Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

5-21-1988

Microcomputer graphic intelligence

Thomas Tillman Shaffner

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Shaffner, Thomas Tillman, "Microcomputer graphic intelligence" (1988). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

ROCHESTER INSTITUTE OF TECHNOLOGY

A Thesis Submitted to the Faculty of
The College of Fine and Applied Arts
in Candidacy for the Degree of

MASTER OF FINE ARTS

Microcomputer Graphic Intelligence
By
Thomas Tillman Shaffner

May 21st, 1988

APPROVALS

Advisor: Robert Keough

Date: 6-3-88

Associate Advisor: James Ver Hague

Date: 6/6/88

Associate Advisor: Norman Williams

Date: 6/7/88

Special Assistant to the
Dean for Graduate Affairs: Phillip Bornarth

Date: 6/10/88

Dean, College of
Fine and Applied Arts: Dr. Robert Johnston

Date: June 10, 1988

I, Thomas T. Shaffner, hereby grant permission to the Wallace Memorial Library of RIT, to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: June 17, 1988

THE BEGINNING

This thesis offers evidence that the marriage of artificial intelligence and microcomputer graphics is a mixed blessing. This pairing will save time and effort for the operator but the strain on the computer may destroy the union.

To begin with, artificial intelligence, or AI, is a mysterious companion for a computer program. A related field to AI, expert systems is much better understood. "An expert system simply defined is a computer program that aids in solving complex problems that would usually require a human expert."[1] Artificial intelligence provides the same service but for generic problems. Another distinguishing characteristic is that an expert system can only carry on a good dialog about one specific topic, and only in its own terms. An AI program can discuss its solutions in any terms, with any user. Because of its friendliness, artificial intelligence gets an odd reputation not from confusion with expert systems, but with conversational interfaces. Any program can mimic dialog in a general way, only a smart one can learn from the interchange.[2]

If artificial intelligence is so misunderstood, is it a necessary partner to microcomputer graphics? A study of computer graphic designers shows that something is needed to improve production quality. Untrained operators on unintelligent computer systems will make any combination of values. Inconsiderate graphical choices can combine to ruin a computer generated slide: "garbage in, garbage out".

Professional designers need not worry about being replaced by automation. Although artificial intelligence may protect the sensibilities of the public, a computer can

---

[1]Excerpt from a report on AI and Expert Syestems by Robert Keough, Rochester Institute of Technology, Rochester, New York, 14 April 1987.

[2]A classic example of a dialog processing program that has no further "intelligence" than for conversation is ELIZA by Joseph Weizenbaum. (See Marion Long, "Turncoat of the Computer Revolution," New Age Journal, December 1985, pp.47-78.)

never be creative. AI will improve image quality and quantity, but innnovation will never be automated.

Artificial intelligence is an unusual, necessary, and non-threatening thesis topic. For an MFA degree however, the issues are more exciting than the imagery. While AI will improve design aesthetics, the best effect will occur through subtle coercion: and since good design is transparent, artificial design improvement will be invisible. Therefore, the best illustration of AI in microcomputer graphics is of the entire process and not the final effect. The accompanying exhibition for this MFA thesis was much more a diagrammatical illustration than an aesthetic masterpiece. An identical illustration of the structure of a sample dialog with a color smart program is included here, as well as a sample conversational interchange (see figs. 1 & 2.)

This paper will continue to address the implications of the combination of artificial intelligence and graphics through an account of a smart, design program. The listing included in this thesis gives a microcomputer the intelligence to assist an operator in choosing the copy, drop shadow, and background colors for a high resolution word slide.

