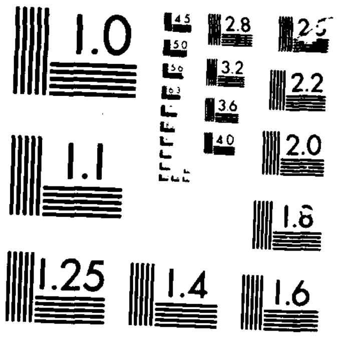
AD-A169 053   TACTICAL DISPLAY SIMULATION ON THE H/Z-100(U) NAVAL   1/2
POSTGRADUATE SCHOOL MONTEREY CA K W COOMES MAR 86

UNCLASSIFIED   F/G 9/2   NL



MICROCOP' CHART

2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TACTICAL DISPLAY SIMULATION ON THE H/Z-100

by

Kenneth W. Coomes

March 1986

Thesis Advisor: Uno R. Kodres

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL *(If applicable)* Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000	7b ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL *(If applicable)*	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS *(City, State, and ZIP Code)*	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE *(Include Security Classification)*

TACTICAL DISPLAY SIMULATION ON THE H/Z-100

12 PERSONAL AUTHOR(S)
Coomes, Kenneth W.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	1986 March 27	143

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	TACTICAL DISPLAY SIMULATOR, H/Z-100, GRAPHICS, ZBASIC, MACRO-86, ASSEMBLY LANGUAGE

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis explores the feasibility of developing a tactical display simulator on the H/Z-100 microcomputer. A prototype simulator is implemented in ZBASIC, some graphics functions reoutines are implemented in Macro-86, and timing and performance measurements are performed for comparison.
Listings of the programs developed are presented, as well as instructions for their effective use. Directions for the modification of the code, and suggested profitable areas of exploration and further development are included.
It is concluded that a tactical display simulator is feasible, and that the final implementation should be in Macro-86.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Uno R. Kodres	22b TELEPHONE *(Include Area Code)* 408 646-2197 22c OFFICE SYMBOL 52Kr

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete
SECURITY CLASSIFICATION OF THIS PAGE

Tactical Display Simulation on the H/Z-100

by

Ken Coomes
Lieutenant, United States Navy
B.S.E.E., University of Washington, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March, 1986

Author: [illegible]
Ken Coomes

Approved by: Uno R. Kodres
Uno R. Kodres, Thesis Advisor

[illegible]
Ron Rautenberg, Second Reader

[illegible]
Vincent Y. Lum, Chairman, Department of
Computer Science

[illegible]
Kneale T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This thesis explores the feasibility of developing a tactical display simulator on the H/Z-100 microcomputer. A prototype simulator is implemented in ZBASIC, some graphics functions routines are implemented in Macro-86, and timing and performance measurements are performed for comparison.

Listings of the programs developed are presented, as well as instructions for their effective use. Directions for the modification of the code, and suggested profitable areas of exploration and further development are included.

It is concluded that a tactical display simulator is feasible, and that the final implementation should be in Macro-86.

THESIS DISCLAIMER

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis will be listed below, following the firm holding the trademark:

1. Zenith Data Systems Corporation

    Zenith

    H/Z-100

    Z-DOS

    ZBASIC

2. Microsoft Corporation

    Microsoft

    MS-DOS

    GML

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

To Dr. Uno Kodres, whose patience and cooperation paved the way for the completion of this thesis. To CDR Ron Rautenberg, whose editorial comments improved the quality of the finished product. To LCDR Paul Callahan, who graciously donated time to proof-read the finished product. And especially to my wife Joanne, in appreciation of the loving support and surrender of time together, in order that I would be able to finish on time.

I. INTRODUCTION

A. BACKGROUND

The Naval Postgraduate School has arranged to purchase approximately fifty Zenith H/Z-100 microcomputers for the microcomputer laboratories of the Computer Science Department. There were numerous reasons for that particular microcomputer to be chosen, one of which is the fact that it possesses a dual micro-processor architecture [Ref. 1: p. E.1]. It has the 8085, an 8-bit microprocessor, which is typical of a simple architecture and instruction set, and able to run software under the CP/M operating system. It uses a simple, single-segment model of memory. The H/Z-100 also comes equipped with the newer and more powerful 8086, a 16-bit microprocessor, with an eight bit memory interface. It makes use of a more complex architecture, more internal registers (some useable as 8- or 16-bit), extended addressing modes, and a more complex memory management scheme, with segmentation registers.

Another attractive feature of the H/Z-100 is its graphics capability. In the character display mode, the H/Z-100 can display 25 rows of 80 characters. A very important graphics ability is the degree of resolution provided. The H/Z-100 provides high-resolution dot addressability, with a dot resolution of 640 horizontal

dots by 225 vertical dots. In the interlace mode, the H/Z-100 provides 640 x 512 pixels. It also comes equipped with three 64K pages of video RAM memory, eight colors in the color option with a color monitor, or eight grey scale levels with the color option and a monochrome monitor , light pen circuitry, and the potential for two pages of video display to be stored simultaneously.[Ref. 1: p. E.2] As a natural consequence of this purchase, the micro-computer laboratory is interested in developing a variety of special-purpose software products to maximize the value of these computers. A primary use for these software products will be in courses which emphasize tactical applications of computers. Graphical displays are a vital and integral part of tactical applications. Software products which provide graphical support for tactical applications and demonstrations of interactive graphical displays that enhance tactical decision-making is very
desirable.

B. OPERATING SYSTEM SUPPORT

The operating system provided with the H/Z-100 computers purchased by the Naval Postgraduate School (NPS) is the Microsoft Disk Operating System. It is our opinion that the MS-DOS operating system is a powerful and practical one. It provides the user with 54 commands, 27 resident commands and 27 transient commands [Ref. 2: pp.

9.2-5.4]. Several of these commands are useful tools for the software developer.

There is a deficiency in the MS-DOS operating system, however, related to graphics-oriented applications. There is virtually no operating system support of graphics functions provided by MS-DOS, the one exception being the CLS (clear screen) command [Ref. 2: p. 5.2].

There is a crucial need for graphics function support of tactical applications involving real-time displays. The system is required to provide a current graphical display of the tactical situation at the same time that changes in that tactical situation are being evaluated. Computations of the time-dependent tactical elements, display of the current situation and acceptance of user inputs to the system must be performed simultaneously, or nearly so. The tactical display, which may be changing rapidly in the case of high-speed targets such as missiles and slowly where stationary and slow-moving tactical units are concerned, must be updated with a minimum frequency of once per second to be useful. This requirement generates the need for effective, special purpose, time efficient code.

C. PURPOSE

We propose, as an initial system development project to meet the purposes stated above, the development of a Naval Tactical Data System (NTDS) display simulator, to be implemented on the H/Z-100 microcomputer. A rapid prototype of a display simulator will be developed, to provide guidelines for developing graphics support tools for tactical systems applications such as those mentioned earlier. The systematic approach used in developing this prototype will demonstrate many of the considerations that graphics support tools must entail. The displays this prototype provides will be illustrative of typical tactical displays that these applications require. Portions of the prototype will be transferred into more time efficient code, in order to ascertain the order of magnitude of efficiency gain that may be expected and to demonstrate the transfer process.

The development of an entire NTDS display simulator is a major undertaking. Although it sounds good as a concept, the logical first step would be a preliminary feasibility study. This study will be an initial look at the feasibility of developing an NTDS display simulator on the H/Z-100. We will be attempting to ascertain that feasibility by exploring algorithms necessary to support the graphics sub-system of such a system, since graphic display is an integral part of the system being simulated and will largely determine the real-time aspects of the system. We will develop code for portions of the graphic display subsystem, and perform some performance tests on that code. We intend and hope for this to be the basis for the development of a graphics system which permits NTDS display subsystem simulation.

D. THESIS ORGANIZATION

In Chapter II we present algorithms devel ped for this initial project. We describe the symbol generator subsystem algorithm in detail. Design focuses primarily on the symbol generator itself. We also provide some of the design considerations made, brief explanations of reasons for specific choices, and discuss some ramifications of alternatives. Algorithms employed to develop sample NTDS-type displays are also presented here.

Chapter III describes the code developed and debugged to date for the display simulator. We have a successful prototype implemented in ZBASIC and some initial routines for graphics support in Macro-86 assembler language.

The next chapter, Chapter IV, is devoted to efficiency issues. Much of the code provided in the Appendixes has sacrificed efficiency for clarity. We felt that, in a ground-breaking project such as this one, the use of easily traceable logic in the code was more valuable than the use of tricky code which might execute more rapidly. Initial performance tests and timing runs are summarized.

The final chapter, Chapter V, provides suggestions for the most effective use of what has been done to date. It also suggests some of the potential future directions for follow-on work. That is where the conclusions of this work are presented. The focus is on what has been learned that may be of value to other students and researchers.

Appendixes A-M are listings of the ZBASIC programs developed to date as a working prototype of the NTDS display simulator. A User's Manual which gives a line-by-line description of the programs in Appendixes A-M, and important considerations when modifying the code, is contained in Appendix N. Listings of Macro-86 programs are in Appendixes O-U. Appendix V is a User's Manual for the code in those Appendixes.

II. ALGORITHMS

A. BASIC SIMULATOR ALGORITHM

To perform the functions of an NTDS display simulator, we must implement an algorithm similar to that in Figure 2.1. The initialization depends somewhat on the implementation chosen. The display simulator requires that some basic items be displayed as a minimum, such as a working area on the screen (window), a reference grid, possibly some land or other special areas within the window and basic symbols. A more detailed explanation of what is displayed in our prototype is contained in the next chapter on code.

```
begin
     initialize display, system
     repeat
          repeat
               delay
               update all tracks
          until (there is a user service request)
          perform service requested
     until (user requests exit)
end
```

Figure 2.1 Basic Simulator Algorithm

The user service requests are also defined by the NTDS environment. We have chosen six NTDS-type user functions to implement, two of which are related to use of this simulator. Other functions may be added to the simulator easily enough, as explained in the code chapter. They are not limited by those required by the simulation. The system, particularly as a graphics subsystem, is eventually able to support multiple purposes, including services that are not NTDS related. The services implemented in the prototype are a sampling of services, and are not meant to be exhaustive.

B. EXPANDED SYSTEM ALGORITHM

The basic algorithm has been expanded through a series of step-wise refinements to the algorithm shown in Figure 2.2. The intervening steps are not included as they would not provide sufficient information that is not contained in the final version to warrant inclusion.

In some statements greater detail than is normally associated with algorithmic language has been brought forward to the algorithm itself and its explanation. This is to begin to acquaint the reader with the prototype. Some sections of the algorithm, whether they are represented as single or multiple statements in Figure 2.2, are expanded even further, where clarification was felt to be necessary or desired. Where greater expansion has been provided it is the aim of the author to provide insight in o the decision-making process during the design of such a prototype system. There has been more effort expended in attempting to provide clear logic in the algorithms and code presented than in driving for efficiency.

```
begin
     clear screen
     initialize variables, tracks
     display function key menu
     read (window parameters)
     display window
     read (# of land masses to display)
     if (# of land masses > 0) then
          display land masses
     read (Y-axis parameters, X-axis parameters)
     display coordinate axes
     read (# of tracks)
     if (# of tracks > 0) then
     begin
          for (# tracks) times
          begin
               read (current track parameters)
               look-up symbol to match parameters
               make track active
               calculate incremental movement of track
               look-up speed leader to match parameters
          end
     end
     repeat
          while (no user input)
          begin
               update all tracks
               delay
          end while
          if (user input is a function key) then
               do indicated function
     until (user input = halt)
end
```

Figure 2.2 Expanded System Algorithm

1. Expansion of "Display Land Masses"

Figure 2.3 is an expansion of the "display land masses" statement of the Expanded System Algorithm. It is shown because the lack of true dynamic memory allocation in the prototype created special problems. This algorithm expansion illustrates one type of solution to those problems. The problem is one that will not be evident in the programming language which will be used in the future to fully develop the graphics system. We accepted the problem here, and dealt with it in this manner, in order to develop the rapid prototype.

In this prototype we provide for three land masses. The elements of the PTS array must be initialized, because they are used to determine the amount of memory to allocate for the land mass arrays. As a consequence, multiple iterative loops are contained in the algorithm. Because they are pre-initialized, the elements of the PTS array are used as flags to indicate when to stop drawing land masses. The lower bound of two was selected to allow this module to draw even tiny representations on the display.

This is an example of a design decision point. There are other ways that the lack of dynamic memory allocation could have been handled. This is not the most efficient choice, but it presents uncomplicated logic. The data in this solution establishes a pattern. The number of points and color for the maximum number of land masses the

```
begin
     for (# land masses) times
          read (PTS, LCOL)   (# points, color for
                                each land mass )
     for (PTS(1)) times
          read (X, Y)        ( points for LAND1 )

     for (PTS(2)) times
          read (X, Y)        ( points for LAND2 )

     for (PTS(3)) times
          read (X, Y)        ( points for LAND3 )

     draw LAND1
     read (CENTX, CENTY)     ( interior pt. )
     paint LAND1

     if (PTS(2)) >= 2 then   (if there is LAND2)
     begin
          draw LAND2
          read (CENTX, CENTY)
          paint LAND2
     end
     else end

     if (PTS(3)) >= 2 then   (if there is LAND3)
     begin
          draw LAND3
          read (CENTX, CENTY)
          paint LAND3
     end
end
```

Figure 2.3 Display Land Algorithm

system will handle must be provided as data, in order to allow dimensioning of arrays. Since a number of points for any dummy land (one point) is required in the data, this solution requires that each dummy land mass have that one point in the data. This also ensures that a user who wishes to modify one data file rather than generate a new one has created space in the data module for the added land masses.

We note again that these are problems that will be nonexistent in Ada, as well as in Macro-86 graphics functions. They result from the use of ZBASIC in this prototype, which was selected simply for the rapidity with which a working demonstration of the display simulator could be developed. This provides early feasibility determination, guidelines for future development, and demonstrates graphics concerns.

2. Expansion of "Display Coordinate Axes"

This algorithmic step is expanded for those who have little or no computer graphics background. The simple algorithm shown in Figure 2.4 deals with the problem of maintaining the proper aspect ratio between the horizontal and vertical axis scales. It is based on a program cited in the code and written by James C. Adams [Ref. 3: p. 9-15].

The aspect ratio in computer graphics is the ratio of horizontal to vertical on the display. The H/Z-100 monitor provides 640 pixels in the horizontal direction and 225 pixels in the vertical direction [Ref. 2: p. E.1]. In order for the scale divisions on the two axes to appear equal they must have different pixel spacing.

```
begin
     calculate horizontal scale
     calculate vertical scale
     draw vertical axis
     draw horizontal axis
     draw horizontal scale divisions
     draw vertical scale divisions
end
```

Figure 2.4 Display Axes Algorithm

3. Expansion of "Do Indicated Function"

The "do indicated function" is expanded because it is a case statement. That fact is not evident by looking at the code, since it is written in a language that does not provide a case statement construct.

We considered only partially expanding this section of the system algorithm. Case statements are widely understood, and tend to become repetitive. Since one of the concerns of this project is to provide development history, and another is to document some decision process in practice, we elected to fully expand it. The full expansion is illustrated in Figure 2.5.

```
begin
     case function of:
     begin
          halt:  begin
                    clear screen
                    restore function keys
                    exit system
                 end
          suspend/continue:
                 begin
                    wait for user input
                 end
          hook track:
                 begin
                    if (some track hooked) then
                       unhook track
                    read (track to hook)
                    hook track
                 end
          enter track:
                 begin
                    read (track parameters)
                    calculate track movement
                    look-up speed leader, symbol
                    display track
                 end
          modify track:
                 begin
                    if (some track hooked) then
                       unhook track
                    read (track to modify
                    hook track
                    for (each track field)
                    begin
                       ask user if OK
                       if not OK then
                          make change
                    end
                 end
          delete track:
                 begin
                    read (track to delete)
                    make track inactive
                 end
     end (case)
```

Figuie 2.5 Do Indicated Function Algorithm

C. CONCLUSION

These algorithms are simply the rough-hewn blueprints for producing the prototype display simulator. Comparing them with the code in Appendices A-M discloses some of the design choices which had to be made during implementation. Inspection of the algorithms alone reveals some of the pre-thinking that they embody. Knowing the capabilities and limitations of the target system and the programming language influenced the construction of the algorithms. In some instances that was beneficial, in others less so.

Algorithms could have been shown for each module of the working prototype, no matter how trivial. This would have served no useful purpose. The important design lesson here is that the more carefully the algorithm was developed and thought out before the module was coded the more rapidly the coding was successfully carried out.

III. CODE

A. LANGUAGE

An early concern in any software development is the choice of a programming language. There are a plethora of languages and language subsets to choose from. Many times the choice is heavily dependent on the experience and preferences of the programmer(s) and the availability of the preferred language on the target machine. These dependencies may narrow the choices but do not define absolutely the language to use in most cases.

This NTDS display simulator is no exception to the considerations listed above. The development phase of a new piece of software is not the best of times to attempt to learn a new language. Therefore only languages familiar to us were considered. The H/Z-100 computers that NPS is purchasing will not come with software support for all currently existing programming languages. They are capable of using languages which possess most of the currently available language features.

The driving consideration in many projects, certainly one of the most important issues if not the most, is the project itself. Each language has features which are better for some applications and suffer inefficiency (or even impotence) for others. For the display simulator the two critical issues are graphics support and speed. The more graphics support the language provides, the more rapidly a prototype may be implemented and tested. The more efficient the language in terms of processing speed the better it is able to approach real-time updates of the display and its symbology.

We feel that Macro-86 assembly language offers the most real-time capability. If the NTDS display simulator were written in assembly language, it would probably meet or exceed the time requirements to provide realistic, dynamic displays. Macro-86 maximizes the utilization of the power of the H/Z-100 computer through segmentation and paging, as well as allowing direct access to the color planes for video control.

Macro-86 assembly language also offers a wide range of interfaces with high-level languages. In particular we are interested in Ada, which is available for the H/Z-100 and which is playing an ever-increasing role in Department of Defense applications. Ada has the ability to make use of Macro-86 routines. This would allow Ada packages to be written for numerous applications, making use of Macro-86 routines which provide basic graphics functions.

Macro-86 assembly language does not provide direct support for the graphics functions the display simulator needs. Even direct input and output requires special handling, register control, and several lines of code.

The considerations above led us to choose ZBASIC for an initial prototype implementation, and Macro-86 for ultimate development and production. We felt that a working prototype could be developed in considerably less time in ZBASIC (in fact, a prototype of the graphics display subsystem has been completed and is included) and used for experimentation, testing and further enhancement. We did develop some basic Macro-86 routines for graphics support of the simulator and testing, and they are included. The major drawback of ZBASIC for further development is its lack of compatibility with other high-level languages.

B. DATA STRUCTURES

It may help to visualize each track in this system as a record of the type illustrated in Figure 3.1. ZBASIC does not provide data structures which are composed of fields with different types. A separate array, therefore, represents each field of the TRACK record.

The other arrays are used as look-up tables for various attributes. The type of symbol assigned to a track is found in the array SYM$, the speed leaders are in LDR$, the number of points determining a land mass(up to three land masses are provided for) are found in PTS with each corresponding land mass color in LCOL. These tables (SYM$ and LDR$) are initialized in lines 180-440 (see Appendix A).

There are provisions for ten pre-defined symbols, and seven symbols are defined in this prototype. By changing the dimension of SYM$ and its initialization, any number of symbols (up to memory limitations) are defineable. Changes to the symbols should be accompanied by changes to the Match module, which assigns symbols to tracks based on their CLASS$. The dimension of CLASS$ would not be changed--its dimension, along with that of the other fields of each TRACK record (see Figure 3.1), determine the number of tracks the system will accommodate. These are some of the problems inherent in a ZBASIC rapid prototype which will disappear with Ada packages, and/or Macro-86 implementation of the display simulator functions.

```
type TRACK = record
     CLASS$ : array [1..9] of char;    (class)
     CUS    : integer;                 (course)
     SPD    : integer;                 (speed)
     TCOLOR : integer;                 (color)
     TX     : integer;                 (x position)
     TY     : integer;                 (y position)
     XINC   : integer;                 (x increment)
     YINC   : integer;                 (y increment)
     T$     : array [1..80] of char;   (symbol)
     L$     : array [1..2] of char;    (speed leader)
     HK$    : array [1..2] of char;    (scale)
     ACTIVE : integer;                 (active/ref pt)
end; (TRACK)
```

Figure 3.1 TRACK Record Structure

There are eight pre-defined speed leaders, which correspond to the four cardinal and four inter-cardinal directions. This, too, could be modified, with changes here and in the Move module, which assigns speed leaders to each TRACK based on course and speed.

The string construction of the elements of the SYM$ and the LDR$ arrays makes use of a "language within a language" that ZBASIC provides for graphics applications. This language is the Microsoft "Graphics Macro Language" (GML). [Ref. 3: p. 5.5]

There is no provision for subtypes in the ZBASIC language. For that reason some of the fields of the TRACK structure may inadvertently be assigned meaningless values. In some cases that will result in program termination and the display of an error message by the interpreter. In other cases it may result in undesirable side effects, or unexpected displays. Subtypes could have been enforced by programming subtype checking--that is providing checks on the bounds of variables that would be classified as subtypes and generating error messages when they were assigned improper values or re-assigning them automatically to values within their bounds. For a prototype implementation of this nature it was felt that the user could be responsible for the correct assignment of values to variables.

The T$ and L$ fields are strings which determine the symbol's appearance. The contents of the T$ field is the character string required to draw the symbol. It is copied from the table of symbols (SYM$), based on the classification (CLASS$) of the track. The speed leader is looked up in the LDR$ table, based on the track's course, and stored in the L$ field for the track.

The only remaining character string field of the TRACK record is HK$. This is a string indicating whether the track is hooked or not (a hooked track is one that has its parameters displayed on the right side of the display, at the user's request). Regardless of whether monochrome or color is used, a means is required to identify which track is currently hooked (if any). We elected to indicate this by enlarging the symbol. The HK$ string, indicating scale, is always added to the T$ string when the symbol is drawn.

The only other data structures of note are the three arrays for land masses which are dimensioned in the Land module. They are two dimensional arrays in which the (x,y) coordinate pairs for points defining the borders of land masses (or other special areas) are stored. The Land module then displays these areas by connecting the points with a line the color of the land area and painting the area of the screen bordered by that line the same color.

