NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INTERACTIVE NAVAL GUNFIRE SUPPORT TRAINING

by

Warren Anthony Mazanec

March 1992

Thesis Advisor: Michael P. Bailey

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
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) OR	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5006
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Interactive Naval Gunfire Support Training

12. PERSONAL AUTHOR(S)
Warren Anthony Mazanec

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, month day)	15. PAGE COUNT
Master's thesis	FROM_____TO_____	March, 1992	165

16. SUPPLEMENTARY NOTATION The views expressed in this paper are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Object-oriented programming; MODSIM; NGFS; Naval Gunfire Support; Men-In-The-Loop Simulation; Training

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

The goal of this thesis is to develop an interactive computer system for Man-In-The-Loop Naval Gunfire Support (NGFS) training. In attaining this goal, it provides the end user with the full range of exercises a ship sees at the NGFS ranges. The emphasis is on coordinated teamwork to facilitate a seamless transition from the training (non-firing) environment to live-round firings. The benefits are three-fold. First, a more cohesive primary team will be developed. Second, the declining DoD budget demands that fewer training rounds are expended and the training and maintenance budget is reduced. Finally, a more efficient use of range time will result from better shipboard teamwork.

The NGFS training model encompasses the five testable scenarios from COMNAVSURFLANTINST 3570.2D (Gunsmoke) and incorporates the appropriate mix of point, area, and counter-battery fire. A built-in umpiring capability impartially monitors time-line events and award the appropriate points and penalties. A training report is generated to document the exercise and compare results to historical performance. A statistical analysis of improvements by *milestones in mission* vs. the norm is done to emphasize areas in critical need for improvement.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [x] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	21. ABSTRACT SECURITY CLASSICIATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL M. Bailey	22b. TELEPHONE (Include Area Code) (408) 646-2085 22c. OFFICE SYMBOL OR/Ba

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE
S/N 0102-LF-014-6603 UNCLASSIFIED

Approved for public release; distribution is unlimited

Interactive Naval Gunfire Support Training

by

Warren Anthony Mazanec
Lieutenant, United States Navy
BS, United States Naval Academy, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

From the

NAVAL POSTGRADUATE SCHOOL
March 1992

Author: _____
Warren Anthony Mazanec

Approved by: _____
Michael P. Bailey, Thesis Advisor

_____
Wayne P. Hughes, Second Reader

_____
Peter Purdue, Chairman
Department of Operations Research

ABSTRACT

The goal of this thesis is to develop an interactive computer system for Man-In-The-Loop Naval Gunfire Support (NGFS) training. In attaining this goal, it provides the end user with the full range of exercises a ship sees at the NGFS ranges. The emphasis is on coordinated teamwork to facilitate a seamless transition from the training (non-firing) environment to live-round firings. The benefits are three-fold. First, a more cohesive primary team will be developed. Second, the declining DoD budget demands that fewer training rounds are expended and that the training and maintenance budget be reduced. Finally, a more efficient use of range time will result from better shipboard teamwork.

The NGFS training model encompasses the five testable scenarios from COMNAVSURFLANTINST 3570.2D (Gunsmoke) and incorporates the appropriate mix of point, area, and counter-battery fire. A built-in umpiring capability impartially monitors time-line events and awards the appropriate points and penalties. A training report is generated to document the exercise and compare results to historical performance. A statistical analysis of improvements by *milestones in mission* vs. the norm is done to emphasize areas in critical need for improvement.

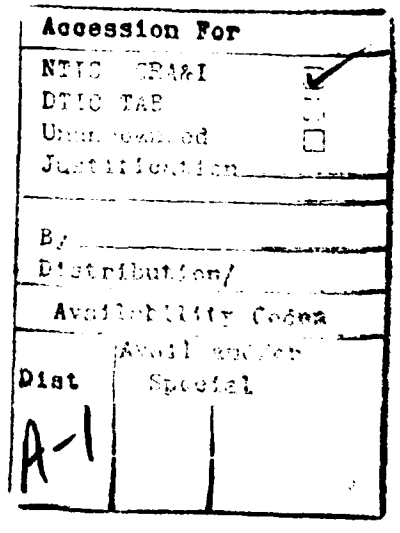

TABLE OF CONTENTS

I. INTRODUCTION

With the increased availability of desk top computers, specialized ship, team and individual training can now leave high-cost, dedicated training facilities and become integrated into ship-board training. Still in use today are the canned scripts that guide ship's company through their drills. Computer systems are now being used to replace handwritten scripts. However, drill monitors are still required to intervene and make the appropriate script available. Emerging technologies such as voice recognition and interaction are available, user-friendly, inexpensive, and on desktop computers. When properly integrated with an interactive training computer program, training can approach the degree of realism once only associated with shore-based facilities and high cost equipment.

A. GOAL

The goal of this thesis is to develop an interactive computer system for Man-In-The-Loop Naval Gunfire Support (NGFS) training. In attaining this goal, it provides the end user with the full range of exercises a ship sees at the NGFS ranges [Ref. 1]. The emphasis is on coordinated teamwork to facilitate a seamless transition from the training (non-firing) environment to live-round firings. The benefits are three-fold. First, a more cohesive primary team will be developed. The primary team consists of the individuals the ship uses during the graded exercise; while a secondary team is similarly trained but not as proficient. Second, the declining DoD budget demands that fewer training rounds are expended and that the training and maintenance

budget be reduced. Finally, a more efficient use of range time will result from better shipboard teamwork.

The NGFS training model encompasses the five testable scenarios from COMNAVSURFLANTINST 3570.2D (Gunsmoke) and incorporates the appropriate mix of point, area, and counter-battery fire[Ref. 1]. A built-in umpiring capability impartially monitors time-line events and awards the appropriate points and penalties. A training report is generated to document the exercise and compare results to historical performance. A statistical analysis of improvements by *milestones in mission* vs. the norm is done to emphasize areas in critical need for improvement.

B. PROSPECTIVE USE

A training system of this nature is designed for use on board to increase primary team coordination and upgrade the level of proficiency of the secondary teams. Strong secondary teams ease the transitions caused by losses in primary personnel from transfers, injury and fatigue. The real utility comes from being able to run scenarios that test three distinct levels of training; ship-wide training, the NGFS team alone and on the lowest level, individual practice. Ship-wide training is the primary purpose, but if a scenario can be run to exercise small teams or individuals, then specific areas requiring training can be targeted without impacting the entire ship.

C. METHODOLOGY

Each of the five exercises are broken down into their component parts to allow the identification of common routines. Intra-ship and ship-to-shore communications are examined to generate the scripts used in each of the exercises. Correctness of the communications between each possible spotter

type and the ship is stressed. A top-level design serves to integrate the scripts and physical models in a modular, object-oriented programming approach. Upon model validation with pre-existing training scripts, perturbations are used to test inter-team coordination. Deviations include, but are not limited to spotter bias, equipment failure, ammunition reliability and accuracy.

A modular, object oriented programming approach is used to develop the models. The programming language MODSIM II is used as it is transferrable to almost any desktop platform encountered in the fleet today without code modifications. An object-oriented language was chosen to facilitate the integration of the many facets associated with platform movement, ammunition accuracy and equipment reliability. [Ref. 2]

D. SCOPE

This study demonstrates the feasibility and desirability of using a desktop Man-In-The-Loop simulation for weapons system training. The training program will be made available to all surface ships so that crews can train themselves in a somewhat realistic environment while in-port or at sea.

This study consists of the construction of:

- the engagement simulation system
- exercise scoring mechanism
- exercise assessment report generator

User acceptance of the system was explored with Surface Warfare Officers found at NPS.

Funding and exercise data for this research effort have been provided by the Major Caliber Ammunition Group at Naval Surface Weapons Center, Crane Division, Crane, Indiana.

II. MEETING THE TRAINING CHALLENGE IN A REALISTIC, MEANINGFUL MANNER

A. ALLOCATING THE SCARCEST RESOURCE OF ALL: TIME

"Time is Money" Benjamin Franklin is credited with saying. The Naval Officer's corollary to that could very well be "Time is Everything." Indeed, one of the biggest challenges facing the sea-going Officer is the allocation of scarce resources, the most important of which is himself. Not only is he faced with the challenge of working at least an eight hour day on watch, but he must also schedule time for administrative tasks, planning, training and overseeing maintenance; never mind eating and sleeping. It is a busy world where the use of pre-existing paperwork is a must because there is little time for the creation of quality, original work. If it exists, and it works, there are more pressing needs that require attention.

It is difficult enough to meet the established training requirements, let alone make the training both realistic and meaningful. The complexities of coordinating a large ship-wide training exercise are enormous. Every detail must be considered to include providing the drill monitors with detailed scripts that specify timing, indications that the crew can see, and most important of all, specifying the point at which the monitor should intervene for safety. After the exercise is completed, aggregating data collected by the monitors and analyzing the results is a time-consuming task. To accomplish this well once is a challenge, but to develop a series of permutations on the basic problem is overwhelming. The problem is magnified considerably when a dynamic situation is being simulated. Ship movement, ranges and

bearings, and communications with outside agencies are beyond the scope of most afloat commands.

In the Gunsmoke publication, there are four canned exercise navigation tracks in the appendices. The interactions between the ship and spotter are documented in a sample exercise but there are no instructions or guidelines for preparing a successful training exercise on a ship. [Ref. 1] The opponent here is not a pesky Third World insurgent, but the pre-existing pall that hangs over training. Once that one scenario has been beaten to death, training becomes stale and extracting anything of value is impossible. The way to surmount this problem is to reduce the burden of creating new scenarios, add a little random uncertainty into the exercise inputs and put all involved on their toes to be ready for anything within the possible realm of the exercise.

One of the often-heard complaints by Naval Gunfire Range Observers is the lack of teamwork exhibited by the ships conducting qualification firings. It is required that formal NGFS training is conducted within 90 days of a qualification exercise by all specified members of the team. After that, a ship is responsible for maintaining proficiency. Neither the money nor time underway is available to put to sea and expend rounds to smooth out teamwork problems. Range time is valuable and large-caliber ammunition precious. Shipboard training coordinators are faced with a difficult challenge—maximizing training effectiveness with declining assets.

B. BETTER TRAINING: IT IS NEEDED NOW

It does not take much for an NGFS team to flounder. Recently, a ship of the Atlantic Fleet, after having completed all other requirements for a successful qualification shoot, failed to qualify because of one person. The

ship's Gunfire Liaison Officer (GLO) broke his foot on the passage to the range. The Executive Officer stepped in to fill the spot. The Executive Officer had been a GLO once, and was the only person available with sufficient experience to assume the role. The XO's experience was dated, but with no onboard training available for a workup, he made the best attempt he could. After two full days on the range, only six rounds had been expended with the last round going directly over the Observation Post. Needless to say, the qualification shoot was terminated immediately. A lot of money had been spent keeping personnel at the range for two days. From any Atlantic Fleet homeport, the trip down to the range is long. Expensive rounds were wasted and most importantly, lives had been needlessly endangered. It is not likely any amount of training could have salvaged perfec scores, but it would have either pointed out correctable weaknesses or that the exercise could not be successfully completed and thereby cancelled.

C. OVERCOMING INERTIA: WINNING THE TRAINING WAR

"Nothing on earth consumes a man more quickly than the passion of resentment." This was written by Friedrich Nietzche in Ecce Homo over a hundred years ago, but very aptly describes the attitude about training. Many times deficiencies in training are perceived to be from a lack of emphasis rather than the true problem; the inability to conduct realistic training. These deficiencies are often manifested in training policies established by external agencies that do little more than increase the training burden by adding more required topics or man-hours of training. This situation, training for the sake of training, is common and found at all levels throughout the Navy.

The NGFS Cylon Spotter is designed as a tool for producing better quality training. It is a stand-alone, unbiased evaluator of Naval Gunfire Support training. The Cylon Spotter simulates the world external to the ship providing communications from the spotter, generating simulated shot fall and providing ship positioning information. It is designed to test all or portions of the NGFS team. To illustrate the capabilities of the Cylon Spotter, consider a ship underway from Newport, RI on weekly operations, as it passes close aboard to Block Island. An NGFS exercise has been planned. The chart shows several likely point and area targets, as well as structures for radar ranges. The ship informs the simulated shore spotter that it is on station and ready for call for fire. For several hours the ship and crew are put through their paces, and shortly after the drill's conclusion a written report detailing the ship's performance is available for an on the spot critique. This was an in-house evolution. There was no range time to schedule, no expensive training team, no equipment to hook up and a minimum number of people to supervise the event. This is the essence of the Cylon Spotter.

In creating the Cylon Spotter, reducing the burden of creating realistic, meaningful scenarios was a major goal. Evaluation of the NGFS team can be conducted on three distinct levels. The highest level of training is a ship-wide exercise with built-in casualties based on historical Mean Time Between Failure (MTBF) data. The next level is internal to the ship, involving the NGFS team. Finally, the program can be run to conduct individual element training on specific tasks with an emphasis on proficiency through repetition. Timing and recording events is done by the system. This minimizes the monitoring team required, and automates most of the data analysis.

A major drawback to current interactive training, like the RESA system, is that all of the commands to the system are typed in. This is slow and frustrating. It takes time to become proficient enough to operate the system without a manual full of commands. A small-scope system like the NGFS Cylon Spotter, with a limited number of commands from the user to the system, can be implemented with the commands grouped by logical headings in pull-down style menus. Actual interaction consists of mouse clicks and use of the <RETURN> key. Inputs are prompted by the appearance of dialog boxes on the screen tailored to the specific information required by the system. In order for the exercise coordinator to more closely monitor the evolution, audible prompts call attention to required input and incoming Spotter message traffic. No manual is needed, just read, pick a command and click; the program takes care of the rest.

