1930 data 60,00,FC,1E,42,67,4E,41,2A,7
C,00,00,11,34,7A,02
1940 data 61,00,06,20,30,39,00,00,11,3
4,2A,7C,00,00,10,D6
1950 data 3A,3C,00,05,B0,5D,67,08,51,C
D,FF,FA,60,00,02,8C
1960 data 0C,79,00,02,00,00,11,1A,66,0
E,0C,39,00,02,00,00
1970 data 11,35,67,10,60,00,02,24,0C,3
9,00,02,00,00,11,35
1980 data 67,00,02,18,61,00,05,D4,61,0
0,03,62,08,39,00,07
1990 data 00,00,11,34,67,00,00,C6,2A,7
C,00,00,1E,64,2A,3C
2000 data 00,00,8C,A0,61,00,05,BC,2A,7
C,00,00,1E,64,28,79
2010 data 00,00,10,E2,4A,39,00,00,11,3
5,66,12,7A,03,28,3C
2020 data 00,00,00,C7,26,3C,00,00,00,A
0,74,06,60,28,0C,39
2030 data 00,01,00,00,11,35,66,12,7A,0
1,28,3C,00,00,00,C7
2040 data 26,3C,00,00,00,A0,74,02,60,0
C,7A,00,28,3C,00,00
2050 data 01,8F,76,50,74,00,33,C5,00,0
0,11,32,42,81,42,80
2060 data 10,1D,6B,16,26,4C,D7,C1,16,9
D,52,81,08,01,00,00
2070 data 66,02,D2,82,51,C8,FF,EE,60,1
8,44,00,1C,1D,26,4C
2080 data D7,C1,16,86,52,81,08,01,00,0
0,66,02,D2,82,51,C8
2090 data FF,EE,B2,83,6D,C8,92,83,54,8
1,51,CD,FF,C2,3A,39
2100 data 00,00,11,32,D9,C3,51,CC,FF,B
4,28,7C,00,00,15,DC
2110 data 3A,3C,00,1F,18,DD,51,CD,FF,F
C,60,24,2A,79,00,00
2120 data 10,E2,2A,3C,00,00,7D,00,61,0
0,04,F8,2A,7C,00,00
2130 data 15,DC,7A,21,61,00,04,EC,B0,B
C,00,00,00,20,66,60
2140 data 2A,7C,00,00,15,F4,3A,3C,00,0
3,30,3C,00,81,90,55
2150 data 3A,C0,51,CD,FF,F6,2A,7C,00,0
0,15,DC,28,7C,00,00
2160 data 15,E4,26,7C,00,00,15,EC,3A,3
C,00,03,30,1C,6B,1E
2170 data B0,7C,00,0F,6E,18,32,1D,6B,1
4,B2,7C,00,0F,6E,0E
2180 data B0,41,6F,0A,4A,53,6B,06,0C,5
3,00,02,6F,04,36,BC
2190 data 00,01,54,8B,51,CD,FF,D6,33,F
C,00,02,00,00,11,08
2200 data 61,00,02,B0,60,00,01,66,2A,7
C,00,00,11,34,7A,02
2210 data 61,00,04,70,61,00,04,6C,4A,7
9,00,00,11,34,66,00
2220 data 00,EA,61,00,04,56,2A,7C,00,0
0,48,1C,7A,5C,61,00
2230 data 04,52,4A,39,00,00,48,28,67,0
0,FE,6E,13,F9,00,00
2240 data 48,29,00,00,11,36,13,F9,00,0
0,48,2B,00,00,11,37
2250 data 61,00,02,06,61,00,01,B6,2A,7
9,00,00,10,E2,2A,3C
2260 data 00,00,7D,00,61,00,04,1C,60,9
6,2A,7C,00,00,11,34
2270 data 7A,01,61,00,04,0E,0C,39,00,0
3,00,00,11,34,6D,16
2280 data 57,39,00,00,11,34,2A,7C,00,0
0,11,36,7A,04,61,00
2290 data 03,F2,61,00,01,C4,13,F9,00,0
0,11,34,00,00,11,35
2300 data 42,39,00,00,11,34,0C,79,00,0
2,00,00,11,1A,66,20
2310 data 0C,79,00,02,00,00,11,34,67,2
0,2A,7C,00,00,0F,7C
2320 data A0,09,3A,3C,00,01,61,00,04,F
E,A0,0A,60,00,00,9E
2330 data 0C,79,00,02,00,00,11,34,67,E
0,61,00,03,9E,2A,7C
2340 data 00,00,11,7A,7A,02,61,00,03,9
A,2A,7C,00,00,11,7C
2350 data 7A,02,61,00,03,8E,0C,79,29,A
B,00,00,11,7A,62,0A
2360 data 0C,79,3E,80,00,00,11,7C,63,0
8,2A,7C,00,00,0F,48
2370 data 60,AE,2A,7C,00,00,1E,64,42,8
5,3A,39,00,00,11,7A
2380 data 61,00,03,60,23,C0,00,00,11,7
E,E3,F9,00,00,11,7C
2390 data 2A,7C,00,00,48,1C,42,85,3A,3
9,00,00,11,7C,61,00
2400 data 03,42,23,C0,00,00,11,82,42,8
5,3A,39,00,00,11,7A
2410 data BA,B9,00,00,11,7E,66,B2,3A,3
9,00,00,11,7C,BA,B9
2420 data 00,00,11,82,66,A4,61,14,61,0
0,01,48,3F,39,00,00
2430 data 11,18,3F,3C,00,3E,4E,41,58,8
F,4E,75,61,00,00,8E
2440 data 42,80,20,79,00,00,10,E2,32,3
9,00,00,11,7A,53,41
2450 data 22,7C,00,00,1E,64,24,7C,00,0
0,48,1C,42,43,16,19
2460 data 4A,03,6A,06,44,03,61,2A,60,2
2,4A,03,67,06,B6,3C
2470 data 00,01,66,16,18,19,E1,4C,18,1
9,55,41,C9,43,4A,04
2480 data 66,04,61,1A,60,06,61,0A,60,0
2,61,12,51,C9,FF,CE
2490 data 4E,75,53,43,3A,1A,61,12,51,C
B,FF,FA,4E,75,53,43
2500 data 3A,1A,61,06,51,CB,FF,FC,4E,7
5,38,00,E3,4C,31,85
2510 data 40,00,D0,7C,00,50,B0,7C,3E,7
F,6F,0E,90,7C,3E,7C
2520 data B0,7C,00,4F,6F,04,90,7C,00,4
F,4E,75,48,79,00,00
2530 data 0B,D2,3F,3C,00,26,4E,4E,5C,8
F,30,39,00,00,11,3A
2540 data 22,7C,00,00,11,5A,32,3C,00,0
F,32,C0,51,C9,FF,FC
2550 data 48,79,00,00,11,5A,3F,3C,00,0
6,4E,4E,5C,8F,3F,3C
2560 data 00,25,4E,4E,54,8F,4E,75,48,7
9,00,00,11,3A,3F,3C
2570 data 00,06,4E,4E,5C,8F,4E,75,42,8
0,42,81,10,39,00,00
2580 data 11,36,E8,08,23,C0,00,00,10,F
E,12,39,00,00,11,36
2590 data C2,3C,00,0F,23,C1,00,00,11,0
2,92,00,67,2C,33,C1
2600 data 00,00,11,0E,42,79,00,00,11,0
A,4A,39,00,00,11,37
2610 data 6A,0E,44,39,00,00,11,37,33,F
C,FF,FF,00,00,11,0A
2620 data 33,FC,00,01,00,00,11,08,60,0
6,42,79,00,00,11,08
2630 data 4E,75,4A,79,00,00,11,08,67,1
8,48,79,00,00,0A,AA
2640 data 3F,3C,00,26,4E,4E,5C,8F,4A,4
0,6A,06,42,79,00,00
2650 data 11,08,61,00,FF,74,23,FC,00,0
0,10,2C,00,00,0D,50
2660 data 2A,7C,00,00,11,9E,3A,BC,00,0
3,3B,7C,00,01,00,02
2670 data 3B,7C,00,01,00,04,3B,7C,00,0
1,00,06,61,00,02,DE
2680 data 08,39,00,01,00,00,12,9F,67,0
8,42,6D,00,06,61,00
2690 data 02,CC,4A,79,00,00,11,08,67,0
E,48,79,00,00,0A,E0
2700 data 3F,3C,00,26,4E,4E,5C,8F,48,7
```

```
9,00,00,15,BC,3F,3C
2710 data 00,06,4E,4E,5C,8F,48,79,00,0
0,0B,C6,3F,3C,00,26
2720 data 4E,4E,5C,8F,4E,75,20,79,00,0
0,04,56,30,3C,00,07
2730 data 4A,90,67,08,58,88,51,C8,FF,F
8,4E,75,42,79,00,00
2740 data 11,0C,42,B9,00,00,15,FC,42,B
9,00,00,16,00,20,BC
2750 data 00,00,0A,EE,23,C8,00,00,10,F
A,4E,75,20,79,00,00
2760 data 10,FA,20,BC,00,00,00,00,4E,7
5,0C,79,00,02,00,00
2770 data 11,08,67,6C,52,39,00,00,11,0
C,10,39,00,00,11,0C
2780 data B0,39,00,00,11,37,66,4A,42,3
9,00,00,11,0C,4A,79
2790 data 00,00,11,0A,6A,08,20,39,00,0
0,10,FE,60,06,20,39
2800 data 00,00,11,02,E3,48,D0,BC,00,F
F,82,40,20,40,33,D0
2810 data 00,00,11,10,32,39,00,00,11,0
E,53,41,4A,79,00,00
2820 data 11,0A,6A,10,30,E8,00,02,51,C
9,FF,FA,30,B9,00,00
2830 data 11,10,4E,75,30,A8,FF,FE,55,8
8,51,C9,FF,F8,60,EC
2840 data 7A,03,28,05,D8,84,2A,44,DB,F
C,00,00,15,EC,0C,55
2850 data 00,01,67,48,28,44,D9,FC,00,0
0,15,FC,52,54,30,14
2860 data B0,6C,FF,F8,66,36,42,54,28,4
4,D9,FC,00,00,15,E4
2870 data 30,14,90,6C,FF,F8,33,C0,00,0
0,11,0E,42,80,4A,55
2880 data 66,0E,33,FC,FF,FF,00,00,11,0
A,30,2C,FF,F8,60,08
2890 data 42,79,00,00,11,0A,30,14,61,0
0,FF,6A,51,CD,FF,A4
2900 data 4E,75,13,F9,00,00,11,1B,00,F
F,82,60,4E,75,13,F9
2910 data 00,00,11,35,00,FF,82,60,4E,7
5,2A,7C,00,00,11,3A
2920 data 7A,20,2F,0D,2F,05,3F,39,00,0
0,11,18,3F,3C,00,3F
2930 data 4E,41,DF,FC,00,00,00,0C,4A,8
0,4E,75,A0,0A,61,04
2940 data A0,09,4E,75,2A,79,00,00,10,E
2,2A,3C,00,00,1F,3F
2950 data 42,9D,51,CD,FF,FC,4E,75,23,F
C,00,00,10,72,00,00
2960 data 0D,50,42,79,00,00,11,9E,60,0
0,01,12,23,FC,00,00
2970 data 10,68,00,00,0D,50,23,FC,00,0
0,0D,BC,00,00,13,BC
2980 data 61,00,00,FA,33,F9,00,00,12,A
0,00,00,11,2A,33,F9
2990 data 00,00,12,A2,00,00,11,2C,20,7
C,00,00,0D,BC,59,79
3000 data 00,00,11,2C,59,68,00,12,0C,7
9,00,02,00,00,11,1A
3010 data 66,0A,59,79,00,00,11,2C,59,6
8,00,12,33,F9,00,00
3020 data 12,A4,00,00,11,2E,58,79,00,0
0,11,2E,33,F9,00,00
3030 data 12,A6,00,00,11,30,58,79,00,0
0,11,30,23,FC,00,00
3040 data 10,54,00,00,0D,50,2A,7C,00,0
0,11,9E,42,55,42,AD
3050 data 00,02,42,AD,00,06,3B,79,00,0
0,11,2A,00,0A,3B,79
3060 data 00,00,11,2C,00,0C,3B,79,00,0
0,11,2E,00,0E,3B,79
3070 data 00,00,11,30,00,10,61,64,23,F
C,00,00,10,40,00,00
3080 data 0D,50,42,55,3B,7C,00,05,00,0
2,3B,79,00,00,11,2A
3090 data 00,04,3B,79,00,00,11,2C,00,0
6,3B,79,00,00,11,2E
3100 data 00,08,3B,79,00,00,11,30,00,0
A,61,30,23,FC,00,00
3110 data 10,4A,00,00,0D,50,42,55,61,2
2,33,F9,00,00,12,9E
3120 data 00,00,11,28,4E,75,23,CD,00,0
0,13,BC,33,C5,00,00
3130 data 11,9E,23,FC,00,00,10,5E,00,0
0,0D,50,22,3C,00,00
3140 data 0D,50,20,3C,00,00,00,C8,4E,4
2,4E,75,00,00,11,86
3150 data 00,00,13,9E,00,00,11,9E,00,0
0,12,9E,00,00,13,BC
3160 data 00,00,14,BC,00,00,0E,C8,00,0
0,0E,F9,00,00,0E,F9
3170 data 00,05,00,06,00,02,10,00,00,0
0,FF,FF,00,10,00,01
3180 data 00,00,0E,D8,00,00,0E,F9,00,0
0,0E,F9,00,03,00,06
3190 data 00,00,10,00,00,00,FF,FF,00,0
C,00,01,00,00,0E,E4
3200 data 00,00,0E,F9,00,00,0E,F9,00,0
5,00,06,00,02,10,00
3210 data 00,00,FF,FF,00,16,00,01,FF,F
F,00,01,00,01,00,14
3220 data 00,00,00,20,00,02,11,32,00,0
0,00,00,00,27,00,0E
3230 data 00,00,00,02,00,0A,00,14,00,0
0,00,00,00,FF,11,00
3240 data 00,02,00,01,00,23,00,0C,00,0
6,00,03,00,05,00,14
3250 data 00,00,00,00,00,FF,11,71,00,0
0,00,00,00,23,00,03
3260 data 00,04,FF,FF,FF,FF,00,15,00,0
0,00,00,00,00,0D,68
3270 data 00,0A,00,00,00,0F,00,01,00,0
5,FF,FF,FF,FF,00,15
3280 data 00,00,00,00,00,00,0D,84,00,0
C,00,01,00,0B,00,01
3290 data 00,02,FF,FF,FF,FF,00,15,00,0
0,00,00,00,00,0D,A0
3300 data 00,07,00,02,00,15,00,01,00,0
7,FF,FF,FF,FF,00,1C
3310 data 00,00,00,00,00,00,0E,FA,00,0
6,00,04,00,16,00,01
3320 data 00,08,FF,FF,FF,FF,00,1A,00,0
5,00,00,00,00,0F,11
3330 data 00,02,00,06,00,09,00,02,00,0
9,FF,FF,FF,FF,00,1A
3340 data 00,05,00,00,00,00,0F,17,00,0
D,00,06,00,09,00,02
3350 data 00,0A,FF,FF,FF,FF,00,1A,00,0
5,00,00,00,00,0F,1B
3360 data 00,18,00,06,00,09,00,02,00,0
1,FF,FF,FF,FF,00,1A
3370 data 00,27,00,00,00,00,0F,20,00,0
D,00,09,00,09,00,02
3380 data 00,00,0D,BC,53,54,2D,4C,6F,6
7,20,50,72,65,73,65
3390 data 6E,74,73,00,41,52,54,20,47,4
1,4C,4C,45,52,59,00
3400 data 62,79,20,43,68,61,72,6C,65,7
3,20,46,2E,20,4A,6F
3410 data 68,6E,73,6F,6E,00,57,68,69,6
3,68,20,74,79,70,65
3420 data 20,6F,66,20,70,69,63,74,75,7
2,65,3F,00,44,45,47
3430 data 41,53,00,4E,45,4F,00,54,49,4
E,59,00,45,58,49,54
3440 data 00,00,00,00,00,00,0F,2C,49,5
4,45,4D,20,53,45,4C
3450 data 45,43,54,4F,52,00,20,20,41,7
2,74,20,47,61,6C,6C
3460 data 65,72,79,00,5B,33,5D,5B,54,6
8,69,73,20,66,69,6C
3470 data 65,20,68,61,73,20,61,6E,20,6
9,6E,63,6F,72,72,65
3480 data 63,74,20,7C,66,6F,72,6D,61,7
```