There is no true dynamic memory allocation in ZBASIC. To circumnaviagate this limitation, the PTS array stores the numbers of points defining each land mass (three are provided for in this prototype); then the array elements of PTS are used to dimension the land arrays when the Land module is called. Multiple calls to the module with different values in the PTS array generate errors. The initialization of the lower array subscript to one will cause errors if the elements of the PTS array are lower than one. For that reason the PTS array elements are initialized to ones. This problem also required that each land mass have an array with a separate name. That led, in turn, to the repetition of similar code in the Land module, once for each land mass.

C. DESIGN DECISIONS

Some of the decisions made have already been discussed. Others are described in an effort to present some project history and design philosophy that may enlighten others, or remove some of the mystique from "software design". The documentation of these decisions and their reasons should also enhance maintainability, and extend the life cycle of the project by creating an environment of modifiability.

The decision to use the special function keys for user input was born of human factors engineering. The concept is to make it simple for the user to make use of the system. In order to make it as simple as possible, a special function key menu is displayed at the bottom of the display (close to the special function keys) which reminds the user what each of them does. These keys are defined in the Init module, and restored in the Keys module upon exit from the system. We chose to make extensive use of data statements for initializing the display. Because we also designed this prototype with mergable files (the data is in one file) only the data file needs to be different for an entirely different initial display.

The Axes module incorporates three simple yet significant design choices. The first of these is the scaling of the divisions along the axes. This scaling was discussed in Section II.B.2.

The second decision was the way to draw the axes. As shown in Figures 3.2-3.3, presenting four distinct areas (background, land, symbols, reference grid) on a display with only two colors (monochrome system) can be difficult. Even on a color monitor problems arise when two or more of these graphic entities of the same color are drawn in the same area on the display. This prototype is written for full color implementation. The sample runs illustrated were run with the colors black and white, because the H/Z-100 monitors currently in use at NPS are monochrome monitors.

Figure 3.2 demonstrates the obvious problem. Where the land and the reference grid overlap, the reference grid does not appear. In this figure the land was drawn first. Simply reversing the order of drawing, as expected, does not solve the problem. It may introduce a different problem, as shown in Figure 3.3.

There are two simple solutions to the problem. They are shown in Figures 3.4-3.5. Figure 3.4 presents the most pleasing appearance. It was created by calculating where

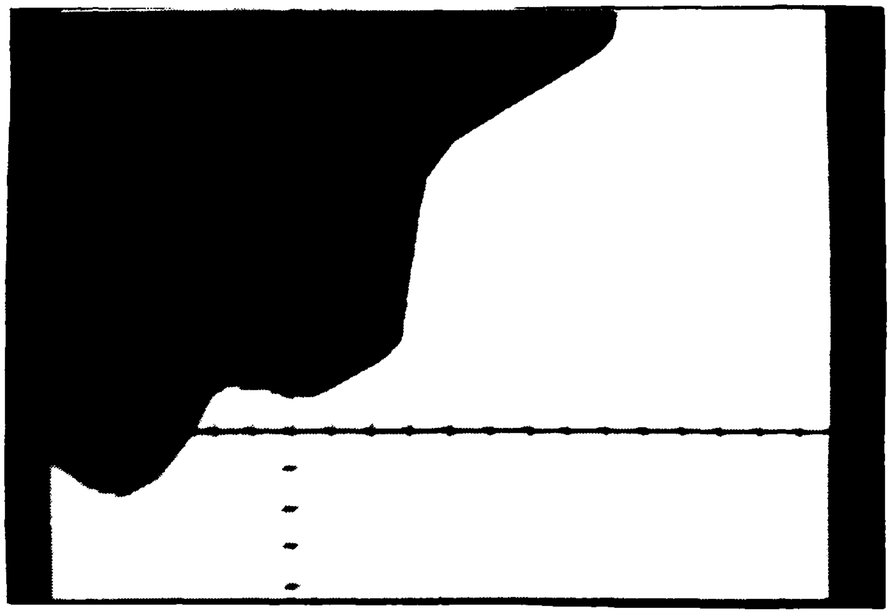


Figure 3.2 Reference Grid Eclipsed by Land

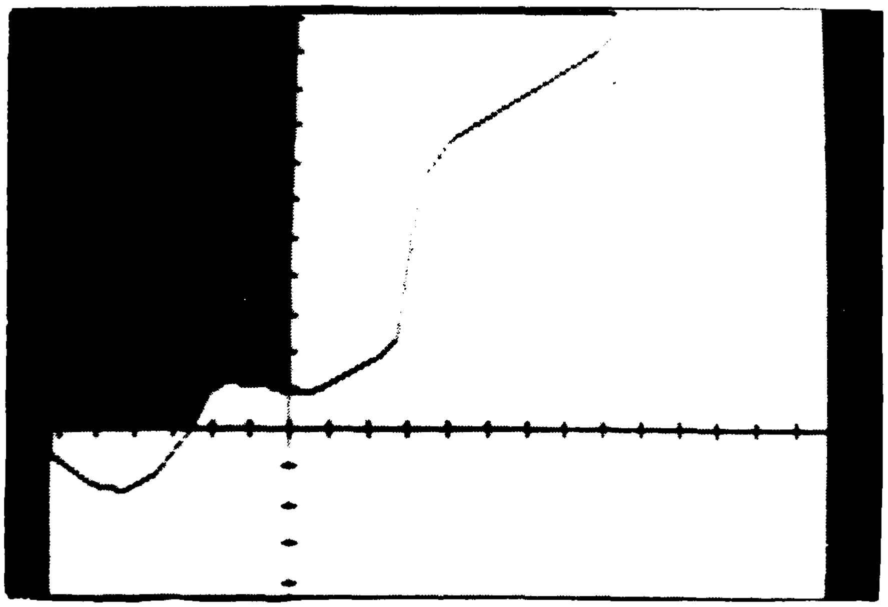


Figure 3.3 Land Mass only Partially Painted

the conflicts would arise and mapping the reference grid in sections, each section the color opposite that of the background. This may present the most elegant display, but requires modification of the actual Axes module every time different land is drawn.

The second method of solving the conflict is illustrated in Figure 3.5. After the land is drawn, a wide line the color of the background (i.e., opposite that of the land) is drawn where the grid will appear, then the grid is drawn on top of it in a slightly narrower line. The picture is not quite as elegant, and some of the land features are obscured. This solution does offer the advantage of writing one Axes module which will work whatever the land for any particular display is. We employed this solution in the prototype for just that reason.

The Update module may be considered the heart of the system. It presented several interesting design alternatives, many because of the language limitations of ZBASIC. The reader may want to refer to Appendix G, which contains the listing of the code of this module, during the reading of the following discussion.

The first choice, which is not evident in examining the code, was whether to place the update loop within this module or in the calling routine. The simulator should periodically perform an automatic update of all tracks, independent of user input. This requirement seemed to infer

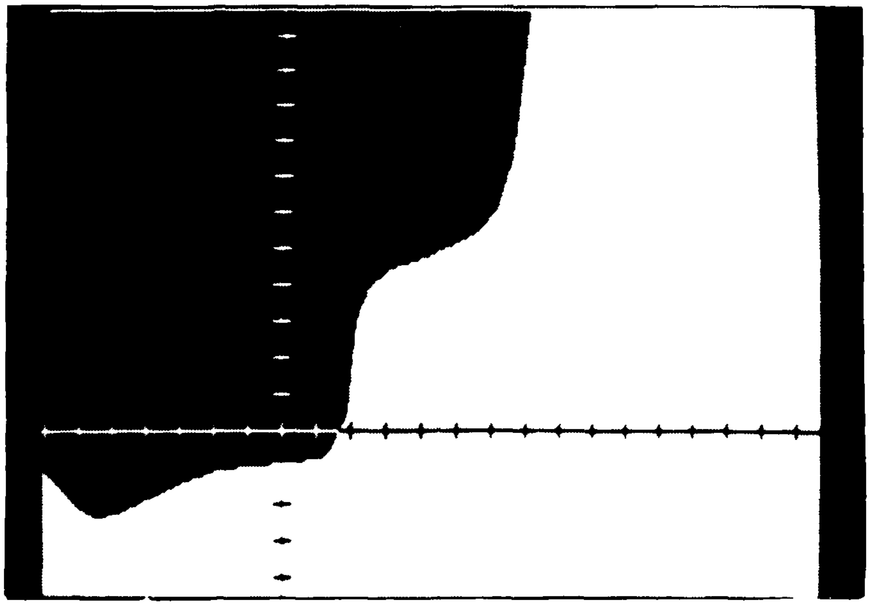


Figure 3.4 Solution One

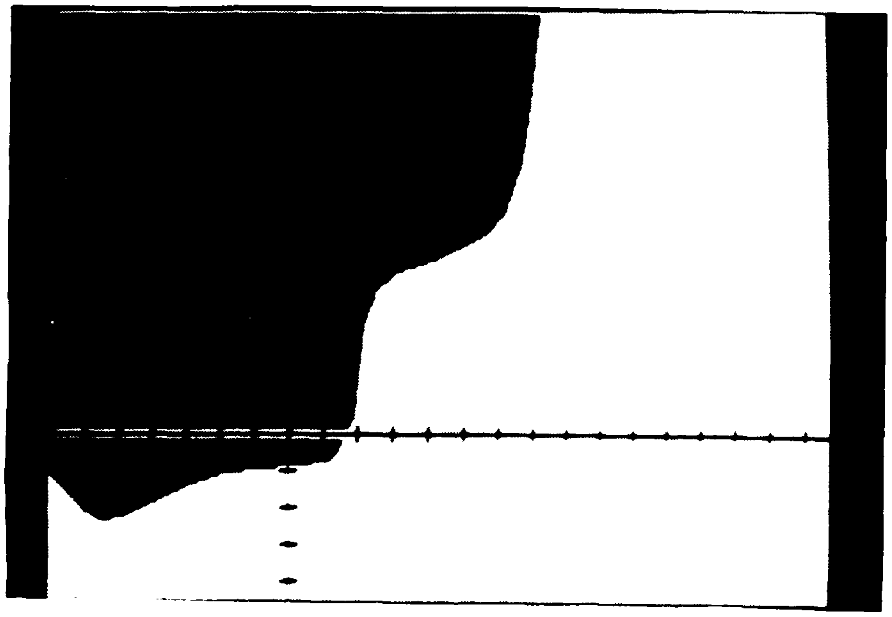


Figure 3.5 Solution Two

that the loop should be internal. Initially we provided the loop in this Update module. That worked fine, until user interaction was added.

At any time the user may elect to delete, insert, hook or modify a track. Ideally he/she would not have to wait for the next update of all tracks to see the effect of the service request, but should see it implemented immediately. For this reason the loop was externalized, allowing this module to be called for the single track being deleted, inserted, hooked or modified.

We have stated that many decisions were made in the interest of clarity rather than efficiency. One of the exceptions to this rule is here, in the early lines of the Update module. Several times within this module reference is made to elements of the TRACK record structure (Figure 3.1). Calculations of the actual address of an element in an array are more time-consuming than references to simple variables. Almost all of the array elements that are referenced are copied into local simple variables to save time.

Because the ACTIVE field is referred to at most twice within Update, it was not copied but is referred to directly. The ACTIVE field serves two purposes with three allowable values. If the value of this field is zero, the track is inactive (the user no longer desires it to be displayed), and it is only erased. A value of one indicates an active track, and it is erased, updated, and re-drawn. A value of two indicates a special type of "inactive" -- a symbol which does not move (i.e., reference point, or target with speed equal to zero). It is not erased or updated, merely re-drawn (in case it has been partially over-written by another track).

The next decision is one involving a sampling technique. One common method for erasing a figure in computer graphics is to re-draw it in the same color as its background. That is the method we employ here.

This erasure/re-drawing could be performed automatically, in ZBASIC, through the use of the PUT and GET statements. The code would have been easier to write, perhaps even quicker to execute. The problem with this easy solution is illustrated in Figures 3.6-3.7.

The use of the PUT and GET statements is a three-step process. The figure that is going to be moved is first drawn somewhere on the screen. This has been done in Figure 3.6, providing the right-most symbol. This step precedes the actual use of either the PUT or the GET statement. Then the GET statement is utilized, in the form GET (x1, y1) - (x2, y2), <figure name> (A necessary preliminary step, before even drawing the figure, is the use of a DIMension statement to allocate memory for the drawing). The subscripted x and y values are coordinates, the first pair for the upper left-hand corner of a box on the screen which

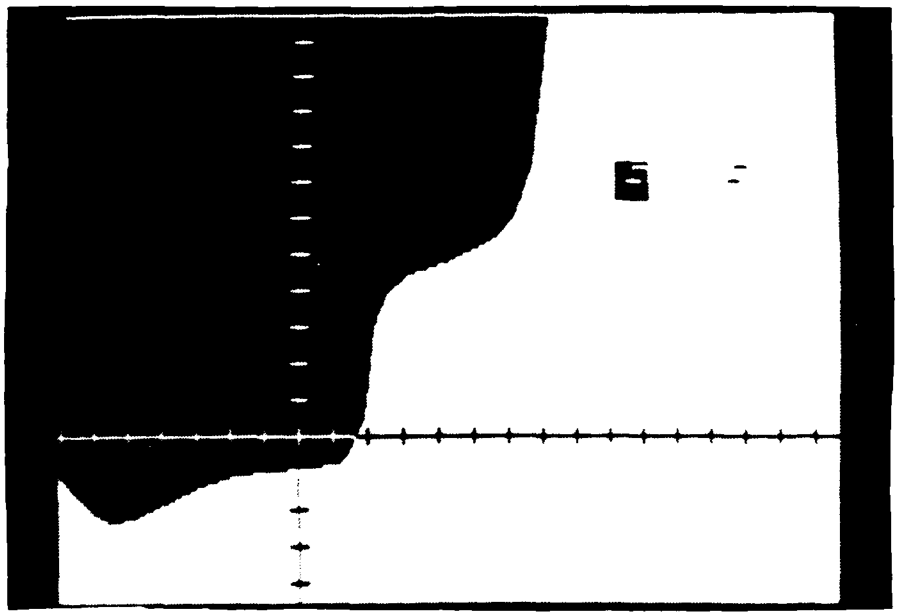


Figure 3.6 Drawing a Symbol for use with PUT and GET

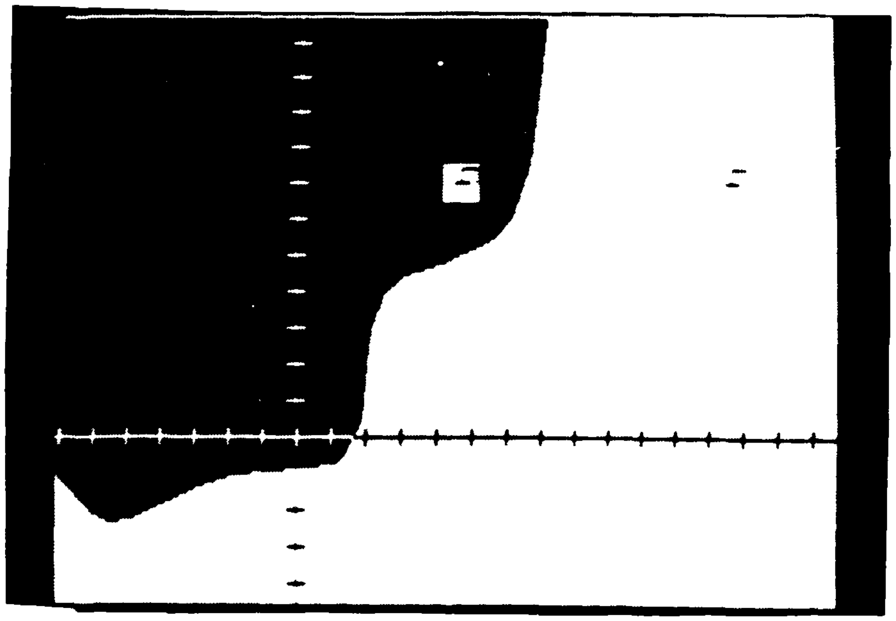


Figure 3.7 Moving a Symbol using PUT

encloses the figure, the second pair for the lower right-hand corner. The amount of space to set aside in memory through the DIM statement is dependent upon the size of the pixel box enclosing the figure. Figure name is a variable name that is assigned to the portion of memory where this pixel box is stored. The third and final step, the use of the PUT statement, is of the form PUT (x, y), <figure name>. The coordinates x and y are of the upper left-hand corner of an area on the screen where the upper left-hand corner of the original figure's containing box is to be placed. The result will be the placement of the original figure where the PUT statement has directed. Placing of a symbol with the PUT statement is demonstrated by the white symbol in the black box in Figure 3.6.

There are "action verbs", optional commands which follow the <figure name> in the PUT statement, which determine the relationship in the new location between the figure's box and the background existing at the specified screen location. Proper use of these action verbs relieves the programmer of the need to be concerned with what the background looks like by automatically ensuring that the desired effect is produced when the figure is drawn, and the background is restored when the figure is erased. The problem with all of this is the requirement that an entire box of pixels, enclosing the symbol, must be moved. The results of this are illustrated in Figures 3.6 and 3.7. [Ref 3: pp. 6.5-6.24]

A solution which provides much cleaner displays, and obscures less background wherever a symbol is placed, is shown in Figures 3.8-3.9. The symbols are drawn using the Graphics Macro Language (GML) of ZBASIC [Ref. 3: p. 5.5] in Figure 3.8. The results of moving them with the Update module are shown in Figure 3.9.

We need, therefore, some method of determining the background color (the symbol may even rest on a background containing two colors in a monochrome display, or more than two in a color display). We elected a point sample, for speed and simplicity. There are problems attendant with this method when the symbol rests on a multi-colored background. Any sampling technique, other than examining every pixel the symbol occupies, suffers from the same problem. Rather than employ this time-consuming process (sampling every point) to solve what we expect to be an infrequent problem, we elected to use a single point sample. We chose to look at a point two pixels to the right and one pixel below the symbol center.

When this sampling is applied to the new symbol position we face another decision. If the new background is the same color as the symbol, what color should the new symbol be drawn in? For a monochrome display the answer is already determined. We elected to provide the same choice

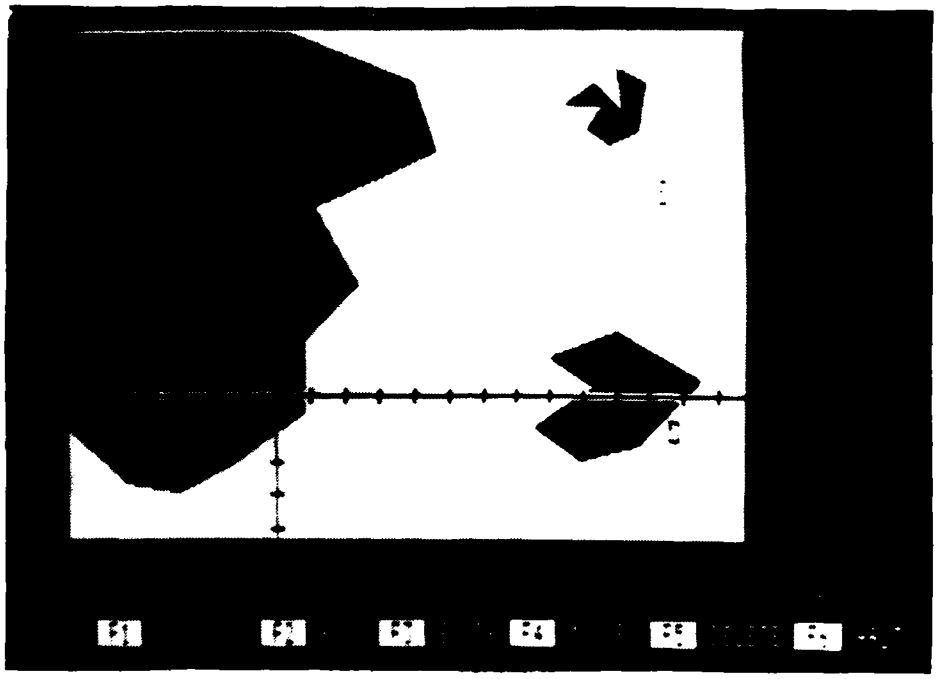


Figure 3.8 Symbols Drawn using GML

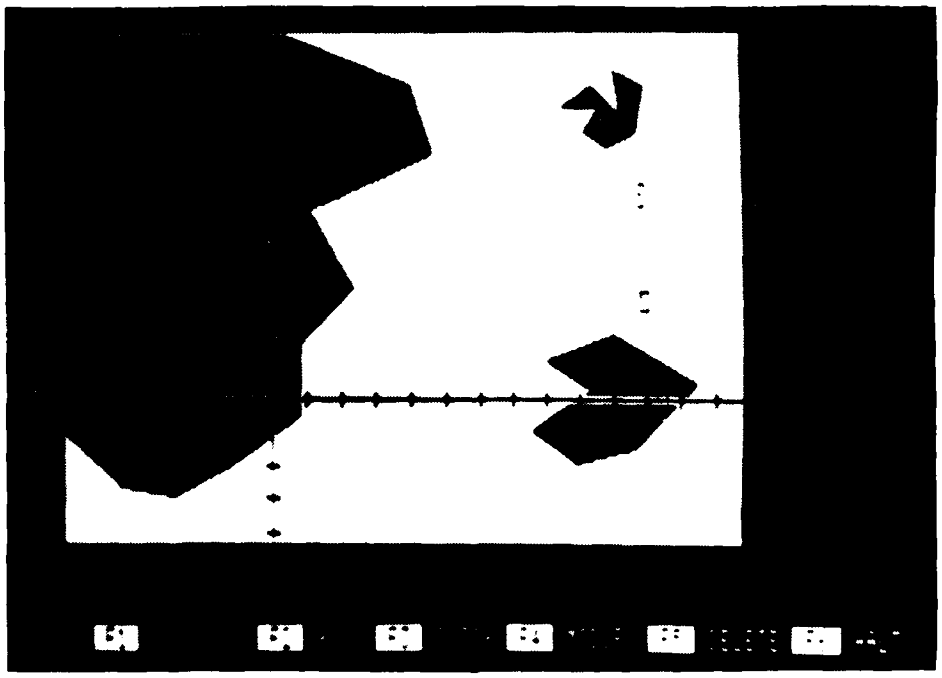


Figure 3.9 Symbols Moved using Update

in the color display. For the two darkest backgrounds, black and blue, the symbol is drawn in white. All others, if they match the actual symbol color, result in a black symbol.

The final decision involved track history. We chose not to store the information anywhere, partly due to the lack of dynamic memory allocation. We did experiment with a track history display, however. It was simple to implement, interesting to see, and is left as an exercise for the reader.

The decisions in the Move module have been addressed earlier, after a fashion. We chose to use only eight speed leader directions. For determination of the incremental movement of the symbol across the display we divided a circle into 20 sections (see Figure 3.10).