D. SCOPE OF THE PROGRAM

"On station and ready for call for fire." These words begin every Naval gunfire exercise. The spotter replies with the fire mission and the ship readies the fire control system to deliver a shore bombardment. The impact area is checked for proper trajectory, time of flight is reported to the beach. The ship waits on the order to fire. The Cylon Spotter responds with "Fire, Over." The rounds are spotted and the ship is guided onto the target. When the rounds are centered to cause maximum damage, the spotter directs the ship to "Fire for effect." Accurate, high-volume large caliber rounds saturate the target area. At the completion of the bombardment, the spotter signifies another successful mission with "End of mission, target destroyed, out." Currently, this interaction occurs only when there are assets available to

provide spotting services. These assets are either airborn spotters, ground-based forward observers or the Observation Posts at the designated Naval Gunfire Ranges. The Cylon Spotter replaces all of these assets. Training with the Cylon Spotter is self-contained, and the environmental interactions become intra-ship. Training is now an on demand evolution; whenever and where-ever it is desired, it can happen.

It is not possible nor is it desirable to model every aspect of the shore bombardment process. Projectile trajectories are modeled on historical impact dispersion distributions. The phenomena of gunbarrel heating and sea state, among others, are ignored, as the overall goal is improving teamwork and shipboard coordination. The dialog between ship and spotter covers the basic components of shore bombardment and includes Call For Fire, point and area targets, coordinated illumination and the appearance of counterbattery fire.

The geographic display editor provided has been designed for the input of actual geographic locations. This facilitates the use of targets of opportunity. An increase in area awareness can be expected at either an exercise or actual bombardment site by familiarizing the team with landmarks and likely targets. The integrated simulation graphics available with the MODSIM language eases the programming requirements for providing real-time position information as opposed to canned exercises. [Ref. 3]

The program provides objective scoring of an exercise in accordance with the established guidelines [Ref. 1]. The elimination of umpire bias and the burden of manually timing events allows the exercise supervisor more time to monitor the evolution instead of becoming bogged down with data collection. Supervisor comments on the exercise are permitted to be entered

in real time and those comments along with the events occurring during the exercise are available in a ship's log style narrative.

III. DEVELOPMENT AND METHODOLOGY BEHIND THE MODEL

Developing the model was a sequential process. The identification of the five testable exercise scenarios was necessary to bound the model in terms of capabilities. A careful analysis of each scenario yielded the basic elements comprising each scenario. This resulted in a composite list of the required Ship–Spotter interactions and specified the data displays the Fire Control party needs. The model pursued two divergent courses at this point. First, the exercise building blocks needed to be completely scripted. A specialized manager was then developed to link the elemental blocks in a logical order according to the exercise specifications. In parallel, the requirements of the Graphical Users Interface, GUI, were detailed. These included the ability for real-time dynamic updates for a geographic display and the dynamic monitoring and display of key values essential to the Fire Control party. Finally, the development environment was chosen so that all of these requirements could be built in a modular, and separately testable manner.

A. DESIGN CONSIDERATIONS

1. Ship–Spotter Interactions

The interaction between the ship and the spotter are detailed in the five graded scenarios located in the appendices of Reference 1. They range from the straightforward; point, area and coordinated illumination fires to the complex; testing the ship's ability to deliver sustained, high volume D-Day shore bombardments. The building blocks from which any exercise can be constructed are as follows:

- Call For Fire

- Point Fire
- Area Fire
- Coordinated Illumination
- Counterbattery Fire

In order to make the program conducive to rapid start-up at the commencement of the exercise, an exercise manager was created to allow editing and planning prior to the drill period. At the start of an exercise, specifying which script for the program to execute causes the right sequence of events to occur. Thus, the highly detailed planning and careful verification of an exercise scenario is accomplished at a time convenient for the user. The options available for the exercise planner include the U.S. Navy's five basic qualification scenarios, and the ability to customize specific scenarios tailored to the training needs of the ship. For variation and flexibility, any exercise/geographic map combination is allowed. This serves one additional purpose: the navigation team becomes an optional element of the exercise. In this manner, the process of fixing ship's position by the normal geographic and electronic means can replace the model's automatic navigation algorithm.

2. Graphical User Interface Requirements

Flexibility in the ability to train requires that the GUI be sufficient to permit training with none of the usual ship's sensors available. In this sense, the GUI provides enough information to the trainees to allow a successful exercise to be run from the classroom. The geographic display in Figure 1 is showing ship's position and orientation relative to the geographic area of operations and is provided for visual validation purposes. As the ship icon

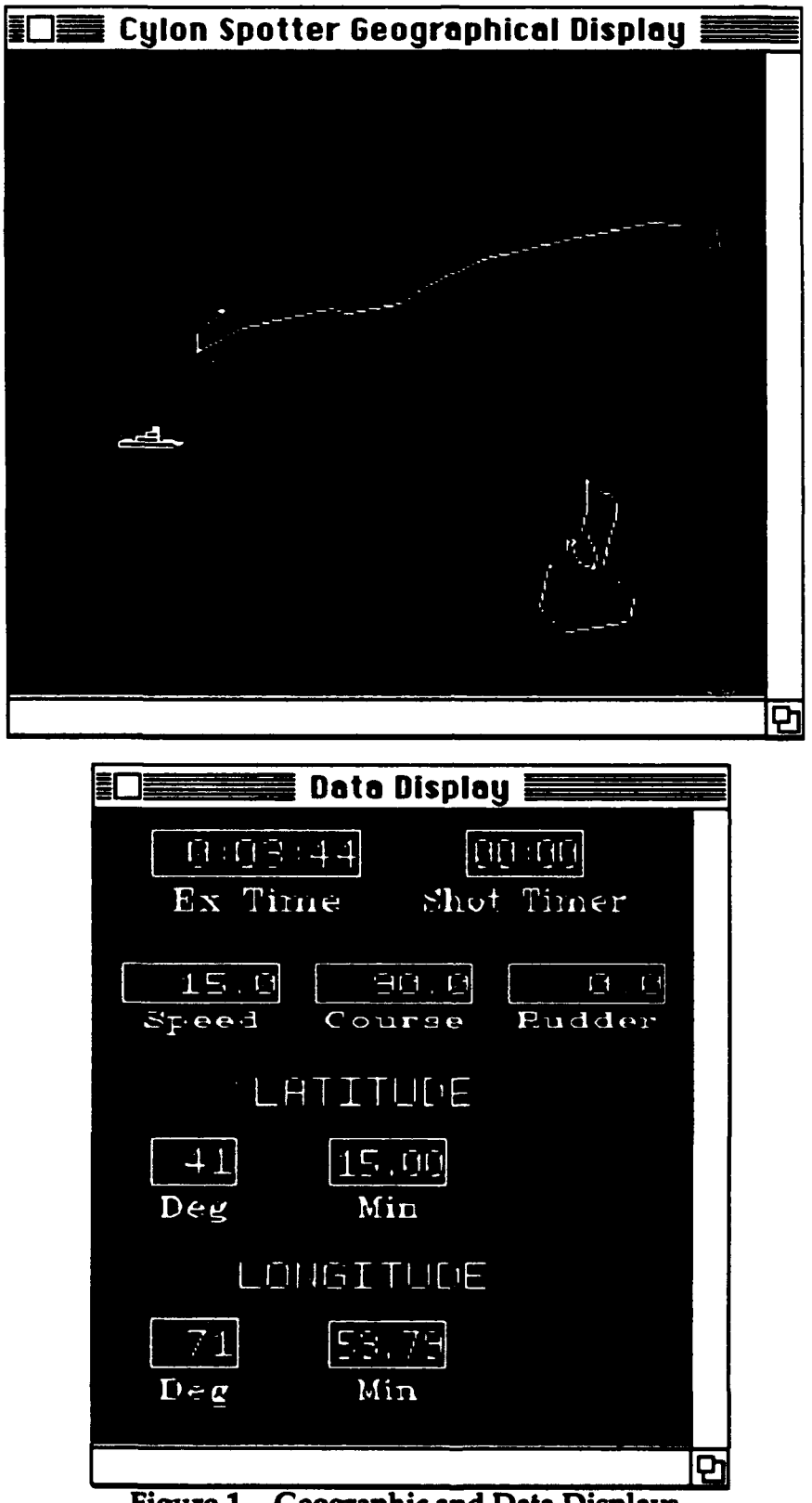

Figure 1. Geographic and Data Displays

rotates with course changes, the intended path can be inspected for correctness. An area 25 nautical miles square is chosen as the default world size of the Geographic Display window. This allows a distance of 25,000 yards from a boundary to the center of the window; more than adequate for five-inch batteries which are normally fired within 15,000 yards of the target. In addition, this allows sea room for maneuvering away from the window's periphery. This feature, though not required by the ship or the spotter, is essential to provide a mental image of where the ship is physically positioned.

Once the ship is initially positioned for the start of the exercise, constant position feedback is required as input to the fire control system and for use on the plots. A separate window is dedicated for displaying longitude and latitude in one of two formats: degrees, minutes, seconds or degrees, minutes and fractions of minutes, depending on the type of chart in use. Time is marked from the beginning of the exercise as opposed to being set to clock time. When the exercise needs to be temporarily suspended for regrouping the fire control team, time can be re-commenced without being fragmented. Finally, ship's parameters are displayed to indicate speed, course and rudder angle. Data output is implemented with digital displays for accuracy.

3. Selecting an Appropriate Developmental Platform

MODSIM II with the integrated SIMGRAPHICS II was selected for its unique capabilities. First, it provides an object-oriented structure that allows the program to be constructed in a distinct and logical modular form. Of major importance is the ability to separately compile and independently test each module. Aberrant behavior of a module is easier to identify and correct

without confounding the model with secondary and tertiary affects. The integrated graphics and the creation of graphical objects simplifies the modeling of ship dynamics. Changes in motion are handled by a library of pre-defined methods, and the digital displays are directed to automatically update to reflect changes in an object's descriptive fields. [Ref. 2]

A standard for desktop computing presently does not exist for the U.S. Navy. The ability to port code to other operating systems without major software rewriting is considered important. MODSIM provides a standard front-end language that can compile into stand-alone applications on most operating systems.

B. DETAILS OF THE GRAPHIC USER'S INTERFACE

The Graphic User's Interface is designed to provide the trainee with logical prompts to reduce the amount of keyboard interaction that is required during the execution of an exercise.

1. Pull-Down Menus

Pull-down menus are the command interface for user to system commands. They are grouped functionally to place similar command choices in the same pull-down block. Figure 2 shows all pull-down menu headings and sub-headings available under each heading.

To prevent improper selection of sub-headings, they are coded with a flag that when set makes them unselectable. The excluded sub-heading item appears dimmed to the user to indicate that the item is not available for selection. For example, until START is selected under the COMMEX heading in Figure 2, the remaining items, STOP, PAUSE and RESUME are unavailable for selection. The last major heading selected changes appearance from black

text on a white background to white text on a black background as a reminder. As each sub-heading is selected, an appropriate dialog box appears to prompt the user for further required actions.

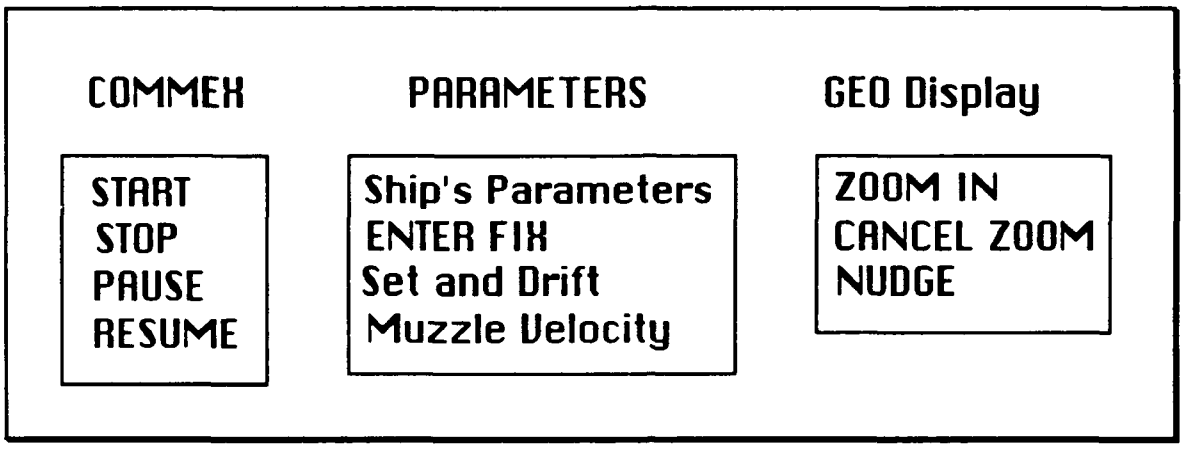

Figure 2. Menu Bar Headings and Sub-Headings

The three menu categories relate to program execution, changing the operating characteristics of the ship and altering the geographic display window. Under the COMMEX heading, program execution begins with START and is terminated by STOP. PAUSE suspends execution and a dialog box appears prompting the user for the cause of the suspension. This is included in the exercise summary. RESUME cancels the pause command and simulation time is once again advanced.

2. Dialog Boxes

Dialog boxes are graphical objects created in a graphical editor and attached to a window. Each dialog box is comprised of several sub-elements. Value boxes are for inputting numeric data, either integer or real. Check boxes accept boolean input, that is, the item is checked and evaluated as TRUE or FALSE, not checked. Text boxes display descriptive text helpful to successful use of the dialog boxes. The last type of sub-element is a button.

Most dialog boxes request input of some type. Accompanying the request for information are two buttons labeled Change and Cancel. Both are terminating buttons. Terminating buttons cause the dialog box to erase when a mouse click is sensed on that button. Change causes the new data entered into the value and check boxes to be accepted. Selecting Cancel invalidates any new data that has been entered and provides a way to exit a dialog box that has been inadvertently selected by the user.

a. Ship Parameters

The Ship's Parameter dialog box, shown in Figure 3, displays information pertaining to ship's course, speed and rudder angle. This dialog box is the result of selecting the sub-heading of the same name in the Parameter's menu. Changing course and speed invokes a method of the ship which causes the rate of change from the old to the new value to follow values contained in the ship's data file.