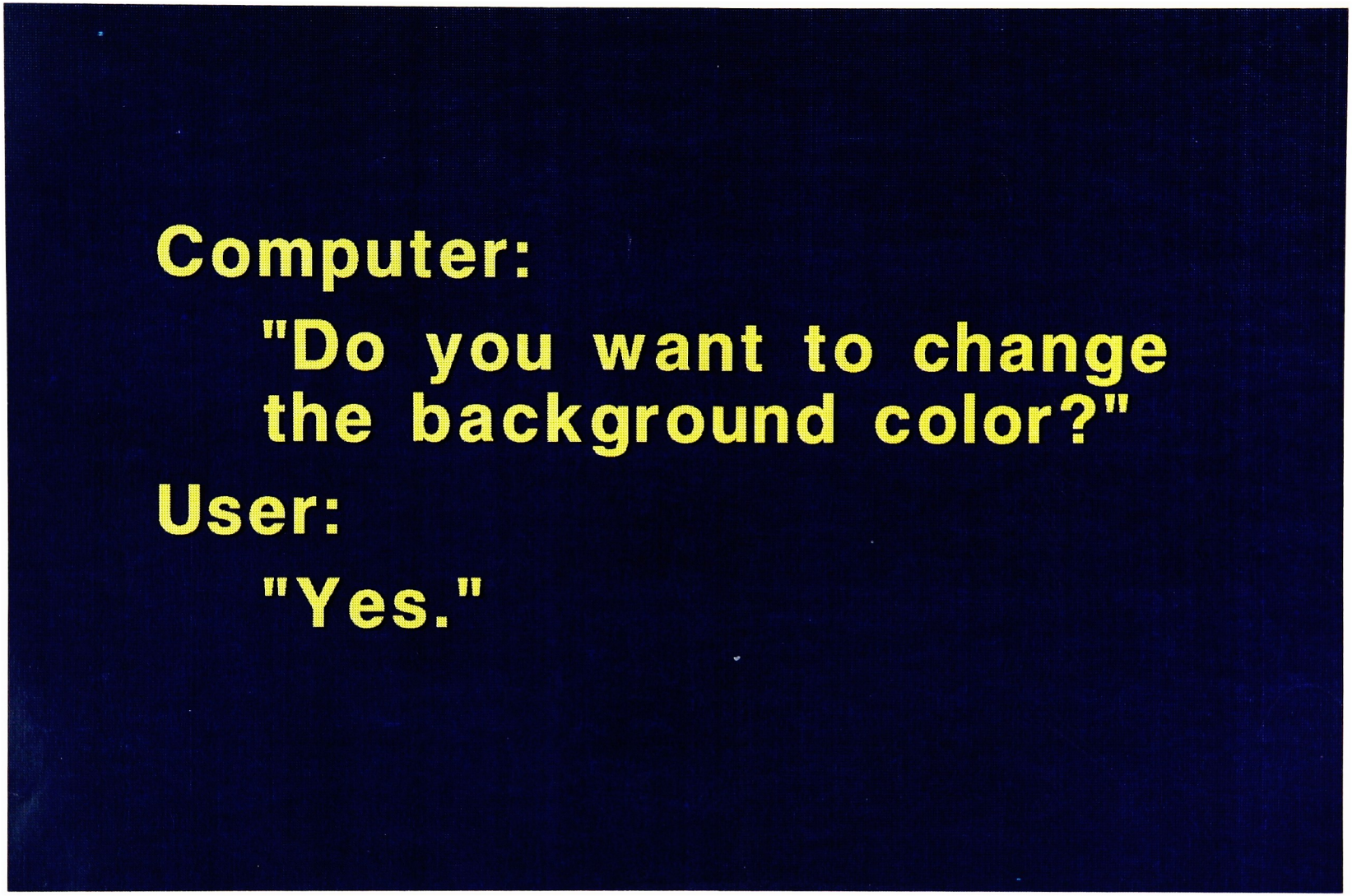

Figure 1. A sample conversational interchange.

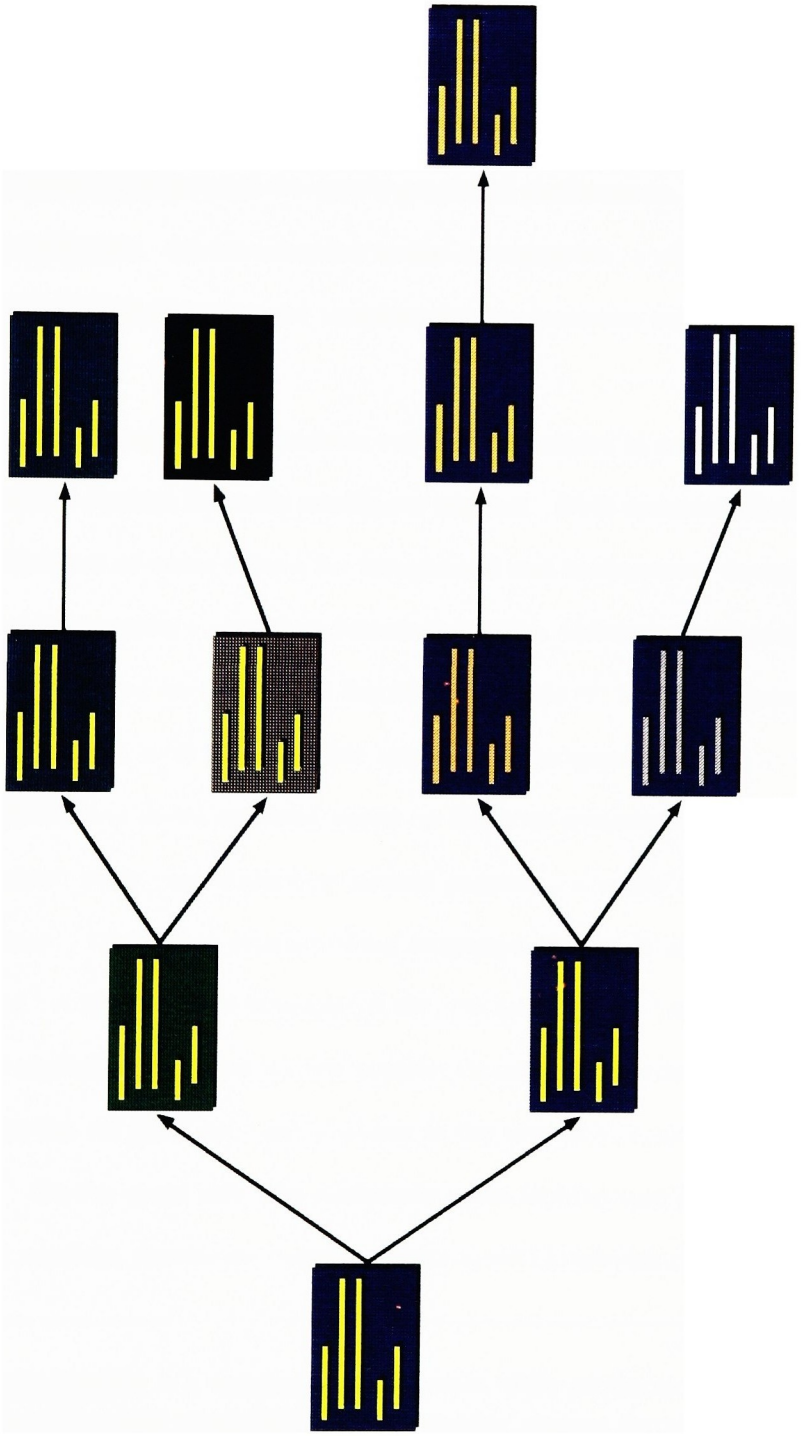

Figure 2. The binary structure of a potential dialog.