The actual listing of the Move module code, Appendix H, looks more complex than it is. The first several lines of code make the division of the circle (courses ranging from 000 degrees to 360 degrees), directing execution of the appropriate statements to assign the increments for the horizontal and vertical movement, as well as the speed leader direction. The GOTO statements simply complete the construction of a giant case statement, transferring program control to the speed section.

Here we encounter another decision: how many different states of speed to recognize. We chose three, representing slow targets (such as surface vessels), medium speed targets (aircraft) and high-speed targets (jet aircraft and missiles). Based on the speed category, the speed leader and the movement increments are scaled. The special case of a track with zero speed is also treated, by assigning no speed leader and no incremental movement.

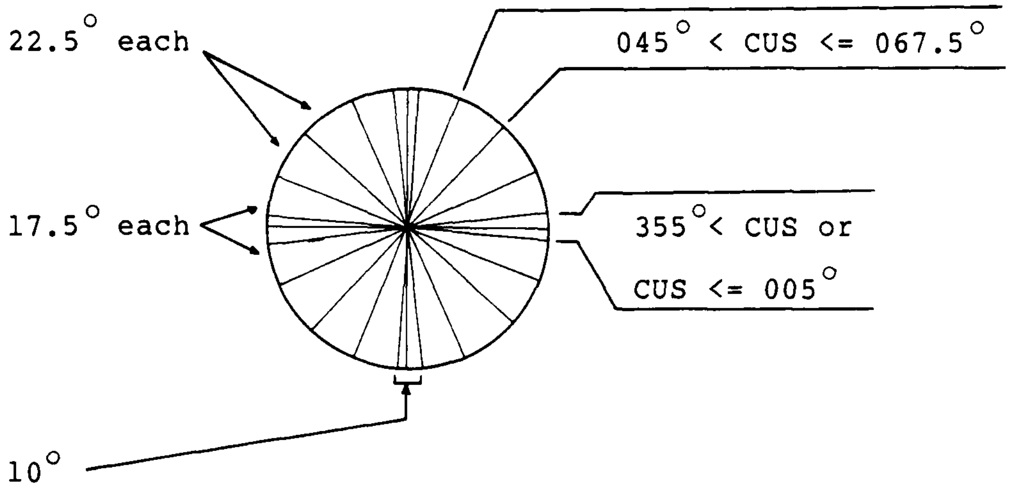


Figure 3.10 Target Course Increments

In the Tracking module we chose to provide for the possibility of existing tracks in the initial display. This feature allows for the establishment of various training modules, each with different initial track selections. If there are none, the code that treats them is skipped.

The next section of this module affects the delay between updates. It also provides the opportunity to the user to make a service request. While all tracks are being updated, the user input is ignored. Then, for the length of time between delays, or until the user makes a service request, the keyboard is repeatedly checked for pressed keys. In the Macro-86 implementation, this could be handled through an interrupt service request. If the user makes a request, it is checked for validity. We decided to ignore invalid input rather than generate error messages. This was felt to be more "user friendly" and robust. Valid input is the use of any of the special function keys that have defined functions. Those which are defined are displayed at the bottom of the screen in the special function key menu, as seen in Figures 3.8, 3.9, 3.11 and 3.12.

The decisions made in the Keys module were driven more by requirement than choice, in many cases. The "Halt" request clears the screen of the display and restores the special function keys, as a matter of good programming practice. The "Suspend/Continue" request waits for another input from the keyboard. We chose to accept any key to continue, again in the interests of robustness and "user friendliness".

There are some interesting requirements generated by a request to "Hook" a track. We have elected to have only one track hooked at a time. The first thing this module does, then, is to check to see if there is a track already hooked. If there is, it must unhook it. This involves more than merely changing the HK$ field in its record.

Hooked tracks are indicated, in this prototype, by enlarged symbols. The enlarged symbol of the previously hooked track must be erased, or the next erasure will only erase a small part of it. After managing any previously hooked track, the user is asked to specify the number of the track to hook. It is then hooked, its symbol enlarged, and the parameters of the track displayed to the right of the screen. An example of a hooked track display is pictured in Figure 3.11. In later implementations, it would be desirable to indicate a track to hook by either its track number or by placing the cursor near it. This prototype does not provide for cursor-dependent user input.

The same problem, cursor-independent user input, surfaces in the "Enter" request. The user is asked to enter track parameters, including "Grid X" and "Grid Y". Details of these parameters are included in the User's Manual, Appendix N. Figure 3.12 shows a request for the user to enter the class of a track at the top of the screen, in response to a depression of the "Enter" <F3> key by the user. The problem re-appears in the "Modify" function, where the "Grid X" and "Grid Y" of the track are verified or modified by the user.

The Match module matches the symbol type to the CLASS$ field of the TRACK record. Arbitrary choices were made here, and have no bearing on actual NTDS symbology. The symbols chosen and the corresponding classifications were

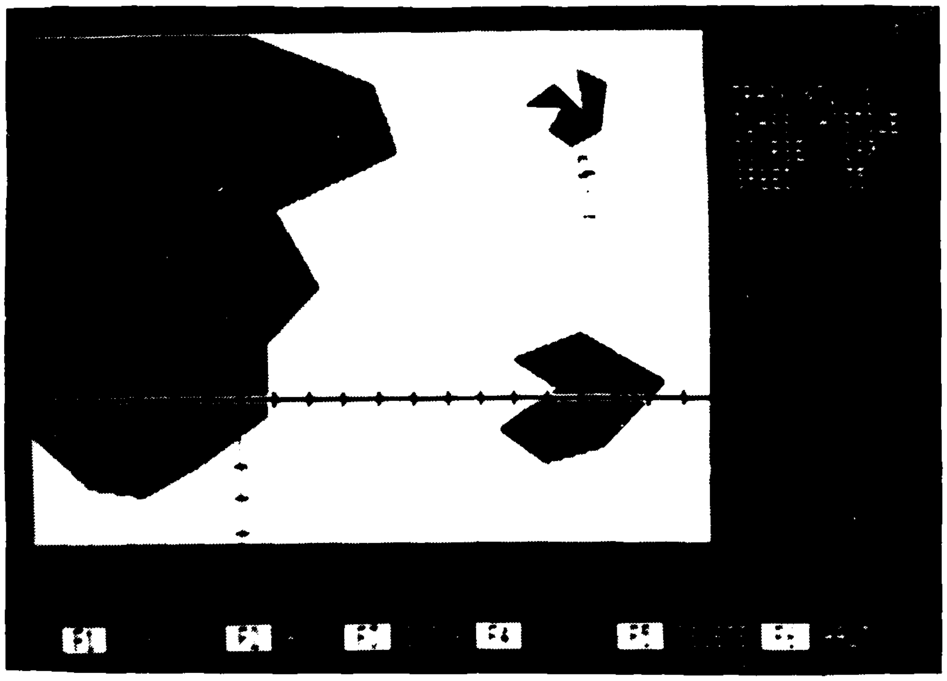


Figure 3.11 A Hooked Track

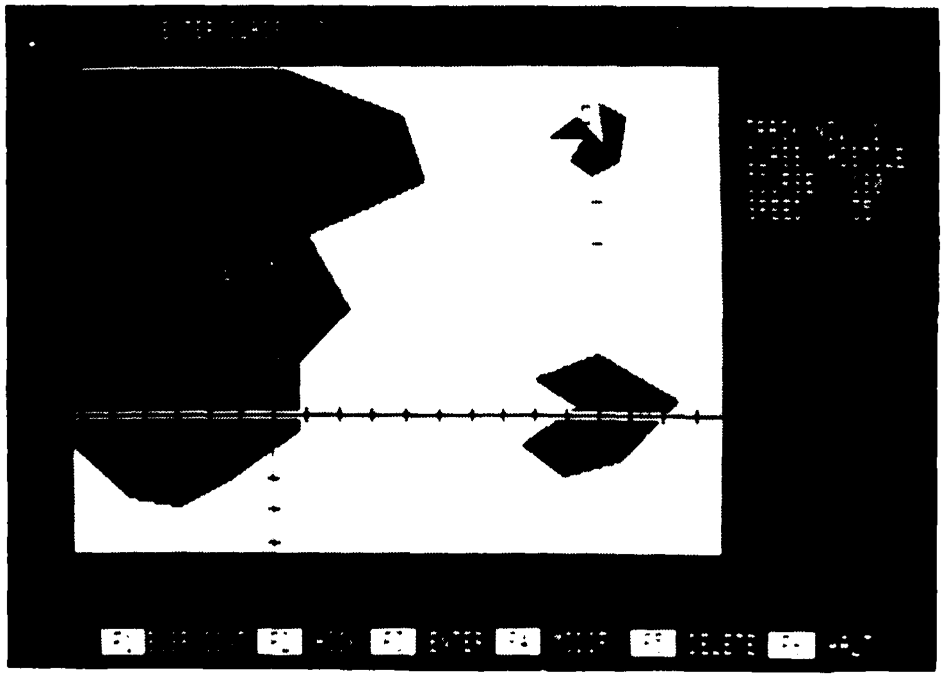


Figure 3.12 User Input Requested

made simply to demonstrate the proper functioning of the prototype.

IV. PERFORMANCE TESTS

A. TIMING

We present here the results of selected timing tests that were performed for purposes of comparison. The first tests performed were timing two primitive functions by the prototype modules in each of the two languages proposed and used. The results are presented in Table 4.1 and discussed below.

TABLE 4.1 TIMING COMPARISONS, ZBASIC AND MACRO-86

	TEST #1 Window Generation	TEST #2 Symbol Generation
ZBASIC	.6854	.0846
Macro-86	.1300	.0015

All times in Table 4.1 are given in seconds. Test #1 was the generation of the window and the reference grid. For the ZBASIC timing the Window and Axes modules were used to generate 100 displays. The times were noted and the elapsed time computed and divided by 100 to obtain the data in Table 4.1. To time the Macro-86 routine, which does generate a reference grid but does not ensure freedom from color conflict and does not provide scale divisions, one display was drawn. Timing interrupts were generated to allow the system clock to time the performance. Although the test conditions were not identical, the test results are indicative of the order of magnitude of the different performance of the two languages.

Test #2 timed the generation of one symbol. For the Macro-86 test the screen was filled with 2000 symbols, the time to do so recorded and divided by 2000 to obtain the result recorded above. For ZBASIC 1000 symbols were generated using the Update module. Because Update erases each symbol and re-draws it, this was the time required, in effect, to draw 2000 symbols. The relative efficiencies of the two languages are again exposed, rather than the precise difference in time required to perform the same task.

Further timing tests were conducted using the entire display simulator prototype. The results are tabulated in Table 4.2 and disussed in the following paragraph.

The first three timing runs involve numbers of symbols that are multiples of each other. The times do not follow that exact correspondence. This is due to the overhead involved in a call to a subroutine and a return, performed in an iterative loop. Even at 99 symbols, where the overhead per symbol becomes negligible, the update time per symbol may be seen to exceed .15 seconds, almost twice the time per symbol when just the Update module was tested. The suggested limitation of the system this data provides is not as serious as it first seems. When only twenty symbols were displayed, it required the operator more than 4.02 seconds to digest the graphic information displayed. Of course, in high density tactical pictures, only the targets of immediate priority are concentrated on.

TABLE 4.2 PROTOTYPE SYMBOL GENERATION TIMING

# of Symbols	Time required to update all tracks
10	2.07 seconds
20	4.02 seconds
40	7.89 seconds
99	15.27 seconds

These initial timing results confirm our earlier suggestion that the prototype should be (and has been) developed in ZBASIC in order to be available for use and experimentation sooner, but the final system implementation should be developed in Macro-86 assembly language. Toward that end the Macro-86 listings in Appendices O-U are provided, and the User's Manual for them in Appendix V.

B. EFFICIENCY

There are other efficiency issues to be addressed. As has been noted more than once, the prototype we have implemented is not the most efficient possible. It may be that when the code has been tightened up as much as can be the ZBASIC implementation may suffice. It is our opinion that it is, and may continue to be, quite useful, but that a final implementation in Macro-86 would be well worth the effort.

Chapter III addressed some of the places in which the ZBASIC code sacrificed efficiency for clarity. There are other indications of possible coding improvements suggested in the User's Manual for the prototype, Appendix N. More suggestions for follow-on efforts are addressed in the next chapter.

V. CONCLUSION

A. USES OF THIS PROTOTYPE

This NTDS display simulator prototype has been developed as proof of the feasibility of such a simulator on the H/Z-100 microcomputer. It demonstrates the graphic ability of the H/Z-100 to support such a simulator, gives any users actual displays to experiment with and learn from, and shows that such a simulator might present real-time graphic updates. It may also be used to demonstrate the graphics capability of the H/Z-100.

The code listings provided, coupled with the design discussions in this text, document some of the thought processes and decision criteria involved in developing such a system. It may be used without modification or improvement as a simplistic display simulator for some of the purposes put forth in Chapter I. It may become the kernel of a more fully developed NTDS system, using Ada as the higher level language and Macro-86 when required.

The Macro-86 modules provided may be incorporated as they are in other systems to perform very primitive graphic functions. They model the type of development that may be considered for other assembly language functions users may want to incorporate in this or other systems. They are also models of at least one form of internal documentation.

B. FOLLOW-ON WORK

The display simulator prototype in ZBASIC performs some of the functions such a system is required to perform. Functions such as track history (earlier alluded to), automatic track sequencing, trouble tracks (tracks which have not been updated recently enough by the user), etc., could be added. Interfaces between the assembly language modules included and high-level languages (such as Ada) could be developed.

C. LESSONS LEARNED

Many valuable lessons were learned during the development of this prototype. It is not easy to assign priority to them. The order in which they are presented is not meant to imply such a prioritization. Each lesson here was valuable, and should be given careful consideration in any future development of this project.

Internal documentation is very important. Even as a simple, one-programmer project, the internal documentation of the code made corrective maintenance much easier, and should enhance the maintainability of the code for other programmers working with it in the future.

During prototype development such as this, clear logic is more important than elegant code. Keeping the logic clear caused a proliferation of variable names, and the repetition of some sections of code, but it greatly enhanced testing and debugging.

ZBASIC is a more versatile language than it appears to be. This may verge on heresy in computer science circles, and it came as a surprise to us. The ease with which some other high-level language constructs could be constructed in ZBASIC was an eye-opener.

At the same time, this project exposed some of the problems with ZBASIC for systems work. The lack of data structure definition was a difficult problem to overcome. The inability to specify sub-types was also an unpleasant reality. The real surprise was revealed in Table 4.1. ZBASIC had not seemed visibly much slower than Macro-86, but the timing tests revealed the magnitude of the difference.

D. CONCLUSIONS

The display simulator prototype has been developed. The feasibility question has been answered. The H/Z-100 is definitely capable of providing the user with an NTDS-type display, and with some of the NTDS user functions.

The display updates are noticeable, regardless of the number of symbols in the system. This feature will remain in any implementation language, because the re-location of the symbols is in discrete steps.

The speed of the updates is a function of the language employed. The data in Table 4.1 provides evidence of the savings in time achievable through the use of Macro-86 routines. The window area alone may be generated in less than one-fifth the time in assembly language. Drawing more complex shapes on the screen take even longer in ZBASIC. This is evidenced by the 56-1 time differential in drawing a simple symbol.

The data in Table 4.2 is even more revealing of the inefficiencies of ZBASIC for real-time applications. It is readily apparent that the symbol generation within the prototype, rather than in a separate test module, takes almost twice as long. The additional time delay is due to the call-return sequence, utilizing the Update module for each symbol re-location.

This does not mean that the prototype itself is useless. Fire control solutions require accurate solutions at precise instants in time. Tactical displays may lag the real-time situation by as much as a few seconds and still be useful to the human operators. A display simulator, which utilizes keyboard input rather than radar and other equipment inputs, is useful at even slower speeds.

We conclude that this prototype has value, as discussed above. Future implementation of a tactical display simulator on the H/Z-100, in assembly language, and/or another high-level language, is desirable and encouraged.