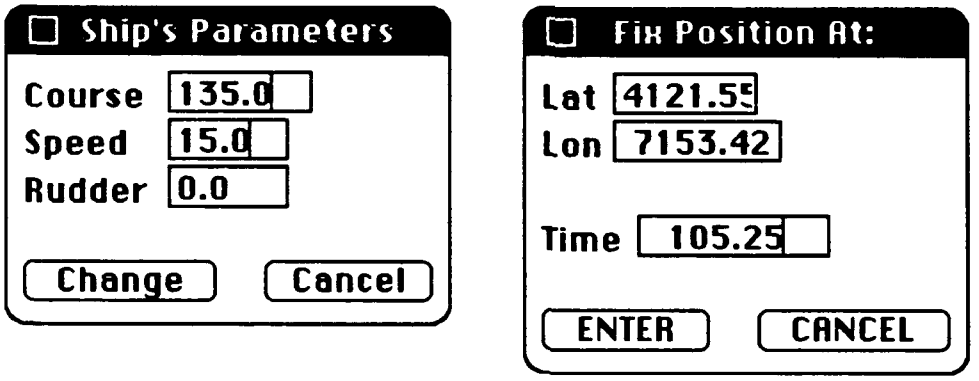

Figure 3. Dialog Boxes For Ship's Parameters

Selecting the the ENTER FIX sub-heading results in the second dialog box shown in Figure 3. All fixes of ship's position are time late. A method of dead reckoning using current ship's course and speed to determine

actual position is included. Entering the time of the fix in HHMM.SS format causes the ship's position to update to current problem time by applying a displacement based on the direction vector multiplied by the elapsed time since the fix.

b. Physical World Parameters

The parameters of the simulated physical world are not constant. The largest effect the physical world imposes on a ship is the phenomena of set and drift. The dialog box for set and drift is shown in Figure 4. Based on the input parameters of speed and direction, an offset is applied to the ship's position every time it is updated. As current conditions change throughout the evolution, the data in the dialog box must be updated by the user.

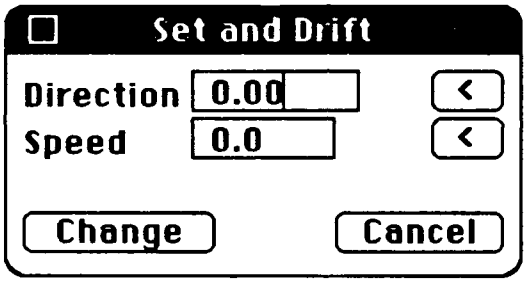

Figure 4. Set and Drift Dialog Box

c. Time of Flight

To accurately calculate projectile time of flight, muzzle velocity of the projectile must be known. As shown in Figure 5, the Muzzle Velocity dialog box has two buttons for selection. The buttons set time of flight velocities to either normal or reduced charge. This is followed by a prompt to ask the user if the previously recorded velocity is correct. There is also input for cases when normal trajectories are not being used. A check box is present

for cases where the target is in defilade and higher than normal trajectory paths are used.

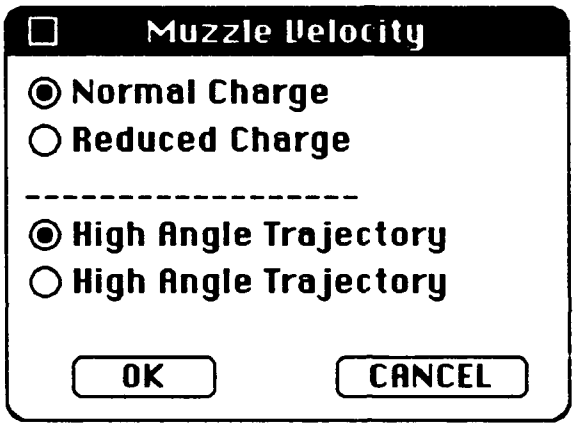

Figure 5. Muzzle Velocity Dialog Box

c. Zoom

The ZOOM menu causes the area of the geographic display window to shrink. This feature is provided for two reasons. First, it allows the accuracy of a geographic feature to be examined on a more precise scale. Secondly, the default window dimensions do not permit viewing of target locations. When the 4:1 zoom option is selected (Figure 6), a flag is set to make the target locations visible. The target the ship is currently engaging blinks. Canceling the zoom feature returns the window to its default dimensions of 50,000 yards by 50,000 yards.

The two zoom dialog boxes appear sequentially. The first selection sets the level of magnification. The Zoom Centering allows the user the option of zooming centered on a specific location, such as target location or zooming centered on the last mouse click position. If a zoom operation is scheduled to occur and a boundary of the geographic display is violated, the

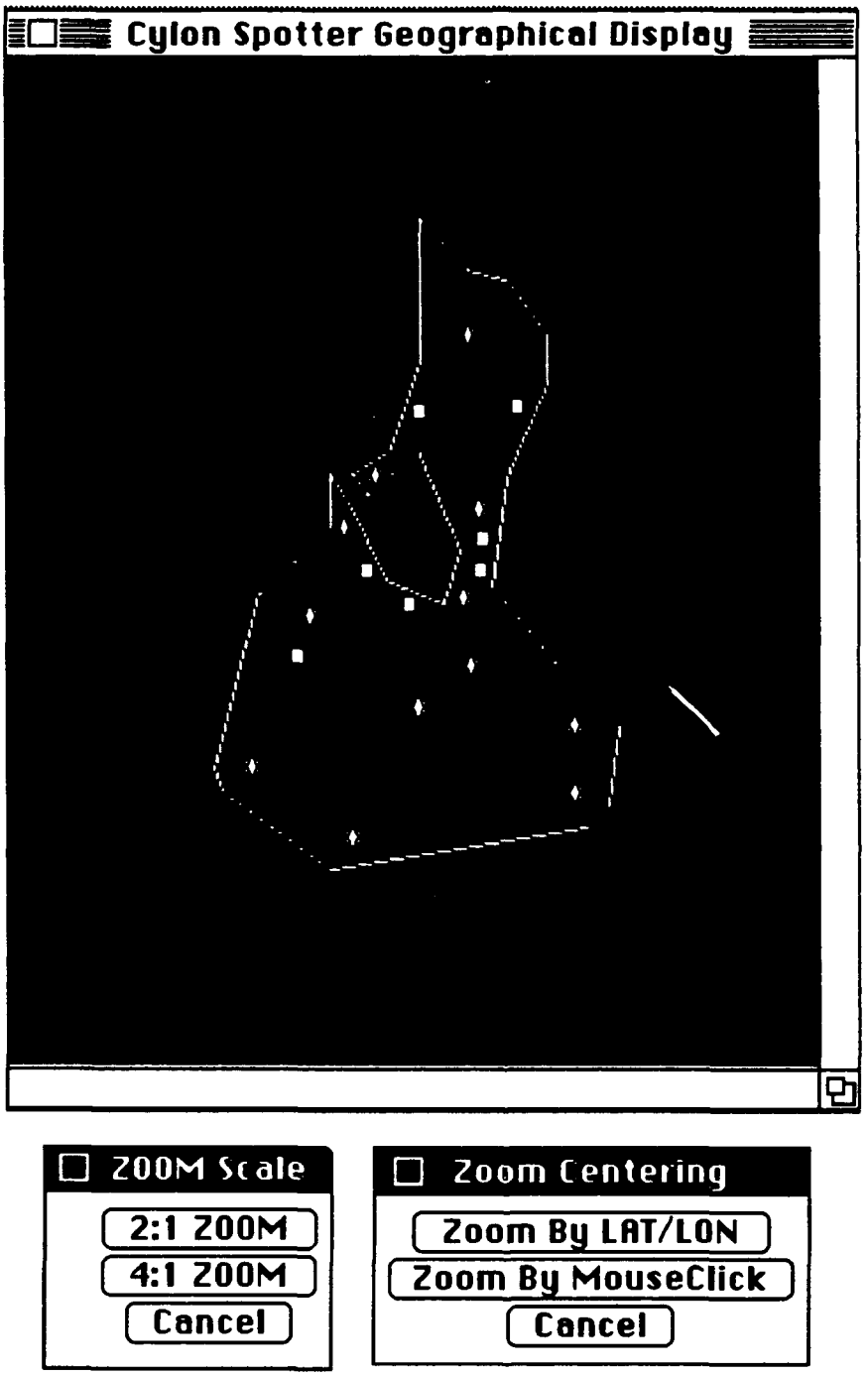

Figure 6. Geographic Display and Dialog Boxes For Zoom

zoom window shifts to coincide with the boundary of the geographic window. To prevent unnecessary entering and exiting of the zoom option

because the positioning of the zoom window is incorrect, a nudge option is provided. The Nudge box, shown in Figure 7, allows the the zoomed portion to shift in the four cardinal directions. The total distance of each nudge is based on the zoom scale selected. Additionally, a fine nudge option is available to move the zoom window half of the normal distance. Until the DONE button is selected, the Nudge box re-draws itself after every nudge operation.

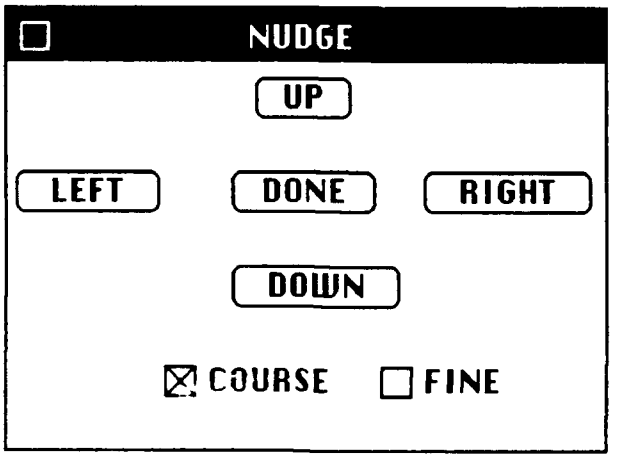

Figure 7. Nudge Dialog Box

d. Additional Dialog Boxes

There are several dialog boxes that are not associated with any menu item. The most important is the Supervisor's Comment dialog box. This box is for the supervisor to input additional comments regarding the current exercise. Documenting erroneous ship-to-spotter communications is accomplished in this fashion. A button in the dialog box is dedicated for this purpose.

C. MODELING THE PHYSICAL SYSTEM

As stated previously, precise modeling of the physical attributes of the ship is not the purpose of this model. However, certain characteristics could not be overlooked. Most important are the rates at which a ship accelerates and decelerates and changes heading for a given rudder/speed combination. The level of precision that is required by the model is determined by two criteria, both of which are necessary for grading an exercise. The grading criteria are displayed graphically in Figure 8. First, the ship is required to inform the spotter of changes in projectile time of flight of five seconds or greater. The lateral velocity of a round is approximately 100 yards per second. It is assumed that measuring the accuracy of projectile time of flights by the observation team is to the nearest second. Based on the lateral velocity, this is equivalent to an error in range of 100 yards. The second criteria is a change in the imaginary line drawn between the ship and the target, known as the Gun Target Line, by five degrees of arc or more. Normally, shore bombardment operations occur at ranges in excess of 6,000 yards. At 6,000 yards, the length of the arc subtended by a five degree angle is approximately 525 yards. Measuring angles to the nearest whole degree is assumed to be the level of accuracy that can be obtained on a chart. At this close range, the corresponding error is 105 yards. The goal for positional accuracy is ± 100 yards. The ship in the model has been maneuvered extensively and the model's ship positions were determined to be within ±100 yards of the hand plotted positions. It is important to note that after a course change, a ship is required by NGFS doctrine to fix its position as soon as steady on the new

course. Errors introduced by the model during a maneuver are reset to zero after the new fix is entered.

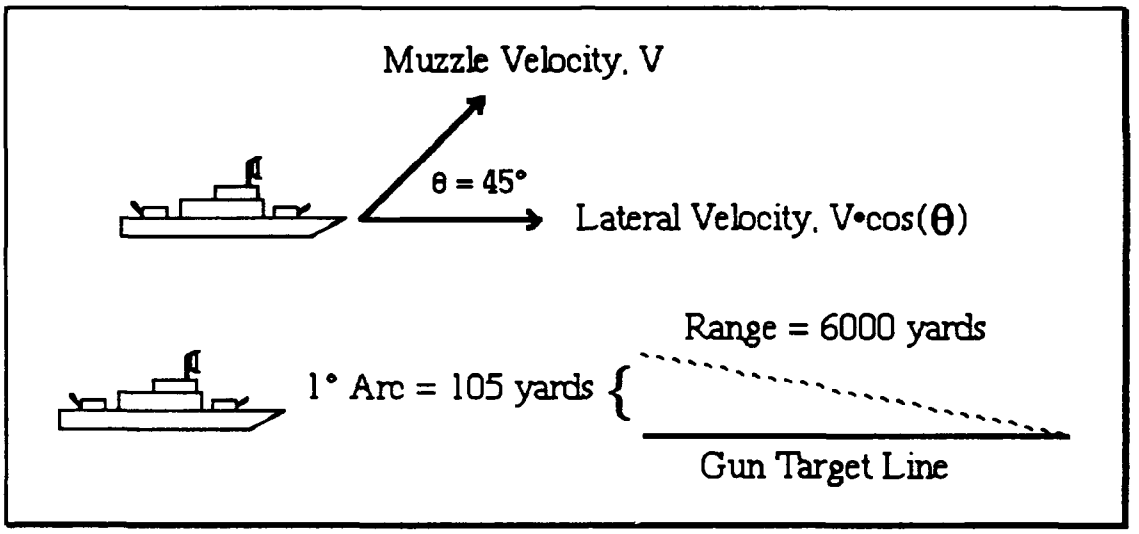

Figure 8. Bounding Model Accuracy

1. The Mercator Projection Problem

There are two types of charts currently in use utilizing Mercator and Gnomonic projections respectively. For large-scale charts (small area), the Mercator projection is standard [Ref. 4].

Two arcs of longitude are not equidistant from the equator to a pole. A method was developed to account for the contraction between two arcs of longitude as a pole is approached. The Mercator projection is such that two arcs of longitude are assumed to be parallel. This in itself is not critical, but as chart locations are shifted from ones close to the equator to ones off coastal United States, the same displacement measured in yards increases per degree of longitude displaced. The physical world represented in the simulation is measured only in constant units of displacement. A filter has been put in place so that when positions are entered into the system for any specified latitude, it is translated into the appropriate units of yards of displacement

from a reference point. This filter must also work in reverse, so that when ship's position is displayed, it is shown in correct latitude and longitude.