THE MIDDLE

The first thesis attempt at joining computer graphics and artificial intelligence was on 8-bit, Apple II computers. Simple illustration and typography programs were enhanced with a utility that learned the user's graphical preferences, and related the knowledge through easier use. An introduction to the Autographix, a slide-generating, computer system, suggested that even the simplest design decisions can be based on previous interaction.[3]

This thesis's program manipulates color in title slides in exactly the same manner as the Autographix system but with intelligent support. Both systems allow the independent specification of color values for foreground and background design elements. However, the Autographix's closed architecture inhibits software development, so AI reasearch had to be done on a different machine. An IBM PC XT was employed to generate the needed graphics with the available intelligence programmability.

The IBM PC is a 16-bit machine based on the 8088 processor. The XT used in this thesis has an 8087 math coprocessor, extended graphics, window interface, and a graphics routine library. The native programming language, BASIC, is adequate for the many tasks addressed. A higher-level language (LISP, PROLOG, C, etc.) would have allowed acceleration and sophistication, but not the simple, flexible and methodical nature of BASIC.

The IBM has its good and bad qualities in the research on artificial intelligence with graphics. On the good side, the extremely open architecture of the system permits the addition of medium resolution color graphics to 640 kilobytes of RAM and a 20 mega-

---

[3]The Autographix is a microcomputer based, slide production system. The equipment allows for rapid production of high quality images through the use of guidelines to aid the composition of information in low resolution. The data for the slide is then sent to a high resolution film recorder. (For specifications see the Instruction Manual [Waltham: Autographix Inc., 1982], p.11.)

byte hard disk. The hard drive contains: a window environment, graphics routines, languages, DOS, the thesis program and all data files. On the other hand, these many concurrent tasks do not promote easy development. If one utility crashes, the whole system goes down.

Even though it is an uncompiled language, BASIC's hardware residence allows for high execution speed and indigenous security. While the language supports standard string parsing instructions, it has no real english language control. The lack of list processing in BASIC requires the constant use of FIELD statements to support random access files.

The graphics ability of the PC is limited to sixteen color, 640 x 480 resolution, raster display. Type-color selection is pivotal to the program, and though the IBM does not support the use of its entire palette for text, a supplementary graphics library permits the creation of graphical type in any color.

All these equipment details become secondary in the public's need for artficially intelligent software, but these specifications have been listed here to define the many factors hindering the development of such programs. Just as color availablity is crucial to this program's development, the difficulty in managing graphics **and** AI is critical to the conclusion of this thesis.

THE PROGRAM

The next section of this thesis is devoted to the careful explanation of the central, experimental program, MGI. The software was developed for the IBM PC XT and when correctly entered and supported, the resulting package will be an artificially intelligent color selection system for word slide production.

As stated, the recommended hardware is an XT, but the program will run unaltered on any IBM PC. While a hard drive is recommended, it is not necessary to the operation of the software. The following supplemental packages are also recommended for use of the program: an 8O286 coprocessor, IBM DOS 3.1O, Microsoft Windows, and Media Cybernetics' HALO.

With alterations, the essential algorithms of this program can be transported to any computer. To make the listing easier to implement, the program structure, including the line numbers, has been typeset in the simplest possible design. The commands remain the native BASIC instructions, but are universal to almost all programming languages.

The accompanying comments and remarks normally included in the program listing itself, have been removed to the opposing pages for easier analysis. The boldface section labels are structural separators, not instructions. Given the equipment restrictions and experimental deadlines, this program only accepts and repeats its education on three color parameters for a word slide, but with careful development, the scope of the package is limitless.

MGI program

Initialize programming environment

```
ON ERROR GOTO 3000
KEY OFF
```

Initialize graphical environment

```
GOSUB 4000
DEF SEG=&H3800
BLOAD "haloi.bin",0
device$="haloibme.dev"                CALL SETDEV(device$)
```

Initialize constants and variables

```
DIM file$(19), variable$(2), color%(2), color$(16), color(3,5)
doyouwant$="do you want "
color%(0)=1:  color%(1)=14:  color%(2)=0
COLOR 14,1
```

Declare graphical variables

```
DATA  "background color",  "copy color",  "drop shadow color"
FOR i=0 TO 2
      READ variable$(i)
NEXT i
```

Declare "expert" graphics constants

```
DATA  0, 1, 2, 3, 6, 8,  15, 14, 11, 10, 9, 7,  0, 1, 2, 3, 6, 8
FOR i=0 TO 2
      FOR color=0 TO 5
            READ color(i,color)
      NEXT color
NEXT i
```

Declare english names for colors

```
DATA "black",  "blue",  "green",  "cyan",  "red",  "magenta",  "brown",  "gray",
      "dark gray",  "blue",  "green",  "cyan",  "red",  "magenta",  "yellow",  "white"
FOR i=0 TO 15
      READ color$(i)
NEXT i
```

MGI stands for **M**icrocomputer **G**raphics **I**ntelligence.

Initialize programming environment

Before sending instructions to the computer, normal system errors must be trapped. A simple fault in any unit can cause the whole system to sieze. In the case of file creation, if an error is detected, a new file is created for that session.

Initialize graphical environment

This software utilizes Media Cybernetics' HALO graphics library. The addresses for each routine are defined in hexadecimal order at the end of the program. The binary functions are loaded from disk resident software, along with the appropriate graphics driver, which is then installed.