APPENDIX A: LISTING OF NEWEST.BAS

```
10 '          SAMPLE NTDS DISPLAY SIMULATOR
20 '
30 '          FRAME #7
40 '
50 '          PROTOTYPE DISPLAY
60 '
70 CLS        'CLEAR THE DISPLAY
80 '
90 '
100 ' * * * * * * * INITIALIZATION AND TABLES * * * * * * *
110 '
120 '
130 OPTION BASE 1          'ARRAY SUBSCRIPT LOWER BOUND = 1
140 '
150 DIM CLASS$(10), CUS(10), SPD(10), TCOLOR(10)
160 DIM TX(10), TY(10), XINC(10), YINC(10), T$(10), L$(10)
170 DIM SYM$(10), LDR$(8), PTS(3), LCOL(3), HK$(10), ACTIVE(10)
180 '
190 '          SYMBOL TABLE
200 '
210 SYM$(1) = "BM+0,-3 R3 D3 BM-6,0 D3 R3 BM+0,+3"
220 SYM$(2) = "BM+0,-3 L3 D3 BM+0,+3"
230 SYM$(3) = "BM+0,-3 R3 D6 L6 U6 R3 BM+0,+3"
240 SYM$(4) = "BM+0,-3 R2 F2 D3 G2 L4 H2 U3 E2 R2 BM+0,+3"
250 SYM$(5) = "BM+0,-3 R3 D3 BM-6,0 U3 R3 BM+0,+3"
260 SYM$(6) = "BM+0,-3 F3 G3 H3 E3 BM+0,+3"
270 SYM$(7) = "U3 R3 D6 L6 U6 R3 D6 U3"
280 SYM$(8) = ""
290 SYM$(9) = ""
300 SYM$(10) = ""
310 '
320 '
330 '          SPEED LEADER TABLE
340 '
350 LDR$(1) = "U4"
360 LDR$(2) = "E3"
370 LDR$(3) = "R5"
380 LDR$(4) = "F3"
390 LDR$(5) = "D4"
400 LDR$(6) = "G3"
410 LDR$(7) = "L5"
420 LDR$(8) = "H3"
430 '
440 '
450 '          START WITH NO TRACKS
460 '
```

```
470 TRACKS = 0
480 '
490 '         INITIALIZE PTS ARRAY ELEMENTS TO 1
500 '
510 FOR I = 1 TO 3: PTS(I) = 1: NEXT I
520 '
530 '         DEFINE FUNCTION KEYS
540 '
560 KEY 1, CHR$(27) + "S"
570 KEY 2, CHR$(27) + "T"
580 KEY 3, CHR$(27) + "U"
590 KEY 4, CHR$(27) + "V"
600 KEY 5, CHR$(27) + "W"
605 KEY 6, CHR$(27) + "P"
610 '
620 '         INITIALIZE HK$ AND ACTIVE
630 '
640 FOR I = 1 TO 10
650 HK$(I) = "SO"
660 ACTIVE(I) = 0
670 NEXT I
680 '
690 '         DISPLAY FUNCTION KEY FUNCTIONS
700 '
710 COLOR 0,7
720 LOCATE 25, 5
730 PRINT " F1 "
740 LOCATE 25, 19
750 PRINT " F2 "
760 LOCATE 25, 29
770 PRINT " F3 "
780 LOCATE 25, 40
790 PRINT " F4 "
800 LOCATE 25, 52
810 PRINT " F5 "
820 LOCATE 25, 64
830 PRINT " F6 "
840 COLOR 7,0
850 LOCATE 25, 9
860 PRINT "SUSP/CONT"
870 LOCATE 25, 24
880 PRINT "HOOK"
890 LOCATE 25, 34
900 PRINT "ENTER"
910 LOCATE 25, 45
920 PRINT "MODIFY"
930 LOCATE 25, 57
940 PRINT "DELETE"
950 LOCATE 25, 69
960 PRINT "HALT"
1000 '
```

```
1010 '        GET WINDOW PARAMETERS
1020 '
1030 READ XUL, YUL, XLR, YLR, CWIND
1040 '
1050 '        DRAW THE WINDOW
1060 '
1070 GOSUB 5000
1080 '
1090 '        GET LAND PARAMETERS
1100 '
1110 READ CONTS                        'HOW MANY LAND MASSES?
1120 '
1130 '        DRAW LAND MASSES
1140 '
1150 GOSUB 8000
1160 '
1170 '        GET GRID PARAMETERS
1180 '
1190 READ XYAX, YTOP, YBOTT, YCOL
1200 READ YXAX, XLEFT, XRITE, XCOL
1210 '
1220 '        DRAW THE GRID
1230 '
1240 GOSUB 5200
1250 '
1260 '        RUN UPDATE TESTS
1270 '
1280 GOSUB 11000
1290 '
1300 '
4999 END
5000 ' * * * * * * *   DRAW  WINDOW  SUBROUTINE   * * * * * * *
5010 ' *                                                       *
5020 ' *  INPUTS:  XUL, YUL - UPPER LEFT-HAND COORDINATES      *
5030 ' *           XLR, YLR - LOWER RIGHT-HAND COORDINATES     *
5040 ' *           CWIND    - COLOR OF WINDOW                  *
5050 ' *                                                       *
5060 ' *  OUTPUT:  SOLID WINDOW, XLR - XUL PIXELS WIDE,        *
5070 ' *           YLR - YUL PIXELS DEEP, COLOR CWIND          *
5080 ' *                                                       *
5090 ' * * * * * * * * * * * * *  * * * * * * * * * * * * * * *
5100 '
5110 '
5120 LINE (XUL, YUL) - (XLR, YLR), CWIND, BF
5130 '
5140 RETURN
5150 '
5160 '
5200 ' * * * * * * *   COORDINATE  AXES  SUBROUTINE   * * * * * * * * *
5210 ' *                                                               *
5220 ' *  INPUTS:  XYAX, YTOP, YBOTT  - VERTICAL AXIS COORDINATES      *
```

```
5230 ' *          YXAX, XLEFT, XRITE - HORIZONTAL AXIS COORDINATES   *
5240 ' *          XCOL, YCOL         - GRID COLORS                    *
5250 ' *                                                              *
5260 ' *  OUTPUT: PROPERLY SCALED SET OF COORDINATE AXES,             *
5270 ' *          OF XCOL AND YCOL                                    *
5280 ' *                                                              *
5290 ' * * * * * * * * * * * * * *  * * * * * * * * * * * * * * * * * *
5300 '
5310 '
5320 HSCALE = (XRITE - XLEFT)/20          'HORIZONTAL SCALE MULTIPLIER
5330 VSCALE = HSCALE * .46                'VERTICAL SCALE MULTIPLIER,
5340                                      'FOR PROPER ASPECT RATIO
5350 '        DRAW VERTICAL AXIS
5360 '
5365 LINE (XYAX-1, YTOP) - (XYAX+1, YBOTT), CWIND, BF
5370 LINE (XYAX, YTOP) - (XYAX, YBOTT), YCOL
5380 '
5390 '        DRAW HORIZONTAL AXIS
5400 '
5405 LINE (XLEFT, YXAX-1) - (XRITE, YXAX+1), CWIND, BF
5410 LINE (XLEFT, YXAX) - (XRITE, YXAX), XCOL
5420 '
5430 '        DRAW HORIZONTAL SCALE DIVISIONS, LEFT
5440 '
5450 FOR H = XYAX TO XLEFT STEP -HSCALE
5460   LINE (H, YXAX-2) - (H, YXAX+2), XCOL
5470   LINE (H+1, YXAX-2) - (H+1, YXAX+2), XCOL
5480 NEXT H
5490 '
5500 '        DRAW HORIZONTAL SCALE DIVISIONS, RIGHT
5510 '
5520 FOR H = XYAX TO XRITE STEP HSCALE
5530   LINE (H, YXAX-2) - (H, YXAX+2), XCOL
5540   LINE (H+1, YXAX-2) (H+1, YXAX+2), XCOL
5550 NEXT H
5560 '
5570 '        DRAW VERTICAL SCALE DIVISIONS, UPPER
5580 '
5590 FOR V = YXAX TO YTOP STEP -VSCALE
5600   LINE (XYAX-4, V) - (XYAX+4, V), YCOL
5610 NEXT V
5620 '
5630 '        DRAW VERTICAL SCALE DIVISIONS, LOWER
5640 '
5650 FOR V = YXAX TO YBOTT STEP VSCALE
5660   LINE (XYAX-4, V) - (XYAX+4, V), YCOL
5670 NEXT V
5680 '
5690 RETURN
5700 '
5710 '
```

```
5720 '        THIS AXES SUBROUTINE IS BASED ON THE PROGRAM
5730 '        9-2, PAGE 9-15, IN THE CONTINUING EDUCATION
5740 '        CORRESPONDENCE COURSE "COMPUTER GRAPHICS",
5750 '        WRITTEN FOR HEATHKIT/ZENITH BY
5760 '                JAMES  C.  ADAMS
6000 ' * * * * * *   UPDATE  TRACKS  SUBROUTINE   * * * * * * *
6010 ' *                                                      *
6020 ' *  INPUTS:  UPD - # OF TRACK TO UPDATE                 *
6030 ' *                                                      *
6040 ' *  OUTPUT:  TRACK UPD IS UPDATED                       *
6050 ' *                                                      *
6060 ' * * * * * * * * * * * * * *  * * * * * * * * * * * * * *
6070 '
6080 '
6100 '
6120 '
6130 '        PERFORM ALL LOOK-UPS ONLY ONCE
6140 '
6150 UPDX = TX(UPD)
6160 UPDY = TY(UPD)
6170 UPDT$ = HK$(UPD) + T$(UPD)
6180 UPDL$ = L$(UPD)
6190 HORZUP = XINC(UPD)
6200 VERTUP = YINC(UPD)
6210 COLUP = TCOLOR(UPD)
6220 '
6230 UPGND = POINT(UPDX+2, UPDY+1)
6240 ON UPGND+1 GOSUB 6520, 6530, 6540, 6550, 6560, 6570, 6580, 6590
6250 WANT$ = COL$ + UPDT$: ALSO$ = COL$ + UPDL$
6255 '
6260 PSET (UPDX, UPDY), UPGND                'DRAW OLD SYMBOL IN
6270 DRAW WANT$                              'REVERSE COLOR
6280 PSET (UPDX, UPDY), UPGND
6290 DRAW ALSO$
6295 DRAW "SO"
6300 '
6305 IF ACTIVE(UPD) = 0 THEN 6490
6310 UPDX = UPDX + HORZUP                    'UPDATE POSITION
6320 UPDY = UPDY + VERTUP
6330 '
6340 UPGND = POINT(UPDX+2, UPDY+1)           'CHECK BACKGROUND COLOR
6350 '                                        OF NEW LOCATION
6360 IF UPGND <> COLUP THEN 6375             'MAKE SYMBOL OPPOSITE
6370 IF UPGND < 2 THEN COLUP = 7 ELSE COLUP = 0    'OF BACKGROUND
6374 '
6375 ON COLUP+1 GOSUB 6520, 6530, 6540, 6550, 6560, 6570, 6580, 6590
6376 WANT$ = COL$ + UPDT$: ALSO$ = COL$ + UPDL$
6380 '
6390 PSET (UPDX, UPDY), COLUP                'DRAW NEW SYMBOL
6400 DRAW WANT$
6410 PSET (UPDX, UPDY), COLUP
```

```
6420 DRAW ALSO$
6425 DRAW "SO"
6430 '
6440 TX(UPD) = UPDX                              'STORE NEW POSITION
6450 TY(UPD) = UPDY
6460 '
6480 '
6490 RETURN
6500 '
6510 '
6520 COL$ = "C0": RETURN
6530 COL$ = "C1": RETURN
6540 COL$ = "C2": RETURN
6550 COL$ = "C3": RETURN
6560 COL$ = "C4": RETURN
6570 COL$ = "C5": RETURN
6580 COL$ = "C6": RETURN
6590 COL$ = "C7": RETURN
7000 ' * * * * * * *   SYMBOL  MOVEMENT  CALCULATOR   * * * * *
7010 ' *                                                       *
7020 ' *  INPUTS:  MOVE   - TRACK TO CALCULATE FOR             *
7030 ' *                                                       *
7040 ' *  OUTPUT:  XINC, YINC, SCALE FACTOR FOR SPEED          *
7050 ' *           LEADER OF EACH ACTIVE TRACK ARE             *
7060 ' *           CALCULATED AND STORED                       *
7070 ' *                                                       *
7080 ' * * * * * * * * * * *  * * * * * * * * * * * * * * * * *
7090 '
7100 '
7110 '
7130 '
7140 '       CALCULATE INCREMENTS BASED ON COURSE
7150 '
7160 IF CUS(MOVE) <=     5 THEN 7400
7170 IF CUS(MOVE) <=  22.5 THEN 7410
7180 IF CUS(MOVE) <=    45 THEN 7420
7190 IF CUS(MOVE) <=  67.5 THEN 7430
7200 IF CUS(MOVE) <=    85 THEN 7440
7210 IF CUS(MOVE) <=    95 THEN 7450
7220 IF CUS(MOVE) <= 112.5 THEN 7460
7230 IF CUS(MOVE) <=   135 THEN 7470
7240 IF CUS(MOVE) <= 157.5 THEN 7480
7250 IF CUS(MOVE) <=   175 THEN 7490
7260 IF CUS(MOVE) <=   185 THEN 7500
7270 IF CUS(MOVE) <= 202.5 THEN 7510
7280 IF CUS(MOVE) <=   225 THEN 7520
7290 IF CUS(MOVE) <= 247.5 THEN 7530
7300 IF CUS(MOVE) <=   265 THEN 7540
7310 IF CUS(MOVE) <=   275 THEN 7550
7320 IF CUS(MOVE) <= 292.5 THEN 7560
7330 IF CUS(MOVE) <=   315 THEN 7570
```

```
7340 IF CUS(MOVE) <= 337.5 THEN 7580
7350 IF CUS(MOVE) <=   355 THEN 7590
7360 '
7370 '
7400 XINC(MOVE) = 8: YINC(MOVE) = 0: L$(MOVE) = LDR$(3): GOTO 7600
7410 XINC(MOVE) = 7: YINC(MOVE) = -3: L$(MOVE) = LDR$(2): GOTO 7600
7420 XINC(MOVE) = 5: YINC(MOVE) = -5: L$(MOVE) = LDR$(2): GOTO 7600
7430 XINC(MOVE) = 5: YINC(MOVE) = -5: L$(MOVE) = LDR$(2): GOTO 7600
7440 XINC(MOVE) = 3: YINC(MOVE) = -7: L$(MOVE) = LDR$(2): GOTO 7600
7450 XINC(MOVE) = 0: YINC(MOVE) = -8: L$(MOVE) = LDR$(1): GOTO 7600
7460 XINC(MOVE) = -3: YINC(MOVE) = -7: L$(MOVE) = LDR$(8): GOTO 7600
7470 XINC(MOVE) = -5: YINC(MOVE) = -5: L$(MOVE) = LDR$(8): GOTO 7600
7480 XINC(MOVE) = -5: YINC(MOVE) = -5: L$(MOVE) = LDR$(8): GOTO 7600
7490 XINC(MOVE) = -7: YINC(MOVE) = -3: L$(MOVE) = LDR$(8): GOTO 7600
7500 XINC(MOVE) = -8: YINC(MOVE) = 0: L$(MOVE) = LDR$(7): GOTO 7600
7510 XINC(MOVE) = -7: YINC(MOVE) = 3: L$(MOVE) = LDR$(6): GOTO 7600
7520 XINC(MOVE) = -5: YINC(MOVE) = 5: L$(MOVE) = LDR$(6): GOTO 7600
7530 XINC(MOVE) = -5: YINC(MOVE) = 5: L$(MOVE) = LDR$(6): GOTO 7600
7540 XINC(MOVE) = -3: YINC(MOVE) = 7: L$(MOVE) = LDR$(6): GOTO 7600
7550 XINC(MOVE) = 0: YINC(MOVE) = 8: L$(MOVE) = LDR$(5): GOTO 7600
7560 XINC(MOVE) = 3: YINC(MOVE) = 7: L$(MOVE) = LDR$(4): GOTO 7600
7570 XINC(MOVE) = 5: YINC(MOVE) = 5: L$(MOVE) = LDR$(4): GOTO 7600
7580 XINC(MOVE) = 5: YINC(MOVE) = 5: L$(MOVE) = LDR$(4): GOTO 7600
7590 XINC(MOVE) = 7: YINC(MOVE) = 3: L$(MOVE) = LDR$(4): GOTO 7600
7595 XINC(MOVE) = 8: YINC(MOVE) = 0: L$(MOVE) = LDR$(3): GOTO 7600
7600 '
7610 '        CALCULATE AMOUNT OF INCREMENT, SPEED LEADER
7620 '        SCALE, BASED ON SPEED
7630 '
7640 IF SPD(MOVE) >= 100 THEN 7690
7641   IF SPD(MOVE) <> 0 THEN 7650
7642     XINC(MOVE) = 0
7643     YINC(MOVE) = 0
7644     L$(MOVE) = ""
7645     GOTO 7770
7650  XINC(MOVE) = INT(.5 * XINC(MOVE))
7660  YINC(MOVE) = INT(.T * YINC(MOVE))
7670  L$(MOVE) = "S2" + L$(MOVE)
7680  GOTO 7770
7690 IF SPD(MOVE) <= 600 THEN 7770
7700   XINC(MOVE) = INT(2 * XINC(MOVE))
7710   YINC(MOVE) = INT(2 * YINC(MOVE))
7720   L$(MOVE) = "S8" + L$(MOVE)
7760 '
7770 RETURN
7780 '
8000 ' * * * * * * *   DRAW  LAND  SUBROUTINE   * * * * * * * *
8010 ' *                                                      *
8020 ' *  INPUTS:  PTS - ARRAY OF #s OF BORDER POINTS         *
8025 ' *           CONTS - # OF LAND MASSES                   *
8030 ' *                                                      *
```

```
8040 ' *  OUTPUT:  PLOTTED LAND MASSES, IN SPECIFIED COLORS   *
8050 ' *                                                      *
8060 ' * * * * * * * * * * * * *  * * * * * * * * * * * * * * *
8070 '
8075 IF CONTS = 0 THEN RETURN                  'NO LAND MASSES, NO DRAW
8080 '
8090 FOR I = 1 TO CONTS
8100   READ PTS(I), LCOL(I)
8110 NEXT I
8120 '
8125 DIM LAND1(PTS(1), 2), LAND2(PTS(2), 2), LAND3(PTS(3), 2)
8130 FOR ISLE = 1 TO PTS(1)
8140   READ LAND1(ISLE, 1), LAND1(ISLE, 2)
8150 NEXT ISLE
8160 '
8170 FOR ISLE = 1 TO PTS(2)
8180   READ LAND2(ISLE, 1), LAND2(ISLE, 2)
8190 NEXT ISLE
8200 '
8210 FOR ISLE = 1 TO PTS(3)
8220   READ LAND3(ISLE, 1), LAND3(ISLE, 2)
8230 NEXT ISLE
8240 '
8250 PSET (LAND1(1,1), LAND1(1,2)), LCOL(1)
8260 FOR ISLE = 2 TO PTS(1)
8270   LINE - (LAND1(ISLE, 1), LAND1(ISLE, 2)), LCOL(1)
8280 NEXT ISLE
8290 '
8300 READ CENTX, CENTY
8310 '
8320 PAINT (CENTX, CENTY), LCOL(1), LCOL(1)
8330 '
8340 IF PTS(2) < 2 THEN RETURN
8350 '
8360 PSET (LAND2(1,1), LAND2(1,2)), LCOL(2)
8370 FOR ISLE = 2 TO PTS(2)
8380   LINE - (LAND2(ISLE, 1), LAND2(ISLE, 2)), LCOCL(2)
8390 NEXT ISLE
8400 '
8410 READ CENTX, CENTY
8420 '
8430 PAINT (CENTX, CENTY), LCOL(2), LCOL(2)
8440 '
8450 IF PTS(3) < 2 THEN RETURN
8460 '
8470 PSET (LAND3(1,1), LAND3(1,2)), LCOL(3)
8480 FOR ISLE = 2 TO PTS(3)
8490   LINE - (LAND3(ISLE, 1), LAND3(ISLE, 2)), LCOCL(3)
8500 NEXT ISLE
8510 '
8520 READ CENTX, CENTY
```

```
8530 '
8540 PAINT (CENTX, CENTY), LCOL(3), LCOL(3)
8550 '
8560 RETURN
10000 ' ************   DATA   **************
10010 '
10020 '  XUL, YUL, XLR, YLR, CWIND
10030 '
10040 DATA 15, 27, 470, 190, 7
10050 '
10060 ' CONTS
10070 '
10080 DATA 3
10090 '
10100 '  PTS, LCOL, # OF TIMES THERE ARE LAND MASSES
10110 '
10120 DATA 13, 0
10130 DATA 8, 2
10140 DATA 10, 1
10150 '
10160 '  BORDER POINTS FOR LAND MASSES
10170 '
10180 DATA 16, 155, 55, 172, 90, 175, 130, 165, 175, 150, 175, 127
10190 DATA 210, 110, 180, 85, 260, 67, 245, 45, 160, 28, 16, 28, 16, 155
10200 '
10210 '
10220 '
10230 DATA 330, 155, 360, 165, 400, 160, 440, 140, 385, 125, 340, 133
10240 DATA 370, 142, 330, 155
10250 '
10260 '
10270 '
10280 DATA 385, 40, 405, 45, 400, 60, 380, 65, 365, 60, 375, 52
10290 DATA 350, 52, 370, 45, 390, 55, 385, 40
10300 '
10310 '   CENTERS OF LAND MASSES
10320 '
10330 DATA 100, 90
10340 '
10350 DATA 380, 150
10360 '
10370 DATA 380, 60
10380 '
10390 '   XYAX, YTOP, YBOTT, YCOL
10400 '
10410 DATA 157, 27, 190, 0
10420 '
10430 '   YXAX, XLEFT, XRITE, XCOL
10440 '
10450 DATA 145, 15, 470, 0
10460 '
```

```
10470 '       NUMBER OF TEST TRACKS
10480 DATA 3
10490 '
10500 '  CLASS$, CUS, SPD, TCOLOR, TX, TY  FOR EACH TEST TRACK
10510 '
10520 DATA "HOSTILE  ", 180, 35, 0, 420, 80
10530 '
10540 DATA "SURVEILL ", 4, 135, 0, 50, 100
10550 '
10560 DATA "UNKNOWN  ", 110, 650, 0, 430, 170
10570 '
10580 '       NUMBER OF MOVES TO TEST UPDATING
10590 '
10600 DATA 10
11000 ' * * * * * * *   TEST   TRACKING   SUBROUTINE    * * * * *
11010 ' *                                                       *
11020 ' *  INPUTS:  TRACKS - # OF TEST TRACKS                    *
11030 ' *                                                       *
11040 ' *  OUTPUT:  SAMPLE OF TRACKS BEING UPDATED              *
11050 ' *                                                       *
11060 ' * * * * * * * * * * * *  * * * * * * * * * * * * * * *
11070 '
11080 READ TRACKS
11090 '
11100 IF TRACKS = 0 THEN 11200
11110 FOR I = 1 TO TRACKS
11120   READ CLASS$(I), CUS(I), SPD(I), TCOLOR(I), TX(I), TY(I)
11125   UPD = I
11130   GOSUB 20000
11135   ACTIVE(I) = 1
11140 NEXT I
11150 '
11160 '
11165 FOR MOVE = 1 TO TRACKS
11170   GOSUB 7000
11175 NEXT MOVE
11180 '
11190 '
11200 DO$ = ""
11210 '
11220 WHILE DO$ = ""
11225   FOR UPD = 1 TO TRACKS
11230     GOSUB 6000
11235   NEXT UPD
11240   FOR I = 1 TO 2000
11250     DO$ = INKEY$
11255     IF DO$ = "" THEN NEXT I ELSE 11280
11260 WEND
11270 '
11280 IF DO$ <> CHR$(27) THEN 11200 ELSE DO2$ = INKEY$
11300 '
```

```
11310 IF DO2$ = "P" THEN GOSUB 12000
11320 IF DO2$ = "S" THEN GOSUB 12100
11330 IF DO2$ = "T" THEN GOSUB 12200
11340 IF DO2$ = "U" THEN GOSUB 12500
11350 IF DO2$ = "V" THEN GOSUB 12800
11360 IF DO2$ = "W" THEN GOSUB 13500
11370 '
11380 GOTO 11200
11390 '
12000 ' * * * * * * *   FUNCTION  KEY  SUBROUTINES   * * * * *
12010 '
12020 '
12030 ' % % % % %  HALT PROGRAM
12040 '                FUNCTION KEY F6
12050 '
12060 CLS
12062 KEY 1, "LIST "
12064 KEY 2, "RUN" + CHR$(13) + CHR$(10)
12066 KEY 3, "LOAD" + CHR$(34)
12068 KEY 4, "SAVE" + CHR$(34)
12070 '
12072 KEY 5, "CONT" + CHR$(13) + CHR$(10)
12074 KEY 6, "PRINT "
12080 END
12085 RETURN
12090 '
12100 ' % % % % %  SUSPEND/CONTINUE  PROGRAM
12110 '                FUNCTION KEY F1
12120 '
12130 GO$ = ""
12140 '
12150 WHILE GO$ = ""
12160   GO$ = INKEY$
12170 WEND
12180 '
12190 RETURN
12200 ' % % % % %  HOOK  TRACK
12210 '                FUNCTION KEY F2
12220 '
12230 LOCATE 2, 10
12240 '
12250 IF HOOK = 0 THEN 12270
12252 ACTIVE(HOOK) = 0
12254 UPD = HOOK
12256 GOSUB 6000
12258 ACTIVE(HOOK) = 1
12259 HK$(HOOK) = "SO"
12260 '
12270 INPUT "TRACK TO HOOK: ";HOOK
12275 LOCATE 2, 10
12276 PRINT "                                  "
```

```
12280 '
12282 ACTIVE(HOOK) = 0
12284 UPD = HOOK
12286 GOSUB 6000
12288 ACTIVE(HOOK) = 1
12290 HK$(HOOK) = "S8"
12300 '
12310 LOCATE 6, 62
12320 PRINT "TRACK NO. ";HOOK
12330 LOCATE 7, 62
12340 PRINT "CLASS  ";CLASS$(HOOK)
12350 LOCATE 8, 62
12360 PRINT "COURSE  ";CUS(HOOK)
12370 LOCATE 9, 62
12380 PRINT "SPEED   ";SPD(HOOK)
12390 '
12400 '
12410 RETURN
12420 '
12500 ' % % % % %  ENTER NEW TRACK
12510 '                FUNCTION KEY F3
12520 '
12530 TRACKS = TRACKS + 1
12540 MOVE = TRACKS
12550 '
12560 LOCATE 2, 10
12570 INPUT "ENTER CLASS   ";CLASS$(TRACKS)
12571 SIZECL = LEN(CLASS$(TRACKS))
12572 IF SIZECL < 9 THEN ADD = 9 - SIZECL
12573 IF ADD = 0 THEN 12575
12574 FOR I = 1 TO ADD:CLASS$(TRACKS) = CLASS$(TRACKS) + " ":NEXT I
12575 LOCATE 2, 10
12576 PRINT "                              "
12580 LOCATE 2, 10
12590 INPUT "ENTER COURSE  ";CUS(TRACKS)
12595 LOCATE 2, 10
12596 PRINT "                                  "
12600 LOCATE 2, 10
12610 INPUT "ENTER SPEED   ";SPD(TRACKS)
12615 LOCATE 2, 10
12616 PRINT "                                  "
12620 LOCATE 2, 10
12630 INPUT "ENTER GRID X  ";TX(TRACKS)
12635 LOCATE 2, 10
12636 PRINT "                                  "
12640 LOCATE 2, 10
12650 INPUT "ENTER GRID Y  ";TY(TRACKS)
12655 LOCATE 2, 10
12656 PRINT "                                  "
12660 LOCATE 2, 10
12670 INPUT "TRACK COLOR  ";TCOLOR(TRACKS)
```

```
12680 LOCATE 2, 10
12690 PRINT "                                        "
12700 '
12705 GOSUB 20000
12710 GOSUB 7000
12712 UPD = MOVE
12715 GOSUB 20000
12716 HK$(UPD) = "SO": ACTIVE(UPD) = 1
12717 GOSUB 6000
12720 '
12730 RETURN
12740 '
12750 '
12800 ' % % % % %  MODIFY  TRACK
12810 '                    FUNCTION KEY F4
12820 '
12830 IF HOOK = 0 THEN 12840
12832 ACTIVE(HOOK) = 0
12834 UPD = HOOK
12836 GOSUB 6000
12838 ACTIVE(HOOK) = 1
12839 HK$(HOOK) = "SO"
12840 LOCATE 2, 10
12850 INPUT "TRACK TO MODIFY: ";HOOK
12855 LOCATE 2, 10
12856 PRINT "                                        "
12860 '
12870 GOSUB 12300
12872 ACTIVE(HOOK) = 0
12874 UPD = HOOK
12876 GOSUB 6000
12878 ACTIVE(HOOK) = 1
12879 HK$(HOOK) = "SO"
12880 '
12890 LOCATE 2, 10
12900 INPUT "IS CLASS OK ";A$
12910 IF A$ <> "Y" THEN LOCATE 2, 40: INPUT "NEW CLASS :";CLASS$(HOOK)
12915 LOCATE 2, 10
12916 PRINT "                                        "
12920 '
12930 LOCATE 2, 10
12940 INPUT "IS COURSE OK ";A$
12950 IF A$ <> "Y" THEN LOCATE 2, 40: INPUT "NEW COURSE:";CUS(HOOK)
12955 LOCATE 2, 10
12956 PRINT "                                        "
12960 '
12970 LOCATE 2, 10
12980 INPUT "IS SPEED OK ";A$
12990 IF A$ <> "Y" THEN LOCATE 2, 40: INPUT "NEW SPEED:";SPD(HOOK)
12995 LOCATE 2, 10
12996 PRINT "                                        "
```

```
13000 '
13010 LOCATE 2, 10
13020 INPUT "IS COLOR OK ";A$
13030 IF A$ <> "Y" THEN LOCATE 2, 40: INPUT "NEW COLOR:";TCOLOR(HOOK)
13035 LOCATE 2, 10
13036 PRINT "                                    "
13040 '
13050 LOCATE 2, 10
13060 INPUT "IS GRID X OK ";A$
13070 IF A$ <> "Y" THEN LOCATE 2, 40: INPUT "NEW GRID X:";TX(HOOK)
13075 LOCATE 2, 10
13076 PRINT "                                    "
13080 '
13090 LOCATE 2, 10
13100 INPUT "IS GRID Y OK ";A$
13110 IF A$ <> "Y" THEN LOCATE 2, 40: INPUT "NEW GRID Y:";TY(HOOK)
13115 LOCATE 2, 10
13116 PRINT "                                    "
13120 '
13130 MOVE = HOOK
13140 GOSUB 7000
13145 UPD = HOOK
13147 GOSUB 6000
13150 '
13160 RETURN
13170 '
13500 ' % % % % %   DELETE A TRACK
13510 '                FUNCTION KEY F5
13520 '
13530 LOCATE 2, 10
13540 INPUT "TRACK TO DELETE: ";DEL
13550
13560 ACTIVE(DEL) = 0
13565 LOCATE 2, 10
13566 PRINT "                                    "
13570 '
13580 RETURN
13590 '
20000 ' * * * * * * *   SYMBOL  ASSIGNMENT   * * * * * * *
20010 ' *                                                *
20020 ' *  INPUTS:  UPD - TRACK TO HAVE SYMBOL ASSIGNED  *
20030 ' *                                                *
20040 ' *  OUTPUT:  TRACK(UPD) IS ASSIGNED A SYMBOL      *
20050 ' *           THAT MATCHES ITS CLASSIFICATION      *
20060 ' *                                                *
20070 ' * * * * * * * * * * * *  * * * * * * * * * * * * *
20080 '
20090 '
20100 IF CLASS$(UPD) = "HOSTILE  " THEN T$(UPD) = SYM$(4): GOTO 20240
20110 '
20120 IF CLASS$(UPD) = "HOST SURF" THEN T$(UPD) = SYM$(6): GOTO 20240
```

```
20130 '
20140 IF CLASS$(UPD) = "UNKNOWN  " THEN T$(UPD) = SYM$(3): GOTO 20240
20150 '
20160 IF CLASS$(UPD) = "UNK AIR  " THEN T$(UPD) = SYM$(5): GOTO 20240
20170 '
20180 IF CLASS$(UPD) = "FIGHTER  " THEN T$(UPD) = SYM$(2): GOTO 20240
20190 '
20200 IF CLASS$(UPD) = "SURVEILL " THEN T$(UPD) = SYM$(1): GOTO 20240
20210 '
20220 IF CLASS$(UPD) = "REF PNT  " THEN T$(UPD) = SYM$(7): GOTO 20240
20230 '
20240 RETURN
20250 '
```