To model the physical world so this problem is accounted for by the program would be an error, and a case of over-engineering. Instead, a simple measurement of a constant displacement is taken and reported in degrees of longitude. From this, a scale value is calculated and applied to longitude values going into the model world, as well as a position from the model to a corresponding longitude.

At the equator one minute of latitude equals one minute of longitude which is equal to one nautical mile or two thousand yards. From this, five minutes of longitude is equal to 0.0833 degrees of longitude.

$$5' \text{ longitude} \times \frac{1^\circ}{60'} = 0.0833^\circ \text{ longitude}$$

The procedure uses this fact and treats the latitude and longitude of the lower left hand corner of the chart, chosen and specified by the user, as the reference point. It is from this position that all other latitude/longitude combinations are displaced. A distance of 10,000 yards is measured from the reference longitude. This is the scaling longitude. A ratio from these two longitudes is the Scale Factor. The Scale Factor is equal to 1.0 at the equator and increases as a pole is approached. Equation (2) adjusts the displacement to units of longitude along the equator, then converts longitude to yards. Simplifying equation (2) yields equation (3) which is representative of the code implementation. This approach was chosen because of its simplicity and requires only that the user supply the latitude and longitude of the reference corner and the scaling longitude.

$$(1)\ \text{Scale Factor} = \frac{\text{reference longitude - scaling longitude}}{0.0833}$$

$$(2)\ \text{distance in yards} = \frac{\text{reference longitude - longitude of object}}{\text{Scale Factor}} \times \frac{10{,}000}{0.0833}$$

substituting in for Scale Factor yields:

$$(3)\ \text{distance in yards} = \frac{\text{reference longitude - longitude of object}}{\text{reference longitude - scaling longitude}} \times 10{,}000$$

2. Advance and Transfer during a Maneuver

For each ship, tables of advance and transfer are calculated during sea trials.

> Advance is the distance gained in the direction of the original course until the ship steadies on her final course. Transfer is the distance gained at right angles to the direction of the original course until the ship steadies on her final course. [Ref. 4:p. 221]

The values contained in these tables represent combinations of ship speeds and rudder angles measured in yards of advance and transfer for different turn angles. This is acceptable for hand-plotting ship's position, but presents a formidable programming challenge. The hand-plotting method is discrete; the start of the turn is plotted, then advance and transfer is applied to determine the ship's position where the turn ends. The Cylon Spotter model employs continuous dynamic updating and the discrete position offset method will not work. Instead, measuring the rate of turn in degrees per second over a specified arc is easily integrated into the continuous updating of the model's ship. Three speeds of five, fifteen and twenty-five knots, each the mean value of a ten knot speed range, are used. The same division for rudder angles is used. These values are displayed on tables in Appendix B; which is a sample ship's data file included for clarity. The time to shift 60

degrees in course from the original heading is measured at four points and used as input parameters for the ship. By the time ship's head has shifted by 60 degrees, the steady state turning rate for that speed/rudder combination has been approached. This results in a total of 36 data points input into the computer. Each of these points represents a constant value for the range of rudder angle, speed and course change values.

3. Time of Flight

Time of flight calculations are made using just two facts; range to the target and muzzle velocity of the projectile. Velocity, V, is broken into the component vectors

$V_x = V \bullet \cos(\theta)$ and $V_y = V \bullet \sin(\theta)$. At perigee the vertical velocity is zero. Utilizing classic physics, V_y = acceleration due to gravity • (1/2 time of flight). Range = V_x • time of flight. Substituting for V_x and V_y results in the equation (4).

$$(4) \quad \frac{2 \cdot \text{Range} \cdot g}{V^2} = \cos(\theta) \cdot \sin(\theta)$$

Using the trigonometric identity $\sin(2\theta) = 2\sin(\theta)\cos(\theta)$, equation (5) describes the initial trajectory angle. Then, with the value for t and the V_x vector, time of flight is solved without the user entering gunbarrel elevation for every round. The time of flight calculation is made assuming the difference in sea level and the target's elevation has a negligible effect on the time of flight. An optional attribute, Target In Defilade, is included to select the higher trajectory solution for the time of flight problem. Maximum range is achieved at 45° barrel elevation. The barrel elevation for the high trajectory solution is 90°—barrel elevation for the normal solution.

(5) $$\arcsin\left(\frac{4 \cdot \text{Range} \cdot g}{V^2}\right) = 2\theta$$

These assumptions are made with the belief that this is good enough to accomplish the underlying purpose of the model, measuring crew performance. Using more detailed rudder angle, speed or course change ranges would serve no purpose.

IV. ANALYSIS AND FEEDBACK OF CREW PERFORMANCE

The data collected by the model consists of error rate counts and time to conduct certain evolutions. A separate analysis program takes the data and scores the results. The results are to be used by non-analysts. The people who review the exercises need to know where to focus attention, what is satisfactory performance and determine the rate of progress. Keeping this in mind, the questions to be answered by the analysis need to be straightforward and must provide grades on the timed portions of the qualification exercises and highlighting areas that do not meet established standards. In developing the analysis portion of the model, it was decided that hardcopy feedback that can be included with training reports as a permanent record took priority over on-screen graphical analysis. All confidence intervals use an alpha value of 0.05; and the t-tests are used, since most sample sizes are less than 30 until a large number of runs are completed. The Aspin-Welch test is used to compare sample means because this test handles the case where variance and size of two samples are not equal. When known, ship's performance is compared to fleet performance statistics.

There are four basic types of events to analyze. Each event type may have many individual components, but only the base case is discussed. Extrapolation to other cases is straightforward, and any major differences are discussed here. The four basic types of events are detailed in the following paragraphs.

A. COMMUNICATION ERROR RATE

Errors in communications between the ship's Radio Telephone Talker and the spotter are measured by the rate of error. There are seven communications errors outlined in paragraph 301.d of Reference 1. Errors of specific type are both individually tallied and aggregated and compared to the total number of required communications by the ship's Radio Telephone Talker. Each error type has a different point value, and is recorded accordingly. A summary of errors is provided as well as a confidence interval on the percentage of total errors committed. A hypothesis test comparing the means of the current exercise against the aggregate of the last five exercises and the total aggregate is included. The report for this portion lists the errors committed, the mean error rate and the bounds of the confidence intervals.

B. PASS/FAIL TIMED EVOLUTIONS

Pass/fail timed evolutions are those evolutions that if violated, cost the ship penalty points. No credit is awarded for doing better than the minimum standard. A summary of the errors committed is listed, and the penalty points awarded.

C. SLIDING-SCALE TIMED EVOLUTIONS

Evolutions in this category have a maximum point value and depending upon the ship's performance of these evolutions, a point value less than the maximum may be awarded. The general points awarded formula takes the form:

$$\text{points awarded} = \text{maximum points} - \frac{(\text{time to complete} - \text{constant1})}{\text{constant2}}$$

Most categories apply to all ships, but a few are fire control system specific. These evolutions are timed and graded. Single event occurrences are recorded and compared to historical data. Multiple event occurrences have the mean and confidence intervals calculated and compared to fleet and ship historical data.

D. EQUIPMENT CASUALTIES

If the option for random equipment failures is selected, equipment failures occur at historical Mean Time Between Failure rates. Time to recover from the equipment casualty is then measured and reported. A confidence interval based on previous performance is calculated and compared with the current casualty event.

E. TRAINING REPORTS

Three different exercise summaries are generated by the analysis. The first is an event summary in a ship's log format. The second report is the graded summary of the exercise detailing points attained, the total possible points and points deducted. Included in the report heading is the starting seed of the random number generator, so that this exercise can be repeated exactly. The final report contains the statistics calculated from the exercise.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The objective of this research is to develop an easy-to-use desktop computer training model to improve shipboard coordination during Naval Gunfire Support missions. This model incorporates object-oriented programming techniques and is the first step in building a large library of general purpose modules. The flexibility inherent in modular design will become the basis from which any variant of interactive training can be easily and quickly created. Within the scope of limited feedback and pending user acceptance testing, this goal has been accomplished.

First, the interaction between the ship and the spotter has been completely scripted and implemented. All of the testable scenarios may be run on this model. Additionally, the versatility of adapting a testable scenario to a target of opportunity increases the utility of the model to provide even more realistic training than can be accomplished by a computer model alone.

Second, an easy-to-use graphical interface is developed which eliminates the need for a user's manual during an exercise. The necessity of minimal interface requirements frees the user from running the system to allow a more thorough monitoring of exercise events.

Third, analysis of data generated by an exercise is aggregated into a compact, easy-to-understand form that answers: What is the level of proficiency? where does training need to be concentrated? is there an improving trend?

Finally, the fusion of the concepts developed in the model increases the realism to a level that was unattainable by an afloat command. This increase in realism results in improved shipboard coordination when conducting Naval Gunfire Support missions. More efficient use of Naval Gunfire ranges are now possible with less waste of expensive ammunition.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

1. Fleet Feedback

It is recommended that after this model has had some end use, the feedback from the users be incorporated into the next release of the model. Areas being considered for updating are model dynamic motion, moving targets, operating system loading and training report format. The methodology for representing a ship's dynamic motion should be assessed for the expansion of the ship's data. This affects model fidelity. The model was left intentionally sparse for operating system performance concerns. It is likely that the end-use platform is a 286-based PC operating at its upper limits with either Windows or OS/2. After the completion of user acceptance testing, the trade-offs between model fidelity and operating system performance should be evaluated and the model adjusted if desirable. The scope, depth and utility of the statistical analysis assessed by the program can be further tailored to training needs. Based on the feedback, computational adjustments should be made accordingly. After completing the adjustments, graphical methods for evaluating exercise data should be incorporated into the analysis module.

2. Voice Recognition Technology

The introduction of inexpensive, commercially available voice recognition hardware opens the door to training realism previously unattainable. Specifically, products like Voice Navigator manufactured by Articulate Systems are designed to interface with pre-existing software that use pull-down menus as an interface in both directions. It recognizes speech in addition to generating speech responses from text-based data files. Expanding the menu bar to include a flat file listing of ship-to-spotter communications allows interaction in the most realistic sense. This will provide a hands-off mode of operation.

3. Incorporate Simvideo

The third leg of the integrated software development platform is simvideo. Simvideo allows the user to "capture" an exercise run in tape file. This tape file can be operated as if it were in a VCR. Forward, rewind, pause and slow playback are all options of this package. This is a valuable tool for post-exercise critiques.

4. Implement Spreadsheet Primitive

Currently, the model allows for entering and editing data via a question and answer mode, which is slow. The spreadsheet primitive would act as a front end to a data file and allow the user to view, enter and edit data from a familiar spreadsheet form. This in turn would eliminate the need for most of the prompts for individual pieces of data.

APPENDIX A. USER'S MANUAL

A. GETTING STARTED

1. Installing the Cylon Spotter

1. Create a directory on your hard drive named "Cylon". Initially, the program and its required files needs two megabytes of space. As the program is used and more data files are created, the memory requirements increase.
2. Copy all of the files into the directory Cylon. The following files are required.
   - BlockIs.atgt
   - BlockIs.data
   - BlockIs.ptgt
   - BlockIs.shore
   - CYLON.exe Note: This may not have the .exe extension.
   - EX.list
   - Gui.lib
   - map.lis
   - Seed
   - Ship.data
3. Map files from the same location use the same root name and are uniquely identified by the extensions. An example of this is the BlockIs map family. Included in a map family are area and point target list specifications. Located in the data file are the geographic specifications used to translate latitude and longitude to the Geographic Display window dimensions. The shore file contains the points used by the program to plot shorelines in the Geographic Display. All four of these map family file extensions need to be present, even if they aren't being used. The four file extensions are:
   - .atgt Area Target list
   - .data Geographic data file
   - .ptgt Point Target list
   - .shore Shoreline representation file

2. Running The Cylon Spotter

Run the CYLON program. A selection menu, like Figure A-1, is presented. After entering the selection, the user is prompted for further information.

```
Loading data, one moment please
Welcome USS Merkin To the NGFS Cylon Spotter


Input the number of the option:


    1. Verify Ship's Data
    2. Change Ship's Data
    3. Plan the next Excercise
    4. COMMEX
    5. Create New Geographic Display Chart

    HIT <return> to TERMINATE this session
```

Figure A-1. The Introduction Menu

a. Verify Ship's Data

Option 1 presents the user with information contained in Ship.data, the ship's data file. An example of this is shown in Appendix B.

b. Change Ship's Data

Option 2 allows the user to review the information presently in the ship's data file and change it. The most important piece of data is the operating characteristics of the ship. Specifically, these are the matrices that contain turning rates for different speed/rudder combinations and the ship's acceleration/deceleration rates. Deceleration rates are based on changing speeds by ordering up the bell that is the ship's final speed. Propulsion assisted deceleration, meaning the use of backing bells, is not taken into

account. It is recommended that a copy of Ship.data is printed prior to changing any data. This enables the user to mark up the sheet with new information in the form that is requested by the program.

c. Planning the Next Exercise

Planning an exercise involves deciding what combination of area, point and coordinated illumination fire to use. Choice of an exercise name is important, as it could possibly write over a pre-existing exercise. Before entering the new exercise name, review the list provided, shown in Figure A-2, to ensure that a good exercise is not inadvertently over-written. The user is then prompted for the Map Family used in the exercise. If a new geographic area is being implemented, the user may go straight to the Geographic Editor from this menu, as seen in Figure A-3. This feature eliminates the need to back out of the Scenario Editor to enter the new chart. The Scenario Editor allows you to choose to follow a Gunsmoke exercise exactly or to create an exercise suited to the needs of the ship as shown in FigureA-4. When making a selection, the number of Fire For Effect rounds is requested. This is the number of rounds in addition to the rounds needed to spot onto the target. At present, the scenario editor does not allow mixing custom exercises with the Gunsmoke specified exercises. Mixing of the Gunsmoke exercises is allowed. To avoid ambiguity on what name the Gunsmoke exercise is called, the exercise designation is used. Once the planning is completed, a screen informs the user to print out the target list for the exercise, see Figure A-5. An example target list is contained in Appendix C.

```
The Following Excercises Are Already Named

              test
              demo
Input The Name Of The Excercise You Wish To Create
Re-use of previously named file will cause that data to be lost.

NOTE:     Keep the name to under 8 characters to prevent
              conflicts with the operating system.
```