Initialize constants and variables

Here the program's parameters are defined. Multiple variables are declared as arrays to ease repetitive processing. The keyword "do you want" is stored as a string variable, to facilitate dialog handling. The default color values for the background, foreground, and dropshadow colors are defined. The text window's colors are set to the defaults.

Declare graphical variables

In this section, the english names for the three color variables are declared and stored in the array, VARIABLE$. Of course, these appelations need not be english, they need not even be words at all. The computer relates its knowledge about these and other parameters independently of their definition.

Declare "expert" graphics constants

The term "expert" refers to the fact that these colors, once declared here in computer terms, are immutable. Neither the computer nor the user can alter these values or their order, without affecting further program use. The color values have been grouped according to the controlling color variables.

Declare english names for colors

Again, the names of the colors are completely arbitrary, in that the strings that replace them can be any alphanumeric characters. The duplicatation of color names is a reflection of the dual brightness levels of the EGA display mode. The last eight colors are high intensity versions of the first. The arrays VARIABLE$ and COLOR$ can be altered at any time, and will only change subsequent interactions with no effect on previously stored data. The two dimensional, real number array COLOR, should never be altered, because the interaction and memory of the system depend on that expertise.

Open workstation

```
CLS
INPUT"initials:    ",initials$
IF RIGHT$(DATE$,2)="80" THEN
        CLS
        INPUT "date:    ";date$
        DATE$=date$
volume$=LEFT$(initials$,3)+LEFT$(DATE$,2)+RIGHT$(DATE$,2)
```

Open volume

```
100   OPEN volume$ FOR INPUT AS #1
IF badfile THEN
        CLOSE
        GOSUB 200
i=0
INPUT #1, file$(i)
PRINT i, "new file"
WHILE NOT EOF (1)
        i=i+1
        INPUT #1, file$(i)
        PRINT i, LEFT$(file$(i),8), RIGHT$(file$(i),3)
WEND
CLOSE
PRINT i+1, "quit"
PRINT " Which    "+doyouwant$;
INPUT number
IF number=i+1 THEN
        END
IF number=0 THEN
        number=i+1
        GOSUB 300
```

Open file

```
file$=file$(number)
OPEN file$ FOR INPUT AS #1
FOR i=0 TO 2
        INPUT #1, color%(i)
NEXT i
CLOSE
```

Open workstation

At this point, the program becomes separate procedures and functions of loops and subroutines. This section is the log-on routine for computer recognition. The user enters his initials, or any three characters and the computer opens an account. If the system does not have a clock on line, it asks the user the date. This information becomes concatenated into a workstation account title that can be organized by user, date or both. As this system is date dependent, the user renews his account every month.

Open volume

The account itself is a list of the user's clientele and job titles. Here, updating the account directory on the screen, is the first part of the infinite loop that constitutes the main structure of the program. All subsequent instructions outside of this loop are subroutines for this loop. The first instruction is to open the account volume, and failing that, go to the routine to create one.

Another loop is initialized, with the variable 'i' as the increment. A null client is appended as the zeroth client, with the title of "new file". The client list is then gathered from the account by the use of a while loop that tests for the end of the file. Each client/ job title listing is parsed and printed on screen. and the file closed at the ith listing. The ith plus one listing becomes a null client with the title "quit".

The user is then prompted to select from the above menu of titles. The user's choice, stored in the variable 'number,' is compared with the zero and ith plus one increments. If the user chooses zero, the program branches to the file creation routine. If the user chooses ith plus one, the program executes a clean ending.

Open file

If the user opts to neither create a new file, nor terminate the session, then the implied selection is the number of the file the user wishes to open. If this is true, the program opens the appropriate client file, and retrieves the file's color parameters. This retrieval is done with a simple, three-line, FOR/NEXT loop and the file is then closed.

Display selected constants

```
mode%=4                                 CALL INITGRAPHICS(mode%)
x=0: y=0: x2=1: y2=1                    CALL  SETVIEWPORT(x,y,x2,y2,color%(0),color%(0))
x=-200: y=-150: x2=200: y2=150          CALL SETWORLD(x, y, x2, y2)
                                        CALL SETCOLOR(color%(0))
                                        CALL CLR
x%=4: y%=4: path%=0: mode%=0            CALL SETTEXT(x%, y%, path%, mode%)
x=-107: y=17                            CALL MOVTCURABS(x,y)
                                        CALL SETTEXTCLR(color%(2), color%(0))
text$="Sample text"                     CALL TEXT(text$)
x=-110: y=20                            CALL MOVTCURABS(x, y)
                                        CALL SETTEXTCLR(color1%, color0%)
                                        CALL TEXT(text$)
INPUT answer$                           CALL CLOSEGRAPHICS
CLS
GOTO 100
```

New volume

```
200   OPEN volume$ FOR OUTPUT AS #1
      PRINT #1,"New file"
      CLOSE #1
      FOR i=0 TO 2
            memory$=LEFT$(initials$,3)+LEFT$(variable$(i),4)+".dat"
            OPEN memory$ AS #1 LEN=85
            FIELD #1,80 AS branch$,2 AS yes$,2 AS no$
            LSET yes$=MKI$(2)
            LSET no$=MKI$(4)
            PUT #1,1
            LSET branch$=doyouwant$+"a different    "+variable$(i)
            LSET yes$=MKI$(3)
            LSET no$=MKI$(4)
            PUT #1,2
            LSET branch$="07"+doyouwant$+color$(7)
            PUT #1,3
            color$=RIGHT$(STR$(color%(i)),LEN(STR$(color%(i)))-1)
            IF color%(i)<10 THEN
                  color$="0"+color$
            LSET branch$=color$+doyouwant$+color$(color%(i))
            PUT #1,4
            CLOSE #1
      NEXT i
```

Display selected constants

Once the client's color preferences are determined, the program displays them using the graphics library that was loaded at the beginning of the program. The many CALLs here are instructions to that binary library. The graphics environment itself must now be initiated, and the viewing parameters adjusted. The screen jumps from the operational text defaults, to the new graphics color in the client's file. The coordinates for the environment and display copy are established from default values.