```
4,21,7C,20,5D,5B,20
3490 data 53,6F,72,72,79,21,20,5D,5B,3
3,5D,5B,54,68,69,73
3500 data 20,70,69,63,74,75,72,65,27,7
3,20,69,6E,20,74,68
3510 data 65,20,77,72,6F,6E,67,20,7C,7
2,65,73,6F,6C,75,74
3520 data 69,6F,6E,21,7C,20,5D,5B,20,5
3,6F,72,72,79,21,20
3530 data 5D,20,57,68,69,63,68,20,44,4
5,47,41,53,20,70,69
3540 data 63,74,75,72,65,3F,20,00,20,5
7,68,69,63,68,20,4E
3550 data 45,4F,43,48,52,4F,4D,45,20,7
0,69,63,74,75,72,65
3560 data 3F,20,00,20,57,68,69,63,68,2
0,54,49,4E,59,20,70
3570 data 69,63,74,75,72,65,3F,20,00,3
A,5C,2A,2E,50,3F,3F
3580 data 00,3A,5C,2A,2E,4E,45,4F,00,3
A,5C,2A,2E,54,4E,59
3590 data 00,41,43,43,00,0A,00,00,00,0
1,00,00,00,00,00,17
3600 data 00,00,00,01,00,01,00,00,00,1
9,00,10,00,07,00,01
3610 data 00,00,00,23,00,01,00,01,00,0
1,00,00,00,2A,00,06
3620 data 00,01,00,01,00,00,00,32,00,0
1,00,02,00,01,00,00
3630 data 00,33,00,09,00,01,00,01,00,0
0,00,34,00,01,00,01
3640 data 00,01,00,00,00,36,00,00,00,0
5,00,01,00,00,00,4E
3650 data 00,01,00,01,00,01,00,00,00,5
A,00,00,00,02,00,02
3660 data 00,00,00,64,00,05,00,01,00,0
0,00,00,00,65,00,05
3670 data 00,05,00,00,00,00,00,66,00,0
1,00,01,00,00,00,00
3680 data 00,67,00,01,00,01,00,00,00,0
0,00,68,00,02,00,05
3690 data 00,00,00,00,00,00,6B,00,01,00,0
1,00,00,00,00,00,72
3700 data 00,01,00,01,00,01,00,00,00,7
8,00,00,00,01,00,02
3710 data 00,00,80,00,80,01,80,02,00,0
0,00,01,00,02,00,00
3720 data 00,04,06,06,04,06,04,06,04,0
A,12,12,06,32,06,06
3730 data 06,06,04,0A,04,06,04,06,04,0
A,04,0E,12,06,06,0C
3740 data 06,06,16,0A,08,0A,0C,26,18,0
8,06,04,06,04,0A,0E
3750 data 08,04,06,04,0C,08,06,08,04,0
8,12,0A,14,24,06,10
3760 data 04,06,12,04,06,04,06,04,06,0
4,06,04,08,08,08,08
3770 data 0C,0A,04,06,04,06,06,08,08,0
8,0E,12,06,16,0A,0E
3780 data 10,08,04,06,04,06,04,08,0A,0
8,04,06,04,06,04,08
3790 data 08,04,06,04,06,04,08,06,06,0
6,12,04,06,04,06,04
3800 data 0E,06,0C,08,0E,0C,16,08,22,1
0,14,0C,08,10,18,0C
3810 data 12,06,0C,04,06,04,0A,04,0A,0
E,0E,04,06,12,0C,06
3820 data 18,0A,0E,14,0A,10,06,06,1C,2
6,48,0C,12,10,14,16
3830 data 06,06,38,0E,10,0E,0C,0A,04,0
6,04,0E,12,0E,08,06
3840 data 10,04,06,08,0A,08,18,0C,0C,0
E,0A,08,08,08,0A,06
3850 data 06,08,0A,08,06,08,06,0E,16,0
6,08,06,76,0E,06,10
3860 data 18,14,08,06,0A,0A,06,06,08,0
8,08,08,08,08,12,0A
3870 data 04,06,22,10,08,0E,0E,26,06,0
6,06,06,08,10,08,06
3880 data 06,08,06,08,08,10,06,08,10,1
C,0E,14,0C,0E,0C,12
3890 data 0C,0C,0C,1E,14,04,06,0A,04,0
6,04,0A,04,06,04,06
3900 data 06,0C,08,0A,04,06,06,04,06,0
6,04,06,10,08,08,08
3910 data 0A,04,0E,08,08,08,0A,04,0A,0
4,08,06,06,04,06,0E
3920 data 04,04,04,04,04,04,04,04,14,0
4,04,14,04,04,68,18
3930 data 18,18,18,18,18,18,0C,64,00
3940 data *
```

ST CHECKSUM DATA

```
 100 data  69, 948, 117, 614, 503, 242
, 410, 427, 14, 109, 3453
 190 data  674, 357, 623, 578, 824, 80
0, 653, 906, 950, 648, 7013
 1080 data  940, 712, 907, 803, 760, 6
52, 810, 756, 802, 732, 7874
 1180 data  854, 756, 26, 752, 707, 74
5, 13, 53, 129, 880, 4915
 1280 data  763, 837, 906, 710, 677, 8
19, 758, 821, 681, 756, 7728
 1380 data  34, 54, 719, 823, 708, 795
, 700, 762, 666, 622, 5883
 1480 data  664, 819, 707, 611, 590, 7
06, 785, 64, 843, 727, 6516
 1580 data  689, 799, 824, 730, 825, 7
60, 852, 808, 906, 791, 7984
 1680 data  802, 807, 794, 827, 908, 8
63, 840, 779, 769, 1, 7390
 1780 data  898, 855, 942, 662, 697, 6
59, 875, 692, 836, 23, 7139
 1880 data  706, 857, 793, 813, 749, 8
15, 691, 993, 645, 620, 7682
 1980 data  689, 754, 799, 701, 712, 6
51, 694, 740, 807, 962, 7509
 2080 data  873, 206, 981, 976, 780, 8
73, 769, 3, 829, 886, 7176
 2180 data  850, 887, 699, 666, 783, 7
98, 664, 745, 684, 655, 7431
 2280 data  669, 738, 618, 689, 825, 7
56, 723, 788, 751, 799, 7356
 2380 data  809, 728, 709, 950, 693, 8
12, 715, 708, 784, 796, 7704
 2480 data  894, 992, 17, 963, 854, 82
7, 942, 831, 816, 786, 7922
 2580 data  774, 810, 630, 878, 665, 7
77, 820, 779, 766, 728, 7627
 2680 data  672, 807, 883, 831, 788, 9
69, 838, 837, 757, 659, 8041
 2780 data  743, 711, 920, 631, 947, 1
87, 970, 859, 69, 909, 6946
 2880 data  48, 947, 919, 785, 763, 92
0, 833, 993, 737, 819, 7764
 2980 data  849, 771, 616, 724, 721, 6
95, 752, 713, 677, 630, 7148
 3080 data  692, 650, 677, 729, 778, 7
19, 738, 703, 824, 663, 7173
 3180 data  758, 756, 671, 835, 522, 5
84, 519, 594, 879, 875, 6993
 3280 data  558, 905, 865, 591, 879, 8
69, 544, 908, 873, 554, 7546
 3380 data  840, 770, 818, 865, 796, 7
64, 681, 767, 854, 840, 7995
 3480 data  865, 869, 810, 901, 854, 7
81, 798, 835, 773, 848, 8334
 3580 data  860, 552, 500, 517, 494, 5
16, 534, 519, 537, 513, 5542
 3680 data  543, 545, 513, 557, 576, 6
16, 653, 606, 575, 650, 5834
 3780 data  586, 574, 670, 646, 668, 6
69, 665, 652, 635, 620, 6385
 3880 data  659, 656, 603, 644, 585, 6
34, 222, 4003
```

Ian's Quest

ST news, information and opinion

by Ian Chadwick

This was a difficult column to write. Normally, I have a target, some idea of what to do. This time, I tried two or three things and they all felt unsatisfactory.

Why? Well, I used to work for a company involved in desktop publishing. For the four months I was there, I worked almost every evening and weekend writing, editing, training and testing. In the final seven weeks alone, I put in 260 hours overtime. I was burning out. I spent little time with my wife, abandoned the garden, ignored the cats, and my social life became a black hole. And when, over personal conflicts inside the company, I was asked to resign, it almost seemed a relief rather than a hardship.

Since that time, I haven't done too much. I've been relaxing, letting the tension slowly evaporate. So when I sat down to write this column, I realized I hadn't done much of anything with my ST in months. I had nothing to pick on, nothing to stand on my soap box and rail about.

Then I got to thinking about those few programs I *did* use during the "hard times." The computer became my escape vehicle, my dream machine. I didn't play many games—they're a lot like television: canned and preprogrammed. What little time I had was spent with Tom Hudson's CAD 3D 2.0. It became my personal transport to that place in my own mind where I could relax.

I like my relaxation to be stimulating, rather than passive. I read at every opportunity. There are piles of books in every room in my house, and I carry books with me everywhere: in the john, in the car, in my briefcase and in my jacket. I even take a book when I walk the dog. TV interferes with my reading, so I try to avoid it.

But there are few computer recreations that stimulate me as CAD 3D does. I get lost inside it. I build things, create objects I've never seen before. I've assembled a model of my own house and another of a city street full of skyscrapers. I built several spaceships, a few cars and a chess set. None of them has been very well built or designed—I'm sure there are hundreds of talented artists out there who could do them all better and in less time—but I've received an enormous amount of pleasure from the process.

What I'm doing is extending my childhood hobby of model building. When I was a young teenager—back around the Miocene—I built plastic models. Then, I was limited by my paltry allowance and the constraints placed on the actual contents of a model kit. Now, I'm free to build and create what was only limited by my imagination.

The real fun I had with models wasn't in following the instructions (something I'm poor at); it was in the customizing. A lot of my models looked like hell, but I had lots of fun in the building. And that's what appeals to me about CAD 3D: it's simply a helluva lotta fun.

I remember one morning last year at around 6 a.m., my wife Susan came into the computer room where I was hunched over my ST, tapping away. I must have looked a might dishevelled.

"When did you get up this morning?" she asked, rubbing sleep from her eyes.

"Ah, well, I never exactly got to bed last night," I replied sheepishly.

"What have you been doing all night?"

"Come here, I'll show you," I said, beaming with pride and still running on the adrenaline that had kept me going all night.

Susan sat down beside me and patiently watched as I proudly displayed my model of Stonehenge, crafted with the beta version of the first CAD 3D. I rotated it, zoomed into the center, pointed out the various rings of stones, blew it up, shrank it down to an indistinguishable dot. After about 15 minutes I turned to her.

"Well?" I asked.

"Well what?"

"What do you think of it?"

"What does it do?"

"Do?" I cried, aghast, "What do you mean, *do?*"

Well, it went on like that for the next few minutes. Susan, a practical woman, couldn't quite appreciate the joy I felt after laboring eight hours to produce the collection of lines she saw on the screen. Susan, of course, never built models as a kid, so couldn't quite see the flickering phosphor as much of anything.

The end result of a piece of software is pretty fragile: a clump of magnetic oxide with magnetic poles aligned in specific directions so that it can be decoded later. Hardly something an archeologist will recognize a few millennia from now.

What I write with my computer becomes hard copy, which is eventually stored somewhere in archival form that can be handled and read. The output from my spreadsheet is forever enshrined in some government tax office (a dubious immortality, I'll admit). These are practical, understandable applications with concrete output. The computer merely performs as a tool for a task that existed before the computer itself.

But the output of CAD 3D (as well as Aegis Animator and other such programs) is entirely on disk, since their output is meant to be directed to a computer's screen, not the flatness of paper. It somehow makes the result ethereal, like a ghost, something not quite real. We're more accustomed to end objects that can be touched or have some permanence. When I turn off my computer, I get a disquieting feeling I've lost something, even though I know it's been saved to disk.

The computer as an imagination device is more or less ignored by the sales folk, who like to nail down in solid form all the wonders the machine can do. I've never heard anyone call it a dream machine. But isn't that one of those marvelous things we want it for? In that 8-hour period, I erected the circles of stone and dug the pits and rings in the earth, just as the mysterious engineers did some 3,000 years ago on a windy plain near Salisbury. In your dreams you are truly free.

I worked a lot with CAD 3D during the few months I was involved with its manual. I developed a great respect for Tom Hudson, the author, who seems to be building tools that inveterate tinkerers like myself enjoy so much. I really grew to like the program, due in part, no doubt, to Tom's willingness to listen to my suggestions, even if harebrained. I still have an armload of suggestions, if he's in the market for them. But the program has come a long way since I first played with it.

It's nice to see a new version of any program you appreciate. It signifies that the author/publishers *care*: they're continuing the development and design work begun with 1.0. It means evolution for the user —you can grow into new versions and learn new features easily. It also means the program won't become outmoded or outdated, because new versions are always on their way. In a way, it's a lot like buying a car. I like my current set of wheels, but five or six years from now, when I'm ready for another, I'll want a newer, state-of-the-art vehicle—not the 1986 model again.

So CAD 3D 2.0 is the new model. By itself, there's not a lot new in the design or intent. The main purpose is to let you create and manipulate 3D objects in space, and view them from a variety of angles. However, the overall appearance and performance of the program has been significantly improved.

The user interface has been polished, more objects are allowed (forty), and there's finally the option to load and save templates for extruded objects. There's "real-time" camera movement in super view and optional stereo separation for the StereoTek glasses (which I've seen, but not used). There's greater color control, better (and faster) shading effects and a recolor option. You can arrange objects in four different groups and manipulate each individually.

For me, the most important additions are the truly flexible placement of lighting sources and the ability to measure and scale objects. The latter dilutes one of the criticisms of the earlier version, that without a method of measurement and scaling, the program is not "professional."

The manual is new, too—very clear and well written. It has a tutorial, tips, advanced techniques, lots of illustrations, even a keyboard command card.

These enhancements pale in comparison to the animation feature. CAD 3D 2.0 comes with the Cybermate animation language program. It is now possible to create complex animation "stories" limited only by your available disk space. Here's where I confess my weakness. I haven't done much more than make a few halting attempts to master the animation. I've seen the results. Darrel Anderson not only wrote a tutorial on using Cybermate, but produced an impressive demonstration involving a spaceship-disk flying into the ST—quite amazing and somewhat beyond my current ability. As I said, I've been busy. I'm saving this delight for another time.

But my delight is still in making the models, trying to build in the CAD universe. I always feel a little like a deity, since I have such absolute control there. I have the power to create and destroy, to shape and move. It's like working magic.

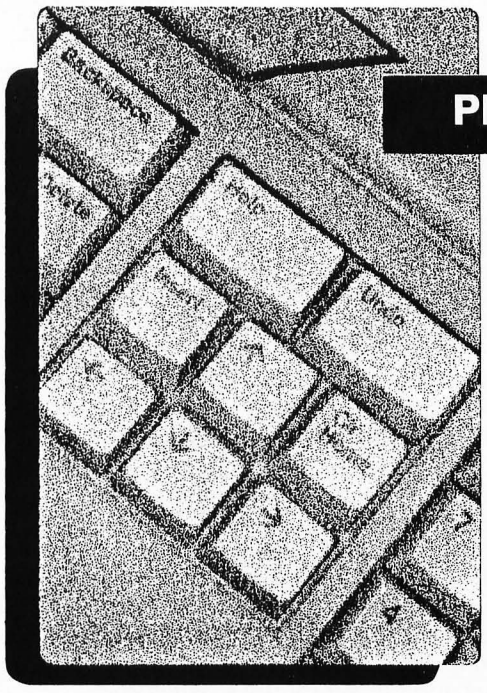
In trying to create the model of my own house, I had to wander through it—both in my mind and in reality—to find something I'd never noticed before, or to clarify some detail I'd ignored. I got to know my house more intimately than I expected. Of course, I made the whole thing look good, but it's not accurate. I plan to measure everything in the house one day and rebuild the model to scale. Luckily, the program's resizing feature will let me do it to the existing model, without having to start again from scratch.

When I had my little house built, I copied it a few times. There! I had my block. I had the same sense of delight a friend gets when he builds model railroad sets. (Of course, his stands in his basement, where he can look at it anytime.)

Probably the best line in the entire manual (at the end of the first tutorial) parallels my own approach: "Get completely lost in the CAD 3D universe." I can't think of a better way to discover the wonders of this program. ///

Ian Chadwick is a Toronto-based writer. He's been an author, book editor, magazine editor and roustabout in the publishing business for many years. In his spare time, he brews his own beer and plays wargames.

Assembly line



A text enciphering program—Part 1.

by Douglas Weir

Although this is only the sixth installment of **Assembly line**, we've already managed to cover a large part of the 68000's instruction set, and almost all of its addressing modes. This time we'll meet the last of the latter— Address Register Indirect with Index; plus, one of the most complex of all the instructions, link.

Last time we learned a technique for passing parameters to subroutines, by pushing them on the stack, then using the Address Register Indirect with Displacement mode to read them from within the subroutine—without actually popping the values and disturbing the stack itself. The only difficulty with this, as used in the program, is that the displacement values have to be tediously recalculated for every subroutine, depending on how many registers the subroutine saves on the stack when it begins. However, there's an easier way to go about this.

The link instruction.

One of the nicest features of the so-called structured programming languages like Pascal or C is local variables. You can write a function or procedure with its own set of variables, some of which may have names identical to those in other subroutines, but which are active and accessible only within the subroutine to which they "belong." They exist only as long as the subroutine is active, and when it terminates, they do also.

This feature can be duplicated in assembly language; the 68000 implementation is founded on one instruction which handles practically everything. This instruction, link, is probably the single most complicated in the entire 68000 set. To make a long story short, it reserves a block of memory on the stack. This memory can then be used (via an address register used as a base pointer) as space for "local" variables just as if it had been reserved in your data segment. A second instruction, unlk (unlink), is used to deallocate this space when you're finished with it.

The link instruction requires two operands, separated (as usual) by a comma. The first is an address register (usually not a7, for reasons which will become clear later). The second is an immediate value. Let's assume that a subroutine, which we'll call blop, begins with the following code:

```
blop:
        link        a6,#0
        movem.l     d0-d4/a0-a5,-(a7)
```

Here's what happens when the first instruction, link, is executed:

The contents of the address register specified, in this case a6, are saved on the stack. Then, the updated contents of the stack pointer, a7, are copied into a6. Finally, the immediate value specified as the second operand is added to the current contents of the stack pointer. In this case the value is 0, so the value of a7 remains the same.

So now the situation is as follows. Register a6 points to a constant location within the stack: in fact, it points to a copy of its own previous value, which was the last thing stored on the stack. Four bytes "up" from this value is the subroutine return address, which was automatically pushed when the bsr or jsr that got us here was executed. Beginning at an offset of 8 bytes from the current value of a6, we will find any parameters that were pushed just prior to the subroutine call. For example, suppose we call blop with the following sequence of instructions:

```
move.l      d0,-(a7)    push a longword
move.w      d1,-(a7)    push a word
move.l      d2,-(a7)    push a longword
bsr         blop        branch to subroutine
```

Within blop, after the link instruction has been executed, it's easy to access these parameters as follows:

```
move.l      8(a6),d7    get 3rd parm pushed
move.w      12(a6),d6   get 2nd parm pushed
move.l      14(a6),d5   get 1st parm pushed
```

Even though blop saves several registers on the stack immediately after the link is executed, the offsets to a6 remain the same. This is because the stack operations affect the value of a7, the stack pointer—not a6, the "link register" specified.

That's one feature of link: it sets up an address register as a useful marker into the stack, something like a bookmark. Just as important, however, is link's memory-allocation function.

The second operand to link, as mentioned, is an absolute value which is added to the contents of the stack pointer a7. If you consider what the stack looks like just after the execution of a bsr, it's clear that *adding* a value to a7 will probably result in the loss of the subroutine return address, which is the value currently at the top of the stack (before the link is executed). You'll recall that we've been adding values to a7 *after* the execution of subroutines, when the parameters had been pushed on the stack before the call. The result in those cases was that the stack was "cleaned up"; in other words, it was just as if we had laboriously popped all the values we had pushed before the call, and the stack pointer was restored to its original value. Who would want to do this to the stack pointer at the beginning of a subroutine?

The answer is, no one. The immediate value (if it's not 0) in a link is usually negative, which results in its being *subtracted* from the value of a7:

```
link        a6,#-20         allocate stack space
```

Register a6 is saved, then loaded with the updated value of a7, just as before. The 20 is subtracted from the value of a7. This has the effect of reserving 20 bytes on the stack. The next data pushed on the stack after the link has been executed—if it's a word—will be written at an address 22 bytes "lower" than the last data pushed, which was the longword value of a6. (Notice, I said *22* bytes, not *20*. Remember, the stack pointer is decremented by an amount equal to the size of the data being written, before it is written.)

How do we access these 20 bytes? We could use positive offsets from a7. But if we wanted to use the stack for other miscellaneous purposes during the subroutine, we'd have to keep track of any operations that might affect the value of the stack pointer, because they would require the offsets to be changed. An easier way is to use negative offsets from the link register a6. For example:

```
move.l      d0,-20(a6)      write a longword
```

Now you can see how local variables in, say, C could be implemented on a 68000 machine. The number of local variables in a function would be calculated, and the total amount of memory space required would be allocated as a negative immediate value in a link instruction. The variable names could then be equated to negative offsets from the link register. Thus, if you had a variable named glop, it could be written to with the following instruction:

```
move.w      d1,glop(a6)
```

. . .where something like the following data declaration appears elsewhere:

```
glop        equ   -16
```

In future installments of **Assembly line**, we'll come back to this use of link for memory allocation. Right now, we're interested in it mainly as a convenient way to set up a stack marker; when its second operand is 0, nothing is subtracted or added to a7, and it, and the designated link register, have the same value after the link is executed.

The effects of link are undone with unlk, which takes only one operand—namely, the link register. The stack pointer, a7, is loaded with the value from the link register. This restores the situation we had just after the link register was originally saved by link, and deallocates any memory then reserved. Then, the top longword of the stack is popped back into the link register, and everything is just as it was before the link was executed. Therefore, it goes without saying, that you should *never* alter the value of the link register between a link and unlk—at least, not without being extra careful to restore it before the unlk is executed.

Value-Added Addresses.

Our last addressing mode, Address Register Indirect with Index, is not really very different from the Displacement mode we learned last time. The concept is practically the same: a value is added to the contents of an address register, and the result is taken as the "effective address" of the operand. The difference is that the value added, instead of being an unalterable constant coded into the program, is instead contained in a designated data register. This allows you to implement assembly language versions of arrays, where the address register contains the base address of the array, and the contents of the data register—incremented or decremented as need be—are used as an index from the base. Hence, the mode's name.

The format of this mode is easily described. It's an expansion of the Displacement mode format. As before, the base (address) register is enclosed in parentheses; also within the parentheses, however, is the index (data) register, preceded by a comma. The data register is followed by a size specifier: "w" means that the contents of the low word of the register is to be used as the index value; "l" means that the entire longword contents of the register is to be used. You must also indicate a displacement in front of the parentheses, as before. You can specify a zero displacement if you want none. Any address register can be used as a base register, and any data register can be used to hold the index.

Here's how it looks. Suppose you've loaded the base address of a table of byte-size SAT scores into register a0. The following code will read the sixth element of this table into register d2:

```
move.w      #5,d0           load index register
move.b      0(a0,d0.w),d2   read sixth element
```

The index register is loaded with a 5 to get at the sixth element, because we're counting from offset 0, as with a C array. If you wanted to count from offset 1, you could specify a displacement of -1:

```
move.w      #6,d0
move.b      -1(a0,d0.w),d2
```

Note that the d0.w in the parentheses specifies what part of d0 is to be used as an index register; it has nothing to do with the size of the data read, which happens to be a byte here. If, by the way, our table had consisted of word-size elements, then an immediate value of 10 would have been required in d0 (in the first example, with 0 displace-

ment). The technique of taking an ordinal subscript value, and generating from it an offset which can be used as an index into an array of multi-byte-size elements is known as "scaling," a technique we'll be using in the future.

Of course, the real power of this addressing mode is seen in loops, as we'll see in this month's program.

An enciphering program.

This month, I wanted to try something a bit different. The result is a program that enciphers text: that is, it inputs a line of characters you type, terminated by a carriage return, and outputs a line of unintelligible gibberish. The difference between this program and, say, advertising copy is that: (1) the program only does a line at a time; and (2) it follows a set of rules in its enciphering, thus allowing the text to be deciphered . . . I hope. We'll see if that's true in our next installment.

Instead of explaining what happens line by line—the program is really too long for that—I'll briefly describe how it behaves when run, then how it enciphers text—its algorithm, to use the technical term. Our new addressing mode, as well as link, unlk and a few other new (but not nearly so complicated) instructions are used.

The program begins by displaying a brief sign-on message, then waits for your input. It will allow you to type about a line's worth of characters. Hitting RETURN will end the input, or, if you type too many characters, the program will end it for you (that's to prevent the enciphered string from extending into forbidden memory). The TAB key works; BACKSPACE doesn't. This was the best way to keep the program simple, given that we're processing characters keypress by keypress. When input is concluded, the enciphered string will be written on the line below the input string, and the program then ends. To encipher another string, you have to run the program again. I got this dandy idea from my local bank's automatic teller machines.

The method of enciphering is as follows. The characters are input, of course, as ASCII codes. Alphabetic codes are converted into lower case—if necessary—and then, by subtracting the code for lower case "a" into a number from 0 to 25. (Non-alphabetic codes are passed through and not enciphered, to show how ranges of values can be checked for in assembly language.) The converted character code is now used as an index into a table of ASCII character codes (*ciphers* in the program's data segment).

The table consists only of all the lower-case ASCII codes, from a to z, in ascending order. Obviously, if we don't do any further converting, the result of indexing into this table of byte-size values will be that we'll simply get an element containing a code identical to the index.

So a fixed value is added to every index before the table is used. If, for example, the value is 2, then an "a" will be translated into a "c," a "g" into an "i," and so on. If the resulting index is larger than the size of the table (which happens if, say, a "z" is to be translated), then the size of the table is subtracted from the index, resulting in a "wrap-around" effect ("z" is translated into "b," "x" is translated into "a").

However, anyone who's read Edgar Allan Poe's *The Gold Bug* knows that this kind of straight substitution cipher is ridiculously easy to break: you look for the most common letters (usually beginning with "e"), obvious combinations, and so on. Something more is needed.

Instead of having one constant add-in value, we'll use a table of them (incs). Each value will be used for a certain number of characters, then the next value will be used, and so on. When we reach the end of the table, we'll start at the beginning again. Not being a cipher expert, I have no idea how effective this ramshackle scheme really is, but I must admit to a certain satisfaction as I was testing the program whenever I noticed letter pairs that happened to be enciphered into non-matching letters.

Go for it.

Aside from mentioning the instructions and techniques that we haven't used before, that's as far as I'm going to explain the program for now. This is probably all you'll need, although I will go into more detail next time, when I'll also offer a deciphering routine. In the meantime, I think you can have some fun trying to sketch out how to implement the deciphering.

The first thing I want to discuss is how the character code ranges are checked. This occurs between the labels e__nxt0 and e__nxt2. The method is simple, and is used several times. To check if a code lies between, say, "A" and "Z" inclusive, I first compare the code for "A" to it. If the code is lower than "A," it can't possibly be within the range. If it's not lower than "A," I then compare the code for "Z," *plus one*, to it. If the code is lower than this value of ASCII "Z" plus one (which happens to be the code for "["), then I know that the code does fall within the A to Z range; otherwise, it doesn't.

As you might expect, a cmp instruction is used to implement these comparisons. However, we're not checking for simple equality of operands, as we were in previous programs; here, we're checking to see if the destination operand is less than the source operand (an immediate value, which explains the cmpi, "compare immediate value"). Checking the zero flag in the Condition Codes Register won't do us any good. Instead, we check the carry flag. It tells us whether or not the last instruction generated a "carry" or "borrow," which happens when a value is subtracted from a smaller value.

Suppose you wrote down on paper the simple subtraction of 9 from 10. Starting with the rightmost column, you find that 9 can't be subtracted from 0, so you borrow 10 from the left column, and 10 minus 9 is 1. In this case, the 10 was there to be borrowed. On the 68000, the carry flag would be set if 9 was subtracted from 0: a borrow was required, but was unavailable. Thus, the state of the carry flag after a compare or subtract will tell us if the value being subtracted was greater than the second operand. But, *only* if it was greater. That's why I check the top end of the range with an immediate value one *greater* than the range itself. If I used the ASCII value of "Z" to check the upper limit in the example above, the value of "Z" itself would never be detected as being in range. Subtracting ASCII "Z" from ASCII "Z" will set the zero flag, but clear the carry flag.

Thus, the cmpi instructions are followed by bcc (branch on carry clear), or bcs (branch on carry set) instructions,

as appropriate. There are many other ways to check value ranges, using other CCR flags, but this seemed the simplest to me at the time. I'll have more to say about binary arithmetic in future installments.

Under label e_nxt2, you can see how I get the current increment from incs and add it to the ASCII code to be enciphered. If the result is greater than the size of the table, I just subtract (with a subi, "subtract immediate value" instruction) the size of the table from it. This results in the "wrap-around" effect I mentioned above. The technique is also used under e_raw on the index into incs itself. By the way, note how nice it is that our tables all have byte-size elements: each element's ordinal number is the same as its offset in the table. I must warn you, things won't always be this easy.

Finally, there are a couple of novelties in the data area at the bottom of the program. First, there are two "variables," space reserved to be written to as well as read from: last_ch and c_string. This space is reserved with the ds (define storage) directive, which is followed by a dot and a size specifier. Then, after at least one space, comes the number of elements to be reserved. One byte is reserved for last_ch (which always contains the last character typed; this allows me to override the action of BACKSPACE, by simply reprinting the last character), and 100 for c_string, where the enciphered string is written, prior to its being displayed on the screen. (One hundred is 20 too many, but I'm excessively cautious in these matters.)

The string and table size constants (S_SIZE, C_MAX and I_COUNT) are defined by having the assembler calculate the number of bytes between the current value of the assembler's location counter and the value of the table's base address (symbolized by its label—for example, ciphers). The nice thing about this is that you can change the size of the table simply by inserting or deleting values; the assembler will calculate the new size. Again, the operation is easy here because the elements are byte-size.

We'll continue with this project next time. Until then—uoou mcempun. //

Listing 1.
Assembly listing.