Figure A-2. Reviewing Existing Exercises

```
      Input the number of the Map Family for this Excercise

      1.   Blockls
      2.   shapes
      3.   test
      4.   Create New Geographic Display Chart.
```

Figure A-3. Selecting a Map Family

```
    THESE OPTIONS ARE TO ASSST IN THE PREPARATION OF
                AN NGFS EXCERCISE.

          1. Point Fire Excercise
          2. Area Fire Excercise
          3. Coordinated Illumination Excercise
          4. Z-40-G
          5. Z-42-G
          6. Z-43-G
          7. Z-44-G
          8. Z-45-G
          9. Exit

      Input the number of the option:
```

Figure A-4. Scenario Options

```
An exercise target list will be generated and stored in a file
called exercisenameTgt.lis.  Print this list prior to commencing the
excercise to allow the plots a chance to create an excercise chart.
THIS IS REQUIRED BY THE CYLON SPOTTER
```

Figure A-5. Target List Message

d. COMMEX

An exercise may be successfully run after ship's data is verified to be correct, a Map Family has been created and an exercise has been planned. The user is presented with a menu similar to Figure 3, only requesting the exercise name. The user is then prompted for one of three random number seed options as seen in Figure A-6.

```
The last exercise used 2116429302  as the Starting Random Number Seed.

Input the number corresponding to your choice for Random Number Seed.
  1.  Re-Use Last Seed:    To re-run last drill Exactly
  2.  Choose New Seed:     To re-run an old exercise Exactly
  3.  All Other Cases:     Choose this most of the time.
```

Figure A-6. Random Number Seed Options

If the user wants to re-run the last exercise, the seed is automatically re-set. If the user wants to re-run an old exercise, the seed number is found on the top of the second page of the appropriate training report. An example is found in Appendix D. For most exercises, it is recommended that the third option be used. It is still possible to re-run an exercise, but the spots and number of rounds to spot onto the target as well as random gun system failures will be different. The ship is then given a moment to set up for the exercise. Once the ship is ready, hitting the <RETURN> key commences the exercise. What

follows is the dialog that is normally associated with ship and spotter communications.

e. Creating a New Geographic Chart

Care must be taken when entering geographic areas. Select the chart that will be used during the actual exercise. There are two possible systems for latitude/longitude input. The first uses degrees/minutes/seconds, the second uses degrees/minutes/fractions of minutes. The Cylon spotter does not have the capability to accept input in one format and display positional information in the other. The input has been designed so that the absolute difference between latitudes and longitudes is plotted; this eliminates the necessity for labeling N/S or E/W. Problems may be encountered when the chosen area straddles either a prime meridian or the equator. Southern Hemisphere latitudes need to be preceded by a negative sign (-) to create a north oriented chart, otherwise the image will be upside down.

The geographic display represents an area 25 NM square. The reference point is the lower left-hand corner. Choose this point to maximize the amount of useful geographic area while providing sufficient sea room for the model's ship to maneuver. It is possible that the initial ship's position may be plotted off of the displayed area, in which case the gunfire exercise should not commence until the ship has driven into the window and is visible. After selecting the reference point, a scaling longitude is entered. This enables the program to correlate distance in yards to minutes of longitude at that given latitude. Remember that the distance represented by a longitudinal measurement decreases as the distance from the equator increases.

Select a name for the chart area. This becomes the root name for the Map Family. All target lists, data and geographic files share this root name. The files are distinguished by the extension. You are asked for graphic objects, area targets and point targets in that order. Counter-battery targets are selected from the point target list. When prompted for the number of geographic object to input, enter the number of distinct shorelines to appear in the window. In the BlockIs example file, there are three shorelines; the South East coast of Connecticut, Block Island proper and the northern tip of Long Island. For island type objects, the last point entered must be the same as the first point for the object to be closed. A lake should be added separately, as it is an enclosed feature within another shoreline. Appendix C is an example of a target list and shows just what information is required by the program to maintain a target list.

Once a geographic map has been generated, there is no way presently to edit the files. Using a text editor, the files can be altered if necessary. **Make a copy of the file before editing!** Edit the duplicate file only; this will prevent loss of data. Save the edited file as an ASCII text file only. If it is saved as a word processor file, the file will contain embedded characters that cause the Cylon Spotter to bomb. Rename the original file so that it can be found easily.

(1) Adding/Deleting Shorelines. The .shore file is organized in the following manner. The first integer in the file is the number of shorelines. Then there are the number of points in each of the shorelines following. The list of each shore point follows, with each shore point having four data elements, the latitude, longitude and the window x and y positions. To add a shoreline to a pre-existing, run Cylon, choose Create New

Geographic Chart and select a temporary name for the chart file. Input the same data as the chart you will add these shorelines to. Input the shorelines. Quit the program and open the file you wish to add to. Figure A-7 points to the appropriate numbers. Increment the first integer to reflect the new total number of shorelines, and the second number to indicate the total number of points in the file. Preceding the first occurrence of the X position in the window, insert the numbers corresponding to the number of points in each of the new shorelines being added. Append the list of shoreline data points to the bottom of the file.

```
    2            <- Number of Shorelines
   48            <- Total number of points to be plotted
   27            <- Number of points in shoreline 1
   21            <- Number of points in shoreline 2
38449.848024     <- X position in the window
27840.000000     <- Y position in the window
 4113.920000     <- Latitude
 7134.700000     <- Longitude
38480.243161
25580.000000
 4112.790000
 7134.680000
```

Figure A-7. Sample .shore File

Deleting a shoreline is a little more tedious. Repeat the file duplication steps as before. Open the file to be operated on. Decrement the first integer by the number of shorelines you wish to remove. It is suggested that only one shoreline at a time is taken out. Decrease the second integer by the corresponding number of shorepoints to be deleted. Delete the line for the number of points in that shoreline. Count down and delete the correct number of points. Save the file. It helps to print out the file beforehand to line out and adjust. A future release of the Cylon Spotter will have a more elegant way around this problem.

f. Conducting the Analysis

After the exercise has been completed, select STOP from the COMMEX menu, this quits the program. Return to the operating system prompt and run the program **Analysis**. You are prompted for the name of the exercise from a menu. The user is informed when the analysis is complete. The training report is found in the file of the same name as the exercise with the root .rpt.

B. NAVIGATING THE GRAPHIC USER INTERFACE

1. The Menu Bar

The menu bar is the interface for commands from the user to the model. Under each heading there are sub-headings as shown in Figure A-8. The headings are organized as follows: operations affecting program execution are located in COMMEX, changing the ship's characteristics are under PARAMETERS and altering the magnification of the geographic display under GEO Display. When a menu item is not applicable to the program, it appears dimmed and cannot be selected by the mouse. An example of this is before program execution begins, the only selectable menu tem is START; all others are dimmed.

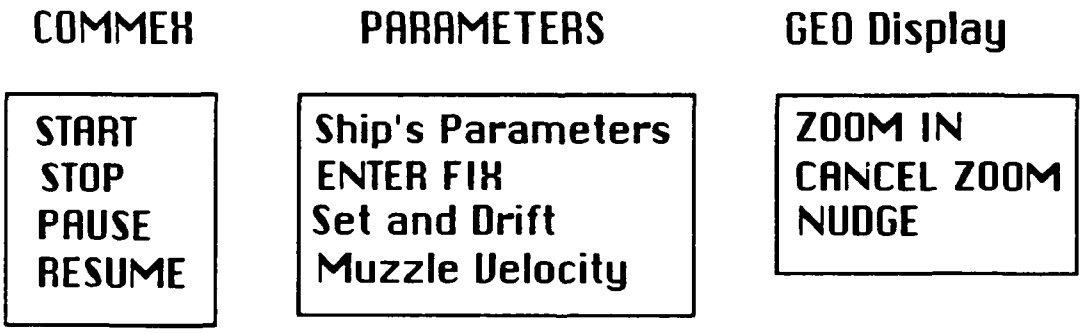

Figure A-8. Menu Bar Headings and Sub-headings

a. COMMEX Sub-Menus

Program execution begins when START is selected and terminates with STOP. PAUSE allows the program to suspend operation. This option is to allow the drill to be interrupted without quitting the program. Time is suspended and all current values retained until execution is resumed with the RESUME command.

b. Parameter Sub-Menus

The PARAMETER sub-menus change data that affects ship position, course, speed, and ordered rudder angle. Set and Drift establishes the effects of current and speed for the geographic display. Muzzle Velocity allows the user to toggle between full and partial charge, normal and high trajectory flight.

c. ZOOM Sub-Menus

These sub-menus affect the magnification and centering of the geographic display. The positioning method, either mouse-click or latitude/longitude, and the amount of magnification can be chosen. Once the position and size are chosen, nudging the center of the zoom is permitted in either course or fine increments.

2. The Dialog Boxes

Dialog boxes appear as the result of selecting a sub-menu. Only the information pertinent to the menu item selected is displayed. Each dialog box is comprised of several sub-elements. Value boxes are for inputting numeric data, either integer or real. Check boxes accept boolean input, that is, the item is checked and evaluated as TRUE or FALSE, not checked. Text boxes display descriptive text helpful to successful use of the dialog boxes. The last type of sub-element is a button. Most dialog boxes request input of some type.

Accompanying the request for information are two buttons labeled Change and Cancel. Both are terminating buttons. Terminating buttons cause the dialog box to erase when a mouse click is sensed on that button. Change causes the new data entered into the value and check boxes to be accepted. Selecting Cancel invalidates any new data that has been entered and provides a way to exit a dialog box that has been inadvertently selected. Once a value has been set in a dialog box, the value does not change until the user re-opens the dialog box and changes that specific value.

a. Parameter Dialog Boxes

Selecting Ship's Parameters causes the left-hand dialog box in Figure A-9 to appear. Changing the values in any or all of the three value boxes takes effect when the Change button is clicked. Rudder convention is positive numbers for right rudder and negative numbers for left rudder. A rudder order with no ordered course at present has no effect. When either the new course or speed is attained by the model, the program beeps, and a message from the system appears to indicate which parameter has been reached. When steady on ordered course, the rudder angle resets to rudder amidships. Deceleration assumes normal bell changes and does not consider propulsion assisted changes, that is, no backing bell has been ordered to decrease the time required for the new speed to be attained. **If in the opinion of the user, ship maneuver's are beyond the scope of the model, a new fix must be entered as promptly as possible.**

Entering a new ship's position requires that the latitude and longitude be entered along with the time of the fix in the HHMM.SS format. That is, 1 hour 5 minutes and 25 seconds are entered as 105.25. The program automatically positions the ship from the fix to its current position using

dead reckoning methods. **Only enter a fix after the ship has steadied on ordered course and speed.** The model uses current course and speed at the time the Enter Fix command is ordered and erroneously positions the ship if it is not steady on ordered course and speed.

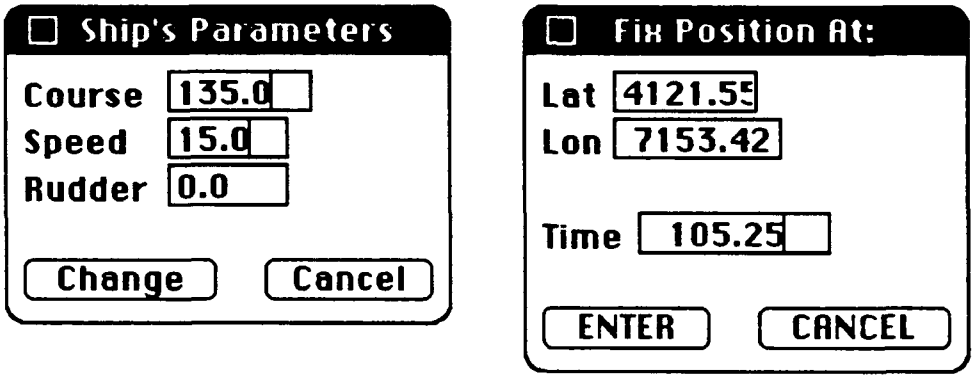

Figure A-9. Dialog Boxes For Ship's Parameters

Set and drift are entered as a direction, degrees true, in the dialog box shown in Figure A-10. The current values of set and drift may be checked at any time by selecting the Set and Drift sub-heading under PARAMETERS. The current value is displayed in the dialog box and selecting the CANCEL button does not alter current settings. The set and drift values may be changed at any time during program execution.

The muzzle velocity dialog box, Figure A-11, has two sets of radio buttons for determining the charge and trajectory. In each case only one button may be selected. After clicking on the Change button, the user is shown a second dialog box to verify that the current muzzle velocity setting is still accurate. At that time, a change to the muzzle velocity may be entered.

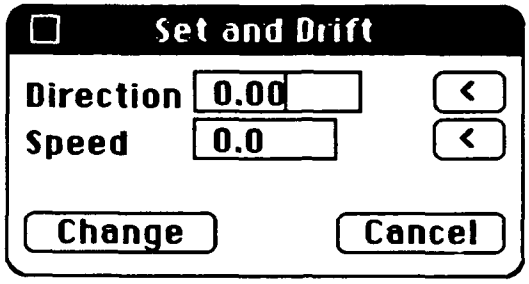

Figure A-10. Set and Drift Dialog Box

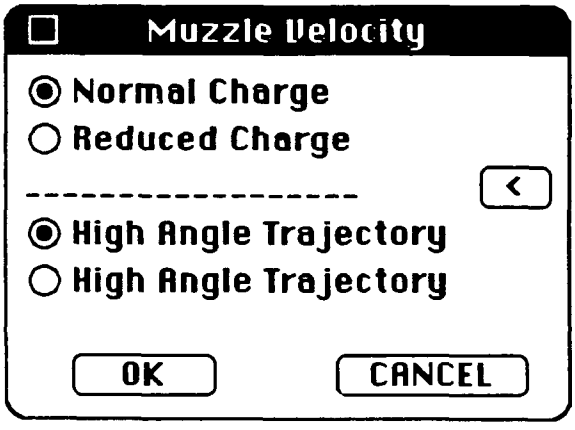

Figure A-11. Muzzle Velocity Setting

b. ZOOM Dialog Boxes

The ZOOM menu causes the area of the geographic display window to shrink. This feature is provided for two reasons. First, it allows the accuracy of a geographic feature to be examined on a more precise scale. Also, the default window dimensions do not permit viewing of target locations. When the 4:1 zoom option is selected (Figure A-12), a flag is set to make the target locations visible. The target the ship is currently engaging blinks. Point targets are represented by a star mark while area targets are represented by a square mark. 2:1 zoom magnifies the viewing area to 25,000

yards square. 4:1 zoom magnifies the viewing area to 12,500 yards square. Canceling the zoom feature returns the window to its default dimensions of 50,000 yards by 50,000 yards.