APPENDIX B: LISTING OF HEADER.BAS

```
10 '                    SAMPLE NTDS DISPLAY SIMULATOR
20 '
30 '                    FRAME #1
40 '
50 '                    DISPLAY WINDOW WITH GRID
60 '
70 CLS                          'CLEAR THE DISPLAY
80 '
90 '
```

APPENDIX C: LISTING OF INIT.BAS

```
100 ' * * * * * * * *  INITIALIZATION AND TABLES  * * * * * * * *
110 '
120 '
130 OPTION BASE 1                         'ARRAY SUBSCRIPT LOWER BOUND = 1
140 '
150 DIM CLASS$(10), CUS(10), SPD(10), TCOLOR(10)
160 DIM TX(10), TY(10), XINC(10), YINC(10), T$(10), L$(10)
170 DIM SYM$(10), LDR$(8), PTS(3), LCOL(3), HK$(10), ACTIVE(10)
180 '
190 '          SYMBOL TABLE
200 '
210 SYM$(1) = "BM+0,-3 R3 D3 BM-6,0 D3 R3 BM+0,+3"
220 SYM$(2) = "BM+0,-3 L3 D3 BM+0,+3"
230 SYM$(3) = "BM+0,-3 R3 D6 L6 U6 R3 BM+0,+3"
240 SYM$(4) = "BM+0,-3 R2 F2 D3 G2 L4 H2 U3 E2 R2 BM+0,+3"
250 SYM$(5) = "BM+0,-3 R3 D3 BM-6,0 U3 R3 BM+0,+3"
260 SYM$(6) = "BM+0,-3 F3 G3 H3 E3 BM+0,+3
270 SYM$(7) = "U3 R3 D6 L6 U6 R3 D6 U3"
280 SYM$(8) = ""
290 SYM$(9) = ""
300 SYM$(10) = ""
310 '
320 '
330 '          SPEED LEADER TABLE
340 '
350 LDR$(1) = "U4"
360 LDR$(2) = "E3"
370 LDR$(3) = "R5"
380 LDR$(4) = "F3"
390 LDR$(5) = "D4"
400 LDR$(6) = "G3"
410 LDR$(7) = "L5"
420 LDR$(8) = "H3"
430 '
440 '
450 '          START WITH NO TRACKS
460 '
470 TRACKS = 0
480 '
490 '          INITIALIZE PTS ARRAY ELEMENTS TO 1
500 '
510 FOR I = 1 TO 3: PTS(I) = 1: NEXT I
520 '
530 '          DEFINE FUNCTION KEYS
540 '
560 KEY 1, CHR$(27) + "S"
```

```
570 KEY 2, CHR$(27) + "T"
580 KEY 3, CHR$(27) + "U"
590 KEY 4, CHR$(27) + "V"
600 KEY 5, CHR$(27) + "W"
605 KEY 6, CHR$(27) + "P"
610 '
620 '           INITIALIZE HK$ AND ACTIVE
630 '
640 FOR I = 1 TO 10
650 HK$(I) = "SO"
660 ACTIVE(I) = 0
670 NEXT I
680 '
690 '           DISPLAY FUNCTION KEY FUNCTIONS
700 '
710 COLOR 0,7
720 LOCATE 25,5
730 PRINT " F1 "
740 LOCATE 25, 19
750 PRINT " F2 "
760 LOCATE 25, 29
770 PRINT " F3 "
780 LOCATE 25, 40
790 PRINT " F4 "
800 LOCATE 25, 52
810 PRINT " F5 "
820 LOCATE 25, 64
830 PRINT " F6 "
840 COLOR 7, 0
850 LOCATE 25, 9
860 PRINT "SUSP/CONT"
870 LOCATE 25, 24
880 PRINT "HOOK"
890 LOCATE 25, 34
900 PRINT "ENTER"
910 LOCATE 25, 45
920 PRINT "MODIFY"
930 LOCATE 25, 57
940 PRINT "DELETE"
950 LOCATE 25, 69
960 PRINT "HALT"
```

APPENDIX D: LISTING OF HARNESS.BAS

```
1000 '
1010 '          GET WINDOW PARAMETERS
1020 '
1030 READ XUL, YUL, XLR, YLR, CWIND
1040 '
1050 '          DRAW THE WINDOW
1060 '
1070 GOSUB 5000
1080 '
1090 '          GET LAND PARAMETERS
1100 '
1110 READ CONTS                         'HOW MANY LAND MASSES?
1120 '
1130 '          DRAW LAND MASSES
1140 '
1150 GOSUB 8000
1160 '
1170 '          GET GRID PARAMETERS
1180 '
1190 READ XYAX, YTOP, YBOTT, YCOL
1200 READ YXAX, XLEFT, XRITE, XCOL
1210 '
1220 '          DRAW THE GRID
1230 '
1240 GOSUB 5200
1250 '
1260 '          RUN UPDATE TESTS
1270 '
1280 GOSUB 11000
1290 '
1300 '
4999 END
```

APPENDIX E: LISTING OF WINDOW.BAS

```
5000 '  * * * * * * *   DRAW  WINDOW  SUBROUTINE   * * * * * * *
5010 '  *                                                       *
5020 '  *  INPUTS:  XUL, YUL - UPPER LEFT-HAND COORDINATES      *
5030 '  *           XLR, YLR - LOWER RIGHT-HAND COORDINATES     *
5040 '  *           CWIND    - COLOR OF WINDOW                  *
5050 '  *                                                       *
5060 '  *  OUTPUT:  SOLID WINDOW, XLR - XUL PIXELS WIDE,        *
5070 '  *           YLR - YUL PIXELS DEEP, COLOR CWIND          *
5080 '  *                                                       *
5090 '  * * * * * * * * * * * * *  * * * * * * * * * * * * * * *
5100 '
5110 '
5120 LINE (XUL, YUL) - (XLR, YLR), CWIND, BF
5130 '
5140 RETURN
5150 '
5160 '
```

APPENDIX F: LISTING OF AXES.BAS

```
5200 '  * * * * * * *   COORDINATE  AXES  SUBROUTINE   * * * * * * * * *
5210 '  *                                                               *
5220 '  *  INPUTS:  XYAX, YTOP, YBOTT  - VERTICAL AXIS COORDINATES      *
5230 '  *           YXAX, XLEFT, XRITE - HORIZONTAL AXIS COORDINATES    *
5240 '  *           XCOL, YCOL         - GRID COLORS                    *
5250 '  *                                                               *
5260 '  *  OUTPUT:  PROPERLY SCALED SET OF COORDINATE AXES,             *
5270 '  *           OF XCOL AND YCOL                                    *
5280 '  *                                                               *
5290 '  * * * * * * * * * * * * * *  * * * * * * * * * * * * * * * * * *
5300 '
5310 '
5320 HSCALE = (XRITE - XLEFT)/20               'HORIZONTAL SCALE MULTIPLIER
5330 VSCALE = HSCALE * .46                     'VERTICAL SCALE MULTIPLIER,
5340                                           'FOR PROPER ASPECT RATIO
5350 '         DRAW VERTICAL AXIS
5360 '
5365 LINE (XYAX-1, YTOP) - (XYAX+1, YBOTT), CWIND, BF
5370 LINE (XYAX, YTOP) - (XYAX, YBOTT), YCOL
5380 '
5390 '         DRAW HORIZONTAL AXIS
5400 '
5405 LINE (XLEFT, YXAX-1) - (XRITE, YXAX+1), CWIND, BF
5410 LINE (XLEFT, YXAX) - (XRITE, YXAX), XCOL
5420 '
5430 '         DRAW HORIZONTAL SCALE DIVISIONS, LEFT
5440 '
5450 FOR H = XYAX TO XLEFT STEP -HSCALE
5460   LINE (H, YXAX-2) - (H, YXAX+2), XCOL
5470   LINE (H+1, YXAX-2) - (H+1, YXAX+2), XCOL
5480 NEXT H
5490 '
5500 '         DRAW HORIZONTAL SCALE DIVISIONS, RIGHT
5510 '
5520 FOR H = XYAX TO XRITE STEP HSCALE
5530   LINE (H, YXAX-2) - (H, YXAX+2), XCOL
5540   LINE (H+1, YXAX-2) - (H+1, YXAX+2), XCOL
5550 NEXT H
5560 '
5570 '         DRAW VERTICAL SCALE DIVISIONS, UPPER
5580 '
5590 FOR V = YXAX TO YTOP STEP -VSCALE
5600   LINE (XYAX-4, V) - (XYAX+4, V), YCOL
5610 NEXT V
5620 '
5630 '         DRAW VERTICAL SCALE DIVISIONS, LOWER
5640 '
```

```
5650 FOR V = YXAX TO YBOTT STEP VSCALE
5660   LINE (XYAX-4, V) - (XYAX+4, V), YCOL
5670 NEXT V
5680 '
5690 RETURN
5700 '
5710 '
5720 '          THIS AXES SUBROUTINE IS BASED ON THE PROGRAM
5730 '          9-2, PAGE 9-15, IN THE CONTINUING EDUCATION
5740 '          CORRESPONDENCE COURSE "COMPUTER GRAPHICS",
5750 '          WRITTEN FOR HEATHKIT/ZENITH BY
5760 '                  JAMES  C.  ADAMS
```

APPENDIX G: LISTING OF UPDATE.BAS

```
6000 '  * * * * * *   UPDATE  TRACKS  SUBROUTINE   * * * * * * *
6010 '  *                                                       *
6020 '  *  INPUTS:  UPD -  OF TRACK TO UPDATE                   *
6030 '  *                                                       *
6040 '  *  OUTPUT:  TRACK UPD IS UPDATED                        *
6050 '  *                                                       *
6060 '  * * * * * * * * * * * * *  * * * * * * * * * * * * * * *
6070 '
6080 '
6100 '
6120 '
6130 '         PERFORM ALL LOOK-UPS ONLY ONCE
6140 '
6150 UPDX = TX(UPD)
6160 UPDY = TY(UPD)
6170 UPDT$ = HK$(UPD) + T$(UPD)
6180 UPDL$ = L$(UPD)
6190 HORZUP = XINC(UPD)
6200 VERTUP = YINC(UPD)
6210 COLUP = TCOLOR(UPD)
6220 '
6225 IF ACTIVE(UPD) = 2 THEN 6375
6230 UPGND = POINT(UPDX+2, UPDY+1)
6240 ON UPGND+1 GOSUB 6520, 6530, 6540, 6550, 6560, 6570, 6580, 6590
6250 WANT$ = COL$ + UPDT$: ALSO$ = COL$ + UPDL$
6255 '
6260 PSET (UPDX, UPDY), UPGND                  'DRAW OLD SYMBOL IN
6270 DRAW WANT$                                'REVERSE COLOR
6280 PSET (UPDX, UPDY), UPGND
6290 DRAW ALSO$
6295 DRAW "SO"
6300 '
6305 IF ACTIVE(UPD) = 0 THEN 6490
6310 UPDX = UPDX + HORZUP                      'UPDATE POSITION
6320 UPDY = UPDY + VERTUP
6322 IF UPDX<15 OR UPDX>470 THEN 6482
6324 IF UPDY<27 OR UPDY>190 THEN 6482
6330 '
6340 UPGND = POINT(UPDX+2, UPDY+1)             'CHECK BACKGROUND COLOR
6350 '                                          OF NEW LOCATION
6360 IF UPGND <> COLUP THEN 6375               'MAKE SYMBOL OPPOSITE
6370 IF UPGND < 2 THEN COLUP = 7 ELSE COLUP = 0     ' OF BACKGROUND
6374 '
6375 ON COLUP+1 GOSUB 6520, 6530, 6540, 6550, 6560, 6570, 6580, 6590
6376 WANT$ = COL$ + UPDT$: ALSO$ = COL$ + UPDL$
6380 '
6390 PSET (UPDX, UPDY), COLUP                  'DRAW NEW SYMBOL
```

```
6400 DRAW WANT$
6410 PSET (UPDX, UPDY), COLUP
6420 DRAW ALSO$
6425 DRAW "SO"
6430 '
6440 TX(UPD) = UPDX                                'STORE NEW POSITION
6450 TY(UPD) = UPDY
6460 '
6480 '
6482 ACTIVE(UPD)=0
6490 RETURN
6500 '
6510 '
6520 COL$ = "C0": RETURN
6530 COL$ = "C1": RETURN
6540 COL$ = "C2": RETURN
6550 COL$ = "C3": RETURN
6560 COL$ = "C4": RETURN
6570 COL$ = "C5": RETURN
6580 COL$ = "C6": RETURN
6590 COL$ = "C7": RETURN
```

APPENDIX H: LISTING OF MOVE.BAS

```
7000 '  * * * * * * *   SYMBOL  MOVEMENT  CALCULATOR   * * * * *
7010 '  *                                                        *
7020 '  *  INPUTS:  MOVE   - TRACK TO CALCULATE FOR              *
7030 '  *                                                        *
7040 '  *  OUTPUT:  XINC, YINC, SCALE FACTOR FOR SPEED           *
7050 '  *           LEADER OF EACH ACTIVE TRACK ARE              *
7060 '  *           CALCULATED AND STORED                        *
7070 '  *                                                        *
7080 '  * * * * * * * * * * *  * * * * * * * * * * * * * * * * *
7090 '
7100 '
7110 '
7130 '
7140 '         CALCULATE INCREMENTS BASED ON COURSE
7150 '
7160 IF CUS(MOVE) <=     5 THEN 7400
7170 IF CUS(MOVE) <=  22.5 THEN 7410
7180 IF CUS(MOVE) <=    45 THEN 7420
7190 IF CUS(MOVE) <=  67.5 THEN 7430
7200 IF CUS(MOVE) <=    85 THEN 7440
7210 IF CUS(MOVE) <=    95 THEN 7450
7220 IF CUS(MOVE) <= 112.5 THEN 7460
7230 IF CUS(MOVE) <=   135 THEN 7470
7240 IF CUS(MOVE) <= 157.5 THEN 7480
7250 IF CUS(MOVE) <=   175 THEN 7490
7260 IF CUS(MOVE) <=   185 THEN 7500
7270 IF CUS(MOVE) <= 202.5 THEN 7510
7280 IF CUS(MOVE) <=   225 THEN 7520
7290 IF CUS(MOVE) <= 247.5 THEN 7530
7300 IF CUS(MOVE) <=   265 THEN 7540
7310 IF CUS(MOVE) <=   275 THEN 7550
7320 IF CUS(MOVE) <= 292.5 THEN 7560
7330 IF CUS(MOVE) <=   315 THEN 7570
7340 IF CUS(MOVE) <= 337.5 THEN 7580
7350 IF CUS(MOVE) <=   355 THEN 7590
7360 '
7370 '
7400 XINC(MOVE) = 8: YINC(MOVE) = 0: L$(MOVE) = LDR$(3): GOTO 7600
7410 XINC(MOVE) = 7: YINC(MOVE) = -3: L$(MOVE) = LDR$(2): GOTO 7600
7420 XINC(MOVE) = 5: YINC(MOVE) = -5: L$(MOVE) = LDR$(2): GOTO 7600
7430 XINC(MOVE) = 5: YINC(MOVE) = -5: L$(MOVE) = LDR$(2): GOTO 7600
7440 XINC(MOVE) = 3: YINC(MOVE) = -7: L$(MOVE) = LDR$(2): GOTO 7600
7450 XINC(MOVE) = 0: YINC(MOVF) = -8: L$(MOVE) = LDR$(1): GOTO 7600
7460 XINC(MOVE) = -3: YINC(MOVE) = -7: L$(MOVE) = LDR$(8): GOTO 7600
7470 XINC(MOVE) = -5: YINC(MOVE) = -5: L$(MOVE) = LDR$(8): GOTO 7600
7480 XINC(MOVE) = -5: YINC(MOVE) = -5: L$(MOVE) = LDR$(8): GOTO 7600
```

```
7490 XINC(MOVE) = -7: YINC(MOVE) = -3: L$(MOVE) = LDR$(8): GOTO 7600
7500 XINC(MOVE) = -8: YINC(MOVE) = 0: L$(MOVE) = LDR$(7): GOTO 7600
7510 XINC(MOVE) = -7: YINC(MOVE) = 3: L$(MOVE) = LDR$(6): GOTO 7600
7520 XINC(MOVE) = -5: YINC(MOVE) = 5: L$(MOVE) = LDR$(6): GOTO 7600
7530 XINC(MOVE) = -5: YINC(MOVE) = 5: L$(MOVE) = LDR$(6): GOTO 7600
7540 XINC(MOVE) = -3: YINC(MOVE) = 7: L$(MOVE) = LDR$(6): GOTO 7600
7550 XINC(MOVE) = 0: YINC(MOVE) = 8: L$(MOVE) = LDR$(5): GOTO 7600
7560 XINC(MOVE) = 3: YINC(MOVE) = 7: L$(MOVE) = LDR$(4): GOTO 7600
7570 XINC(MOVE) = 5: YINC(MOVE) = 5: L$(MOVE) = LDR$(4): GOTO 7600
7580 XINC(MOVE) = 5: YINC(MOVE) = 5: L$(MOVE) = LDR$(4): GOTO 7600
7590 XINC(MOVE) = 7: YINC(MOVE) = 3: L$(MOVE) = LDR$(4): GOTO 7600
7595 XINC(MOVE) = 8: YINC(MOVE) = 0: L$(MOVE) = LDR$(3)
7600 '
7610 '         CALCULATE AMOUNT OF INCREMENT, SPEED LEADER
7620 '         SCALE, BASED ON SPEED
7630 '
7640 IF SPD(MOVE) >= 100 THEN 7690
7641   IF SPD(MOVE) <> 0 THEN 7650
7642     XINC(MOVE) = 0
7643     YINC(MOVE) = 0
7644     L$(MOVE) = ""
7645     GOTO 7770
7650   XINC(MOVE) = INT(.5 * XINC(MOVE))
7660   YINC(MOVE) = INT(.5 * YINC(MOVE))
7670   L$(MOVE) = "S2" + L$(MOVE)
7680   GOTO 7770
7690 IF SPD(MOVE) <= 600 THEN 7770
7700   XINC(MOVE) = INT(2 * XINC(MOVE))
7710   YINC(MOVE) = INT(2 * YINC(MOVE))
7720   L$(MOVE) = "S8" + L$(MOVE)
7760 '
7770 RETURN
7780 '
```