Of course, the entire operation of the program could be handled in this environment, but the demand on the library to handle input and output was too much experimentation for the deadlines of this thesis. The graphics library supports many other variables that have not been manipulated here. As mentioned before, a more complete version of this software would be a more powerful graphical tool, but the need for redundancies and safeguards, would make the fundamental algorithm too cumbersome.

Once the program has displayed the values for the current client file, it waits for the user to acknowledge the display with a keypress. Upon response, the program erases the display, and returns to the file menu at the beginning of the main loop.

New volume

Now, the program structure breaks down to component subroutines only. This procedure creates a new account for every new operator, or for a new operational month. Command control is routed to this routine by way of a "file not found" error. If a user logs on that the computer does not recognize, or an old operator renews his account, the syntax of their new account will so confuse the machine that it will branch to this routine to create the necessary files.

In addition to the client file, this subroutine also creates three memory files. Here labelled 'i=O TO 2,' these three files represent the infant intelligence of the program. Each file corresponds to a separate field of experience, where the name of the memory file is a concatenation of the 'ith' variable and the operators initials.

Each memory file is defined as a random access file of record length 85. The first 80 characters of the file are reserved for question and response strings, while the remaining characters are devoted to affirmative and negative branching integer pointers. The first record contains the data on the entire file, including a record total and a pointer to the last record.

Questions are stored straightforward, as an interrogative string, with two pointers to the affirmative and negative records. The answers, whether yes or no, are stored as a concatenation of the numerical color value, and the prompt for that color. No pointers are included, as these branches are terminal.

Once the initial query for a change of state is recorded with the negative value of the default state and the standard affirmative option of light gray, the file is closed and the routine loops on to next color variable. Once all the variables are declared in the infant intelligence of the machine, the routine continues to the next set of instructions rather than returning.

Append volume

```
300   OPEN volume$ FOR APPEND AS #1
      CLS
      INPUT"client:   ",client$
      IF LEN(client$)<8 THEN
            client$=client$+SPACE$(8-LEN(client$))
      CLS
      INPUT"title:   ",title$
      IF LEN(title$)<3 THEN
            title$=title$+SPACE$(3-LEN(title$))
      file$=LEFT$(client$,8)+"."+LEFT$(title$,3)
      PRINT #1,file$
      CLOSE
```

New file

```
      FOR j=0 TO 2
            CLS
            current%=2
            memory$=LEFT$(initials$,3)+LEFT$(variable$(j),4)+".dat"
            OPEN memory$ AS #1 LEN=85

400         FIELD #1,80 AS branch$,2 AS yes$,2 AS no$
            GET #1,current%

            IF LEFT$(branch$,1)="0" OR LEFT$(branch$,1)="1" THEN
                  GOTO 600

500         question$=branch$
            GOSUB 2000

            IF badanswer THEN
                  GOTO 500

            previous%=current%

            IF yes THEN
                  current%=CVI(yes$)

            IF no THEN
                  current%=CVI(no$)

            GOTO 400
```

Append volume

This routine can be branched to directly, for the addition of a new file to an existing account.

Once the computer has a set of rote questions and pat answers, it must now apply its new found knowledge. In this section, the program switches from passive file initialization to interactive file labelling. The computer asks for, and processes the client and job titles. These values are then written to the user's account file, which is then closed. Program flow goes on to the following routine, rather than returning to the main loop.

New file

The next step for a new user is the creation of the data for the new client. A three repetition loop is begun, to gather information about the new client. Each of the three memory files about color are retrieved and FIELDed. The second record is gotten and scanned for a resulting color. If the branching string is not an answer then it is a question and the string is sent to the interrogation routine. Depending on the operator's response, the program branches to the next level, or re-asks the question. If the user satisfactorily answers the question, the pointer to the current branch is adjusted to the pointer specified in the record for that response. The program then loops back to the FIELD statement and gets the specified record.

Check end of branch for satisfactory color

```
600     no2=no
        color%(j)=VAL(LEFT$(branch$,2))
        branch$=RIGHT$(branch$,LEN(branch$)-2)

700     question$=branch$
        GOSUB 2000

        IF badanswer THEN
                GOTO 700

        IF color%(1)=color%(0) AND yes THEN
                GOSUB 1000
                GOTO 200

        IF yes THEN
                color$=branch$
                CLOSE
                NEXT j
```

Set memory aside for learned color

```
FIELD #1,80 AS branch$,2 AS yes$,2 AS no$
GET #1,1
total%=CVI(yes$)+1
last%=CVI(no$)+2
LSET yes$=MKI$(total%)
LSET no$=MKI$(last%)
PUT #1,1
```

Adjust binary structure to accommodate new color

```
FIELD #1,80 AS branch$,2 AS yes$,2 AS no$
GET #1,previous%
IF no2 THEN
        LSET no$=MKI$(last%-1)
ELSE
        LSET yes$=MKI$(last%-1)
PUT #1,previous%
```

Check end of branch for satisfactory color

Once the program has gotten a binary response to a color question that corresponds to an exact color, the next step is to ask whether the selected color is what the user wanted.