```
*****************************************************************
*
* A Text-Enciphering Program: Part 1
*
*   by Douglas Weir  Copyright (c) 1987 ST-Log
*
*****************************************************************

        text

        move.l          #sign_on,-(a7)          sign on message
        move.w          #9,-(a7)                code=print string
        trap            #1                      do it
        addq.l          #6,a7                   pop args

        move.l          #c_string,-(a7)         string space address
        bsr             encipher                encipher a string
        addq.l          #4,a7                   pop arg

        move.l          #p_string,-(a7)         print string address
        move.w          #9,-(a7)                code=print string
        trap            #1                      do it
        move.w          #0,-(a7)                code=exit program
        trap            #1                      do it

encipher:
        link            a6,#0                   frame pointer
        movem.l         d0-d4/a0-a5,-(a7)       save registers

        movea.l         8(a6),a4                point to string space
        movea.l         #ciphers,a3             point to cipher table
        movea.l         #incs,a5                point to increments
        clr.l           d2                      reset inc index
        move.w          #I_PHASE,d3             reset phase counter
        move.w          #S_SIZE,d4              absolute counter
        move.b          #CR,last_ch             initialize last char
e_loop:
        move.w          #1,-(a7)                code=conin
        trap            #1                      do it
        addq.l          #2,a7                   pop arg
        cmpi.b          #CR,d0                  carriage return?
        beq             e_end                   if so
        cmpi.b          #TAB,d0                 tab?
        bne.b           e_nxt00                 if not
        move.b          #SPACE,d0               else insert space
        move.b          d0,last_ch              last char too
        bra.b           e_raw                   and write it

e_nxt00:
        cmpi.b          #BS,d0                  backspace?
        bne.b           e_nxt0                  if not
        move.b          last_ch,d0              else get last char
        move.w          d0,-(a7)                push it
        move.w          #2,-(a7)                code=conout
        trap            #1                      do it
        addq.l          #4,a7                   pop args
        bra.b           e_loop                  and start over
e_nxt0:
        move.b          d0,last_ch              save raw char
        cmpi.b          #A,d0                   lower than 'A'?
        bcs.b           e_raw                   if so, just write it
        cmpi.b          #Z1,d0                  else: upper case?
        bcc.b           e_nxt1                  if not
        addi.w          #U_CONV,d0              else convert to lower
        bra.b           e_nxt2                  case and continue
e_nxt1:
        cmpi.w          #a,d0                   non-alphabetic?
        bcs.b           e_raw                   if so, write it
        cmpi.w          #z1,d0                  else: lower case?
        bcc.b           e_raw                   if not, write it as is

e_nxt2:
        subi.w          #C_CONV,d0              else make it an index
        move.b          0(a5,d2.w),d1           get current increment
        add.w           d1,d0                   add it to index
        cmpi.w          #C_MAX,d0               over maximum?
        bcs.b           e_nxt3                  if not
        subi.w          #C_MAX,d0               else correct it

e_nxt3:
        move.b          0(a3,d0.w),d0           get enciphered code
e_raw:  move.b          d0,(a4)+                write it
        subq.w          #1,d3                   decrement phase counter
        bne.b           e_test                  if phase not exhausted
        move.w          #I_PHASE,d3             else restart counter
        addq.l          #1,d2                   and increment index
        cmpi.w          #I_COUNT,d2             over maximum?
        bcs.b           e_test                  if not
        subi.w          #I_COUNT,d2             else wrap around

e_test:
        dbra            d4,e_loop               go till end

e_end:
        move.b          #LF,(a4)+               append line feed
        move.b          #CR,(a4)+               and carriage return
        move.b          #0,(a4)                 and null

        movem.l         (a7)+,d0-d4/a0-a5       restore registers
        unlk            a6                      deallocate frame
        rts                                     and return

        data

sign_on         dc.b    'encipher...',10,13,0
last_ch         ds.b    1               holds last char typed

p_string        dc.b    10,13           line feed, carriage return
c_string        ds.b    100             space for enciphered string
S_SIZE          equ     *-c_string      length of string

ciphers         dc.b    'abcdefghijklmnopqrstuvwxyz'
C_MAX           equ     *-ciphers       length of table
C_CONV          equ     97              'a' - 97 = 0
U_CONV          equ     32              'A' + 32 = 'a'

incs            dc.b    10,5,2,7,9,1    ciphering increments
I_COUNT         equ     *-incs          size of table
I_PHASE         equ     3               3 chars enciphered per inc

A               equ     65              ASCII 'A'
Z1              equ     91              ASCII 'Z' + 1
a               equ     97              ASCII 'a'
z1              equ     123             ASCII 'z' + 1

CR              equ     13              carriage return
LF              equ     10              line feed
TAB             equ     9               tab
BS              equ     8               backspace
SPACE           equ     32              space
```

C-manship

Using sliders and arrows to change a window's contents.

by Clayton Walnum

We've stated previously that a window is just a box that allows us to display data in a convenient manner, and the programmer is completely responsible for what is done with the window's work area. One of the things that GEM's windows provide, to help the user manipulate the displayed data, is the slider/arrow system. By moving the sliders or clicking the arrows, the user can "move" the data within the window to any position he likes.

This convenience is paid for by the programmer, however, because, when a slider or arrow is used, GEM doesn't do anything except send a message to the program. It's up to the programmer to decide what to do with the message and how to update the window's display.

The listing.

Type in Listing 1 and compile it. (You can also find the source code and compiled program on this month's disk version, or on the ANALOG Atari user group on Delphi.) Please note that the program was developed using Megamax C. If you have a different compiler you may have to make some small changes to the listing.

When the program is run, a window will be opened, the directory of the default drive (the one you ran the program from) will be read, and the filenames found there displayed in the window's work area. You may then use the sliders and arrows in their conventional way to move the data within the window. You can also enlarge or shrink the window by dragging (with the mouse, of course) the lower right corner of the window.

Note that the example program only provides the vertical arrows and sliders. I didn't include the horizontal ones because they're handled almost exactly the same way as their vertical counterparts.

Getting a directory.

The first thing of interest in the sample program is the method with which we can read a disk's directory. The code to accomplish this can be found in the get_fnames() function in Listing 1. Let's take a look at that now.

The first thing we must do is initialize a couple of variables. We'll be using the integer p as an array index, and the integer files.count (this is a member of the structure files, which is declared near the top of the listing) will contain the number of filenames read from the directory.

Next, we set an important address with the call:

```
Fsetdta ( dta );
```

Here, dta is a pointer to character data (in our case, the address of the character array dta[]). The function Fsetdta() is GEMDOS function 0x1a and is declared in OSBIND.H. It sets the address of the DTA (Disk Transfer Address), a 44-byte buffer in which the directory data is stored. We supplied the buffer by defining the array dta[].

When we get around to actually reading a filename, the DTA will contain all sorts of useful information, as shown here:

Byte	Contains
0 - 20	For OS use only
21	File attribute
22 - 23	Integer, file time stamp
24 - 25	Integer, file date stamp
26 - 29	Long integer, file size
30 - 43	Filename

So let's fill that DTA, shall we? We get the first filename with the call:

```
end = Fsfirst ( p, a );
```

Here, p is the pathname you want to use for the file search and a is an integer whose bit settings determine the search's attributes. The function call returns a negative integer in

the case of an error (for instance, when there are no more filenames to be read). In the sample listing, we placed the pathname string, "*.*", directly into the call. This pathname takes advantage of "wild cards," so the search will match any file. The attributes set by a are determined as follows:

Bit	Result
0	Search limited to normal files
1	Read only files included
2	Hidden files included
3	System files included
4	Folder names included
5	Subdirectory files included

By setting bits *0* and *4* in the search attributes argument, we'll read all the file and folder names from the root directory (and, because of the pathname, these will be read from the default drive), just as if you had just opened the drive from the desktop.

Now that we've got the first filename, we set up a while loop to get the rest, as well as to process the filenames into the form we need, later storing them into the two-dimensional array files.fnames[][]. All we're doing in the processing is making sure each entry in the filename array is exactly *15* characters long: the filename padded with spaces and ending with a null.

The rest of the filenames are retrieved, one by one, with the call:

```
end = Fsnext ();
```

This function (GEMDOS 0x4f), defined in OSBIND.H, also returns a negative value for an error condition.

We continue executing the while loop until end becomes negative, or files.count becomes greater than MAX.

Slipping and sliding.

Now that we've got all those filenames stored, we're ready to open our window. We've done all this stuff before; there's nothing new here, except the use of the integer top to keep track of where in the filename list the window's data display is to start (remember, the topmost filename in the display won't necessarily be the first one in our list), and setting up the sliders.

Once we open the window, we need to set the size and position of the slider. Our function calc_slid() takes care of this, requiring three integers as arguments: the handle of the window; the total number of text lines (in this case, the number of filenames in the list); and the total width of the text in columns.

The function first calls wind_get() to get the size of the window's work area, then calculates the number of lines and columns that'll fit that area. The size of the slider is then calculated like this:

```
size = 1000*lines_avail/line_count;
```

The size of the slider can range anywhere from *1* to *1000*, and represents the relative portion of the document displayed. By dividing the number of lines available in the window by the number of lines in the "document," we end up with a value representing the portion of the document that'll fit the window. Multiplying this value by *1000* will give us the equivalent size of the slider.

For instance, let's say we have 10 lines to display, but the window can hold only 6. Dividing 6 by 10 gives us *.6*—the portion of the document that the window can display. When we multiply this value by *1000*, we get *600*—the slider's relative size. Keep in mind that, for greater accuracy, you might want to use floating-point math, rather than integer math.

Once the size of the slider is calculated, it's set with the call:

```
wind_set(w_h,WF_VSLSIZE,size,0,0,0);
```

Here, w_h is the window's handle, WF_VSLSIZE is the function's operation flag (WF_HSLSIZE for horizontal sliders) as defined in GEMDEFS.H, size is an integer value between *1* and *1000*, and the three zeroes are unused arguments.

Next, we need to calculate the slider's position within its track, which is also a value between *1* and *1000*. The following calculation, done with floating-point math, the result of which is cast to an integer, does this:

```
pos = (int) (float) top / (float)
(line_cnt-lines_avail) * 1000;
```

The integer top is the number of the uppermost displayed line in the window, line_cnt is the total number of lines in the document, and lines_avail is the number of lines that'll fit the window.

The slider's position is then set with the call:

```
wind_set(w_h,WF_VSLIDE,pos,0,0,0);
```

Here, w_h is the window's handle, WF_VSLIDE is the function's operation flag (WF_HSLIDE for horizontal sliders) as defined in GEMDEFS.H, pos is an integer between *1* and *1000*, and the three zeroes are unused arguments.

Me and my arrow.

Whenever the user clicks on one of the arrows, or in the slider's tracks, the program will receive a *WM_ARROWED* message. This message is a little different from the others we've worked with. It actually contains a sub-message, which GEM stores in msg_buf[4]. This forces us to do a little more work before we can take care of the user's request. This extra work is tackled in the function do_arrow(), where we use the value stored in msg_buf[4] to determine exactly what the user wants to do.

If the user has clicked on the down arrow, msg_buf[4] will contain a *WA_DNLINE* message, and program execution will continue with the function do_dnline().

Before we look at that function, let's stop and think about what we're doing. What exactly is the user requesting when he clicks on the down arrow?

Generally, it means that we must move the window's display one unit upward and the slider one unit downward. Exactly what that "unit" is depends a great deal on your application. If our window contained some sort of graphic information, such as a map, a unit could be anything from a single scan line to dozens of scan lines. Luckily, our decision is a little easier. We're working with text, so the obvious unit of measurement is a text line.

We know now what we want to do, but how are we going to go about it? The easy way of updating the display would be just to increment top, then call draw_interior() to redraw the window's work area. The problem with that is that it's

too slow, too sloppy. We're going to need something much more elegant. When you're working with the desktop's file windows, the text displays move neatly upward one line each time you click on the down arrow; you can't see the redrawing.

Almost all the information we need is on the screen, right? Only the bottom of the new list isn't shown. You know what that means? We can update most of the window using raster operations. All we have to do is "blit" the area of the window from the second filename down to a position one text line higher, then fill in the bottom with the new filename. And that's exactly what we do in do__dnline.

Let's run through that function now. First, we use wind__get() to get the size of the window's work area, then we calculate the number of text lines that'll fit. Armed with that information, we use an if statement to make sure we don't bother updating the window if there are no filenames left to display. In other words, if the last file in our list is already shown in the window, we want to ignore the request to scroll downward.

If our calculations show that the arrow message is okay to process, we increment top, calculate where in our list of filenames we'll find the new data that needs to be displayed, set clipping to on, turn off the mouse and do our raster stuff.

Because—depending on the size of the window—we may have a partial filename displayed at the bottom, we actually have to print two filenames, and if we're at the end of the list, the second filename should be a line of spaces. Here's what happens in our function:

If the window is a size in which an even number of filenames will fit, our code will blit the display up one line, then print at the bottom the next two filenames in the list. The first will go at the very bottom line, and the second won't appear at all, because we're trying to print it outside of the clipping area.

Now, let's take a case where the size of the window allows a partial filename to show at the bottom. When we do our calculations for the number of lines that'll fit the window, the result is an integer, which means the decimal portion has been truncated (i.e., rounded down to the nearest integer). Because of this, only the number of *complete* lines that'll fit the window are counted. That's why we always want to print two filenames, since the second may represent one that is only partially visible.

In the case of a partially displayed filename, we'll actually have to print one and a half filenames. Sound tricky? Naw. The clipping rectangle makes this little complication easy to handle. We just print two filenames, and anything that lies outside of the area gets clipped off, leaving us with a partial filename (the second one we printed) at the bottom.

Once we've gotten our display written, we have to recalculate the position of the slider. This is done the same way we did it when we first opened our window, with a call to our function calc__slid().

That completes the processing of the down arrow request. The function do__upline does the same thing for the *WA__UPLINE* message as do__dnline did for the *WA__DNLINE* message, except the process is reversed—and a little simpler. In do__upline we're rastering the window's work area down one line, then printing the next filename (thinking backwards) at the top. Because we'll never have a partial line here, we don't have the extra complications we had with do__dnline.

Paging all sliders.

Two other messages we may receive from the original *WM__ARROWED* message (not including messages for horizontal sliders and arrows, which are *WA__LFPAGE, WA__RTPAGE, WA__LFLINE* and *WA__RTLINE*) are *WA__DNPAGE* and *WA__UPPAGE*. These are sent to us by GEM whenever the user clicks in the slider's track, indicating that he wants the next "page" of information, the next windowful of lines following that already shown in the window.

Because the information we want displayed in the window is not available anywhere on the screen, we can't use raster operations. We have to do it the sloppy way: figure out the new line for the top of the window, then call our function draw__interior to do the work. But there are a couple of things we have to watch out for when calculating the new top. We have to make sure that top doesn't end up less than zero and that it doesn't become a value that will cause our display to go beyond the bottom of our text.

Let's look at the function do__dnpage. First, we use a wind__get() call to find the size of the window's work area, then we calculate the number of lines that'll fit. Since we want to move down that number of lines in the document, all we have to do to calculate the new value for top is to

add lines_avail to its existing value. The only thing to check for is the bottom of the document. If our new top, plus the number of lines in the display equal a value greater than the total number of lines in the document, we have to set back the value of top in such a way that the last line of our document will also be the bottom line of the window. Not too tricky, really.

The function do_uppage() (which executes when we receive a *WA_UPPAGE* message) works in the same manner, except we *subtract* lines_avail from top, then check to make sure top isn't less than zero.

Anywhere you like.

Another way the user can change the window's display is to grab the slider with the mouse pointer and move it to a new position. When this occurs we get a *WM_VSLID* message from GEM (or WM_HSLID, if it's a horizontal slider).

This message is almost as easy to handle as the *WA_UPPAGE* and *WA_DNPAGE* messages. The key thing to know here is that the slider's new position is returned in msg_buf[4]. To find the corresponding position in our document, all we have to do is get the size of the window's work area, calculate the number of lines that'll fit the window, then perform the following calculation:

```
top = msg_buf[4] *
(line_cnt - lines_avail) / 1000;
```

We then set the slider to its new position with the call:

```
wind_set(w_h,WF_VSLIDE,
msg_buf[4],0,0,0);
```

A call to our function draw_interior (which will use the new value calculated for top) completes the task.

An important note.

You should be aware that, many times, the calculations for finding the size and position of the sliders may have to be done with 32-bit math (using long integers) to avoid overflow problems, or with floating-point math when you need greater accuracy. Ignoring these possibilities could give you some perplexing results. If ever you find your sliders behaving mysteriously—and you're sure your logic is correct—check your math. ///

Listing 1.
C listing.