APPENDIX I: LISTING OF LAND.BAS

```
8000 '  * * * * * * *   DRAW  LAND  SUBROUTINE   * * * * * * * *
8010 '  *                                                     *
8020 '  *  INPUTS:  PTS - ARRAY OF #s OF BORDER POINTS        *
8025 '  *           CONTS - # OF LAND MASSES                  *
8030 '  *                                                     *
8040 '  *  OUTPUT:  PLOTTED LAND MASSES, IN SPECIFIED COLORS  *
8050 '  *                                                     *
8060 '  * * * * * * * * * * * * *  * * * * * * * * * * * * * * *
8070 '
8075 IF CONTS = 0 THEN RETURN                    'NO LAND MASSES, NO DRAW
8080 '
8090 FOR I = 1 TO CONTS
8100   READ PTS(I), LCOL(I)
8110 NEXT I
8120 '
8125 DIM LAND1(PTS(1), 2), LAND2(PTS(2), 2), LAND3(PTS(3), 2)
8130 FOR ISLE = 1 TO PTS(1)
8140   READ LAND1(ISLE, 1), LAND1(ISLE, 2)
8150 NEXT ISLE
8160 '
8170 FOR ISLE = 1 TO PTS(2)
8180   READ LAND2(ISLE, 1), LAND2(ISLE, 2)
8190 NEXT ISLE
8200 '
8210 FOR ISLE = 1 TO PTS(3)
8220   READ LAND3(ISLE, 1), LAND3(ISLE, 2)
8230 NEXT ISLE
8240 '
8250 PSET (LAND1(1,1), LAND1(1,2)), LCOL(1)
8260 FOR ISLE = 2 TO PTS(1)
8270   LINE - (LAND1(ISLE, 1), LAND1(ISLE, 2)), LCOL(1)
8280 NEXT ISLE
8290 '
8300 READ CENTX, CENTY
8310 '
8320 PAINT (CENTX, CENTY), LCOL(1), LCOL(1)
8330 '
8340 IF PTS(2) < 2 THEN RETURN
8350 '
8360 PSET (LAND2(1,1), LAND2(1,2)), LCOL(2)
8370 FOR ISLE = 2 TO PTS(2)
8380   LINE - (LAND2(ISLE, 1), LAND2(ISLE, 2)), LCOL(2)
8390 NEXT ISLE
8400 '
8410 READ CENTX, CENTY
8420 '
```

```
8430 PAINT (CENTX, CENTY), LCOL(2), LCOL(2)
8440 '
8450 IF PTS(3) < 2 THEN RETURN
8460 '
8470 PSET (LAND3(1,1), LAND3(1,2)), LCOL(3)
8480 FOR ISLE = 2 TO PTS(3)
8490   LINE - (LAND3(ISLE, 1), LAND3(ISLE, 2)), LCOL(3)
8500 NEXT ISLE
8510 '
8520 READ CENTX, CENTY
8530 '
8540 PAINT (CENTX, CENTY), LCOL(3), LCOL(3)
8550 '
8560 RETURN
```

APPENDIX J: LISTING OF DATA.BAS

```
10000 ' ***********   DATA   *************
10010 '
10020 '  XUL, YUL, XLR, YLR, CWIND
10030 DATA  15, 27, 470, 190, 7
10040 ' CONTS
10050 DATA 2
10060 ' PTS(1), LCOL(1), PTS(2), LCOL(2)
10070 DATA 8, 3
10080 DATA 5, 5
10090 ' BORDER POINTS FOR LAND MASS ONE
10100 DATA 100, 125, 120, 150, 130, 140, 125, 135, 155, 134
10110 DATA 160, 127, 125, 125, 100, 125
10120 ' BORDER POINTS FOR LAND MASS TWO
10130 DATA 240, 100, 270, 105, 290, 90, 265, 85, 240, 100
10140 ' BORDER POINTS FOR DUMMY LAND MASS
10150 DATA 1, 1
10160 ' CENTER OF LAND MASS ONE
10170 DATA 120, 135
10180 ' CENTER OF LAND MASS TWO
10190 DATA 265, 95
10200 ' XYAX, YTOP, YBOTT, YCOL
10210 DATA 157, 27, 190, 0
10220 ' YXAX, XLEFT, XRITE, XCOL
10230 DATA 145, 15, 470, 0
10235 DATA 3
10240 ' TRACK(1), CLASS$(1), CUS(1), SPD(1), TCOLOR(1), TX(1), TY(1)
10250 DATA "HOSTILE", 180, 35, 0, 420, 80
10260 ' TRACK(2), CLASS$(2), CUS(2), SPD(2), TCOLOR(2), TX(2), TY(2)
10270 DATA "FRIENDLY", 4, 135, 0, 50, 100
10275 '       TRACK (3)
10280 DATA "UNKNOWN", 110, 650, 0, 430, 170
10290 '       NUMBER OF MOVES TO TEST UPDATING
10300 DATA 5
```

APPENDIX K: LISTING OF TRACKING.BAS

```
11000 '  * * * * * * *   TEST  TRACKING  SUBROUTINE    * * * * *
11010 '  *                                                      *
11020 '  *  INPUTS:  TRACKS - # OF TEST TRACKS                  *
11030 '  *                                                      *
11040 '  *  OUTPUT:  SAMPLE OF TRACKS BEING UPDATED             *
11050 '  *                                                      *
11060 '  * * * * * * * * * * * * * * * * * * * * * * * * * * * *
11070 '
11080 READ TRACKS
11090 '
11100 IF TRACKS = 0 THEN 11200
11110 FOR I = 1 TO TRACKS
11120   READ CLASS$(I), CUS(I), SPD(I), TCOLOR(I), TX(I), TY(I)
11125   UPD = I
11130   GOSUB 20000
11135   ACTIVE(I) = 1
11136 IF CLASS$(I) = "REF PNT  " THEN ACTIVE(I) = 2
11140 NEXT I
11150 '
11160 '
11165 FOR MOVE = 1 TO TRACKS
11170   GOSUB 7000
11175 NEXT MOVE
11180 '
11190 '
11200 DO$ = ""
11210 '
11220 WHILE DO$ = ""
11225  FOR UPD  = 1 TO TRACKS
11230     GOSUB 6000
11235  NEXT UPD
11240  FOR I = 1 TO 2000
11250    DO$ = INKEY$
11255    IF DO$="" THEN NEXT I ELSE 11280
11260 WEND
11270 '
11280 IF DO$ <> CHR$(27) THEN 11200 ELSE DO2$ = INKEY$
11300 '
11310 IF DO2$ = "P" THEN GOSUB 12000
11320 IF DO2$ = "S" THEN GOSUB 12100
11330 IF DO2$ = "T" THEN GOSUB 12200
11340 IF DO2$ = "U" THEN GOSUB 12500
11350 IF DO2$ = "V" THEN GOSUB 12800
11360 IF DO2$ = "W" THEN GOSUB 13500
11370 '
11380 GOTO 11200
```

APPENDIX L: LISTING OF KEYS.BAS

```
12000 ' * * * * * * *   FUNCTION  KEY  SUBROUTINES   * * * * *
12010 '
12020 '
12030 ' % % % % %  HALT  PROGRAM
12040 '                 FUNCTION KEY F6
12050 '
12060 CLS
12062 KEY 1, "LIST "
12064 KEY 2, "RUN" + CHR$(13) + CHR$(10)
12066 KEY 3, "LOAD" + CHR$(34)
12068 KEY 4, "SAVE" + CHR$(34)
12070 '
12072 KEY 5, "CONT" + CHR$(13) + CHR$(10)
12074 KEY 6, "PRINT "
12080 END
12085 RETURN
12090 '
12100 ' % % % % %  SUSPEND/CONTINUE  PROGRAM
12110 '                 FUNCTION KEY F1
12120 '
12130 GO$ = ""
12140 '
12150 WHILE GO$ = ""
12160   GO$ = INKEY$
12170 WEND
12180 '
12190 RETURN
12200 ' % % % % %  HOOK  TRACK
12210 '                 FUNCTION KEY F2
12220 '
12230 LOCATE 2, 10
12240 '
12250 IF HOOK = 0 THEN 12270
12252 ACTIVE(HOOK) = 0
12254 UPD = HOOK
12256 GOSUB 6000
12258 ACTIVE(HOOK) = 1
12259 HK$(HOOK) = "SO"
12260 '
12270 INPUT "TRACK TO HOOK: ";HOOK
12275 LOCATE 2, 10
12276 PRINT "                                  "
12280 '
12282 ACTIVE(HOOK) = 0
12284 UPD = HOOK
12286 GOSUB 6000
```

```
12288 ACTIVE(HOOK) = 1
12290 HK$(HOOK) = "S8"
12300 '
12310 LOCATE 6, 62
12320 PRINT "TRACK NO. ";HOOK
12330 LOCATE 7, 62
12340 PRINT "CLASS  ";CLASS$(HOOK)
12350 LOCATE 8, 62
12360 PRINT "COURSE   ";CUS(HOOK)
12370 LOCATE 9, 62
12380 PRINT "SPEED    ";SPD(HOOK)
12390 '
12400 '
12410 RETURN
12420 '
12500 ' % % % % %  ENTER  NEW  TRACK
12510 '                FUNCTION KEY F3
12520 '
12530 TRACKS = TRACKS + 1
12540 MOVE = TRACKS
12550 '
12560 LOCATE 2, 10
12570 INPUT "ENTER CLASS   ";CLASS$(TRACKS)
12571 SIZECL = LEN(CLASS$(TRACKS))
12572 IF SIZECL < 9 THEN ADD = 9 - SIZECL
12573 IF ADD=0 THEN 12575
12574 FOR I = 1 TO ADD:CLASS$(TRACKS) = CLASS$(TRACKS) + " ":NEXT I
12575 LOCATE 2, 10
12576 PRINT "                          "
12580 LOCATE 2, 10
12590 INPUT "ENTER COURSE  ";CUS(TRACKS)
12595 LOCATE 2, 10
12596 PRINT "                          "
12600 LOCATE 2, 10
12610 INPUT "ENTER SPEED   ";SPD(TRACKS)
12615 LOCATE 2, 10
12616 PRINT "                          "
12620 LOCATE 2, 10
12630 INPUT "ENTER GRID X  ";TX(TRACKS)
12635 LOCATE 2, 10
12636 PRINT "                          "
12640 LOCATE 2, 10
12650 INPUT "ENTER GRID Y  ";TY(TRACKS)
12655 LOCATE 2, 10
12656 PRINT "                          "
12660 LOCATE 2, 10
12670 INPUT "TRACK COLOR   ";TCOLOR(TRACKS)
12680 LOCATE 2, 10
12690 PRINT "                                     "
12700 '
12702 IF CLASS$(MOVE) = "REF PNT  " THEN ACTIVE(MOVE) = 2
```

```
12705 GOSUB 20000
12710 GOSUB 7000
12712 UPD = MOVE
12715 GOSUB 20000
12716 HK$(UPD) = "SO": ACTIVE(UPD) = 1
12717 GOSUB 6000
12720 '
12730 RETURN
12740 '
12750 '
12800 ' % % % % %  MODIFY TRACK
12810 '                FUNCTION KEY F4
12820 '
12830 IF HOOK =  0 THEN 12840
12832 ACTIVE(HOOK) = 0
12834 UPD = HOOK
12836 GOSUB 6000
12838 ACTIVE(HOOK) = 1
12839 HK$(HOOK) = "SO"
12840 LOCATE 2, 10
12850 INPUT "TRACK TO MODIFY: ";HOOK
12855 LOCATE 2, 10
12856 PRINT "                              "
12860 '
12870 GOSUB 12300
12872 ACTIVE(HOOK) = 0
12874 UPD = HOOK
12876 GOSUB 6000
12878 ACTIVE(HOOK) = 1
12879 HK$(HOOK) = "SO"
12880 '
12890 LOCATE 2, 10
12900 INPUT "IS CLASS OK ";A$
12910 IF A$ <> "Y" THEN LOCATE 2, 40 : INPUT "NEW CLASS :";CLASS$(HOOK)
12915 LOCATE 2, 10
12916 PRINT "                                                     "
12920 '
12930 LOCATE 2, 10
12940 INPUT "IS COURSE OK ";A$
12950 IF A$ <> "Y" THEN LOCATE 2, 40 : INPUT "NEW COURSE:";CUS(HOOK)
12955 LOCATE 2, 10
12956 PRINT "                                                     "
12960 '
12970 LOCATE 2, 10
12980 INPUT "IS SPEED OK ";A$
12990 IF A$ <> "Y" THEN LOCATE 2, 40 : INPUT "NEW SPEED: ";SPD(HOOK)
12995 LOCATE 2, 10
12996 PRINT "                                                 "
13000 '
13010 LOCATE 2, 10
13020 INPUT "IS COLOR OK ";A$
```

```
13030 IF A$ <> "Y" THEN LOCATE 2, 40 : INPUT "NEW COLOR: ";TCOLOR(HOOK)
13035 LOCATE 2, 10
13036 PRINT "                                                  "
13040 '
13050 LOCATE 2, 10
13060 INPUT "IS GRID X OK ";A$
13070 IF A$ <> "Y" THEN LOCATE 2, 40 : INPUT "NEW GRID X: ";TX(HOOK)
13075 LOCATE 2, 10
13076 PRINT "                                                  "
13080 '
13090 LOCATE 2, 10
13100 INPUT "IS GRID Y OK ";A$
13110 IF A$ <> "Y" THEN LOCATE 2, 40 : INPUT "NEW GRID Y: ";TY(HOOK)
13115 LOCATE 2, 10
13116 PRINT "                                                  "
13120 '
13130 MOVE = HOOK
13140 GOSUB 7000
13145 UPD = HOOK
13147 GOSUB 6000
13150 '
13160 RETURN
13170 '
13500 ' % % % % %  DELETE A TRACK
13510 '                FUNCTION KEY F5
13520 '
13530 LOCATE 2, 10
13540 INPUT "TRACK TO DELETE: ";DEL
13550 '
13560 ACTIVE(DEL) = 0
13565 LOCATE 2, 10
13566 PRINT "                                                  "
13570 '
13580 RETURN
```

APPENDIX M: LISTING OF MATCH.BAS

```
20000 ' * * * * * * *   SYMBOL  ASSIGNMENT   * * * * * * *
20010 ' *                                                 *
20020 ' *  INPUTS:  UPD - TRACK TO HAVE SYMBOL ASSIGNED  *
20030 ' *                                                 *
20040 ' *  OUTPUT:  TRACK(UPD) IS ASSIGNED A SYMBOL      *
20050 ' *           THAT MATCHES ITS CLASSIFICATION      *
20060 ' *                                                 *
20070 ' * * * * * * * * * * * * *  * * * * * * * * * * * * *
20080 '
20090 '
20100 IF CLASS$(UPD) = "HOSTILE  " THEN T$(UPD) = SYM$(4): GOTO 20240
20110 '
20120 IF CLASS$(UPD) = "HOST SURF" THEN T$(UPD) = SYM$(6): GOTO 20240
20130 '
20140 IF CLASS$(UPD) = "UNKNOWN  " THEN T$(UPD) = SYM$(3): GOTO 20240
20150 '
20160 IF CLASS$(UPD) = "UNK AIR  " THEN T$(UPD) = SYM$(5): GOTO 20240
20170 '
20180 IF CLASS$(UPD) = "FIGHTER  " THEN T$(UPD) = SYM$(2): GOTO 20240
20190 '
20200 IF CLASS$(UPD) = "SURVEILL " THEN T$(UPD) = SYM$(1): GOTO 20240
20210 '
20220 IF CLASS$(UPD) = "REF PNT  " THEN T$(UPD) = SYM$(7)
20230 '
20240 RETURN
20250 '
```

APPENDIX N: USER'S MANUAL FOR DISPLAY SIMULATOR

A. HOW TO USE THIS SIMULATOR

The minimum system configuration requirements for this NTDS display simulator are as follows:

- H/Z-100 or compatible computer
- 128K RAM memory
- 1 DS/DD 5 1/4" disk drive
- Z-DOS 1.25 operating system
- ZBASIC interpreter or compiler
- the file NEWEST.BAS

  or
- the subroutine files making up NEWEST.BAS, which are:
  - -- HEADER.BAS
  - -- INIT.BAS
  - -- HARNESS.BAS
  - -- WINDOW.BAS
  - -- AXES.BAS
  - -- LAND.BAS
  - -- MOVE.BAS
  - -- UPDATE.BAS
  - -- TRACKING.BAS
  - -- DATA.BAS or DATA1.BAS
  - -- MATCH.BAS
  - -- KEYS.BAS

B. GETTING STARTED

"Boot up" the computer system under the Z-DOS operating system. After getting the system prompt ensure that the default disk drive (if there are two or more) contains the file ZBASIC.COM (the ZBASIC interpreter), unless using the compiled version. If working with the compiled version, follow the instructions for compiling and running a ZBASIC program that came with the compiler being used.

1. This step is for users without the file NEWEST.BAS. If that file is present, skip to step B.2.

Type the command "ZBASIC". This will load the ZBASIC interpreter. When it displays its prompt, load any one of the subroutine files. Then type MERGE "filename" for each of the other files, one by one. Once they are all merged type RUN (If any error messages are displayed when attempting to do this, one or more of the files may not be stored properly. In order to store them properly they will have to be loaded when there is no other program stored in memory, and saved with the command SAVE "filename", A. This saves them in an ASCII format, which allows them to be merged with other files).

2. Load NEWEST. Type RUN.

C. INTERACTING WITH THE SIMULATOR

While the simulator is running it accepts user input through the use of the special function keys. The special function key menu is displayed on line 25 of the monitor, below the NTDS display.

The Suspend/Continue key is a double action key -- to suspend the automatic updating of tracks (and all other system functions) depress the F1 key. To resume system operation depress it again. When it is depressed for the "continue" function all tracks will be updated.

The Hook key (F2) allows the display of the track parameters for one of the tracks in the system. When it is depressed the system will request an input at the top of the screen. It will prompt the user to input the track number of the track to hook. The input must be a number between 1 and 10, or an error will result. If the number is the number of an active track that track will have its symbol enlarged as long as it is hooked, and its parameters displayed to the right of the NTDS display area.

The Enter key (F3) allows a new track to be entered. The user will be prompted for inputs at the top of the screen. For CLASS$, input the classification of the new track (no more than nine characters, please). If the classification does not match one the system recognizes, the speed leader only will be displayed for the new track. The currently recognized inputs are: "HOSTILE", "FIGHTER", "UNKNOWN", "HOST SURF", "REF PNT", "SURVEILL", and "UNK AIR". The CUS (course) should be a number between 0 and 360, representing degrees true. Speed (SPD) should be a positive number, greater than or equal to zero, representing speed in knots. Grid X is the X coordinate of the track, in terms of the display. It should be a number in the range of 15-470. Likewise Grid Y is a display coordinate, and ranges from 27-190. Numbers other than these will work, if they are within the range of the pixels of the display. The Update module tests for the coordinates of the track, however, and the track will not move if placed outside the display window. Track Color should be a number between 0-7. On the monochrome systems this will not matter unless it is 0 or 7--on color systems these numbers correspond to the colors listed in the User's Manual.

The F4 key, Modify, will go through the same track parameters that were just discussed for the F3 key. It will first ask which track to modify, and the track number must be input. The system then hooks that track, and goes through each track parameter asking if it is OK. The user should input a "Y" if the parameter is fine, anything else if it is not. If the response is other than "Y" the user will be requested to input the correct value for that parameter, and the hooked track will be modified accordingly.

The Delete key, F5, asks the user to provide a track number, and the track number input will be deleted. Upon the next update of the system it will no longer appear on the screen.

The Halt key, F6, provides a gracious exit from the display simulator system. When it is depressed it restores the special function keys to their ZBASIC settings, clears the screen, and returns the ZBASIC interpreter prompt.

D. MODIFYING THE INITIAL DISPLAY

The initial display, in its entirety, is determined from the DATA module. By modifying the DATA module, or creating a new one, and re-merging it with the system, a new initial display may be created. There is a caution here: if a new (or modified) DATA module is used, the line numbers must match all those of the old module, or the unused old numbers must be deleted, to prevent erroneous assignments.

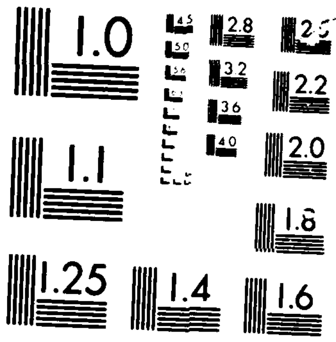
The data should be entered in the order of Figure N.1, within the ranges and for the purposes stated below.

The first five data values relate to the window. The first two of them, XUL and YUL, are the X and Y coordinates of the upper left-hand corner of the display window, respectively. The X value should fall between 0-480, and the Y value between 20-212. The same range restrictions apply to the XLR and YLR values, which are the coordinates for the lower right-hand corner of the window. The color of the box, a number between zero and 7, is given by CWIND.

The window parameters are followed by CONTS, the number of land masses (or special areas) the initial display will contain. The current system limitation is for a maximum of 3, and this number should not be less than zero. If CONTS is zero, the next data value to be read in is XYAX. Otherwise there are CONTS number of entries of the variables specified within the square brackets ([ ]).

AD-A169 053 TACTICAL DISPLAY SIMULATION ON THE H/Z-100(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA K W COOMES MAR 86 2/2

UNCLASSIFIED F/G 9/2 NL



MICROCOP

For each 'continent' there should be a value pair (PTS, LCOL). The PTS is the number of points ((x,y) coordinate pairs) that specify the border of the land area, LCOL is

```
    -    XUL
    -    YUL
    -    XLR
    -    YLR
    -    CWIND
    -    CONTS
  [ -    PTS, LCOL ] - CONTS TIMES
  [ -    X1, Y1, X2, Y2, ... XN, YN ] - CONTS TIMES
  [ -    CENTX, CENTY ] - CONTS TIMES
    -    XYAX
    -    YTOP
    -    YBOTT
    -    YCOL
    -    YXAX
    -    XLEFT
    -    XRITE
    -    XCOL
    -    TRACKS
  [ -    CLASS$, CUS, SPD, TCOLOR, TX, TY ] - TRACKS TIMES
    -    MOVES
```

Figure N.1 Order of Data Entry

the color of that piece of land. After the PTS, LCOL pairs (one pair for each land area to be input) there are CONTS lists of ordered pairs, Xm, Ym, each pair representing a border point of the land mass. The final subscript, N, should match the PTS number for each particular land mass, and XN, YN should match X1, Y1 to ensure that the land mass will be painted properly. After the list of border points is read in, an interior point, CENTX, CENTY, is read in. This should be a point not on the border but within in. This is the point that determines the area of the screen that will be painted in LCOL color.

The following four data values specify the vertical grid parameters, and the four after that the horizontal grid. The XYAX value should fall somewhere between the XUL and XLR values read in earlier. It is the horizontal, or X, coordinate of the Y axis. YTOP and YBOTTOM are the top and bottom of the Y-axis, and should match YUL and YLR respectively, if the grid is to be from the top of the window to the bottom. The grid's color is determined by YCOL, which should be between 0-7.