If the user was given this color as a response to a previous negative statement, then that result is stored in the temporary variable 'no2.' The numerical value of the color and the question are stripped from the string, and the program asks the question. The routine branches to the answer verification subroutine, and returns. If the user does not answer correctly, the program loops back and re-asks the question. If the chosen color conflicts with previous choices, then the machine branches to a warning and the user must re-enter a nonthreatening color.

The conflict warned of here is when the copy color and the background color are identical. Any expert rule that can be expressed in boolean or otherwise mathematical terms, can be added at this point. Also, this is a punishment method of warning. A more advanced and active system may influence the user not through restriction, but through suggestion. This would require a more extensive diagnostic system, and would deviate from the intention of this thesis.

If this color is acceptable to the computer and the operator, then the routine loops to the next color variable.

Set memory aside for learned color

However, if the computer's best match for a color at this point is unacceptable to the operator, the program FIELDs the first record and finds out what the index for the last record is, for memory expansion. The values for the total and last index of all the records are stored in the 'yes' and 'no' fields respectively of the first record. The necessary adjustments are made to these numbers and the new values are PUT back into the first record.

Adjust binary structure to accommodate new color

After the housekeeping for the whole file is done, the computer must determine where in the binary tree to place the next color. As the last value was unsatisfactory, the next record must be located relative to the previous branch. If the previous branch was also unsatisfactory, then its negative pointer becomes the index of the record that immediately preceded the latest record. If the color that the computer presents is bad, but the record before was good then the index for the previous branch is stored in the record's affiramtive pointer. The record is then restored to its location as the preceding branch.

Display available palette

```
800     CLS
        FOR color=0 TO 5
                COLOR color(j,color),7
                PRINT color(j,color);
        NEXT color

        IF j=1 THEN
                CLS
                FOR color=0 TO 5
                        COLOR color(j,color),0
                        PRINT color(j,color);
                NEXT color

        COLOR 14,1
        PRINT "which color "+doyouwant$;          INPUT color
        color$=color$(color):color%(j)=color

        IF color%(1)=color%(0) THEN
                GOSUB 1000
                GOTO 800
```

Learn preferred color

```
        CLS
        FIELD #1,80 AS branch$,2 AS yes$,2 AS no$
        PRINT "why "+doyouwant$+color$;           INPUT question$
        LSET branch$=doyouwant$+question$

900     question$="is this true for "+color$
        GOSUB 2000

        IF badanswer THEN
                GOTO 900

        IF no THEN
                LSET yes$=MKI$(current%)
                LSET no$=MKI$(last%)
        ELSE
                LSET yes$=MKI$(last%)
                LSET no$=MKI$(current%)

        PUT #1,last%-1
```

Display available palette

Now that the computer knows what the relative description of the chosen color must be, it must find out the numeric value of that color. To facilitate the selection of the preferred color, the computer presents a selected menu of its native colors.

The menu is limited by the preset values in the DATA statement at the beginning of the program. These predetermined values represent the passive expertise of the system. By not showing the user the entire range of mechanical colors, a poor choice is more difficult. If the operators wish to use a color that does not appear in the menu, they must enter the value for that color blindly. The machine only provides assistance for ranges of acceptable colors.

In the case of the brighter colors of the foreground copy, a second palette is drawn over the other palettes which is also representative of the preferred choices.

The menu is very simple in technique, and restricted by the native ability of the machine to display all color values in a text menu. But, the display is a very good example of how even a little expertise added to the system provides for more efficient production.

Another expert default example is when the computer compares the user's color with the other color variables, for conflict. Here, an active warning system, rather than a passive restraint menu, is employed. Like the earlier use of this device, the operator cannot ignore the advice of the machine.

Learn preferred color

This program provides the default expertise, but more importantly, learns from the use of the passive and active restraint what the user really wants. Here is where the program really exhibits its intelligence. This set of instructions asks why the user chose the color, and goes on to verify the relationship of the choice versus the color value.

What is not immediately obvious is that an operator has the ability to choose a value that does not satisfy a particular qualification that the computer learns. This is distinguished as a contradiction of the branching statement, and the pointer for the learned color indicates the retrograde direction of the contradiction.

This action is specified as a swapping of the pointers in anything but a negative response. If the chosen color is correct, relative to the learned reasoning, then the affirmative pointer for that value is placed at the index of the last record, and the negative pointer set to the current record. If the chosen color goes against the learned reasoning, then the current and last pointers are reversed. In either case, the approriate information is PUT away in the preceding record.

As confusing as the relationships of the branching reason and results are, the binary tree seems too mechanical to be classified as an intelligent process. Regardless of the automatic nature of the program, the best method of describing the routine is as a learning process. The computer is being trained, but in very subjective terms that may or may not make sense to everyone.