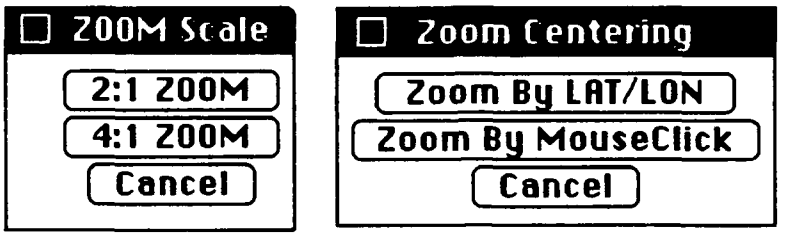

Figure A-12. Dialog Boxes For Zoom

The Zoom Centering allows the user the option of zooming centered on a specific location, such as target location or zooming centered on the last mouse click position. If a zoom operation is scheduled to occur and a boundary of the geographic display is violated, the zoom window shifts to coincide with the boundary of the geographic window. To prevent unnecessary entering and exiting of the zoom option because the positioning of the zoom window is incorrect, a nudge option is provided. The Nudge box, shown in Figure A-13, allows the zoomed portion to be shifted in the four cardinal directions. The total distance of each nudge is based on the zoom scale selected. Additionally, a fine nudge option is available to move the zoom window half of the normal distance. Until the DONE button is selected, the Nudge box will re-draw itself after every nudge operation. The nudge distance for 2:1 zoom is 2500 yards and the 4:1 zoom nudge is 1250 yards.

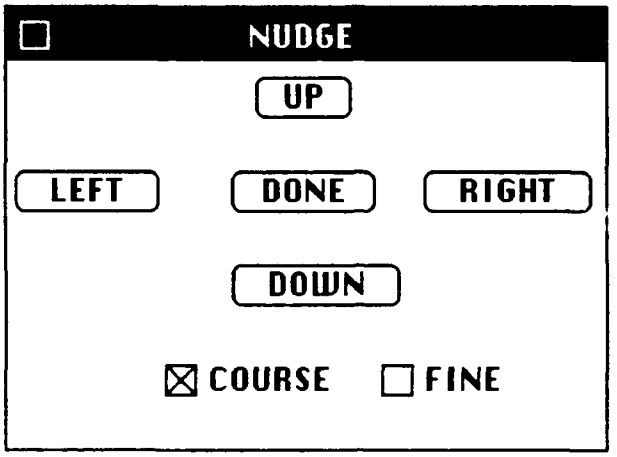

Figure 13. Nudge Dialog Box

APPENDIX B. SAMPLE SHIP'S DATA FILE

```
The following is current ships data as of:

Fri Feb 14 01:57:58 1992

SHIP:          USS Merkin  DDG 666
CO:            CDR Qweeg
XO:            LCDR Annelid
Ship's GFCS:   MK56  mod   4

Spotter's Call Sign:     GS1
Ship's Call Sign:        NAV1

Gun Battery Accuracy, (error is standard deviation):
    range:        50.00
    deflection:   25.00


              Acceleration Data

Time in Seconds it takes To Increase Speed From the
    Low End To the High End of the Speed Range


          KNOTS          SECONDS

        0  TO   5          90.00

        5  TO  10          90.00

       10  TO  15         120.00

       15  TO  20         120.00

       20  TO  25         300.00

       25  TO  30         600.00
```

Deceleration Data

Time in Seconds it takes To Decrease Speed From the
High End To the Low End of the Speed Range

KNOTS			SECONDS
30	TO	25	300.00
25	TO	20	180.00
20	TO	15	60.00
15	TO	10	60.00
10	TO	5	60.00
5	TO	0	60.00

Turn Rate Data

Time in Seconds it takes to turn in Relative Degrees,
At a given Speed and Rudder Angle

TABLE FOR SPEED 5 Knots

Amount Of Turn In Relative Degrees

Rudder Angle	0-15	15-30	30-45	45-60
5	43.00	26.00	25.00	24.00
15	21.50	13.00	12.50	12.00
25	10.50	6.50	6.00	6.00

TABLE FOR SPEED 15 Knots

Amount Of Turn In Relative Degrees

Rudder Angle	0-15	15-30	30-45	45-60
5	43.00	22.00	19.00	19.00
15	21.50	11.00	9.50	9.50
25	10.50	5.50	5.00	5.00

TABLE FOR SPEED 25 Knots

Amount Of Turn In Relative Degrees

Rudder Angle	0-15	15-30	30-45	45-60
5	30.00	17.00	12.50	11.00
15	15.00	8.50	6.50	5.50
25	7.50	4.50	3.50	3.00

APPENDIX C. SAMPLE TARGET LIST

TARGET LIST FOR EXCERCISE test

Created on Mon Feb 17 13:58:52 1992 Using Map File: BlockIs

AREA TARGET LIST DATA

Tgt ID	Latitude	Longitude	Width	Length	Quantity
AB001	4109.560	7136.580	200	100	75
AB002	4109.200	7136.440	500	500	10
AB003	4110.190	7135.590	600	400	20
AB004	4111.200	7135.780	100	100	15
AB005	4111.920	7135.200	100	100	1
AB006	4113.120	7134.580	100	50	45
AB007	4112.330	7133.920	250	100	125

AREA TARGET LIST DESCRIPTION

Tgt ID	Target Type	Protection
AB001	MEN_IN_THE_FIELD	NONE
AB002	ANTENNAE_FIELD	NONE
AB003	MOORED_TROOP_TRANSPORTS	NONE
AB004	HELOS_ON_TARMAC	NONE
AB005	FUEL_DEPOT	NONE
AB006	MEN	IN_TRENCHES
AB007	MEN_IN_THE_FIELD	NONE

POINT TARGET LIST DATA

Tgt ID	Latitude	Longitude	Quantity
BZ001	4110.330	7133.850	1
BZ002	4110.150	7134.820	1
BZ003	4109.170	7133.100	3
BZ004	4110.870	7135.880	1
BZ005	4113.000	7134.000	1
BZ006	4111.000	7135.000	1
BZ007	4112.500	7134.630	1
BZ008	4110.500	7136.250	1
BZ009	4109.500	7136.500	1
BZ010	4113.320	7133.820	1
BZ011	4111.800	7135.450	1
BZ012	4112.330	7134.680	1

POINT TARGET LIST DESCRIPTION

Tgt ID	Target Type	Protection
BZ001	MICROWAVE_TOWER	NONE
BZ002	CONTROL_TOWER	NONE
BZ003	SHORE_GUN_BATTERY	IN_CAVE
BZ004	FUEL_DEPOT	UNDERGROUND
BZ005	COMMAND_CENTER	UNDERGROUND
BZ006	FUEL_PIER	NONE
BZ007	POWER_RELAY_STATION	NONE
BZ008	SAM_SITE	SAND_BAGGED
BZ009	COMMUNICATIONS_CENTER	CEMENT_BUNKER
BZ010	AIRCRAFT_ON_GROUND	NONE
BZ011	GUN_BOAT_IN_HARBOR_ENTRANCE	NONE
BZ012	OBSERVATION_TOWER	NONE