```
/************************************************/
/*            C-manship, Listing 1              */
/*                  ST-Log #19                  */
/*          Developed with Megamax C            */
/************************************************/

#include <gemdefs.h>
#include <obdefs.h>
#include <gembind.h>
#include <osbind.h>

#define TRUE  1
#define FALSE 0
#define PARTS NAME|CLOSER|SIZER|UPARROW|DNARROW|VSLIDE
#define MAX 50
#define SOLID 1
#define MIN_WIDTH 64
#define MIN_HEIGHT 64

/* GEM arrays. */
int work_in[11],
    work_out[57],
    contrl[12],
    intin[128],
    ptsin[128],
    intout[128],
    ptsout[128];

/* Global variables. */
int handle, w_h, top,
    fullx, fully, fullw, fullh,
    char_w, char_h, box_w, box_h,
    wrkx, wrky, wrkw, wrkh;

/* Message buffer. */
int msg_buf[8];

struct {
   char fnames[MAX][15]; /* Char array for filenames. */
   int  count;           /* Number of filenames read. */
} files;
```

```
/* Window title. */
char *title = "C-manship";


main ()
{
   appl_init ();        /* Initialize application.        */
   open_vwork ();       /* Set up workstation.            */
   do_wndw();           /* Go do the window stuff.        */
   v_clsvwk (handle);   /* Close virtual workstation.     */
   appl_exit ();        /* Back to the desktop.           */
}


open_vwork ()
{
   int i;

   handle = graf_handle ( &char_w, &char_h, &box_w, &box_h);
   for ( i=0; i<10; work_in[i++] = 1 );
   work_in[10] = 2;
   v_opnvwk ( work_in, &handle, work_out );
}


do_wndw ()
{
   top = 0;

   get_fnames ();
   wind_get ( 0, WF_WORKXYWH, &fullx, &fully, &fullw, &fullh );
   w_h = wind_create ( PARTS, fullx, fully, fullw, fullh );
   wind_set ( w_h, WF_NAME, title, 0, 0 );
   wind_open ( w_h, 100, 20, 150, 151 );
   calc_slid ( w_h, files.count, 14 );
   graf_mouse ( ARROW, 0L );

   do {
      evnt_mesag ( msg_buf );
      switch ( msg_buf[0] ) { /* msg_buf[0] is message type. */

         case WM_SIZED:
            do_move ();


            break;

         case WM_ARROWED:
            do_arrow ();
            break;

         case WM_VSLID:
            do_vslide ();
            break;

         case WM_REDRAW:
            do_redraw ( (GRECT *) &msg_buf[4] );
            break;
      }
   }
   while ( msg_buf[0] != WM_CLOSED );

   wind_close ( w_h );
   wind_delete ( w_h );
}


do_arrow ()
{
   switch ( msg_buf[4] ) {
```

```
        case WA_UPPAGE:
           do_uppage ();
           break;

        case WA_DNPAGE:
           do_dnpage ();
           break;

        case WA_UPLINE:
           do_upline ();
           break;

        case WA_DNLINE:
           do_dnline ();
           break;

        }
}


do_vslide ()
{
    GRECT r;
    int lines_avail;

    wind_get ( w_h, WF_WORKXYWH, &r.g_x, &r.g_y, &r.g_w, &r.g_h );
    lines_avail = r.g_h / char_h;
    top = msg_buf[4] * (files.count - lines_avail) / 1000;
    wind_set ( w_h, WF_VSLIDE, msg_buf[4], 0, 0, 0 );
    draw_interior ( r );
}


do_uppage ()
{
     GRECT r;
     int lines_avail;

     wind_get ( w_h, WF_WORKXYWH, &r.g_x, &r.g_y, &r.g_w, &r.g_h );
     lines_avail = r.g_h / char_h;
     top -= lines_avail;
     if ( top < 0 )
       top = 0;
     draw_interior ( r );
}


do_dnpage ()
{
    GRECT r;
    int lines_avail;

     wind_get ( w_h, WF_WORKXYWH, &r.g_x, &r.g_y, &r.g_w, &r.g_h );
     lines_avail = r.g_h / char_h;
     top += lines_avail;
     if ( top > files.count - lines_avail )
       top = files.count - lines_avail;
     draw_interior ( r );
}
do_upline ()
{
    FDB s, d;
    GRECT r;
    int pxy[8];

    if ( top != 0 ) {
       top -= 1;
       wind_get ( w_h, WF_WORKXYWH, &r.g_x, &r.g_y, &r.g_w, &r.g_h );
       set_clip ( TRUE, r );
       graf_mouse ( M_OFF, 0L );
```

```
        s.fd_addr = 0L;
        d.fd_addr = 0L;
        pxy[0] = r.g_x;
        pxy[1] = r.g_y + 1;
        pxy[2] = r.g_x + r.g_w;
        pxy[3] = r.g_y + r.g_h - char_h - 1;
        pxy[4] = r.g_x;
        pxy[5] = r.g_y + char_h + 1;
        pxy[6] = r.g_x + r.g_w;
        pxy[7] = r.g_y + r.g_h - 1;
        vro_cpyfm ( handle, S_ONLY, pxy, &s, &d );
        v_gtext ( handle, r.g_x+char_w, r.g_y+char_h,
                        &files.fnames[top][0] );
        set_clip ( FALSE, r );
        calc_slid ( w_h, files.count, 14 );
        graf_mouse ( M_ON, 0L );
    }
}


do_dnline ()
{
    FDB s, d;
    GRECT r;
    int pxy[8];
    int lines_avail, index;

    wind_get ( w_h, WF_WORKXYWH, &r.g_x, &r.g_y, &r.g_w, &r.g_h );
    lines_avail = r.g_h / char_h;
    if ( (top + lines_avail) < files.count ) {
        top += 1;
        index = top + lines_avail - 1;
        set_clip ( TRUE, r );
        graf_mouse ( M_OFF, 0L );
        s.fd_addr = 0L;
        d.fd_addr = 0L;
        pxy[0] = r.g_x;
        pxy[1] = r.g_y + char_h + 1;
        pxy[2] = r.g_x + r.g_w;
        pxy[3] = r.g_y + r.g_h - 1;
        pxy[4] = r.g_x;
        pxy[5] = r.g_y + 1;
        pxy[6] = r.g_x + r.g_w;
        pxy[7] = r.g_y + r.g_h - char_h - 1;
        vro_cpyfm ( handle, S_ONLY, pxy, &s, &d );
        v_gtext ( handle, r.g_x+char_w, r.g_y+(lines_avail)*char_h,
                        &files.fnames[index][0] );
        if ( index != files.count-1 )
            v_gtext ( handle, r.g_x+char_w, r.g_y+(lines_avail)*char_h+char_h,
                            &files.fnames[index+1][0] );
        else
            v_gtext ( handle, r.g_x+char_w, r.g_y+(lines_avail)*char_h+char_h,
                            "                    " );
        set_clip ( FALSE, r );
        calc_slid ( w_h, files.count, 14 );
        graf_mouse ( M_ON, 0L );
    }
}


get_fnames ()
{
    char dta[44];
    int end, p, x, null_found;

    p = 0;
    files.count = 0;
    Fsetdta ( dta );
    end = Fsfirst ( "*.*", 17 );

    while ( (end > -1) && (files.count <= MAX) ) {
```

```
        null_found = FALSE;
        files.count += 1;
        for ( x=0; x<14; ++x ) {
           if ( dta[30+x] == 0 )
              null_found = TRUE;
           if ( null_found )
              dta[30+x] = ' ';
           files.fnames[p][x] = dta[30+x];
        }
        files.fnames[p][14] = 0;
        p += 1;
        end = Fsnext ();
     }
}


calc_slid ( w_h, line_cnt, col_cnt )
int w_h, line_cnt, col_cnt;
{
   int lines_avail, cols_avail, vslid_siz, pos;

   wind_get ( w_h, WF_WORKXYWH, &wrkx, &wrky, &wrkw, &wrkh );
   lines_avail = wrkh / char_h;
   cols_avail = wrkw / char_w;
   vslid_siz = 1000 * lines_avail / line_cnt;
   wind_set ( w_h, WF_VSLSIZE, vslid_siz, 0, 0, 0 );
   pos = (int) ( (float)(top) ) / ( (float)(files.count - lines_avail) ) * 1000;
   wind_set ( w_h, WF_VSLIDE, pos, 0, 0, 0 );
}


do_move()
{
   if ( msg_buf[6] < MIN_WIDTH )
      msg_buf[6] = MIN_WIDTH;
   if ( msg_buf[7] < MIN_HEIGHT )
      msg_buf[7] = MIN_HEIGHT;
   wind_set ( msg_buf[3], WF_CURRXYWH,
              msg_buf[4], msg_buf[5], msg_buf[6], msg_buf[7] );
   calc_slid ( w_h, files.count, 14 );
}


draw_interior ( clip )
GRECT clip;
{
   int pxy[4];
   int x, lines_avail, lines_shown;

   graf_mouse ( M_OFF, 0L );
   set_clip ( TRUE, clip );
   wind_get ( msg_buf[3], WF_WORKXYWH, &wrkx, &wrky, &wrkw, &wrkh );

   vsf_interior ( handle, SOLID );
   vsf_color ( handle, WHITE );
   pxy[0] = wrkx;
   pxy[1] = wrky;
   pxy[2] = wrkx + wrkw - 1;
   pxy[3] = wrky + wrkh - 1;
   vr_recfl ( handle, pxy );

   lines_avail = wrkh / char_h;
   lines_shown = files.count - top;
   if ( lines_avail > lines_shown ) {
      top = files.count - lines_avail;
      if ( top < 0 )
         top = 0;
   }

   for ( x=top; x<files.count; ++x )
```

```
        v_gtext ( handle, wrkx+8, wrky+(x+1-top)*char_h,
                  &files.fnames[x][0] );

    set_clip ( FALSE, clip );
    calc_slid ( w_h, files.count, 14 );
    graf_mouse ( M_ON, 0L );
}


do_redraw ( rec1 )
GRECT *rec1;
{

    GRECT rec2;
    wind_update ( BEG_UPDATE );
    wind_get ( msg_buf[3], WF_FIRSTXYWH,
               &rec2.g_x, &rec2.g_y, &rec2.g_w, &rec2.g_h );

    while ( rec2.g_w && rec2.g_h ) {
       if ( rc_intersect ( rec1, &rec2 ) )
          draw_interior ( rec2 );
       wind_get ( msg_buf[3], WF_NEXTXYWH,
                  &rec2.g_x, &rec2.g_y, &rec2.g_w, &rec2.g_h );
    }

    wind_update ( END_UPDATE );

}


set_clip ( flag, rec )
int flag;
GRECT rec;
{
    int pxy[4];

    pxy[0] = rec.g_x;
    pxy[1] = rec.g_y;
    pxy[2] = rec.g_x + rec.g_w - 1;
    pxy[3] = rec.g_y + rec.g_h - 1;
    vs_clip ( handle, flag, pxy );
}
```

C-manship

The ins and outs of GFA BASIC.

by Ian Chadwick

This is one of two columns in which we'll explore the options of getting data into your program, then getting output. This issue, we cover the former: input. This doesn't cover input through alerts or dialog boxes. Alerts were discussed in a previous column; dialog boxes will be covered later. What we'll discuss here is input through the keyboard.

There are two basic types of input statements: those that stop program execution and wait for input before continuing, and those that do it "on the fly," without interrupting execution:

Program Stops	On the fly
INP	INKEY$
INPUT	
INPUT#	
FORM INPUT	
LINE INPUT	
LINE INPUT#	

There are also commands to read data from files (GET and BGET), which will be discussed in a later column about I/O and files, along with the input commands above suffixed with a number sign (INPUT#, LINE INPUT#). These function the same as their basic commands, except that they wait for input from a previously opened data channel, rather than through the keyboard.

INPUT is the essential command. It allows the user to enter one or more variables—numeric and/or strings can be combined here—as long as each variable in the command is separated by a comma. The number of variables possible is limited to the line length, but I've managed to fit sixty-nine in a single statement. Of course, this is a pointless academic exercise—you want to limit the input to something more readily usable: three or four variables might be best. For example:

```
INPUT CASH_VALUE, SALES_PRICE
INPUT A, B$, C, D$
INPUT NAME$, AGE
```

Multiple variables can either be entered by the user on the same line, separated by commas; or one after another, one variable at a time, by pressing RETURN for each one. Both the comma and RETURN are valid separators in data.

INPUT can also be attached to its own text prompt, which is displayed before the data is requested:

```
INPUT "What is your name and age";name$,age
INPUT "Enter item name, part number and
quantity ", part$,pnum, pquant
```

If the text is separated from the variable by a semicolon, then a question mark is added at the right end of the text. Otherwise, the variable information is typed in immediately after the text, so add a space before the final quotation to separate the two:

```
INPUT "Enter your age ",age
```

Input variables are limited to 255 characters and are terminated by pressing RETURN, which either signifies to continue the program, or, if there are more variables to enter, get the next.

A problem arises when you want to include commas in your data. You can splice them in using MID$, but that's not always effective. Instead, you can use LINE INPUT, which functions like INPUT, except it works for strings only and accepts commas as valid characters in the data or string (only RETURN is recognized as a separator). Be careful when asking for more than one variable in a single statement, since each string *must* be entered on its own line.

```
LINE INPUT "Enter your name and address",
Name$, Address$
```

In all other features, LINE INPUT functions identically to INPUT. FORM INPUT is another variation on INPUT with two important differences: you can specify the number of characters in the string, thus limiting the entry for use in applications such as databases. This is particularly important when creating fixed size records on disk. It is often associated with the FIELD command.

```
FORM INPUT 20, Address$
```

The other difference is that you cannot tag a text line with FORM INPUT. If you want a text prompt, you must use the PRINT statement first.

FORM INPUT doesn't work with a data channel, as do INPUT# and LINE INPUT#. This means it's not good for reading data from a file on disk or from another peripheral. Entry is only through the keyboard. However, data input with this command can be written to disk with no problems.

FORM INPUT AS is not an input command *per se*, but rather a function to change existing strings from user keyboard input. As such, it will be discussed in a later column on string handling.

These three commands are pretty much alike, except for the minor differences described above. You can enter special characters into all three forms by pressing ALT, CONTROL-S or CONTROL-A, as described in the GFA BASIC manual.

INP is somewhat different from INPUT. First, it accepts only a single byte, which is fine for character input, but poor for a lot of disk I/O, and simply not suitable for long string data. That doesn't mean it has no strengths. Since it reads a byte, rather than a character or string, it can read the keyboard keys you can't use for other types of input, including the function and cursor keys, and HELP and UNDO. For example:

```
OPEN "",#1, "CON:"
DO
   A=INP(#1)
   PRINT A, CHR$(A)
   EXIT IF A=32
LOOP
CLOSE #1
END
```

This simply reads the keyboard and tells you what was pressed. If it was the SPACE BAR, that ends the program. INP offers an alternate way of handling input from peripherals, so you don't need to OPEN a channel for a device:

```
DO
   A=INP(2)
   PRINT A, CHR$(A)
   EXIT IF A=32
LOOP
END
```

Aside from this small convenience, the two examples perform the same. You'll note, by the way, when you try this, that the function and SHIFT-function keys produce different results, but that CONTROL-function and ALT-function don't. Experiment a bit to see what results you get from different key combinations.

It's important to note that, unlike the other input functions, INP doesn't have data separators and doesn't wait for RETURN to be pressed. RETURN is treated like any other character.

INP can also be directed to take data from an opened channel like INPUT and LINE INPUT, using INP#. However, if the keyboard is the channel you want to use, then INP(2) is slightly more efficient and shorter.

INKEY$ is a lot like INP in that it reads a single character (*not* byte, although a character in this case is a byte), rather than a string or large number like INPUT and its variations. However, it does it immediately—on the fly, if you will—without waiting for anything to happen. If nothing is pressed, then it continues on. For example:

```
DO
   A$=INKEY$
   PRINT A$,ASC(A$)
   EXIT IF ASC(A$)=32
LOOP
```

This streams by your screen pretty quickly, but you get the drift. The power in INKEY$ is that it works without waiting, as you can see by this simple loop. What you don't see properly is that function, cursor, HELP and UNDO keys produce a two-character string. The first character is always the ASCII code for 0. In order to see the two codes produced, try this example:

```
DO
  REPEAT
    A$=INKEY$
  UNTIL A$<>""
  IF LEN(A$)=2
    PRINT LEFT$(A$),ASC(A$),RIGHT$(A$),
  ASC(RIGHT$(A$)) ELSE
    PRINT A$,ASC(A$)
ENDIF
```

continued

```
    EXIT IF ASC(A$)=32
  LOOP
```

This shows you the function key codes. It also pauses execution, thanks to the REPEAT loop, so that you can see each key as it's pressed individually. In order to get the ASCII code for any two-character key, use:

```
KEY=CHR$(0)+CHR$(ASC(A$))
```

Which method of input you use depends, of course, on what you want to do with the data and how you want to get it. Obviously, if you need strings and multiple entries, then INP and INKEY$ are inadequate.

New products.

MichTron recently sent me a copy of their latest effort, GFA Companion, a Resource Construction Set (RCS) for GFA BASIC. What it does is create dialog boxes, error messages and text boxes with buttons and sliders that you can incorporate within your programs. The process is *vastly* simplified using Companion, as opposed to trying to hack it out with the language and an ST Developer's Kit.

The code generated is an ASCII .LST code you can merge into your own programs. Very easy to use, although the documentation is far too vague and thin for a product of this sort. You'd be well advised to try several examples and read the code they generate, as well as look through the examples provided. The code comes out commented—a nice touch which makes it fairly easy to use and alter. However, more effort in the manual would have been appreciated.

There are some small drawbacks, which really do not detract from the program in any way. First, you can only have a single slider in a box and a limited number of buttons. If you want more, you'll have to hack the source code and create your own. This isn't terribly difficult, but it's an annoying limitation. A public domain program on CompuServe called DIOX lets you create dialogs with 128 radio buttons, so you might look into it if you need more than the seven allowed by Companion.

The other problem is that the source code cannot accompany any program you create unless compiled or PSAVED (protection-saved). I understand the rationale for this step, but I don't agree with its necessity, except as far as their own comments in the output go. I feel this reduces the friendliness of the GFA environment and restricts the product's usefulness.

I like to be able to read other people's code, examine it, and sometimes tinker with it. Not with Companion's output. This isn't a problem if you're planning to compile everything and not sell or give out the source code, but it discourages its use—and you won't see output examples on GEnie, Delphi or CompuServe. A simple request for a line about code being produced with the Companion would have—I think—served them better.

The program comes with a tutorial consisting of several short lessons. I found this of dubious value. I'm not sure I understand why they bothered with it. I'd much rather have seen them offer a window construction set with sliders, so users could create their own tutorials for their programs. This aside, the Companion is a useful development tool and certainly worth the price.

By the way, GFA BASIC and Compiler version 2.02 is now available. There are some improvements in syntax handling and a new ON MENU time command. Contact MichTron for information regarding upgrades. ///

REGULAR FEATURE

Database Delphi

ST matters discussed in the Atari Users' Group SIG on Delphi.

by Andy Eddy

In the same manner that **ST-Log** is changing, the ANALOG/Atari SIG on Delphi has recently gone through some extensive changes. However, the new SYSOP (System Operator) line-up is filled with familiar names: Clayton Walnum (ANALOG4), Matthew Ratcliff (MATRAT), Charles Johnson (CFJ), Maurice Molyneaux (MAURICEM), Charlie Bachand (BACHAND) and myself (ANALOG2). This crew is well versed in all facets of Atari lore and Delphi commands, so any problems can be sent to any of us in mail or forum.

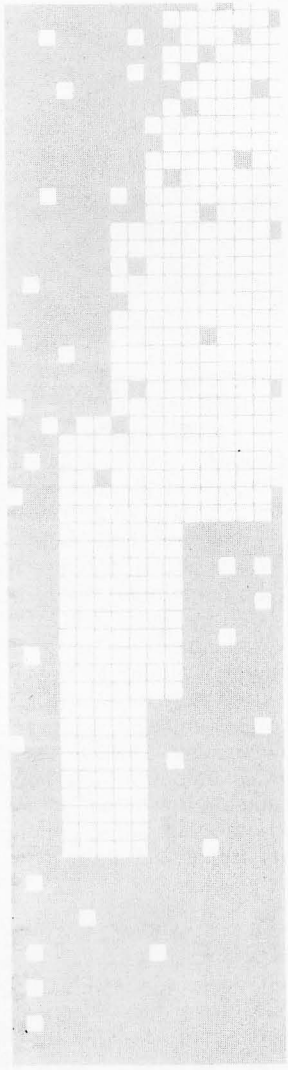
What's new

The databases have also seen some rearranging recently. Among the changes that have been made:

— The ST Programs section, which had previously been a catch-all for most any ST-compatible file, has had some of its entries moved to other new areas.

— The new "Entertainment on the ST" area was added to house the wealth of public domain gameware that exists.

— The category "DEGAS Pictures" has been renamed "Art on the ST," given the number of picture formats that exist and are used more frequently. Picture formats like Tiny and Spectrum are seeing more uploads than DEGAS. As well, homebrewed animations are showing up more frequently, due to the success of CAD 3-D and other animation packages. That variety brought about the more descriptive title.



In addition to these changes, more alterations are being discussed. For a while, the "Applications on the ST" category has suffered an identity crisis, with users not really sure what should go there. Its intended purpose was to hold demos of products and files associated with specific programs (such as spreadsheets and databases, for example). There is also talk of adding other categories, possibly relieving more of the weight that the "ST Programs" partition carries. All this will result in faster searches for desired files. We'll keep you informed of what transpires here, and if you have any suggestions, feel free to drop us a line.

Talk talk

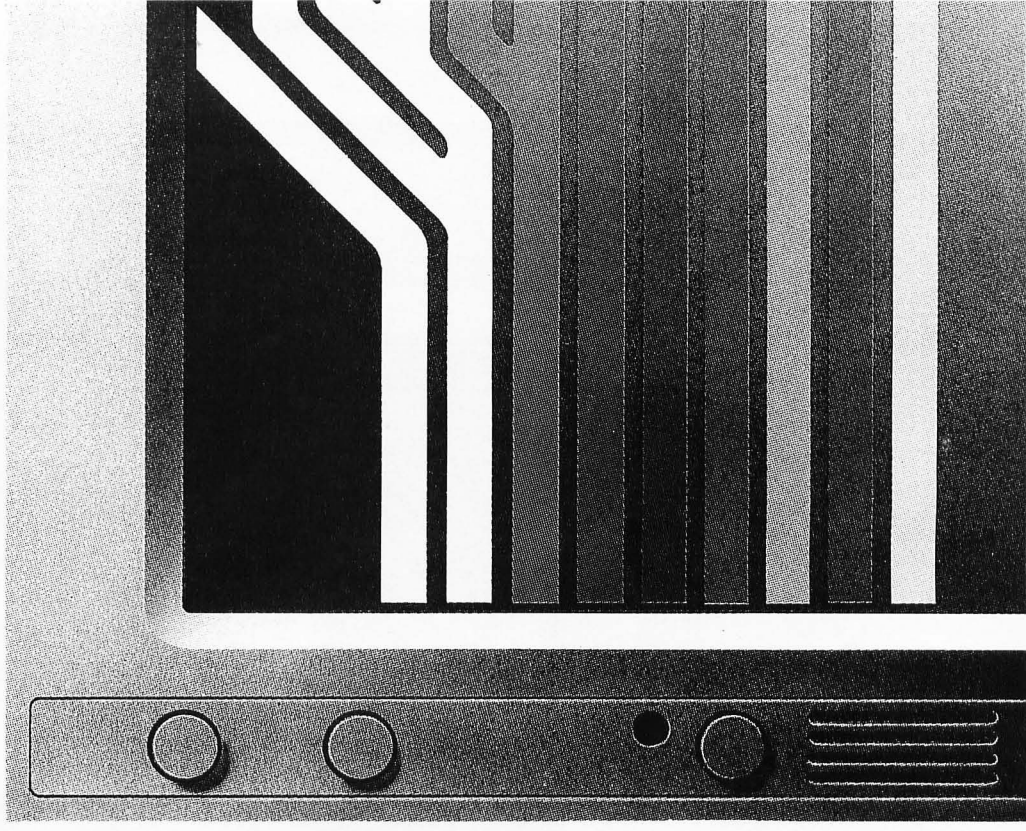
To assist in the dissemination of pertinent information, we've set up a weekly time for Delphi enthusiasts to gather together. Each Tuesday night (at 10 p.m., Eastern time) you can go to the Conference area (type *CO* from the *ANALOG* prompt) and join in with your fellow Atari hobbyists and professionals. While this real-time *klatch* has only been in place for a few weeks (as of this writing), we experienced an increase in attendance and interest. It's a great way to learn about new products, have questions answered and just chat with others who share your brand of computer.

Through this column, we'll attempt to detail what takes place in the SIG of interest to ST owners. This will come in the form of summaries, noting what took place in the formal conferences that take place occasionally, placing certain special guests of the Atari developers' and manufacturers' community in the hot seat. In the past, these sessions have allowed the users to take part in their own press conference, asking questions to many of the vital components in the Atari world.

Similarly, the ongoing discussions that evolve in the forum will be mentioned. Recently, for example, a few of us took part in a discussion of piracy. I've got to tell you, the term "discussion" is a nice way to describe the thread (the term for a contiguous topic in the Forum). It started off reasonably calm, but shortly broke out into some really heated debate. Unfortunately, piracy is a subject that draws strong emotion. One side relayed that *any* piracy diminishes the brand (with much comparison to the Atari 8-bit line and its oft-times woeful existence) and is wrong; the other side mentioned that piracy is predominantly a by-product of poor marketing and poor software quality. I don't know that there will be a solid solution—none was evident after the topic petered out—but both sides agreed that something should be done and soon.

File these away for later

I'll also try to mention some of the better quality uploads that come down the pike. There's a ton of great software in the form of freeware or shareware (where the programmer asks for a minimal donation for his or her effort). One recent example is a program called MEGAMATIC.PRG. Written by ANALOG SIG regular, Lloyd Pulley (MADMODIFIER), Megamatic will provide a bunch of added features and utilities. Placed in your AUTO folder and activated at boot-up, this program lets you choose what it features: among these are memory checking, a choice of different



DESKTOP.INF files (depending on what resolution you pick to start up in), a RAMdisk that is configurable to any size you want, a print spooler (which is a buffer that lets you continue what you are doing on the computer while a file is output on the printer) and a screen saver reminiscent of the 8-bit Atari's attract mode.

Megamatic is a well done, multifaceted utility that gives you a great deal of power and flexibility over its operation. By running the configuration program, you can tailor it to do just what you want and what you don't want. If you don't want to have the screen saver activated, you have that option; if you have a ST4, or memory upgraded ST, and want a large RAMdisk, you have that ability. It's all there for you.

To get to this shareware program, type *DAT ST* (which combines the commands DATABASE and ST PROGRAMS to get you to the desired area), then enter *SEA MEGAMAT* (short for SEARCH Megamatic) to get to the Megamatic description. To encourage satisfied users to donate (the author's request in a shareware program like this one), Pulley has *another* version that he will provide to those who do reward him for his labors. This version contains a few extra goodies above and beyond the one in the databases.

Hey, look me over. . .

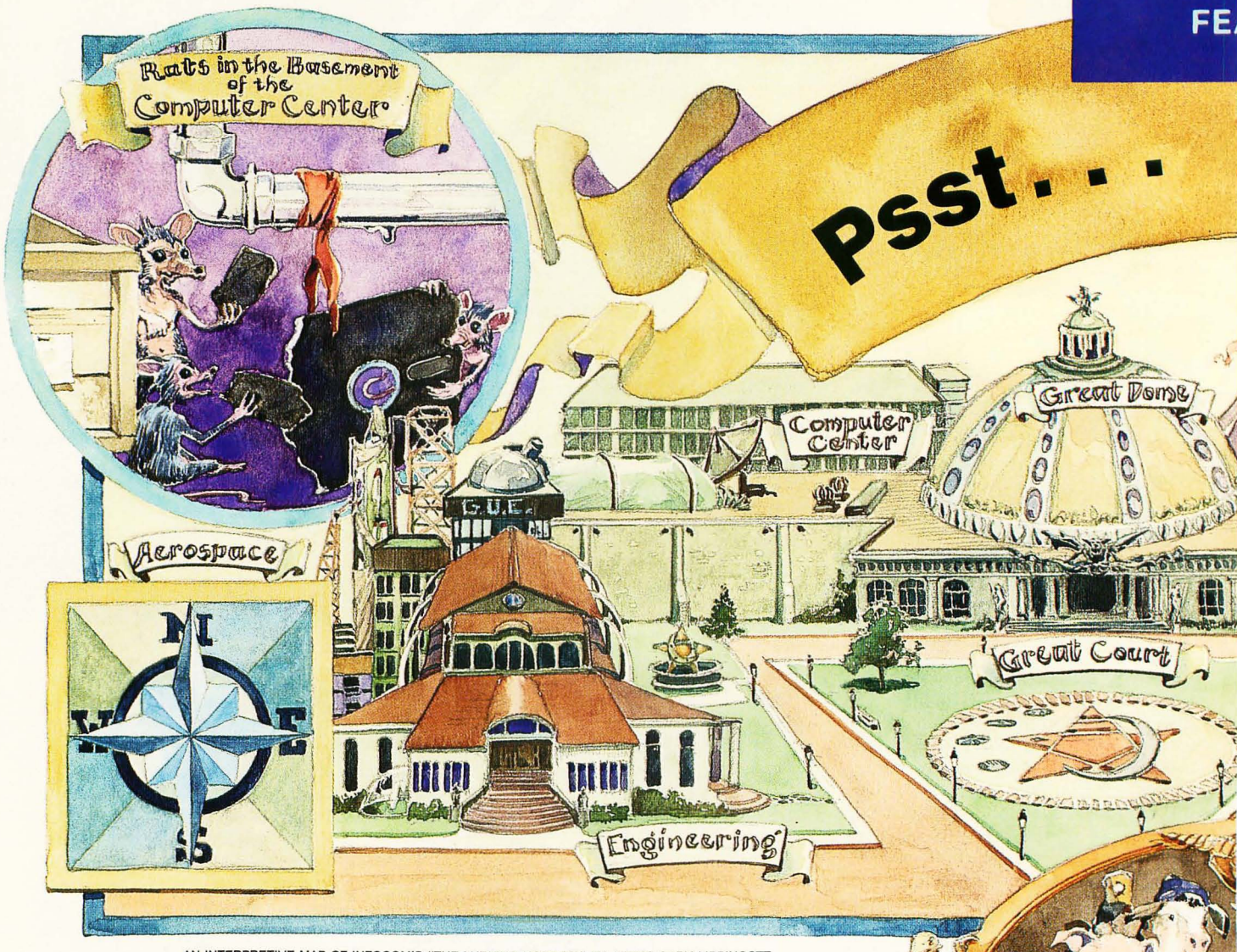
Lastly, this column will help you become more familiar with what Delphi has to offer outside the ANALOG SIG. There is something for everyone here: travel service, news and sports coverage (through a connection with Associated Press), shopping services, as well as many more general and interest-specific areas.

One of the newer areas on Delphi—which opened to the public at the end of February—is *Views on News*. Run by Ellen Kaufman (ELLEN, Manager for the Micro Artists (MANIAC) SIG) and a number of other Delphi regulars, it is found off the News menu (type *NEWS* from the Main prompt, then *VIEWS*). This is mainly a forum for discussing current events—first entries have covered topics ranging from popular quotes, the presidential campaign and the recent woes of television evangelists. In addition, there are also some features (listed in the *Views on News* under the FEATURES menu, logically enough), the wildest of which is a column by Bob Fried (FRIED) called "Articles of Lasting Strangeness, which are frequently humorous and sometimes touching, always thought-provoking.

Another one is NOISE (News of Import Sent Electronically). Each day, it takes one of the lesser-known current event items and comments on it—such as the fellow who was thrown out of a Las Vegas casino for using a computer to keep track of the cards that went by while he was playing Blackjack. The computer would interpret his toe taps, which represented the "count" of the deck and transfer the data to his athletic supporter. Hmmm. . .

The *Views on News* area is quite innovative, bringing a lighter look at our world to the screen as seen through different pairs of eyes. Take a peek at this new service and speak your mind.

Till next month, C U on-line. . . //



AN INTERPRETIVE MAP OF INFOCOM'S "THE LURKING HORROR," BY ARTIST GARY LIPPINCOTT

How to keep your hair

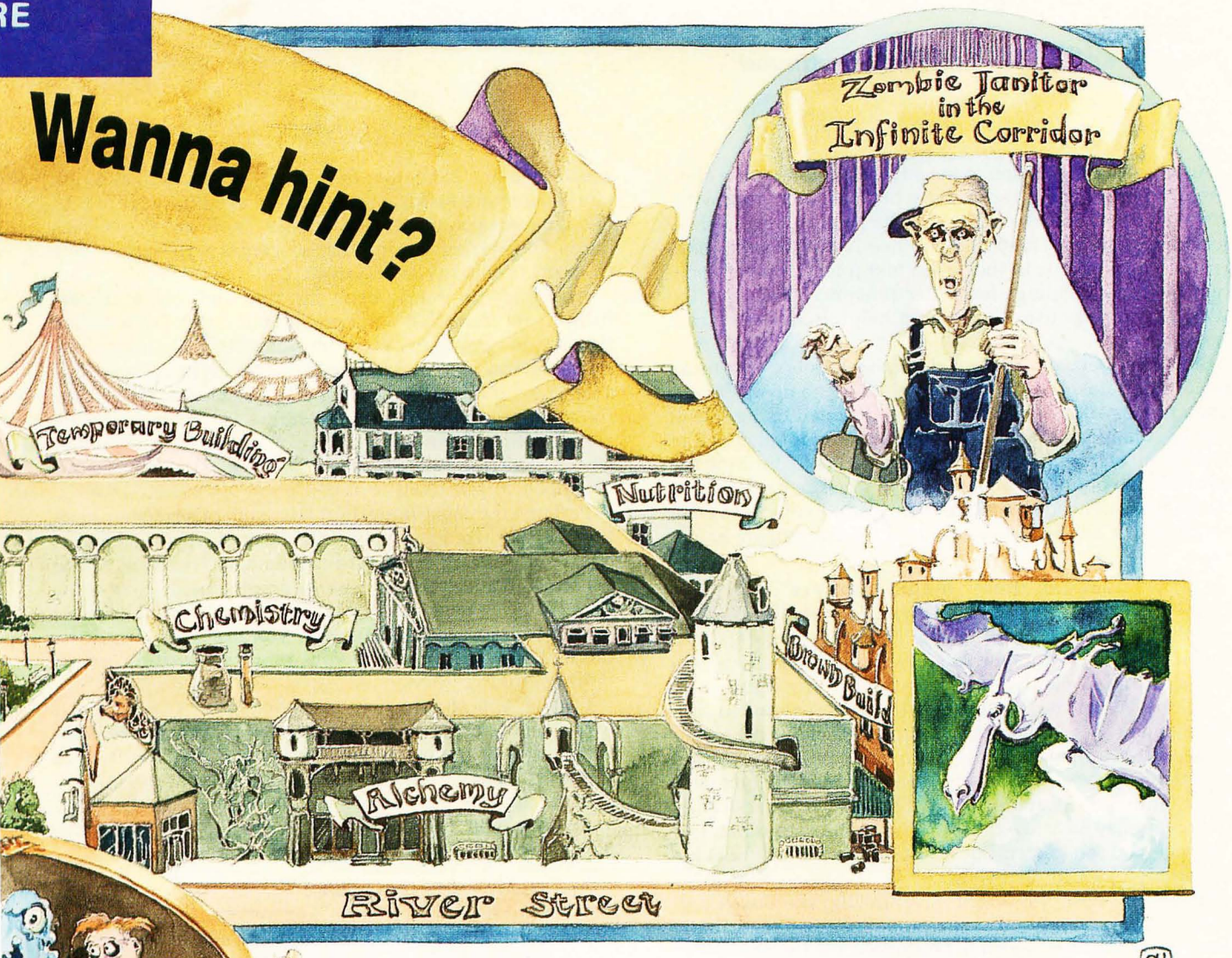
by Andy Eddy

Adventure games are one of the most popular sources of entertainment on personal computers today. Whether they're strictly text versions or graphically assisted—some even going so far as to resemble arcade games in their mode of play—they offer a challenge to the imagination and intuition that other computer entertainment doesn't provide.

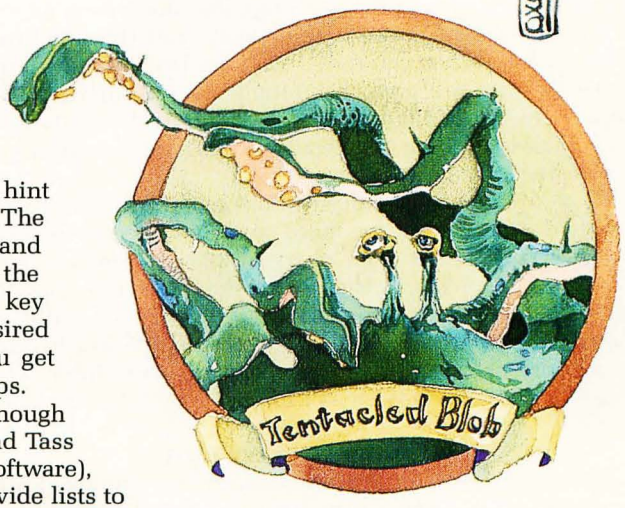
Of course, with any adventure comes the possibility of getting stuck, finding your character painted into a corner of seemingly no escape. We all go through it—maybe you keep getting killed or are hung up in a maze, but try and try as you might, it looks like you're forever trapped. It's got to be the most frustrating (though, for some, it's the most satisfying) aspect for the player.

Fortunately, there are ways around this. Telecommunications networks and user groups have become strong assistants in the search for adventure clues. Unfortunately, user groups meet infrequently, and—overall, with so many games coming out—you may find yourself one of the first to play that particular title, or the first with that specific hold-up.

To the rescue, when all else fails, come the manufacturers of the contests themselves. In the form of tip sheets or hint books, they've discovered a marketing tool that reduces the possibility of the player's dropping the game from sheer discouragement (conceivably lessening future sales to that person), in some cases providing added revenue through the sale of these hints.



while adventuring



If you catch a code

There are two popular methods used for hint publishing: coded clues and invisible ink. The first takes questions the player will likely ask and provides answers in cryptographic form. It's up to the player to decide which questions he needs answers to, and, using a key (A=T, B=I, etc.) as a deciphering tool, he can unscramble the desired clue. This method is helpful, but is often time consuming. If you get jammed often (as I do), you'll use reams of paper deciphering tips.

Activision (a company not well known for their adventures, although they've had success with some, such as Hacker, Borrowed Time and Tass Times In Tonetown) and Polarware (formerly known as Penguin Software), use this process in conveying assistance. These manufacturers provide lists to those who request them, and a Polarware spokesperson told me that the sheets are very popular. They request that you send an SASE (self-addressed, stamped envelope) with your request, and Activision asks for $1.00 to defray costs.

You may choose to do what I did with these hint sheets, and save a wealth of time. I wrote short ST BASIC programs to take the code key and automatically convert the cipher.

See the accompanying listing for a sample program I used to descramble some of Activision's sheets, which employ an inverting of the alphabet (A=Z, B=Y, etc.) as a cipher. It takes all alphabetic characters entered and converts them to the proper letter, while letting others (like punctuation) pass through unchanged.

Wanna hint? *continued*

Another recent addition to the adventure-developing clan, Firebird Software, goes so far as to provide hints from within their game The Pawn. Ditto for the recently released sequel, The Guild of Thieves. If you need assistance in these games, you can type in the coded text (placed under pertinent question headings for better reference) found in the manuals. In response, you'll get a helpful comment from the game.

Similarly, Datasoft, in their detective mystery adventure game, 221 B Baker St., has coded clues come up on-screen when requested. These hints can be decoded using one of twenty code groups found in the game's manual. Each of these groups is a different deciphering scheme, limiting the ability of the opposing team of detectives (the game is best played head-to-head against another team) to discover what the clues are, as they're acquired on the run during gameplay.

Just call them "magic markers"

The other clue method—and, without a doubt, the best—is the invisible ink hint sheet. Though it's costly, Infocom and Sierra On-Line utilize this method. As above, you have a list of frequently asked questions or troubling game situations. When you find the question that pertains to your predicament, you'll discover a number of blank boxes underneath. Within each box is a progressively more helpful hint, that's revealed when you "paint" across the box with a special marker provided with the book. If you're really stuck, you may have to expose all the boxes to find out how to rescue your character. But it's up to the player to decide how many clues to uncover.

In keeping with the fun of adventuring, the boxes will many times contain jokes or snide comments, usually chastising the player for *using* clues. In some instances, I've found the questions refer to things that don't even appear in the game, which can momentarily lead the unwitting victim astray. When you reveal the clues beneath these "red herrings," you are given seemingly helpful answers. But usually, in the last box, they tell you that the hints were meaningless and that you shouldn't unveil clues unless you have a real need.

As mentioned before, these books are more expensive to produce and, in turn, are more costly to receive: Infocom and Sierra On-Line both charge $7.95 per book. Cindy Weiss, Infocom's Manager of Public Relations, informed me that they'll soon start to combine hints for two different games into one book, which will be priced at $9.95. In other situations, "new games will have some 'on-line' hints available," in the same manner as Firebird and Datasoft games.

A few hints of my own

Hint books are nice, but they should be employed only as a last resort. Much of the fun is in discovering how to prevail over the myriad puzzles contained within the adventure. Most ostensibly insurmountable situations can be resolved by using the more basic rules of adventuring.

First and foremost, you should read the documentation that comes with the game *thoroughly*. Often, you'll find tidbits of information that may assist you in some scenarios, but you may also discover idiosyncrasies or commands that are unique to a particular developer's products: how to get your character's status, or the method used to speak to other characters you meet, for example. There are no set standards for writing adventures, so games will vary widely from one company to the next.

Also, pick up every object you find and look at it carefully, to get clues that will benefit your quest. It's not often that the programmers place objects with no purpose in a scenario. You may, though, need certain items before you're able to get others (like getting a key to unlock a door of a room containing necessary treasures). Of course, some adventures limit how many items you can carry, so you may have to backtrack and retrieve them later in the game, when you need to put them to use.

Perhaps the most important advice is to save your position to disk frequently—particularly if you're about to chance a difficult or unsure movement. That way, if you get stuck in a less-than-desirable predicament—like being dragged away by a slimy creature or lost in space without oxygen, for example—or even if you just want to quit for the day, you can return to your previous position, rather than beginning anew.

Finally, you should map every location. The reason is twofold: not only will you avoid retracing your steps—a big waste, as most games are scored by how *few* moves your character makes—but you also can get somewhere quickly.

Well, that's it. I hope all this has helped, as a lot of enjoyment can be realized from a patient and knowledgeable playing style. Happy Trails! //

Andy Eddy works as a cable TV technician in Connecticut, but has been interested in computers since high school. While his family's Atari 800 is four years old, he's been avidly playing arcade games since Space Invaders and is a former record holder on Battlezone.

Listing 1.
ST BASIC listing.