Update binary logic for new color

```
        FIELD #1,80 AS branch$,2 AS yes$,2 AS no$
        colornumber$=RIGHT$(STR$(color%(j)),LEN(STR$(color%(j))))

        IF color%(j)<10 THEN
                colornumber$="0"+colornumber$

        LSET branch$=colornumber$+doyouwant$+color$
        PUT #1,last%
        CLOSE
NEXT j
```

Append file

```
OPEN file$ FOR OUTPUT AS #1
FOR i=0 TO 2
        PRINT #1,color%(i)
NEXT i
CLOSE #1
RETURN
```

Bad design warning

```
1000  COLOR 14,4
      CLS
      INPUT "BAD CHOICE",answer$
      RETURN
```

Check binary structure

```
2000  FOR i=78 TO 1 STEP -1
        IF RIGHT$(question$,i)=SPACE$(i) THEN
                text$=LEFT$(question$,LEN(question$)-i)
      NEXT i
      CLS
      PRINT question$;                          INPUT answer$
      badanswer=0:yes=0:no=0

      IF LEN(answer$)=0 THEN
              badanswer=1
              RETURN
      answer$=LEFT$(answer$,1)

      IF answer$="Y" OR answer$="y" THEN
              yes=1
      ELSE
              no=1
      RETURN
```

Update binary logic for new color

Once the computer has learned the user's color preference, the new knowledge must be committed to memory. The associated color value is set in a two digit syntax, and then concatenated with the 'doyouwant' string, and the english name of the color. The new string is PUT into the last record, the file is closed, and if there are no more color variables to learn, the next routine is run.

Append file

Once the program has completely exhausted all color variables for the job in question, the learned colors must be stored in the associated client file. The operator's account files were closed in the last routine, leaving plenty of RAM for the client file to be opened, and the variables printed into the correct sequential file. This process is accomplished using a three repetition FOR/NEXT loop, the file is closed, and control finally RETURNed to the main program loop.

Bad design warning

In addition to the preceding multipurpose subroutine, four other procedures and functions exist to serve the main loop. The first of these extra routines is the active, bad-design warning. This function interrupts the normal interaction, and displays a yellow and red warning message. The operator removes the display with a keypress, returning control to the main program.

Check binary structure

Another crucial function is the input and verification of binary answers. This subroutine takes a temporary string called 'question$' and truncates the trailing spaces. The main purposes for removing superfluous characters are making the string length of the user's input fit the random-access, record length, and facilitate concatenation.

Once the text string is truncated, it is printed on the screen as a question, prompting further user input. After the input is gathered in the variable 'answer$' the three state parameters are initialized at a null state. The variables 'yes,' 'no,' and 'badanswer' are boolean expressions where the zero state is off and any other value is on.

If the user inputs nothing, the program RETURNs with a 'badanswer,' forcing the reasking of the question 'question$.' Any other input is truncated to only the first letter, which is compared with the upper and lower case versions of the letter "y." If the 'answer$' satisfies the condition, then the boolean, 'yes' becomes true, else the 'no' case is true. Instruction control then RETURNs to the main loop.

Error trap

```
3000  IF ERR=53 THEN
          badfile=1
          CLOSE
          RESUME NEXT
      PRINT ERR,ERL
      END
```

4000 **Graphic library routine addresses**

```
SETCOLOR        =&H0
SETTEXT         =&HCB
SETTEXTCLR      =&HD2
TEXT            =&HD9
MOVTCURABS      =&HE7
SETVIEWPORT     =&H134
SETWORLD        =&H142
SETDEV          =&H261
INITTCUR        =&H2A7
CLOSEGRAPHICS   =&H2C3
CLR             =&H2CA
INITGRAPHICS    =&H302
SETFONT         =&H4A6
SETSTEXT        =&H4AD
SETSTCLR        =&H4B4
STEXT           =&H4BB
INQSTSIZE       =&H4D0
RETURN
```

Error trap

Another subroutine is the error trap function called from the beginning of the program. This set of instructions is critical to the development of the software, for any minor miscarriage of operation could completely destroy the integrity of the system. Such safeguards are necessary in the multitasking environment artificial intelligence and graphics demands.

Fortunately such redundancies as error traps also provide simpler solutions to repetitive problems. The prime example of this kind of algorithm in this program is the use of an error trap code to determine if a file being called exists or not. If this code is encountered, normal execution continues with the next instruction, which in this program, is a file creation function. If the error is of some other program, system, or mechanical problem, program execution is halted and the error code and offending line are printed.

Graphic library routine addresses

The last procedure, starting at line 4000, is a directory of the HALO graphical routines. While for the purposes of this thesis the program does not strictly require such intense graphics as HALO supports, any other incarnation would.

The hexadecimal addresses of the machine language routines are listed in ascending order, with the system's original names. Certainly Media Cybernetics' library is more than adequate to reproduce the features and functions of the Autographix, but the purpose here is not to produce a competitive graphics package, but to demonstrate the feasibility of the combination of graphics and AI.

Including the remainder of the HALO addresses would fill three more pages, but more signicantly, program execution would be endangered by the additional functionality. At this level of complexity, operations fail from hardware inadequacy one tenth to one fifth of the normal logins. These figures are averages of the switching failures of the program in random disk access, graphics addresses, and program instruction. The failures take the form of inconsistent displays, string values, or outright program crashes. The data comes from extensive tests beyond normal development trials, and allows for regular operator error.

THE END

Artificial intelligence and graphics are two of the most information-intensive fields of the computer industry. A computer's sole purpose is to process data, but the demands of graphics, and environment have influenced the shape of the machine at the microcomputer level. As graphics programs have become larger and more comprehensive, the computer's architecture has expanded to handle the increased load. Microcomputers can now mimic multitasking by performing many simple tasks in very rapid succession, but pairing AI and graphics requires such tremendous switching in the serial operation of the computer that the two fields are nearly mutually exclusive.