The vertical grid parameters are followed by those for the horizontal grid, and they are of the same form. The first, YXAX, is the vertical location of the horizontal grid line, and should be between YUL and YLR. The XLEFT and XRITE specify the ends of the horizontal grid, and should match XUL and XLR for a full-window grid. The grid color is independent of the vertical grid colo. and is specified by a number 0-7 for XCOL.

If the initial display is to have any test tracks prior to user input the number of them is read in through the parameter TRACKS. This number should have a value between zero and ten, the system currently being limited to ten tracks. If TRACKS is zero the next data value is MOVES. Otherwise, the data following TRACKS is sets of parameters for the initial tracks.

CLASS$ is the classification of each test track, and should be a character string surrounded by quotes, no longer than nine characters (excluding the quotation marks). The currently recognized classifications are "HOSTILE", "HOST SURF", "UNK AIR", "REF PNT", "FIGHTER", "SURVEILL" and "UNKNOWN". Any classification other than these will result in a symb 1 which consists only of a speed leader for the track.

CUS and SPD are the course and speed of the track. They should be positive or zero. The course is in degrees true (0-360) and the speed is in knots(0-?). TCOLOR is the track color, and should again be a 0-7 number.

The TX and TY are the grid coordinates of the track's initial position. They are pixel coordinates on the screen. TX should be between XUL and XLR, T b twe n YUL and YLR. If they are not one of two things will happen. If they are outside the range of the window but within the range of the screen they will be drawn on the screen in the specified position, and not updated. If they are outside the range of the screen (0-639 for x, 0-224 for y) an error will result, and the system will be exited.

The value of MOVES should be zero if TRACKS is zero. It represents the number of automatic times the system will update the tracks if there is no user input. Actually, this is a hold-over from an earlier version of the system. It may be used if the system is modified--otherwise it will be ignored.

E. UNDERSTANDING THE CODE

Following is a line by line explanation of the code. The subsections correspond to the subroutines that make up the display simulator system. Each subsection is titled according to its subroutine. The code may be examined by following, in order, Appendix A, which is a listing of the assembled subroutines, or by following the appropriate Appendix for each subroutine.

1. Header

We begin with a header, identifying the program and clearing the screen. These statements are lines 10-90.

2. Init

The next section of code, "INITIALIZATION AND TABLES" (lines 100-960) performs several housekeeping chores to set up the prototype. Line 130 sets the array subscript lower bound, and lines 150-170 allocate memory for the necessary arrays. The symbol and speed leader tables (SYM$ and LDR$) are initialized in lines 180-440.

The variable TRACKS is initialized to zero. Later in the program it is read from a DATA statement, to determine how many tracks the system starts with prior to user input. Whenever a track is added, TRACKS is incremented. If it exceeds ten, the dimension of the parts of the TRACK record (see Figure 3.1), a subscript out of range error will result. The prototype does no 'garbage collection', as such, and flags inactive tracks with a value of zero in the ACTIVE field.

Line 510 initializes the elements of the PTS array to one. This is necessary because of the lack of dynamic memory allocation in ZBASIC. The elements of the PTS array are used in the Land module to dimension arrays, and must be greater than or equal to one.

The special function keys are initialized in lines 530-600. This prototype was developed with a ZBASIC interpreter, ZBASIC, under Z-DOS version 1.25. In that environment the special function keys are pre-set to provide ZBASIC commands. These lines re-set them to generate their normal escape sequences when depressed.

The HK$ and ACTIVE fields of each track are initialized in lines 620-670. This prototype was developed for color and/or monochrome use. The current Zenith monitors at NPS are monochrome. For that reason we elected to indicate a hooked track by enlarging its symbol. The HK$ field will always be drawn as part of the symbol. If the track is not hooked it will be scale zero ("S0"), for normal size. For hooked tracks it is changed to "S8", for double size. The ACTIVE field is primarily used to determine which tracks are active. Reference points require special treatment in this prototype, for efficiency. A more detailed explanation is with the Update module. A value of

2 in the ACTIVE field indicates that the track is a reference point.

The final chore performed by the Init module is the display of the function keys menu. Lines 690-960 provide the user with reverse video labels of the active function keys, and normal video display of their purposes on line 25 of the display. This places the menu close to the keys involved and out of the main display area.

3. Harness

The test harness, or Harness module, follows in lines 1000-4999. It grew as the prototype was developed. The final line, 4999, which is an END statement, is no longer necessary, but was prior to the installation of user interaction as a feature. During development and testing all inputs were through program lines and DATA statements. This may be a good point at which to mention that there are some unnecessary lines remaining, many unused line numbers, and some sections of code where line numbers are too close together.

The presence of unnecessary lines does not adversely affect the performance of the prototype. Some of them are left in to allow follow-on researchers to see some history of the thought process and development procedures used before. Most of them are present to allow for spacing and readability of the code, and are left in for those reasons.

In most cases the line numbers are spaced by tens. This allows for the insertion of several lines wherever necessary during the ongoing development of the system. In some cases they are closer together, demonstrating the prior development and debugging. There are wide gaps in some sections of code, illustrating the modular development process. It is particularly important when writing files which will be merged to attempt to assign line numbers which will not risk duplication.

The Harness module requires little explanation. It reads DATA statements to obtain parameters, calls on subroutines to make use of the parameters to draw static portions of the display, and transfers control to the Tracking module in line 1280.

4. Window

The Window module, lines 5000-5160, is also self-explanatory. It is written in general terms, and may be used to draw any size box, anywhere on the screen, in any color and for any purpose.

5. Axes

Most of the modules are written to be useful elsewhere. The Axes module is no exception. We could have made use of the previous module, Window, and re-defined what have been labelled the window parameters, since Axes also draws boxes. This is just one example of extra code being written, and trickiness avoided, for clarity and readability. This feature, abundant code and prolific variable creation rather than re-using the same variable names for different purposes, also enhances maintainability.

Lines 5320-5330 ensure that aspect ratio is maintained when the two axes are scaled. Line 5365 draws a box one pixel wider on each side than the vertical axis line of the window's color. This enables the axis to cross any color land mass without getting lost. Line 5405 does the same thing for the horizontal axis.

6. Update

The Update module is, in many ways, the heart of the system. It is the module that re-positions the track symbols periodically, draws and erases them, and checks to see if they fall within the window limits.

The first thing Update does is look up all array variables that are referenced frequently in the module. This saves time when each variable is used. It is much faster for the interpreter to look up the copy in the local simple variable than to compute the address from an array index. Lines 6150-6210 do the copying of array variables into local simple variables.

Line 6230 samples a background point at the current symbol position. A common method of erasing in computer graphics, and the one employed here, is to re-draw the symbol in the color of its background. Based on the color of the local background one of eight subroutines determines the proper color for the string COL$. The reason the statement using UPGND + 1 is because the colors are from 0-7, but the ON <exp> GOSUB statement requires a number equal to or greater than one to branch.

The string WANT$ is then composed of the color and the symbol, ALSO$ is composed of the color and the speed leader (both of these being the color of the background in this case, to perform an erasure), then each string is drawn at the current symbol position.

In line 6260 the symbol is located at its current position. Line 6270 draws the symbol, line 6280 relocates at symbol center and line 6290 draws the speed leader. The scale is returned to normal in line 6295.

If the symbol is inactive (ACTIVE = 0) this is all that is required and line 6305 directs program flow to the RETURN statement. For active symbols lines 6310-6320 update the position of symbol center and program flow continues.

Line 6340 samples the background at the updated symbol position. If there is no conflict logic similar to that just completed for the erasure, using COLUP (the current symbol color) rather than UPGND (background color) draws the re-located symbol in lines 6375-6425. If there is a conflict line 6370 makes the symbol white for dark backgrounds and black for light backgrounds.

Lines 6440-6450 store the updated symbol position in the TX and TY fields of the track record. Line 6490 returns to the calling routine. Each of the lines 6520-6590 contains two statements, constituting an entire subroutine. These are the subroutines called upon to set COL$, which is used to determine the color the symbol will be drawn in.

7. Move

The Move module determines how many pixels in each direction a symbol will move when it is updated and which speed leader will be assigned to a track, based on track course and speed. Lines 7160-7350 branch to the appropriate line number based on the course, dividing the full circle of directions (courses 0-360 degrees true) into 20 zones. Lines 7400-7590 are the lines branched to, only one of which will be executed. They make the assignment of incremental values of change in the x and y direction and assign one of the eight speed leaders from the speed leader table, then branch to 7600. Together these lines (7160-7590) form one giant case statement.

Line 7640 branches to 7690 if the target is not a slow speed track. For slow speed tracks that do have motion line 7641 branches to 7650. Lines 7642-7645 handle tracks with no motion, ensuring no incremental movement and no speed leader. For slow speed tracks that do move lines 7650-7680 reduce the incremental movement and scale the speed leader down.

Medium speed tracks, treated as the norm, are handled by the branch from line 7690-7770, 7770 being the

RETURN. Lines 7700-7720 handle high speed tracks by increasing the incremental movement and scaling up the speed leader.

8. Land

This is the module that draws the land masses. It currently provides for only three land masses. Because ZBASIC has no dynamic memory allocation the DIM (dimension) statements cannot be executed more than once or an error results. For more land masses to be introduced to the system they must be described by the same number (or fewer) points than one of the first three and one of the three land arrays re-used, or more land arrays must be added to this module in the DIM statement(s). The latter solution is the easiest to implement, and will be the easiest for others to follow later on. That is why it was chosen here, rather than simply dimensioning one array large enough to handle any probable number of points.

Line 8075 guards against execution if there are no land masses to draw. If there are land masses the variable CONTS contains the number, and is used as an index in the loop of lines 8090-8110, which reads in the numbers of points of each of the masses and their color. Line 8125 sets aside memory for the arrays, as mentioned earlier.

This module has been designed for zero or three. The intention was to make the logic clear, and also to provide loops for all three so that only data statements would need to be changed if any number 0-3 were input. That is why there are three loops, lines 8130-8150, 8170-8190 and 8210-8230 which read in the points describing the land masses. If there are fewer than three at least one dummy point must be in the data statements for each of the unused arrays.

Lines 8250-8280 draw the first land mass (if there were none to draw line 8075 would prevent the branch to here). Line 8300 reads the coordinates of an interior point for the first land mass, and line 8320 paints it.

If there is only one land mass line 8340 executes the RETURN. Otherwise lines 8360-8430 perform the same functions for land mass two as 8250-8320 did for land mass one. Lines 8450-8540 perform a similar test and conditional execution of land mass three function. If there were three land masses line 8560 executes the RETURN, otherwise it would have already been executed.

9. Data

We have already gone over the data format. The Data module follows it, interspersing the data with comments for clarity. It is recommended that users follow the same procedure. It makes corrective maintenance and enhancement much easier. Another design philosophy embodied here and encouraged is the matching of one data statement to one read statement. Code could be reduced by combining, for example, the data in lines 10080 and 10120-10140. Instead we opted to keep the data on separate lines matching read statements in the program. This procedure reduces debugging time (it's easy to create mis-matched data/read pairs) and makes the module more readable.

10. Tracking

This module performs the automatic system updating of tracks and monitoring for user input. It is the driver program, in essence, whereas the Harness module is the initialization driver.

Line 11080 determines if there are any initial tracks in the system. If not line 11100 branches to 11200, skipping lines 11110-11140, which read in the initial tracks if there are any.

For initial tracks lines 11165-11175 calculate the appropriate incremental movements and assign speed leaders, through the use of the Move module.

Line 11200 initializes the DO$ variable to an empty string. DO$ is used to tell the system what to do if there is user input.

Lines 11220-11260 drive the system until there is user input. All tracks are automatically updated in lines 11225-11235, the user is given a chance for input during a pause between updates in lines 11240-11255. The constant 2000 in line 11240 determines the length of time between updates when there is no user input. If it is reduced, shortening the delay, a reasonable minimum would probably be

500. If the delay is too short the user reaction may be too slow to input a selection, resulting in at least one more update than desired. The motions of the tracks may also appear too jerky and/or rapid if there is not sufficient delay between updates. When the system detects that the user has depressed a key the program branches to 11280.

Line 11280 reverts to the initialization in line 11200 and repeats the process if the key struck was not a special function key, by checking for the first character to be an "ESCape" (CHR$(27). If a special function key was struck lines 11310 to 11360 branch to the appropriate routine to handle the request. Line 11380 reverts to initialization of DO$ and repeats the update/delay process if the key was not a pre-defined function key, or upon completion of the service of the request.

11. Keys

This module defines the special function key routines. Keys F1-F6 are currently defined, more could easily be added. They should be initialized in the Init module, branches to their routines provided for in the Tracking module, and their routines defined in this one.

Lines 12000-12085 handle the request for a halt. The screen is cleared and the function keys are restored before the END statement is executed. The RETURN statement is not really necessary. Actually only the END statement is needed here, but it is good programming practice to clear the screen when finishing a graphics routine, as well as restoring functions keys defined. The RETURN statement is included for similar reasons, since this is a subroutine.

Lines 12100-12190 perform the suspend/continue function, by simply waiting for another keyboard input to continue.

The hook track function is in lines 12200-12420. The locate statements ensure that messages and input requests appear at the top of the screen. Lines 12250-12259 check to see if there is already a track hooked, and unhook one if one is hooked.

Line 12270 requests the input of the track to hook, lines 12275-12276 clear the request from the screen when the requested input has been provided. The track input as the one to hook is hooked in lines 12282-12290. After it has been hooked and its symbol enlarged (the way a hooked track is displayed) lines 12310-12380 display its parameters to the right of the display window.

Lines 12500-12750 perform the enter new track function. First the number of tracks is incremented in lines 12530-12540. Then lines 12560-12690 request for each of the user inputs and clear the requests when the input has been made (lines 12571-12574 ensure that the classification will be exactly nine characters in length for symbol assignment).

After track parameter input lines 12705-12717 perform necessary calculations and matching to provide the rest of the track parameters and display the new track.

12. Match

This routine simply matches the CLASS$ of a track (classification) to its appropriate symbol. Only the line which finds a string match will be executed, and the RETURN. If there is no match no symbol will be assigned, and only the speed leader will be displayed. This distinguishes unidentified tracks (which may be unconfirmed, bogus, or whatever) from tracks known to exist but unclassified (UNKNOWN).

F. CROSS-REFERENCE

The following cross-reference of variables is provided as an aid to modifying the code in further development. It is for the version of the code listed in Appendix A, the first and un-numbered version.

NAME	PURPOSE	LOCATIONS
CLASS$()	classification of track	INIT, TRACKING, KEYS, MATCH, DATA
CUS()	track's course	INIT, MOVE, TRACKING, KEYS, DATA

NAME	PURPOSE	LOCATIONS
SPD()	track's speed	INIT, MOVE, TRACKING, KEYS, DATA
TCOLOR()	track color	INIT, UPDATE, TRACKING, KEYS, DATA
TX()	track x coord	INIT, UPDATE, TRACKING, KEYS, DATA
TY()	track y coord	INIT, UPDATE, TRACKING, KEYS, DATA
XINC()	horizontal movement	INIT, UPDATE, TRACKING, KEYS, MOVE, DATA
YINC()	vertical movement	INIT, UPDATE, TRACKING, KEYS, MOVE, DATA
T$()	track symbol	INIT, UPDATE, MATCH
L$()	track speed leader	INIT, UPDATE, MOVE
SYM$()	generic symbol	INIT, MATCH
LDR$()	generic speed leader	INIT, MOVE
PTS()	number of points defining land mass	INIT, LAND, DATA
LCOL()	land color	INIT, LAND, DATA
HK$()	scale to draw track	INIT, UPDATE, KEYS
ACTIVE()	state of track	INIT, UPDATE, KEYS
TRACKS	number of tracks	INIT, TRACKING, DATA, KEYS

NAME	PURPOSE	LOCATIONS
I	generic loop counter	INIT, LAND, TRACKING, KEYS
XUL	x coordinate upper left-hand corner of window	HARNESS, WINDOW, DATA
YUL	y coordinate upper left-hand corner of window	HARNESS, WINDOW, DATA
XLR	x coordinate lower right-hand corner of window	HARNESS, WINDOW, DATA
YLR	y coordinate lower right-hand corner of window	HARNESS, WINDOW, DATA
CWIND	window color	HARNESS, WINDOW, AXES, DATA
CONTS	# of land masses	HARNESS, LAND, DATA
XYAX	x coordinate Y-axis	HARNESS, AXES, DATA
YTOP	y coordinate Y-axis top	HARNESS, AXES, DATA
YBOTT	y coordinate Y-axis bottom	HARNESS, AXES, DATA
YCOL	Y-axis color	HARNESS, AXES, DATA
YXAX	y coordinate X-axis	HARNESS, AXES, DATA
XLEFT	x coordinate X-axis left	HARNESS, AXES, DATA
XRITE	x coordinate X-axis right	HARNESS, AXES, DATA
XCOL	X-axis color	HARNESS, AXES, DATA

NAME	PURPOSE	LOCATIONS
HSCALE	horizontal scale	AXES
VSCALE	vertical scale	AXES
H	loop counter	AXES
V	loop counter	AXES
UPDX	x coordinate symbol center	UPDATE
UPDY	y coordinate symbol center	UPDATE
UPDT$	symbol	UPDATE
UPDL$	speed leader	UPDATE
HORZUP	horizontal increment	UPDATE
VERTUP	vertical increment	UPDATE
COLUP	symbol color	UPDATE
UPGND	pixel color	UPDATE
COL$	color string	UPDATE
WANT$	symbol string	UPDATE
ALSO$	speed leader string	UPDATE
UPD	loop counter	UPDATE, KEYS, TRACKING
LAND1(,)	land point	LAND, DATA
LAND2(,)	land point	LAND, DATA
LAND3(,)	land point	LAND, DATA
ISLE	loop counter	LAND
CENTX	x coordinate land point	LAND, DATA

NAME	PURPOSE	LOCATIONS
CENTY	y coordinate land point	LAND, DATA
MOVE	loop counter	TRACKING, DATA, MOVE
DO$	user input	TRACKING
DO2$	user input	TRACKING
GO$	user input	KEYS
HOOK	hooked track indicator	KEYS
SIZECL	length of CLASS$	KEYS
A$	user input	KEYS
DEL	delete track indicator	KEYS

APPENDIX O: LISTING OF TEST_10.ASM

```
TITLE -- TEST OF FILLING SCREEN WITH SYMBOL
;
P_STACK SEGMENT STACK
        DW      100H DUP (0FH)
STK_TP  LABEL   WORD
P_STACK ENDS
;
P_DATA  SEGMENT
SYMBOL  DB      00000000B, 00000000B, 01111110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01100110B
        DB      00000000B, 00000000B, 01111110B
SHAPE   DB      01111110B
        DB      01100110B
        DB      01100110B
        DB      01100110B
        DB      01100110B
        DB      01100110B
        DB      01100110B
        DB      01100110B
        DB      01111110B
TIME    DB      '00:00:00.00', 13, 10, '$'
TEN     DB      10
DATA1   DB      16 DUP (0BH)
DATA2   DW      8 DUP (0B0H)
P_DATA  ENDS
;
        INCLUDE PARM.DEF
        INCLUDE DOS_FUNC.MAC
;
P_CODE  SEGMENT
        ASSUME  CS:P_CODE, DS:P_DATA, SS:P_STACK
;
START:  MOV     AX, P_STACK             ;set up SS through AX
        MOV     SS, AX
        MOV     SP, OFFSET STK_TP       ;set up SP
;
        PUSH    DS                      ;save for far return
        SUB     AX, AX                  ;ensure 0 offset for far rtn
        PUSH    AX
;
```

```
        MOV     AX, P_DATA                  ;set up DS through AX
        MOV     DS, AX
;
        SUB     AX, AX                      ;zero AX, to save IO port
        IN      AL, IO_PORT                 ;status
        PUSH    AX                          ;save status
;
        CALL    CLS                         ;clear the screen
;
;
        PUSH    CX                          ;save registers, then
        PUSH    DX                          ;call the timer routine
        CALL    TIMER
        POP     DX                          ;restore the registers
        POP     CX
;
        MOV     AL, 78H                     ;prepare IO port
        OUT     IO_PORT, AL
        MOV     SI, 0                       ;set up symbol part counter
;
        SUB     BP, BP                      ;zero BP, for offset
        SUB     AX, AX                      ;zero AX, for symbol
        MOV     AL, SYMBOL[SI]              ;top scan-line of symbol
        MOV     BH, L_COUNT                 ;loop counter for loop L
        MOV     BP, LINE                    ;start BP negative, to
        NEG     BP                          ;bring it to 0 at beginning
                                            ;of loop
;
;
;outer loop (L) -- for L=1 to L_COUNT do
;                  begin
;                       fill line (L) with symbol
;                  end
;
LOOP_L: ADD     BP, LINE                    ;move offset to next line
        MOV     CL, I_COUNT                 ;set loop I counter
;
;second loop (I) -- for I= 1 to I_COUNT do
;                   begin
;                       write (symbol) @ line L, position I
;                   end
;
LOOP_I: MOV     DI, 0                       ;reset scanline counter
        MOV     AL, SYMBOL[SI]              ;get top scan-line
        MOV     AH, SHAPE[DI]               ;symbol shape, for clearing
        NOT     AH                          ;space with inverse
;
;third loop (J) -- for J=1 to J_COUNT do
;                  begin
;                       write(symbol, scanline(J)
;                  end
```

```
;
LOOP_J: PUSH    AX                      ;save symbol, use AX to
        MOV     AX, B_PLANE             ;reset ES to blue plane
        MOV     ES, AX
        POP     AX                      ;restore symbol to AL
        MOV     BL, K_COUNT             ;set counter for loop K
;
;inner loop (K) -- for K=1 to K_COUNT do
;                   begin
;                        negate symbol
;                        AND symbol with plane (K)
;                        negate symbol
;                        OR symbol with plane (K)
;                   end
;
LOOP_K: AND     ES:[BP], AH             ;AND shape inv. with plane (K)
        OR      ES:[BP], AL             ;OR symbol with plane (K)
        DEC     BL                      ;count loop K iterations
        CMP     BL, 0                   ;loop K done?
        JLE     K_DONE                  ;Yes, go to end of loop K
        PUSH    AX                      ;No, save symbol, use AX
        MOV     AX, ES                  ;to modify ES for next
        ADD     AX, PLANE               ;color plane
        MOV     ES, AX
        POP     AX                      ;restore symbol in AL
        INC     SI                      ;move to next symbol part
        MOV     AL, SYMBOL[SI]          ;get next symbol part
        JMP     LOOP_K                  ;repeat loop K
;
K_DONE: INC     DI                      ;count loop J iterations
        CMP     DI, J_COUNT             ;loop J done?
        JGE     J_DONE                  ;Yes, go to end of loop J
        ADD     BP, VERT                ;No, move to next scan-line
        INC     SI                      ;get next symbol part
        MOV     AL, SYMBOL[SI]          ;next scan-line of symbol
        MOV     AH, SHAPE[DI]           ;next scan-line of shape
        NOT     AH
        JMP     LOOP_J                  ;repeat loop J
;
J_DONE: DEC     CL                      ;count loop I iterations
        CMP     CL, 0                   ;loop I done?
        JLE     I_DONE                  ;Yes, go to end of loop I
        MOV     SI, 0                   ;back to first part of symbol
        SUB     BP, LINE                ;No, move to start of symbol
                                        ;just done, then
        INC     BP                      ;move to right one byte
        JMP     LOOP_I                  ;repeat loop I
;
I_DONE: DEC     BH                      ;count loop L iterations
        CMP     BH, 0                   ;loop L done?
        JLE     L_DONE                  ;Yes, go to end of loop L
        SUB     BP, X_LINE              ;No, move to start of
                                        ;last character, then
        MOV     SI, 0                   ;reset symbol part counter
        JMP     LOOP_L                  ;repeat loop L
;
L_DONE: PUSH    CX                      ;save registers, and
        PUSH    DX                      ;call timer
        CALL    TIMER
        POP     DX                      ;restore registers
        POP     CX
        POP     AX                      ;restore IO port status
        OUT     IO_PORT, AL
;
;
EXIT    PROC    FAR
        RET
EXIT    ENDP
;
        INCLUDE CLS.SUB
        INCLUDE TIMER.SUB
        INCLUDE BOX.SUB
;
P_CODE  ENDS
        END     START
```