```
1 REM This ST BASIC program accepts ty
ped-in UPPER CASE
2 REM text from certain Activision hin
t sheets and turns
3 REM them into usable clues... Plain
and simple.
4 REM By Andy Eddy for ST-Log Magazine
, 1987
5 DIM A$(255), X$(255)
10 LINE INPUT "";A$
20 Y=LEN(A$)
30 FOR X=1 TO Y
40 X$=MID$(A$,X,1)
50 C=ASC(X$)
60 IF C>90 OR C<65 THEN 80
70 C=(90-C)+65
80 PRINT CHR$(C);
90 NEXT X
99 PRINT
100 GOTO 10
999 END
```

The Karate Kid Part II

by Steve Bak (Artwork by Pete Lyon)
MICHTRON
576 S. Telegraph
Pontiac, Michigan 48053
(313) 334-5700
Low resolution $39.95

by Bill Kunkel

Just what the ST universe needs: yet *another* karate contest!

The Karate Kid Part II is a nice looking, multiscenario action game closely linked to the Ralph Macchio/Pat Morita movie of the same title. As in the film, the player (cast as Macchio's "Daniel") is no longer competing within the comfortable confines of a school auditorium. This time, he's told, there are no trophies, referees or points. Then how do you know who wins?

"Whoever dead," explains Morita's character, "doesn't."

This all sounds very nice, reader-san, but in context of the actual game, it is meaningless. No points, you say? Of course there are; they simply are not tallied on-screen. No referees, eh? But the combatants are still limited to the sixteen legal moves that can be entered through the joystick or keyboard.

The program attempts to follow the film and introduces several bonus sequences right off the celluloid. In one, Daniel must break a block of ice with his naked appendages (his bare hands, that is); in another, the player inexplicably becomes the Pat Morita character and must capture an annoying housefly with a pair of chopsticks. The graphics in both these sequences are "striking," but in terms of game play, the chopsticks-vs-fly gambit is a bit of a bore.

Karate Kid II

Background graphics throughout are beautiful, taking full advantage of magnificent Japanese scenery. The foreground sprite animations, however, are rather diminutive. The smaller an on-screen character, the more limited its body articulation; in a martial arts game, where subtle body movements must be clearly distinguishable, larger characters are needed.

Karate Kid Part II is another nice-looking karate game, despite a few rough edges. If your thirst for martial arts hasn't already been slaked, this may be your cup of tea. ///

Airball

MICRODEAL U.S.A.
Distributed by MICHTRON
576 S. Telegraph
Pontiac, Michigan 48053
(313) 334-8729
Low resolution $39.95

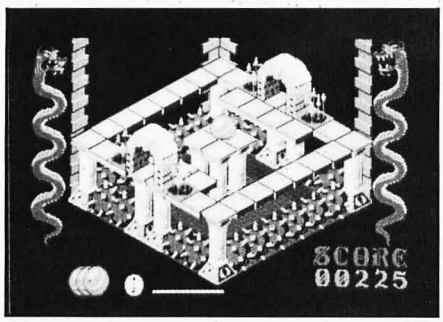
by Joyce Worley

Captured by an evil wizard, you've suffered the most unusual video-gaming fate to date. The dastard turned you into a rubber ball, then bounced you into a hostile world. Rolling and tumbling, the gamer has to navigate through his mansion seeking the spellbook of instructions for regaining human form.

Airball's unusual theme turns this arcade-action game into an adventure. As the rolling ball, the gamer must explore the maze of over 150 rooms. This would be difficult enough, since the castle is a warren of steps, ramps, and narrow passages bounded by sharp objects that can puncture and destroy the fragile balloon. But that's only half the problem.

The **Airball** has a slow leak, and the remaining air pressure is indicated by a yellow bar across the bottom of the screen. To refill, the ball must bounce on top of one of the pumps scattered throughout the mansion. Then the gamer has to choose the right moment to roll the ball off the pump, since too much internal pressure makes the balloon burst.

Helpful objects are scattered through the house, like a flashlight (indispensable in the dark room), gold bars, and other point-scoring prizes, some of which are needed to undo the shape-changing spell. Each time the ball is destroyed by over- or under-pressurizing, or by being punctured with a sharp object, a cross marks the spot where it died. The next time the ball rolls past that point, the gamer can pick up this grisly reminder of past lives; it's one of the ingredients needed in order to return to human form.



Airball

The mansion is viewed one room at a time from a three-quarters overhead perspective. Rolling through an arched doorway blanks that room and brings up the next one. Prizes are randomly scattered throughout the castle. To win, the gamer must find and return the spellbook to the starting room, then locate the items required in a sort of rolling scavenger/treasure hunt. Accumulating all the ingredients wins a message of congratulations—and an offer to play again.

Players control **Airball** with keyboard, joystick or mouse. The ball can be moved up, down, left, right, or bounced, and the player can pick up or drop items found in the mansion. However, it's not an easy play mechanic to master. The three-quarter perspective makes it hard to get the hang of controlling the ball. The beautifully drawn playfields incorporate stone archways, walls and staircases which the ball can pass completely around. This is a lovely effect, but it's frustrating when the ball gets stuck in a blind corner.

Airball is an extremely pleasant, even addictive, action game. The pretty playfields, bounded on each side by ornamental wriggling snakes, and the pleasant musical accompaniment, make rolling the orb a real ball! ///

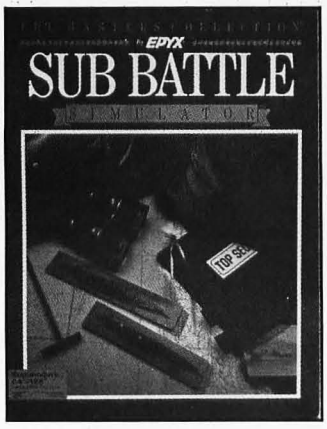
Sub Battle Simulator

by Digital Illusions
EPYX
P.O. Box 8020
Redwood City, CA 94063
(415) 366-0606
Low resolution $39.95

by Bill Kunkel

Sub Battle Simulator is the *third* submarine simulator published for the ST in the past year (Silent Service from MicroProse and GATO from Spectrum Holobyte are the others), and it compares favorably to its peers. Like the competition, **Sub Battle Simulator** casts the user as a circa-World-War-II sub commander with a variety of tactical and strategic options (as well as a couple of high-tech aids undreamt of by submariners of four decades ago).

The game features three modes: Target Practice (vs. an enemy convoy), Single Mission and Wartime Command (which covers the entire war). **Sub Battle** uses a Macintosh-style interface with File (loading, saving, new game, target practice), Information (target tally, ship's log and vanity board), Special (navigator, time compression, send SOS, radio position, transfer torpedo, abandon ship, send shore party and sound controls), and Mission (current orders) menu headers accessible throughout the game.



Sub Battle Simulator

The game screen is comprised of four gauges (view, heading, speed and depth) in a vertical strip occupying the left fourth of the screen; a visual display; dialog box (for crew responses, radio messages, etc.); and twenty-one click-on command boxes (crash dive, run silent, diesel power, charge battery, deck gun, launch forward and aft torpedo, etc.) Smaller secondary screens can be brought up with a mouse click. These include views from the tower, periscope, binoculars, sonar screen, radar screen, map display (with zoom), side display and status readout.

Sub Battle Simulator is a state-of-the-art product, with excellent sound and graphics and an accessible user interface. It most resembles GATO in its look and play style, but it is clearly superior to that slightly dated product. On the other hand, it can't match Silent Service for visceral punch and cinematic ambiance, but does make up for that with its precision and sense of realism.

The excellent documentation not only details the user options and equipment, but provides a well-written historical perspective that enriches an already solid game. ///

GAME

LOW RESOLUTION

Strathello

A SPECIAL INCLUSION

A C version of Capture the Flag

by Todd Kepus

Strathello is an adaptation of the famous game Capture the Flag—always a favorite among computer enthusiasts—and this version will run on the Atari 520ST in low resolution. It's written in C and 68000 assembly language, using the Megamax C compiler, and is available on this issue's disk version or on the Atari SIG on Delphi.

Layout

Strathello is played on a 14x9 grid, with the player's pieces on the left side and the computer's on the right. Both the player and computer have thirty-six pieces, occupying four columns of the screen on each side. Two water holes will appear in the upper middle and lower middle portions of the screen to further complicate strategy. The object of the game is to capture the opponent's base.

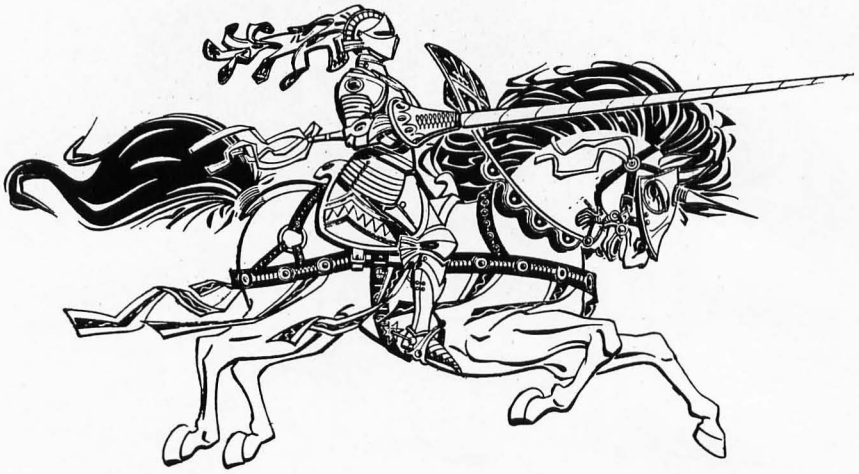
The player and computer have nine different pieces each. Their ranking is as follows: Seeker—lowest rank; Investigator—destroys Seeker; Mercenary—destroys the Seeker and the Investigator; Equalizer—destroys Seeker, Investigator, and Mercenary; and Terminator—destroys everything above.

In addition to the regular rankings, there are three special pieces that have their own special powers: Mine—a stationary object that destroys anything that lands on it—it can only be removed by a Decoder; Secret Service—destroys only the Terminator; and Decoder—destroys the Seeker, Investigator and a Mine.

Finally, there's the base, which can be captured by any of the opponent's movable pieces. The game ends when either base is captured.

Setting up

When the program is run, the credits and playing board appear. To set your pieces on the screen, place the square cursor anywhere in any of the four left columns by moving the mouse. Then press the left mouse button to set the piece. A piece may not be set upon another. The pieces are set on the board in ascending order of rank, that is, the Seekers are placed first, followed by the Investigators, and so on. It's a good idea to place the Seekers in the fourth column and move in to the left as you set the pieces. That way, the base will be placed behind all the players, protecting it from



a quick computer victory.

Playing and moving

After all of the player's pieces are set, the computer's pieces immediately appear and the game begins. To move, place the cursor on a piece that's not surrounded. The message *Click Source* will appear in the upper left corner of the screen with the name of the piece directly beneath it. If the cursor is on a piece that can be moved, the message *Click Source* will begin to blink, indicating that it is a valid source piece. Click the left mouse button. The message *Click Destination* will appear in the upper left-hand corner of the screen, indicating that you must now choose a box in which to move.

A piece can only move one space vertically or horizontally. When the cursor is over a valid destination, the message will begin to blink, indicating a possible choice. Click the mouse when the cursor is on the square to which you wish to move. The computer then makes its move. The game continues until either one of the bases is captured, or until it's impossible to win the game.

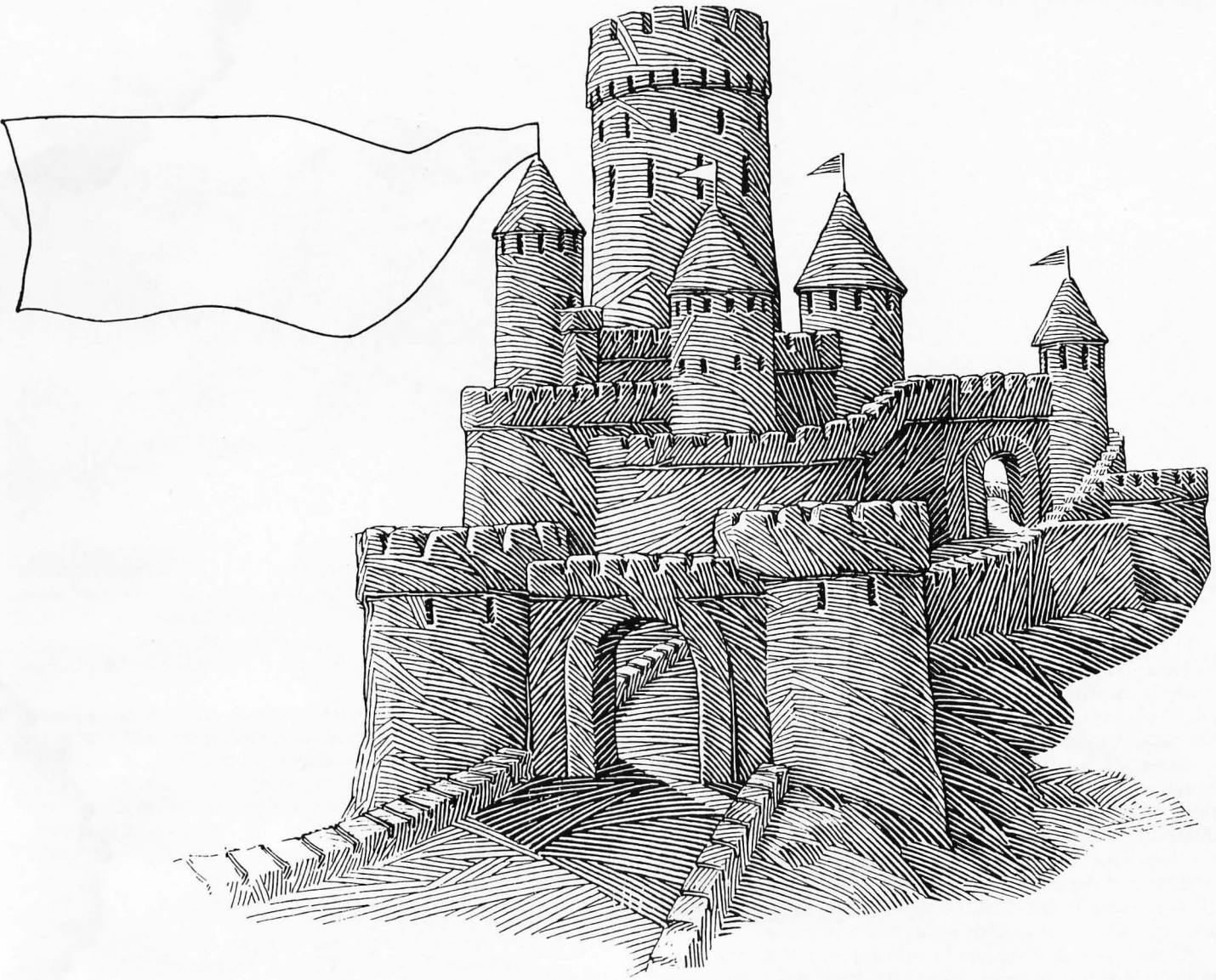
How the program works

Strathello was written using the Megamax C compiler, one of the best C compilers for the Atari ST. I recommend it to any ST programmer, whether hobbyist or professional. One of the nicest features about Megamax is that it allows in-line assembly code. **Strathello** allocates a background screen for drawing images on, and all buffer and screen moves are done with assembly language for quick results.

Two of the most important assembly routines included in the program are object movement routines. They are specifically designed to move 16x16 color objects for maximum speed.

The first routine, position, places a 16x16 object on the screen. It takes the X- and Y-coordinates, the location of the object data, and the address of the screen or buffer in which to copy the image. The data block that defines the object is the same format of screen memory layout. Each object data block takes 256 bytes, including the mask. A special routine, makemask, is used to create the mask from the object's image data.

The second important routine is the replace function, which takes the same X- and Y-coordinates, the buffer to copy from, and the buffer or screen in which to copy. In **Strathello**, replace copies the buffer to the screen, replacing the object and thus, restoring the background. Immedi-



ately after the screen is restored, the new object is drawn to avoid blinking.

Before the main loop is called, which activates player and computer movement, the player must first set his players on the screen. This is accomplished by the function set_player. Afterwards, the computer's pieces are randomly established by the function comp_set. The main loop of the program is the function players_turn. This routine handles the cursor, player and computer movement routines. At the end of the function, it checks for victory by either side and exits the loop if one side has won.

The two important functions in the evaluation of the player's movements are uncndmove and do_attack. If the player moves to a vacant position, an unconditional move is made (uncndmove). If the player happens to move on an opponent, do_attack is called to determine the victor of the move. Two error-checking routines are called before each routine, to determine if the player has correctly chosen a source and destination. They will return an error value if the cursor is over an invalid position. The program determines the mouse's position on the screen by calling get_pos, which returns two numbers for the horizontal and vertical positions.

After the player has moved, computurn is called to move the computer's piece. The computer then establishes a list of all its possible moves, to decide on the best one. The highest priority moves are placed at the top of the list, and if not executed, the second most important move will be checked, continuing down the list until a move is made.

One important function the computer uses to move a piece is victory, which checks for victory of a computer's piece over a player's piece. Another function, protect_piece, protects the specified piece by moving to intercept the opponent. The home_in function moves a computer piece toward another specified position, such as the opponent's base, and is also called when the computer's Secret Service is trying to search and destroy the player's Terminator. This function is the backbone of the computer's strategy, selectively picking targets and reselecting after the target is destroyed.

Before the computer does anything else, it first checks the board for an instant victory. This is accomplished with do_flag, which returns a nonzero value if it can achieve a victory. A quick victory is defined as one that can be made in the next turn. Since the computer relies on its powerful pieces to achieve a quick victory, the retreat function is used

to retract a piece if it's about to be attacked by a player's piece. This function is given second priority, following the flag check. When the game ends, you can play again or quit.

Enhancing *Strathello*

The assembly language graphics routines were created to be directly compatible with the Color Object Editor's source code, from which the graphics characters were created. Since the Color Object Editor doesn't create a mask for the data, the makemask function is used to create the mask.

Strathello was programmed to be easily enhanced. For example, it takes little effort to add more intelligent computer algorithms, to make the computer more logical when determining how to attack the other side. The array realscrn contains all of the piece's codes, and whatever is contained in that array determines the actions of the computer's strategy. For example, one could change the number of Terminators on either side to give a distinct advantage.

Another possible application would be to take the computer's algorithm out and replace it with some code, allowing the array to be sent over the modem line, to play with a friend. In addition, since each piece has a different code in the array, all you have to do is change the value of the piece desired. The codes for all of the pieces are as follows:

Space (no piece) 0
Seeker (player's) 1
Investigator 2
Mercenary 3
Equalizer 4
Terminator 5
Secret Service 6
Decoder 7
Mine 8
Headquarters 9
Unused 10
Seeker (computer's) 11
Investigator 12
Mercenary 13
Equalizer 14
Terminator 15
Secret Service 16
Decoder 17
Mine 18
Headquarters 19
Unused 20
Water 21

For a real challenge, you could make all of the computer's pieces Terminators. Just be sure to increase the array sizes near the beginning of the program. For instance, if you want to increase the amount of Terminators the player has, change the *3* in arrays p5x[3], p5y[3] to the number of Terminators desired.

It's a simple matter to add pieces to the screen, by changing the values in the array and plotting the character to the screen with the function setguy. To use this function, supply the X- and Y-coordinates, along with the address of the graphic form of that particular piece. For example, to place an additional player's mine in the middle of the screen, place this fragment of code directly after set_player in the main loop:

```
/* sets player's mine at pos. 7,4 */
realscrn[7][4]=8;
/* draw image on screen */
setguy ( 7,4, (long)(mine_p) );
```

You must then change the number of allowed mines from five to six. To do this, change the arrays pmx[5] and pmy[5] to pmx[6] and pmy[6]. Changing the number of pieces allowed is the same process as above for any piece; however, you must change the corresponding array.

Notes of strategy

Since the first pieces set up are the Seekers, it's a good idea to place them at the front of the battle line and reserve a space at the far left side of the screen for the base. Then, place mines around the base to block the computer's Decoders. To beat the computer, simply knock out the computer's Decoders; the computer cannot win if it has no Decoders to destroy the mines around your base. If possible, protect the Secret Service piece, since the computer will try to quickly overtake you with its Terminators. ///

Introduced to the Atari 400 just five years ago (in the seventh grade), Todd Kepus is now an Atari ST programmer. He purchased his ST from a local user group immediately after reading about it in Atari Explorer *two years ago. His ST interests include fast animation routines and 3D graphic programming. He is currently a Computer Science major at California State University in Sacramento.*

NOW PLAYING AT A SOFTWARE DEALER NEAR YOU

Available for Amiga, Commodore 64, IBM PC, Apple IIgs, and Atari ST, which are trademarks respectively of Commodore-Amiga, Commodore Electronics, Ltd., International Business Machines, Apple Computer Inc., and Atari Inc.

Cinemaware Corporation, 4165 Thousand Oaks Blvd., Westlake Village, CA 91362

Circle #114 on reader service card.

WordPerfect® in Every Way

If you're looking for software that takes full advantage of your Atari's capabilities while providing an extensive range of features, look no further.

WordPerfect offers the power you need with features like Columns, Indexing, Merge, Macros, Speller, and Thesaurus. You'll find them useful for everything from simple memos to complex reports. All features are easily accessed using the Atari mouse and pull-down menus, or WordPerfect's standard keystrokes.

WordPerfect's GEM-based design taps the Atari's resources with ready access to desktop accessories, full color adjustment for color monitors, a definable mouse pointer and cursor, and the ability to move and size up to four windows. And WordPerfect is written in assembly language to take full advantage of the Atari's speed.

WordPerfect Corporation offers Atari users the stability of a proven product, produced by a reliable leader in software manufacturing. With full documentation, toll-free customer support, and free software upgrades, your investment will be profitable for years to come.

Expand your options with WordPerfect—the most powerful word processor you can buy for the Atari ST. For a demonstration, contact your local dealer.

WordPerfect
CORPORATION

1555 N. Technology Way · Orem, UT 84057
Tel: (801) 225-5000 · Telex: 820618 · FAX: (801) 227-4288

Circle #112 on reader service card.