The use of mass memory storage and sophisticated hardware and software integration have expedited the attempted wedding of AI and graphics. The use of coprocessors improves computational speed, but by diverting serial execution, not by distributing parallel instructions. The single information channel in microprocessors just cannot support artificial intelligence **and** computer graphics. A smart, graphics microcomputer must be a true parallel machine, or a symbiosis of many dedicated, serial machines.[4]

Must a computer have graphics and intelligence? As mentioned before, a longitudinal study of a class of art and design students offers some proof that AI is necessary. In an initial questionnaire (see fig. 3) to the entire population, only six of fourteen computer graphics students (42%) made color choices for word slides that did not conflict with accepted color interaction theory.[5] At the end of the class, the students could effectively operate the equipment, but their average for the quarter only rose to 57% (73 acceptable choices of 127 total slides) in a controlled observation.

---

[4] See the section "Limitations of a Serial Machine", in Understanding Computers: Artificial Intelligence, (Alexandria: Time-Life Books, 1986), pp. 104-110.

[5] Additional rules for correct color determination and usage were adapted from Joseph Alber's The Interaction of Color, (New Haven: Yale University Press, 1975).

The Autographix's guidelines have evolved to enforce the use of bright hues, especially yellow, on dark backgrounds (i.e. blue, green, black) with dark, black, or no drop shadow.[6] However, the students continued to place complementary hues of copy and drop shadows of the same value upon high saturation backgrounds. Given mechanical color templates and prior knowledge of color theory, nearly half of the subjects still did not produce quality images after three months of instruction. When experienced graphics students with dedicated equipment misapply color, the computer must assume more responsibility for that information.

Perhaps the present low demand for artificial intelligence support cannot justify overcoming the difficulties of implementing algorithms at the microcomputer level. Certainly the proliferation of graphical information has promoted a reorganization of the media involved, but the tools need to assume more responsiblity for the data they supply. Bad design affects the legiblity of information, and if a computer can learn the operator's preferences and govern them with regard to good design, the designer will be free to be more innovative with that information.

---

[6]Software templates and rules of thumb for equipment use are well documented. (See the Theory Manual [Waltham: Autographix Inc., 1982], 2.1.3 pp.11-12.)

AUTOGRAPHIX COLOR QUESTIONNAIRE

As you prepare to make yet another Autographix slide, you'll pick your predominant colors from the many possible combinations. When you finally make those decisions, please record them here.

1) What background color did you choose?_____

2) What type color did you choose?_____

3) What drop-shadow color, if any, did you choose?_____

If the colors you just listed are anything like the color guidelines listed below, please circle your choice.

	BUlW	GNlW	BRlW	BKlW
1) Background	**Dark Blue**	**Dark Green**	**Dark Brown**	**Black**
2) Type	**Light Yellow**	**Light Yellow**	**Light Blue-Green**	**Yellow**
3) Drop-shadow	**Black**	**None**	**Dark Blue-Green**	**None**

If your color choices don't match any of these guidelines, why did you choose the colors you listed above (i.e. "I like 'em for their warmth.", "They look earthy." or "These colors best match the manufacturer's PMS colors.")?_____

_____

_____

If you care to volunteer more information, please record any other kinds of color/type relationships. What colors did you use for the copy, bullets, legends, etcetera? If the colors represent some abstract (non-design related) choice, what is that scheme? Are they picked for legiblity? Is there a color you'd like to use, that's not in the palette? How much of the color do you use, that is, how's your type weight versus color density?_____

_____

_____

_____

_____

_____

Figure 3. A sample questionnaire.

BIBLIOGRAPHY

Albers, Joseph. Interaction of Color. New Haven: Yale University Press, 1975.

Bolt, Richard A. The Human Interface, Where People and Computers Meet. Belmont: Lifetime Learning, 1984.

Brand, Stewart. The Media Lab, Inventing the Future at MIT. New York: Viking, 1987.

Comfort, Alex. Reality & Empathy: Physics, Mind, and Science in the 21st Century. Albany: State University of New York Press, 1984.

Edwards, Betty. Drawing on the Artist Within. New York: Simon and Schuster, 1986.

Godman, Arthur. Barnes & Noble Thesaurus of Computer Science. New York: Barnes & Noble Books, 1984.

Haack, Susan. Deviant Logic: Some Philosophical Issues. London: Cambridge University Press, 1974.

Hofstadter, Douglas R. Godel, Escher, Bach: An Eternal Golden Braid. New York: Vintage Books, 1979.

Horn, Delton T. Smart Apples, 31 Artificial Intelligence Experiments. Blue Ridge Summit: Tab Books Inc., 1987.

Keough, Robert P. Rochester Institute of Technology, Rochester, New York. Reports and conversations, 1986-1988.

Long, Marion. "Turncoat of the Computer Revolution." New Age Journal, December 1985: pp. 47-78.

Mishkoff, Henry C. Understanding Artificial Intelligence. Indianapolis: Howard W. Sams & Co., 1985.

Schank, Roger C. and Childers, Peter G. The Cognitive Computer: On language, Learning, and Artificial Intelligence. Reading: Addison-Wesley Publishing, Inc., 1984.

Shirai, Yoshiaki and Tsuji, Jun-ichi. Artificial Intelligence: Concepts, Techniques and Applications. New York: John Wiley & Sons, 1982.

Whitney, Patrick, ed. Design in the Information Environment. New York: Alfred A. Knopf, 1985.

Wiersma, William. Research Methods in Education: An Introduction. Boston:Allyn & Bacon, Inc., 1986.

Wilson, Stephen. Using Computers to Create Art. Englewood Cliffs: Prentice-Hall, 1986.

Instruction Manual, Theory Manual, Reference Manual. Waltham: Autographix Inc., 1982.

Understanding Computers: Artificial Intelligence. Alexandria: Time-Life Books, 1986.