APPENDIX P: LISTING OF TEST_8.ASM

```
TITLE -- EXPERIMENT 8 -- TEST BOX SUBROUTINE
;
P_STK   SEGMENT STACK
        DW      100H DUP (00H)
STK_TP  LABEL   WORD
P_STK   ENDS
;
P_DATA  SEGMENT
TIME    DB      '00:00:00.00', 13, 10, '$'
TEN     DB      10
        DB      20H DUP (?)
P_DATA  ENDS
;
        INCLUDE PARM.DEF
        INCLUDE DOS_FUNC.MAC
;
P_CODE  SEGMENT
        ASSUME  CS:P_CODE, DS:P_DATA, SS:P_STK
;
START:  MOV     AX, P_STK
        MOV     SS, AX
        MOV     SP, OFFSET STK_TP
;
        PUSH    DS
        SUB     AX, AX
        PUSH    AX
;
        MOV     AX, P_DATA
        MOV     DS, AX
;
        SUB     AX, AX
        IN      AL, IO_PORT                 ;save IO port status
        PUSH    AX
;
        CALL    CLS                         ;clear the screen
;
        PUSH    CX
        PUSH    DX
        CALL    TIMER
        POP     DX
        POP     CX
;
        MOV     BH, Y_START                 ;vertical start line
        MOV     AH, Y_STOP                  ;vertical stop line
        MOV     BL, X_START                 ;horizontal start column
        MOV     AL, X_STOP                  ;horizontal stop column
        MOV     CL, COLOR                 ;COLOR box
        MOV     CH, OFFH                  ;solid pixel line
;
        CALL    BOX_F                     ;draw box
;
        MOV     BH, Y_START
        MOV     BL, GRIDX_START           ;draw Y-axis
        MOV     AH, Y_STOP
        MOV     AL, GRIDX_STOP
        MOV     CL, G_COLOR
        MOV     CH, OFOH         ;to blank out space
        CALL    BOX_F
;
        CMP     CL, 0                     ;if grid black,
        JE      X_AXIS                    ;it is already drawn
        MOV     CH, OFOH                  ;half-byte width
        CALL    BOX_F                     ;actually draw Y-axis
;
X_AXIS: MOV     BH, GRIDY_START           ;draw X-axis
        MOV     BL, X_START
        MOV     AH, GRIDY_STOP
        MOV     AL, X_STOP
        MOV     CL, G_COLOR
        MOV     CH, 00H                   ;to blank out space
        CALL    BOX_F
;
        CMP     CL, 0                     ;if grid black,
        JE      OVER                      ;it is already drawn
        MOV     CH, OFFH                  ;solid pixel line
        CALL    BOX_F                     ;actually draw X-axis
;
OVER:   PUSH    CX
        PUSH    DX
        CALL    TIMER
        POP     DX
        POP     CX
        POP     AX                        ;restore IO control port
        OUT     IO_PORT, AL
;
EXIT    PROC    FAR
        RET
EXIT    ENDP
;
        INCLUDE BOX.SUB
        INCLUDE CLS.SUB
        INCLUDE TIMER.SUB
;
P_CODE  ENDS
        END     START
```

APPENDIX Q: LISTING OF TIMER.SUB

```
; this is a subroutine which gets and converts the time,
; then displays it
;
; INPUTS: none
;
; OUTPUTS: the time is displayed on the screen
;
; FLAGS: none
;
; REGISTERS: none
;
TIMER: GET_TIME
       CONVERT CH, TEN, TIME
       CONVERT CL, TEN, TIME[3]
       CONVERT DH, TEN, TIME[6]
       CONVERT DL, TEN, TIME[9]
       DISPLAY TIME
       RET
;
; end of timer subroutine
;
```

APPENDIX R: LISTING OF BOX.SUB

```
; wa-ohaeqls eh topx p whiqt§uhihost osueplraipo posp hl ebs wuossl
;
COMMENT *
 INPUT:  BH = vertical line to start box on (0 - 23)
          BL = horizontal column to start box on (0 - 79)
          AH = vertical line to stop box on (1 - 24)
          AL = horizontal column to stop box on (1 - 80)

  NOTE:  if AX < BX an error will result
         IO control port needs to be saved prior to calling this
         subroutine

 OUTPUT:  generates a colored box on the screen

 FLAGS:   none returned

 REGISTERS: used as noted above, preserved

*
;
BOX_F:  PUSH    AX                      ;save all registers used
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    DI
        PUSH    SI
        PUSH    BP
        PUSH    DS
;
        SUB     DX, DX                  ;zero DX
        MOV     DH, BH                  ;get start line
        SHL     DX, 1                   ;convert to necessary offset
        SHL     DX, 1
        SHL     DX, 1
        MOV     DL, BL                  ;rest of start offset
        MOV     SI, DX                  ;starting offset
;
        SUB     AL, BL                  ;how many bytes across
        MOV     DL, AL                  ;DL <--- horizontal count
;
        SUB     AH, BH                  ;how many lines down
        MOV     DH, AH                  ;DH <--- vertical count
;
        CMP     CL, 4                   ;does color include green?
        JGE     GREEN                   ;Yes, go to green
        CMP     CL, 2                   ;No, does it include red?
        JGE     RED                     ;Yes, go to red
        JCXZ    BLACK                   ;if CL=0, handle black
        MOV     AX, 0C000H              ;No, handle blue
        MOV     DS, AX
        MOV     AL, 38H
        JMP     PREP
;
RED:    MOV     AX, 0D000H              ;handle red, magenta
        MOV     DS, AX
        SUB     CL, 2                   ;is color red?
        JNZ     MAGNTA                  ;No, go to magenta
        MOV     AL, 68H
        JMP     PREP
;
MAGNTA: MOV     AL, 28H
        JMP     PREP
;
GREEN:  MOV     AX, 0E000H              ;handle green, cyan, yellow,
        MOV     DS, AX                  ;and white
        SUB     CL, 4                   ;is color green?
        JNZ     CYAN                    ;No, check for cyan
        MOV     AL, 58H                 ;Yes, handle green
        JMP     PREP
;
CYAN:   CMP     CL, 1                   ;is color cyan?
        JNE     YELLOW                  ;No, try yellow
        MOV     AL, 18H                 ;Yes, handle cyan
        JMP     PREP
;
YELLOW: SUB     CL, 2                   ;is color yellow?
        JNZ     WHITE                   ;No, must be white
        MOV     AL, 48H                 ;Yes, handle yellow
        JMP     PREP
;
WHITE:  MOV     AX, 0C000H
        MOV     DS, AX
        MOV     AL, 08H
        JMP     PREP
;
BLACK:  MOV     AX, 0C000H
        MOV     DS, AX
        MOV     AL, 78H
;
PREP:   OUT     IO_PORT, AL
        MOV     AL, CH
        SUB     CX, CX
        MOV     CL, DL                  ;horizontal count
        XOR     BP, BP
        MOV     DL, DH
        XOR     DH, DH
        MOV     BP, DX                  ;vertical count
        MOV     DI, 128                 ;line spacing
        PUSH    SI
        PUSH    CX
;
LUPE:   MOV     [SI], AL
        MOV     ES:[SI], AL
        INC     SI
        LOOP    LUPE
;
        DEC     BP
        JZ      FINISH
        POP     CX
        POP     SI
        ADD     SI, DI
        PUSH    SI
        PUSH    CX
        JMP     LUPE
;
FINISH: POP     CX
        POP     SI
        POP     DS
        POP     BP
        POP     SI
        POP     DI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
;
        RET
;
; end of subroutine to draw box
;
```

APPENDIX S: LISTING OF CLS.SUB

```
; subroutine to clear the screen, ZENITH
;
; INPUT: none
;
; OUTPUT: none
;
; FLAGS: none
;
; REGISTERS: none
;
CLS:    PUSH    AX              ;save register used
;
        IN      AL, 0D8H        ;prepare to save IO control
        PUSH    AX              ;port status, and save it
;
        MOV     AL, 0FH         ;blank the screen
        OUT     0D8H, AL
;
        IN      AL, 0DBH
        AND     AL, 0F7H        ;SET = 0
        OUT     0DBH, AL
;
        IN      AL, 0D9H
        AND     AL, 0F7H        ;activate CLRSCRN
        OUT     0D9H, AL
;
        MOV     CX, 6680        ;wait
DELA:   NOP
        LOOP    DELA
;
        IN      AL, 0D9H
        OR      AL, 08H         ;de-activate CLRSCRN
        OUT     0D9H, AL
;
        POP     AX
        OUT     0D8H, AL        ;restore IO control port
;
        POP     AX              ;restore register
;
        RET
;
; end of clear screen routine
;
```

APPENDIX T: LISTING OF DOS_FUNC.MAC

```
; this is a file of MS-DOS 2.0 function macros
;
; get_time is a macro which puts the time in CX and DX
;
GET_TIME  MACRO
        MOV     AH, 2CH
        INT     21H
        ENDM
;
; convert is a macro which converts the value parameter into
; a number in the base parameter system, and puts the
; converted value in the destination parameter location
;
CONVERT MACRO   VALUE, BASE, DESTINATION
        LOCAL   TABLE, START
        JMP     START
TABLE   DB      "0123456789ABCDEF"
START:  MOV     AL, VALUE
        XOR     AH, AH
        XOR     BX, BX
        DIV     BASE
        MOV     BL, AL
        MOV     AL, CS:TABLE[BX]
        MOV     DESTINATION, AL
        MOV     BL, AH
        MOV     AL, CS:TABLE[BX]
        MOV     DESTINATION[1], AL
        ENDM
;
; display is a macro which displays a string located in
; memory at the location passed in the parameter string,
; and the string must end with the ASCII code for '$', 24H.
;
DISPLAY MACRO   STRING
        MOV     DX, OFFSET STRING
        MOV     AH, 09H
        INT     21H
        ENDM
;
;
```

APPENDIX U: LISTING OF PARM.DEF

```
;  file of parameter definitions
;
; i/o port address
IO_PORT EQU     0D8H
;
;horizontal(X), vertical(Y) start/stop corners of a box
;
X_START EQU     4
X_STOP  EQU     58
Y_START EQU     6
Y_STOP  EQU     243
;
;color of the box
;
COLOR   EQU     3
;
;horizontal(X), vertical(Y) start/stop corners of boxes that
;will serve as grid lines
;
GRIDX_START     EQU     18
GRIDX_STOP      EQU     19
GRIDY_START     EQU     17
GRIDY_STOP      EQU     19
;
;color of the grid lines
;
G_COLOR         EQU     4
;
;constants for loop counts and symbol location shifting
;
LINE    EQU     1024            ;required to shift one vertical line
                                ;on the screen
;
;length of line in bytes
;
X_LINE  EQU     79
B_PLANE EQU     0C000H          ;start address of blue plane
I_COUNT EQU     80
J_COUNT EQU     9               ;counter for loop J
K_COUNT EQU     3               ;counter for loop K
L_COUNT EQU     20              ;counter for loop I
PLANE   EQU     1000H           ;address difference between color
                                ;planes
HORZ    EQU     1               ;horizontal space shift
VERT    EQU     128             ;vertical space shift
;
```

```
;size of one character, nine scan-lines
;
SIZE    EQU     1152
HITE    EQU     1280
```

APPENDIX V: USER'S MANUAL FOR ASSEMBLY PROGRAMS

A. HOW TO USE THE MACRO-86 PROGRAMS

The Macro-86 assembly language programs included in this thesis have been assembled and linked. To run any of the tests simply type the filename at the system prompt. The file READ.ME on each distribution disk describes what each file is named and what it does.

B. UNDERSTANDING THE CODE

The internal documentation explains the Macro-86 code step by step. We will not indulge in a line by line explanation as Appendix N does for the BASIC code. This Appendix will discuss some of the reasons behind the code in the test file, Appendices O and P.

We set up our own segments for stack, data, and code because we are using the EXE format rather than the COM format. The EXE format is necessary to provide direct control of the video random access memory (VRAM) addresses.

The first entries in the data segment are the bytes which define the test symbol. For these tests we did not establish complete symbol tables and perform table look-ups, as we were interested in establishing simple timing bases for efficiency comparisons with the BASIC prototype. SYMBOL is defined in binary form to allow visualisation of its constituent parts. The first byte on each line of SYMBOL defines its blue plane, the second the red and the third the green planes. This initialization may alter the shape or color of the symbol. The symbol may even be constructed of multi-colored parts. This would not matter in a monochrome system, of course. Since the microcomputer laboratories currently operate only monochrome monitors, this test symbol is defined in the green plane only.

The shape of the symbol is next defined separately. This is necessary because a non-white symbol possesses a shape which differs in each color plane. The test symbol is a prime example -- its shape in the green plane matches exactly that of SHAPE, but it has no shape in the red and blue planes.

In order to maintain the purity of the background and symbol colors, a space must be cleared for the symbol in all three color planes (on a color system, or a monochrome system with the color option installed) to all zeroes. The way in which color is generated (superimposing three pixels, one of each color plane) drives this necessity.

Figure V.1 illustrates the problem. In Figure V.1 the background color planes are represented in part (a), the test symbol in part (b). Parts (c), (d) and (e) of the Figure exhibit the results of an OR, AND and XOR operation, respectively, between the color planes of the symbol and those of the background. None of the results produces the correct result of background and symbol, shown in part (f).

COLOR PLANE								
BLUE			RED			GREEN		
1	1	1	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1

(a) Background

0	1	1	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0

(b) Symbol

0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

(c) Background AND Symbol

1	1	1	0	1	1	1	1	1
1	1	1	0	0	0	1	1	1

(d) Background OR Symbol

1	0	0	0	1	1	1	1	1
1	1	1	0	0	0	1	1	1

(e) Background XOR Symbol

1	1	1	0	1	1	1	0	0
1	1	1	0	0	0	1	1	1

(f) Desired Results

Figure V.1 Results of Operating with Symbol Directly

Figure V.2 illustrates the solution. The background planes are in part (a), as before. Part (b) is the shape of the symbol inverted and stored in each plane, creating a mask which is ANDed with the background to produce part (c). This mask is created by the SHAPE stored in data. By performing the AND operation of the background and the inverse of the symbol shape, a screen area of background with black where the symbol will appear is created (c). Part (d) is the actual symbol, which will appear differently in each color plane unless it is all white. When (c) and (d) are OR'd together the correct background/symbol colors appear in part (e), matching part (f).

The PARM.DEF file which is included next is a file of defined constants. During the development process the collection of all constants in one separate file allowed simpler experimentation and debugging.

Another file, DOS_FUNC.MAC, is included after PARM.DEF. This is a file of Microsoft Disk Operating System Function Macros (hence the name and the extension). These files may be found at the end of chapter four in Reference 4. They are macros that perform some of the basic MS-DOS functions.

The first three lines of the code segment (after the ASSUME statement) initialize the stack segment register and the stack pointer. The AX register is used as an intermediary, because the segment registers should never be written to directly.

COLOR PLANE

BLUE			RED			GREEN		
1	1	1	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1

(a) Background

1	0	0	1	0	0	1	0	0
1	1	1	1	1	1	1	1	1

(b) Symbol Shape, Inverted

1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	1	1	1

(c) Background AND Inverted Shape

0	1	1	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0

(d) Symbol

1	1	1	0	1	1	1	0	0
1	1	1	0	0	0	1	1	1

(e) Symbol OR'd with (c)

1	1	1	0	1	1	1	0	0
1	1	1	0	0	0	1	1	1

(f) Desired Results

Figure V.2 Results of Operating with Shape and Symbol

After that, three lines prepare for a graceful exit. If a Macro-86 program is ended with a far return, and the stack has had the proper addresses saved on it for this type of exit, a simple return to the operating system is effected. The preparation of the stack involves saving the DS register, and an offset of zero. Having saved the DS register for the return, the next two lines initialize it to access our own data segment.

It is not absolutely necessary to include the next three lines of code. They read and save the input/output port status. This is our standard programming practice of utilizing the stack to save values (such as status registers) that our program modifies, in order that they may be properly restored upon completion of our program's execution.

After clearing the screen, the registers involved in creating a window on the screen are loaded with the necessary values. The header at the top of the BOX.SUB file describes what values are needed, and how they are used.

Next we call the timer routine, in order to measure the time efficiency of the routine which draws the symbols. The input/output port is prepared by the next few lines of code to allow our symbol-drawing operation.

The next six lines of code, from SUB BP, BP through NEG BP, make preparation for entering our outer loop. The base pointer (BP) register is initialized to a negative value so that the outer loop may begin each repetition by incrementing a value equal to LINE and still begin the first iteration at a value of zero. The value of zero is equivalent to the upper left-hand corner of the screen.

Labels identify the statements which make up the tops of each of the iterative loops. The internal documentation identifies the initialization required for each loop. The order of steps could be modified to save a few operations involving the accumulator (AX). We elected to write the code this way for clarity of purpose.

The inner loop, LOOP_K, first performs the AND operations discussed earlier, to clear a space for the symbol. Then the OR operation described is performed, to write the appropriate plane of the symbol into the proper color plane. The loop is repeated for each color plane, making use of the ES register to point to the proper location in VRAM.

The third loop, LOOP_J, repeats the inner loop for each scan line of the symbol. LOOP_I fills each line with symbols, and LOOP_L fills the specified number of lines.

C. MODIFYING THE CODE

The easiest changes are to the symbol. Its shape is modified by changing the binary definition of it within the data segment of the driver program. Corresponding changes should be made to the bytes defining SHAPE in the same program. Defining the symbol in different color planes and/or combinations of color planes modifies its color. If the binary representation differs from color plane to color plane a multi-colored symbol is possible. However, the shape is singly defined with the current program version, and the colors desired may not be the ones displayed. The actual symbol display will depend upon background. All other simple changes, those not affecting the program logic but just modifying the display, are made by altering the values of the constants in the PARM.DEF file. Those are discussed in the next section.

D. CONSTANTS

The constants defined in the PARM.DEF file (Appendix U) determine the display characteristics. They are listed in tabular form below, along with their use.

NAME	PURPOSE
IO_PORT	port address for Zenith input/ output port status register
X_START	x coordinate (in pixels) of left side of window
X_STOP	x coordinate (in pixels) of right side of window
Y_START	y coordinate (in pixels) of top of window
Y_STOP	y coordinate (in pixels) of bottom of window
COLOR	color of window

NAME	PURPOSE
GRIDX_START	x coordinate (in pixels) of left of vertical reference grid
GRIDX_STOP	x coordinate (in pixels) of right of vertical reference grid
GRIDY_START	y coordinate (in pixels) of [illegible]
GRIDY_STOP	y coordinate (in pixels) of bottom of horizontal reference grid
G_COLOR	color of reference grid
LINE	address modification required to shift down one line on the screen
X_LINE	number of right-most byte on one character line of the display
B_PLANE	address of blue color plane in VRAM
I_COUNT	number of symbols to write horizontally
J_COUNT	number of scan lines per symbol
K_COUNT	number of color planes
L_COUNT	number of lines to fill with symbols
PLANE	hex difference between color plane addresses
HORZ	number of bytes to shift right while filling symbols in
VERT	difference in address between top scan-line address and bottom scan-line address of the same symbol position on the screen
SIZE	not used
HITE	not used

LIST OF REFERENCES

1. Z-100 User's Manual, Zenith Data Systems Corporation, 1982.

2. Microsoft MS-DOS Version 2, Zenith Data Systems Corporation, 1984.

3. Adams, James C., Computer Graphics, Heath Company, 1985.

4. MS-DOS Version 2 Programmer's Utility Pack, Zenith Data Systems Corporation, 1984.

BIBLIOGRAPHY

Barnes, J.G.P., Programming in Ada, Addison-Wesley Publishing Company, 1984.

Larsen, Lawrence P., Assembly Language Programming, Heath Company, 1984.

Rector, Russell and Alexy, George, The 8086 Book, OSBORNE/ McGraw-Hill, 1980.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Dr. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	3
5.	LT Ken Coomes c/o Peggy Todd 12 Horseshoe Drive Litchfield, New Hampshire 03051	3
6.	CDR Ron Rautenberg, Code 52Rt Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
7.	CAPT J. Donegan, USN PMS 400B5 Naval Sea Systems Command Washington, D.C. 20362	1
8.	PCA AEGIS Data Repository RCA Corporation Government Systems Division Mail Stop 127-137 Moorestown, New Jersey 08057	1
9.	Library (Code E33-05) Naval Surface Warfare Center Dahlgren, Virginia 22449	1

10.	Dr. M. J. Gralia Applied Physics Laboratory John Hopkins Road Laurel, Maryland 20707	1
11.	Dana Small Code 8242, NOSC San Diego, California 92152	1
12.	LCDR Paul Callahan, Code 52Cs Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
13.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93943-5100	1

END

DTIC

7-86