APPENDIX D. PROGRAM CODE

```
MAIN MODULE cylon;

(*-------------------------------------------------------------

PROGRAM NAME:       cylon
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       1/26/92
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the main module of a Naval Gunfire Support Simulation

-------------------------------------------------------------*)

FROM ShipData  IMPORT VerifyShip, ChangeShip;
FROM Plan      IMPORT PlanEx;
FROM ExList    IMPORT COMMEX;
FROM Global    IMPORT GlobalInit, ShipFile, ExData, BeginSeed, Rand;
FROM StdScrn   IMPORT Screen, Lines;
FROM Geo       IMPORT CreateGeoDisplay;
FROM IOMod     IMPORT StreamObj, FileUseType(Output);

VAR
    selection       :    INTEGER;

    FLAG            :    CHAR;

    seedfile        :    StreamObj;

BEGIN
    OUTPUT("   ");
    GlobalInit;

    FLAG := 'Y';

    WHILE FLAG = 'Y'

     selection := 99;
     OUTPUT("Welcome USS ",ShipFile.name," To the NGFS Cylon Spotter");
     OUTPUT(" ");
     OUTPUT(" ");
     OUTPUT("Input the number of the option: ");
     OUTPUT(" ");
     OUTPUT(" ");
     OUTPUT("     1. Verify Ship's Data ");
     OUTPUT("     2. Change Ship's Data ");
     OUTPUT("     3. Plan the next Excercise ");
     OUTPUT("     4. COMMEX ");
     OUTPUT("     5. Create New Geographic Display Chart");
     OUTPUT(" ");
```

```
        OUTPUT("     HIT <return> to TERMINATE this session");

        INPUT(selection);

        CASE selection
             WHEN 1:
                  VerifyShip;

             WHEN 2:
                  ChangeShip;

             WHEN 3:
                  PlanEx;

             WHEN 4:
                  COMMEX;

             WHEN 5:
                  CreateGeoDisplay;

             OTHERWISE
                  FLAG := 'N';

        END CASE;

        NEW(seedfile);
        ASK seedfile TO Open("Seed", Output);
        ASK seedfile TO WriteInt(BeginSeed,32);
        ASK seedfile TO WriteInt(Rand.currentSeed,32);
        ASK seedfile TO Close;
        DISPOSE(seedfile);

    END WHILE;

    OUTPUT(" GEL ");

END MODULE.
```

```
DEFINITION MODULE AreaSpot;

{-------------------------------------------------------------------------

MODULE NAME:        AreaSpot
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/12/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the definition module that controls the spotting for an area
target.

-------------------------------------------------------------------------}


FROM Spotter   IMPORT SpotterObj;
FROM Global    IMPORT TargetRecType, AimPt;

TYPE
    AreaSpotObj = OBJECT(SpotterObj);
     ASK METHOD Prosecute(IN SALVOS:INTEGER; IN mapfamily : STRING);
     ASK METHOD GetAreaTgt(IN mapfamily : STRING);
    END OBJECT;

VAR
    ASpotter    :    AreaSpotObj;
    tgt1        :    TargetRecType;

END MODULE.
```

```
IMPLEMENTATION MODULE AreaSpot;

{-------------------------------------------------------------------------

MODULE NAME:        AreaSpot
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/12/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the module that controls the spotting for an area target.

-------------------------------------------------------------------------}


FROM IOMod      IMPORT StreamObj, FileUseType(Input, Output);
FROM Global     IMPORT TargetRecType, FirstTarget, AimType, Rand, AimPt,
                ExData;
FROM Spotter    IMPORT SpotterObj;
FROM StdScrn    IMPORT Screen, Lines;
FROM UtilMod    IMPORT ClockTimeSecs, ClockRealSecs, Delay;

OBJECT AreaSpotObj;

    ASK METHOD GetAreaTgt(IN mapfamily : STRING);

    VAR
     strm1      :    StreamObj;

    BEGIN
     NEW(tgt1);
     FirstTarget := tgt1;
     NEW(strm1);
     ASK strm1 TO Open ("AreaTgt.list", Input);
     ASK strm1 TO ReadString(tgt1.tgtID);
     ASK strm1 TO ReadReal(tgt1.x);
     ASK strm1 TO ReadReal(tgt1.y);
     ASK strm1 TO ReadInt(tgt1.Aim);
     ASK strm1 TO ReadString(tgt1.type);
     ASK strm1 TO ReadString(tgt1.protection);
     ASK strm1 TO ReadInt(tgt1.width);
     ASK strm1 TO ReadInt(tgt1.length);
     ASK strm1 TO ReadString(tgt1.quantity);
     ASK strm1 TO Close;
     DISPOSE(strm1);
    END METHOD;

    ASK METHOD Prosecute(IN salvos:INTEGER; IN mapfamily:STRING);

    VAR
     dum2           :    CHAR;

     Spotter        :    SpotterObj;

     roundsfired,
```

```
 FFEflag        :    INTEGER;

BEGIN
 NEW(ASpotter);

 ASK ASpotter TO Greeting;
 ASK ASpotter TO GetAreaTgt(mapfamily);
 ASK ASpotter TO GetAimPt;

 OUTPUT("FIRE MISSION  TARGET NUMBER  ", tgt1.tgtID, "OVER");
 Screen(8);
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(3);
 OUTPUT("FIRE MISSION  TARGET NUMBER  ", tgt1.tgtID, " OUT");
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(6);
 OUTPUT("FROM TARGET NUMBER ",tgt1.tgtID);
 OUTPUT("DIRECTION ", AimPt.bearing, "  MAGNETIC");
 OUTPUT(AimPt.rangedir," ",AimPt.range," ",AimPt.elevdir," ",
 AimPt.elev);
 OUTPUT(tgt1.type);
 OUTPUT(salvos," SALVOS IN EFFECT SHORE ADJUST OVER");
 Screen(8);
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(3);
 OUTPUT("FROM TARGET NUMBER ",tgt1.tgtID);
 OUTPUT("DIRECTION ", AimPt.bearing, "  MAGNETIC");
 OUTPUT(AimPt.rangedir," ",AimPt.range," ",AimPt.elevdir," ",
 AimPt.elev);
 OUTPUT(tgt1.type);
 OUTPUT(salvos," SALVOS IN EFFECT SHORE ADJUST OUT");
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(3);
 OUTPUT("GUN TARGET LINE ___ DEGREES MAGNETIC");
 OUTPUT("READY _____  (TIME OF FLIGHT IN SECONDS)");
 OUTPUT("OVER");
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;
```

```
Lines(2);
Screen(6);
OUTPUT("GUN TARGET LINE _ _ _ DEGREES MAGNETIC");
OUTPUT("READY _____  (TIME OF FLIGHT IN SECONDS)");
OUTPUT("BREAK , FIRE , OVER");
Screen(8);
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
ASK ExData TO WriteLn;
Lines(2);

Screen(3);
OUTPUT("FIRE , OUT");
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
ASK ExData TO WriteLn;

FFEflag := 0;
roundsfired := 0;

WHILE (roundsfired < salvos) AND (FFEflag = 0)

     Screen(3);
     OUTPUT("SHOT");
     INPUT(dum2);
     ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
     ASK ExData TO WriteLn;

     Screen(3);
     IF (roundsfired + 1) = salvos
          OUTPUT("SPLASH , ROUNDS COMPLETE , OVER");

     ELSE
          OUTPUT("SPLASH , OUT");

     END IF;
     INPUT(dum2);
     ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
     ASK ExData TO WriteLn;

     IF (roundsfired + 1) < salvos
          Screen(6);
          OUTPUT("RIGHT/LEFT");
          OUTPUT("ADD/DROP");
          OUTPUT("UP/DOWN");
          OUTPUT("OUT");
          Screen(8);
          INPUT(dum2);
          ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
          ASK ExData TO WriteLn;
          Lines(2);

          Screen(3);
          OUTPUT("RIGHT/LEFT");
          OUTPUT("ADD/DROP");
          OUTPUT("UP/DOWN");
```

```
                OUTPUT("OUT");
                INPUT(dum2);
                ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
                ASK ExData TO WriteLn;

            END IF;

            roundsfired := roundsfired + 1;

        END WHILE;

        Screen(6);
        OUTPUT("ROUNDS COMPLETE , END OF MISSION");
        OUTPUT("TARGET NUMBER  ", tgt1.tgtID, "  DESTROYED , OVER");
        Screen(8);
        INPUT(dum2);
        ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
        ASK ExData TO WriteLn;

        Screen(3);
        OUTPUT("ROUNDS COMPLETE , END OF MISSION");
        OUTPUT("TARGET NUMBER  ", tgt1.tgtID, "  DESTROYED , OUT");
        INPUT(dum2);
        ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
        ASK ExData TO WriteLn;

    END METHOD;

END OBJECT;

END MODULE.
```

```
DEFINITION MODULE ExList;

(*-------------------------------------------------------------------------

MODULE NAME:        ExList
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/12/91
LAST MODIFIED:      11/21/91

DESCRIPTION:

This is the definition module that coordinates the event sequence during
an exercise.

-------------------------------------------------------------------------*)


PROCEDURE COMMEX;

END MODULE.
```

```
IMPLEMENTATION MODULE ExList;

(*-------------------------------------------------------------------------

MODULE NAME:        ExList
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/21/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the module that coordinates the event sequence during an
exercise.

-------------------------------------------------------------------------*)


FROM StdScrn        IMPORT Screen, Lines;
FROM IOMod          IMPORT FileUseType(Input,Output),StreamObj,
                    FileExists;
FROM PointSpot      IMPORT PSpotter;
FROM AreaSpot       IMPORT ASpotter;
FROM Global         IMPORT Rand,BeginSeed,EndSeed;

PROCEDURE COMMEX;

TYPE
    ExRecType  =    RECORD
     ex             :    INTEGER;
     salvos         :    INTEGER;
     nextEx         :    ExRecType;
    END RECORD;

VAR
    i,j,
    numex,
    FLAG,
    inpt        :    INTEGER;

    mapfile,
    exname      :    STRING;

    exlist,
    seedfile,
    inl         :    StreamObj;

    TempEx,
    Excercise,
    FirstEx     :    ExRecType;

    drill       :    ARRAY INTEGER OF STRING;

BEGIN
    NEW(exlist);
    ASK exlist TO Open("EX.list", Input);
    ASK exlist TO ReadInt(numex);
```

```
OUTPUT("Select The Number OF The Excercise To Be Run");
NEW(drill, 1..numex);

FOR i:= 1 TO numex
 ASK exlist TO ReadString(drill[i]);
 OUTPUT(i,".   ",drill[i]);
END FOR;
INPUT(i);

WHILE (i < 1) OR (i > numex)
 OUTPUT("The number entered was incorrect,review the list and try
 again");
 FOR i:= 1 TO numex
      OUTPUT(i,".   ",drill[i]);
 END FOR;
 INPUT(i);
END WHILE;

exname := drill[i] + ".ex";


NEW(seedfile);
IF FileExists("Seed") = FALSE
 ASK seedfile TO Open("Seed", Output);
 ASK seedfile TO WriteInt(Rand.originalSeed,32);
 ASK seedfile TO WriteInt(Rand.currentSeed,32);
 ASK seedfile TO Close;
END IF;

ASK seedfile TO Open("Seed", Input);
ASK seedfile TO ReadInt(BeginSeed);
ASK seedfile TO ReadInt(EndSeed);
ASK seedfile TO Close;
DISPOSE(seedfile);

OUTPUT;
OUTPUT;
OUTPUT("The last exercise used ", BeginSeed, "  as the Starting
Random Number Seed.");
OUTPUT;
OUTPUT("Input the number corresponding to your choice for Random
Number Seed.");
OUTPUT("  1.  Re-Use Last Seed:    To re-run last drill Exactly");
OUTPUT("  2.  Choose New Seed:     To re-run an old exercise
Exactly");
OUTPUT("  3.  All Other Cases:     Choose this most of the time.");
OUTPUT;
INPUT(i);

CASE i
 WHEN 1:
      ASK Rand TO SetSeed(BeginSeed);

 WHEN 2:
      OUTPUT("Input Seed - Must be an INTEGER!");
      INPUT(j);
```

```
      ASK Rand TO SetSeed(j);
      BeginSeed := j;

 OTHERWISE
      ASK Rand TO SetSeed(EndSeed);
      BeginSeed := EndSeed;

END CASE;

NEW(in1);
ASK in1 TO Open(exname, Input);
ASK in1 TO ReadString(mapfile);
NEW(PSpotter);
NEW(ASpotter);

FLAG := 0;

Lines(2);
Screen(5);
INPUT(inpt);

Lines(3);
OUTPUT("A review of the scheduled excercise follows:");
Lines(1);
OUTPUT("Using Map Family:  ",mapfile);
Lines(1);

WHILE NOT (ASK in1 eof)

 IF FLAG = 0
      NEW(Excercise);
      Excercise.nextEx := NILREC;
      FirstEx := Excercise;
      FLAG := 1;

 ELSE
      TempEx              := Excercise;
      NEW(Excercise);
      TempEx.nextEx       := Excercise;
      Excercise.nextEx    := NILREC;

 END IF;

 ASK in1 TO ReadInt(Excercise.ex);
 ASK in1 TO ReadInt(Excercise.salvos);

 CASE Excercise.ex

      WHEN 1:
           OUTPUT("Gunfire Excercise Number",Excercise.ex);
           OUTPUT("Point Fire Excercise  ");
           OUTPUT("Number Of Salvos  ",Excercise.salvos);

      WHEN 2:
           OUTPUT("Gunfire Excercise Number",Excercise.ex);
           OUTPUT("Area Fire Excercise  ");
```

```
                OUTPUT("Number Of Salvos  ",Excercise.salvos);

           WHEN 3..10:

           OTHERWISE

         END CASE,

         Lines(1);
         Screen(5);
         INPUT(inpt);
         Lines(1);

       END WHILE;

       FLAG       := 0;
       Excercise  := FirstEx;

       WHILE FLAG = 0

       CASE Excercise.ex

        WHEN 1:
             ASK PSpotter TO
        Prosecute(Excercise.salvos,mapfile);
             IF Excercise.nextEx = NILREC
                  FLAG := 1;
             ELSE
                  Excercise := Excercise.nextEx;
             END IF;

        WHEN 2:
             ASK ASpotter TO
             Prosecute(Excercise.salvos,mapfile);
             IF Excercise.nextEx = NILREC
                  FLAG := 1;
             ELSE
                  Excercise := Excercise.nextEx;
             END IF;

        OTHERWISE

       END CASE;

       END WHILE;

END PROCEDURE;

END MODULE.
```

```
DEFINITION MODULE Geo;

(*-------------------------------------------------------------------------

MODULE NAME:        Geo
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       1/12/92
LAST MODIFIED:      1/26/92

DESCRIPTION:

This is the definition module that creates and maintains the geographic
display.

-------------------------------------------------------------------------*)


FROM Global IMPORT TargetRecType;

PROCEDURE CreateGeoDisplay;

PROCEDURE DConvert(IN mark : REAL; IN system : INTEGER) : REAL;

VAR

    sys         :    INTEGER;

    dlat,
    dlon,
    scale       :    REAL;

    MapFamily   : STRING;

END MODULE.
```

```
IMPLEMENTATION MODULE Geo;

(*-------------------------------------------------------------------

MODULE NAME:        Geo
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       1/12/92
LAST MODIFIED:      2/20/92

DESCRIPTION:

This is the module that creates and maintains the geographic display.

-------------------------------------------------------------------*)


FROM Global     IMPORT TargetRecType;
FROM StdScrn    IMPORT Reminder, Lines;
FROM TgtMkr     IMPORT CreateAreaTgt, CreatePointTgt;
FROM IOMod      IMPORT StreamObj, FileUseType(Output, Input);

(*****************  DConvert  ***********************)

PROCEDURE DConvert(IN mark : REAL; IN system : INTEGER) : REAL;

VAR
    temp,
    deg,
    min,
    sec         :    REAL;

    dummy       : INTEGER;

BEGIN

    IF system = 1
     temp := mark/10000.0;
     dummy := TRUNC(temp);
     deg := FLOAT(dummy);

     temp := (temp - deg) * 100.0;
     dummy := TRUNC(temp);
     min := FLOAT(dummy);
     temp := (temp - min) * 100.0;
     min := min/60.0;
     deg := deg + min;

     dummy := TRUNC(temp);
     sec := FLOAT(dummy);
     sec := sec/3600.0;
     deg := deg + sec;

    ELSE                          (*start with DDMM.mmm*)
     temp := mark/100.0;          (*DD.MMmmm*)
     dummy := TRUNC(temp);
     deg := FLOAT(dummy);
```

```
      temp := (temp - deg) * 100.0;  {00.MMmmm * 100 = MM.mmm}
      min := temp/60.0;
      deg := deg + min;

    END IF;

    RETURN deg;

END PROCEDURE;

{*****************  CreateGeoDisplay  ***********************}

PROCEDURE CreateGeoDisplay;

TYPE
    ShorePtType = RECORD
      x    :    REAL;
      y    :    REAL;
      lat  :    REAL;
      lon  :    REAL;
      next:     ShorePtType;
    END RECORD;

VAR
    lat, lon,
    lon2, dlon2,
    equatorlon,
    temp              :    REAL;

    changeFlag,
    areaFlag,
    ptFlag,
    shoreFlag         :    STRING;

    edit,
    ptctr,
    TotalShorelines,
    TSLctr,
    totshorepts,
    i,j,k             :    INTEGER;

    shorectr          :    ARRAY INTEGER OF INTEGER;

    tempptgt,ptgt,
    cbat,tempctgt,
    firstpt,
    firstcounter      :    TargetRecType;

    FirstShore,
    LastShore,
    StartingShore,
    tempshore,
    shore             :    ShorePtType;

   mapfile,
```

```
    DataFile,
    PtFile,
    ShoreFile       :    StreamObj;

    maps            :     ARRAY INTEGER OF STRING;

    outfile         :    STRING;


BEGIN
    OUTPUT("Check the chart to be used for this excercise and determine
    the system");
    OUTPUT("for latitude and longitude.  It will be in one of the
    following formats");
    OUTPUT("    1. 41-09-45.0 as 410945.0 <DDMMSS.S> ");
    OUTPUT(" or 2. 41-09.75   as 4109.75 <DDMM.mm>   - note they are
    the same number");
    OUTPUT;
    OUTPUT("Decide which system your chart will be using and enter the
    corresponding");
    OUTPUT("number: 1 or 2");
    INPUT(sys);
    OUTPUT;
    OUTPUT("IMPORTANT NOTE!!!  It is not necessary to add the
    identifiers N/S or E/W");
    OUTPUT("                   The geographic reference system uses
    an internal method");
    OUTPUT("                   which depends on the absolute
    difference between the");
    OUTPUT("                   reference point and the entered
    point.     To enter an");
    OUTPUT("                   identifier with the Lat/Lon will
    result in ");
    OUTPUT("                   a system error.");
    OUTPUT;
    OUTPUT("INPUT the LAT/LON of the Lower Left Corner of the Geo
    Display.");
    OUTPUT("This will be used as a reference mark for all other
    positions.");
    OUTPUT;
    OUTPUT("Latitiude? ");
    Reminder(sys);
     NPUT(lat);
    OUTPUT("Longitude? ");
    Reminder(sys);
    INPUT(lon);
    Lines(2);
    OUTPUT("INPUT the Longitude 10,000 yds to the right of the
    Reference Longitude");
    OUTPUT("This longitude will be used to determine the scaling
    factor for");
    OUTPUT("accurate longitude placement.");
    OUTPUT("Scaling Longitude? ");
    Reminder(sys);
    INPUT(lon2);
```

```
    dlat := DConvert(lat, sys);
    dlon := DConvert(lon, sys);
    dlon2 := DConvert(lon2, sys);

    OUTPUT("converted lat is:      ",dlat);
    OUTPUT("converted lon is:      ",dlon);
    OUTPUT("converted scale lon is:",dlon2);

{*********************************************************************
    Function to calculate the scaling factor to account for the amount
of contraction between two longitudes as a pole is approached.

At the equator:
    1 min of longitude = 1 min of latitude = 1 nautical mile = 2000yds
    10,000 yds = 5 min of lon or lat at the equator =>
    60 min of lat/lon = 1 degree of lat/lon
  => 5 min * (1 degree/ 60 min) = 0.083333

As a pole is approached, 1 min lon < 1 min of lat = 2000yds.
Therefore, a constant displacement in yards from a reference longitude
increases in displacement measured in degrees of longitude.

ex:  at longitude 70.000 W a 10,000 yard displacement at the   equator
yields 0.083333 degrees longitude displacement at 41.000 N     (Long
Island Sound) yields 0.10967 degrees longitude displacement.

    This is an increase of 1.316   = SCALE
                  = (ref lon - scaling    lon)/ 0.083333

    To calc the distance in yards from the reference for  a chart
    location =>

                        (ref lon - lon of tgt)          10,000
     distance     =     ----------------------    *    ---------
                              scale                     0.083333

substituting in scale yields =>

                        (ref lon - lon of tgt)
                =     ------------------------------    *   10,000
                      (ref lon - lon of 10k displ)

*********************************************************************}

    scale :=10000.0 / ABS(dlon2 - dlon);

    OUTPUT("scaling factor is  ",scale);

    Lines(2);
    OUTPUT(" Input the name of the Map Family You are creating.");
    OUTPUT("A Map Family will create files with the same name, but
    different extensions");
    OUTPUT("for the appropriate routines.  This is an aid to simplify
    bookkeeping by the user");
    OUTPUT("as a list of existing maps will offered, and only the base
    file name will be ");
```

```
    OUTPUT("required. ");
    OUTPUT;
    OUTPUT("    Limit the name to 8 characters to avoid operating
    system conflicts.");
    OUTPUT;

    OUTPUT("Input the Map Family Name");
    INPUT(MapFamily);
    outfile := MapFamily + ".data";

    NEW(mapfile);
    ASK mapfile TO Open("map.lis", Input);
    ASK mapfile TO ReadInt(j);
    NEW(maps, 1..j+1);

    FOR k := 1 TO j
     ASK mapfile TO ReadString(maps[k]);
    END FOR;

    k:= k+1;
    maps[k] := MapFamily;
    j := j+1;

    ASK mapfile TO Close;
    ASK mapfile TO Open("map.lis", Output);

    ASK mapfile TO WriteInt(j,3);
    ASK mapfile TO WriteLn;

    FOR k := 1 TO j
     ASK mapfile TO WriteString(maps[k]);
     ASK mapfile TO WriteLn;
    END FOR;

    ASK mapfile TO Close;
    DISPOSE(mapfile);

{*************************************************************************
*****          Creating a data file that stores pertinent map family
*****          data in following order:
*****
*****          1.   lat/lon system
*****          2.   reference lat
*****          3.   reference lon
*****          4.   scaling lon
*****          5.   reference lat in decimal form
*****          6.  reference lon in decimal form
*****          7.   scaling lon in decimal form
*****          8.   scaling factor applied to lon differnce
*****
*****          the system procedure FileExists will be called to
*****          determine if a point file exists, if the file does not
*****          exist, a notice to the user will be issued, and program
*****          execution will continue.
*****
*************************************************************************}
```

```
    NEW(DataFile);

    ASK DataFile TO Open(outfile,Output);
    ASK DataFile TO WriteInt(sys,4);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(lat,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(lon,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(lon2,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(dlat,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(dlon,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(dlon2,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO WriteReal(scale,12,6);
    ASK DataFile TO WriteLn;

    ASK DataFile TO Close;
    DISPOSE(DataFile);


{*****************  SHORE POINTS  ************************}

OUTPUT;
OUTPUT;
OUTPUT("How many seperate graphic objects (shorelines) do you wish to
enter?");
INPUT(TotalShorelines);
OUTPUT;
OUTPUT;

IF TotalShorelines > 0
    OUTPUT("*****  Notes On Entering Shoreline Points  *****");
    OUTPUT("1. After entering each point you will be asked of you wish
    to enter another");
    OUTPUT("   point on the current shoreline. When you are through
    with the current");
    OUTPUT("   shoreline, enter N at the prompt. You will then be
    prompted for the points");
    OUTPUT("   for the next shoreline.  When you input N for the last
    shoreline, you will");
    OUTPUT("   prompted for targets.");
    OUTPUT;
    OUTPUT("2. If the geographic object is an island or an enclosed
    object, ensure the last");
```

```
    OUTPUT("   point entered is the same as the first point to
    close off the polygon.");
    OUTPUT;
END IF;

NEW(shorectr, 0..TotalShorelines);
NEW(shore);
StartingShore := shore;
TSLctr := 0;

WHILE TSLctr < TotalShorelines

    TSLctr := TSLctr + 1;
    shoreFlag := "Y";
    changeFlag := "N";
    shorectr[TSLctr] := 0;
    FirstShore := shore;
    WHILE (shoreFlag = "Y") OR (shoreFlag = "y")
     shorectr[TSLctr] := shorectr[TSLctr] +1;
     OUTPUT;
     OUTPUT;
     OUTPUT("Input the Latitude of shore point", shorectr[TSLctr], " ");
     Reminder(sys);
     INPUT(shore.lat);
     shore.y := ABS(DConvert(shore.lat, sys) - dlat) * 120000.0;
     OUTPUT("Input the Longitude of shore point",shorectr[TSLctr],"  ");
     Reminder(sys);
     INPUT(shore.lon);
     shore.x := ABS(DConvert(shore.lon, sys) - dlon) * scale;
     shore.next := NILREC;

     OUTPUT("Another shore point ? <Y/N>");
     INPUT(shoreFlag);

     IF (shoreFlag = "Y") OR (shoreFlag = "y")
          tempshore := shore;
          NEW(shore);
          tempshore.next := shore;
     ELSE
          LastShore := shore;
          changeFlag := "Y";
     END IF;

    END WHILE;

    WHILE (changeFlag = "Y") OR (changeFlag = "y")

     OUTPUT;
     OUTPUT;
     OUTPUT("You have input ",shorectr[TSLctr]," points and they are as
     follows:");
     OUTPUT;
     OUTPUT("Pt no.        lat           lon");
     i := 1;
     shore := FirstShore;
     WHILE i <= shorectr[TSLctr]
```

```
          OUTPUT("  ",i,"     ",shore.lat,"     ",shore.lon);
          shore := shore.next;
          i := i + 1;
     END WHILE;

     OUTPUT;
     OUTPUT;
     OUTPUT("Would You like to Change A Point? <Y/N>");
     INPUT(changeFlag);
     OUTPUT;

     IF (changeFlag = "Y") OR (changeFlag = "y")
          OUTPUT;
          OUTPUT;
          OUTPUT("INPUT the number of the record you wish to edit");
          INPUT(edit);
          OUTPUT;

          i := 1;
          shore := FirstShore;
          WHILE i < edit
               i := i + 1;
               shore := shore.next;
          END WHILE;

          OUTPUT("  ",i,"     ",shore.lat,"     ",shore.lon);
          OUTPUT("Input the new Latitude ");
          Reminder(sys);
          INPUT(shore.lat);
          OUTPUT("Input the new Longitude ");
          Reminder(sys);
          INPUT(shore.lon);
          shore.y := ABS(DConvert(shore.lat, sys) - dlat)*120000.0;
          shore.x := ABS(DConvert(shore.lon, sys) - dlon) * scale;

     END IF;

    END WHILE;

    IF TSLctr < TotalShorelines
     tempshore := LastShore;
     NEW(shore);
     tempshore.next := shore;
    END IF;


END WHILE;

{*****  counting up the total number of shorepoints entered  *****}

i := 1;
totshorepts := 0;

WHILE i<= TotalShorelines
    totshorepts := totshorepts + shorectr[i];
    i := i + 1;
```

```
END WHILE;

{*********************************************************************

creating shorepoints file in the following order:
    1. number of shore objects
    2. total number of shore objects
    3. breakdown of number of pts for each object, each on its own line
    4. listing of all of the points.

*********************************************************************}

IF totshorepts > 0
    OUTPUT("Here in creating map shore files");
    outfile := MapFamily + ".shore";
    OUTPUT("the outfile name is  ", outfile);
    NEW(ShoreFile);
    ASK ShoreFile TO Open(outfile,Output);

    ASK ShoreFile TO WriteInt(TotalShorelines,4);
    ASK ShoreFile TO WriteLn;
    ASK ShoreFile TO WriteInt(totshorepts,6);
    ASK ShoreFile TO WriteLn;

    i := 1;
    WHILE i<= TotalShorelines
     ASK ShoreFile TO WriteInt(shorectr[i],5);
     ASK ShoreFile TO WriteLn;
     i := i + 1;
    END WHILE;

    i := 0;
    shore := StartingShore;

    WHILE i < totshorepts

     ASK ShoreFile TO WriteReal(shore.x,12,6);
     ASK ShoreFile TO WriteLn;

     ASK ShoreFile TO WriteReal(shore.y,12,6);
     ASK ShoreFile TO WriteLn;

     ASK ShoreFile TO WriteReal(shore.lat,12,6);
     ASK ShoreFile TO WriteLn;

     ASK ShoreFile TO WriteReal(shore.lon,12,6);
     ASK ShoreFile TO WriteLn;

     shore := shore.next;
     i := i +1;

    END WHILE;

    ASK ShoreFile TO Close;
    DISPOSE(ShoreFile);
```

```
END IF;

CreateAreaTgt;
CreatePointTgt;


END PROCEDURE;

END MODULE.
```

```
DEFINITION MODULE Global;

(*-------------------------------------------------------------------------

MODULE NAME:        Global
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/7/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the definition module that maintains the variables required to
be set and initialized at time of program execution.

-------------------------------------------------------------------------*)


FROM IOMod IMPORT StreamObj;
FROM RandMod IMPORT RandomObj;

TYPE
    AimType         =    (point, area, counterbattery);


    AimPtRecType    =    RECORD
     bearing    :    INTEGER;
     defl       :    INTEGER;
     defldir    :    STRING;
     range      :    INTEGER;
     rangedir   :    STRING;
     elev       :    INTEGER;
     elevdir    :    STRING;
     Xcoord     :    REAL;
     Ycoord     :    REAL;
     nextRec    :    AimPtRecType;
    END RECORD;


    TargetRecType = RECORD
     tgtID      :    STRING;
     x          :    REAL;
     y          :    REAL;
     lat        :    REAL;
     lon        :    REAL;
     Aim        :    INTEGER;
     type       :    STRING,
     protection:     STRING;
     width      :    INTEGER;
     length     :    INTEGER;
     quantity   :    STRING;
     next       :    TargetRecType;
    END RECORD;

    ShipRecType = RECORD
     name       :    STRING;
     desig      :    STRING;
```

```
      hullno     :    STRING;
      COrank     :    STRING;
      COname     :    STRING;
      XOname     :    STRING;
      XOrank     :    STRING;
      GFCS       :    STRING;
      model      :    INTEGER;
      accel      :    FIXED ARRAY [0  6] OF REAL;
      decel      :    FIXED ARRAY [0  6] OF REAL;
      turnrate   :    FIXED ARRAY [1..3],[1..3],[1..4] OF REAL;
    END RECORD;

    ErrorRecType = RECORD
      range           : REAL;   (error is std deviation in meters)
      deflection      : REAL;
    END RECORD;

PROCEDURE GlobalInit;

VAR
    Xmax,
    Ymax        :    REAL;

    CSbeach,
    CSship      :    STRING;

    ShipAccuracy,
    SpotterBias:     ErrorRecType;

    FirstTarget:     TargetRecType;

    ExData      :    StreamObj;

    ShipFile    :    ShipRecType;

    Rand        :    RandomObj;

    AimPt       :    AimPtRecType;

    TSpeed,
    TRudder     :    FIXED ARRAY [0..3] OF INTEGER;

    THead       :    FIXED ARRAY [0..4] OF INTEGER;

    BeginSeed,
    EndSeed     :    INTEGER;

END MODULE.
```

```
IMPLEMENTATION MODULE Global;

(*-------------------------------------------------------------------------

MODULE NAME:        Global
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/7/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the module that maintains the variables required to be set and
initialized at time of program execution.

-------------------------------------------------------------------------*)


FROM IOMod     IMPORT StreamObj, FileUseType(Input,Output), FileExists;
FROM StdScrn   IMPORT Screen;

PROCEDURE CreateShipDataFile;

    VAR
     ShipDataFile        :    StreamObj;

     i,j,k               :    INTEGER;

BEGIN

    OUTPUT("Welcome To The NGFS Cylon Spotter Program");
    OUTPUT(" ");
    OUTPUT("First, I would like a little information about your
    command");
    OUTPUT(" ");
    OUTPUT("Enter Your Ship's Name:     example - Anthrax");
    INPUT(ShipFile.name);
    OUTPUT(" ");
    OUTPUT("Enter Your Ship's Designation:  example - DDG ");
    INPUT(ShipFile.desig);
    OUTPUT(" ");
    OUTPUT("Enter Your Ship's Hull Number:");
    INPUT(ShipFile.hullno);
    OUTPUT(" ");
    OUTPUT("Enter Your CO's name: example - Jones ");
    INPUT(ShipFile.COname);
    OUTPUT(" ");
    OUTPUT("Enter your CO's Rank: example - CDR ");
    INPUT(ShipFile.COrank);
    OUTPUT(" ");
    OUTPUT("Enter Your XO's name: example - Door ");
    INPUT(ShipFile.XOname);
    OUTPUT(" ");
    OUTPUT("Enter your XO's Rank: example - LCDR ");
    INPUT(ShipFile.XOrank);
    OUTPUT(" ");
    OUTPUT("Enter Your Ship's Gunfire Control System: example - MK68");
```

```
INPUT(ShipFile.GFCS);
OUTPUT(" ");
OUTPUT("Enter Your Ship's Gunfire Control System Modification:
example - 16");
INPUT(ShipFile.model);

OUTPUT;
OUTPUT("During this phase of data input you will be prompted for
data in the following order:");
OUTPUT("1. Acceleration rates by the number of seconds it takes the
ship to");
OUTPUT("   accelerate from the low end to the high end of the speed
range.");
OUTPUT;
OUTPUT("2. Deceleration rates by the number of seconds it takes the
ship to");
OUTPUT("   decelerate from the high end to the low end of the speed
range.");
OUTPUT;
OUTPUT("3. Change of heading by the number of seconds it takes the
ship to");
OUTPUT("   change heading from the low end to the high end of the
speed range.");
OUTPUT;

OUTPUT;
OUTPUT("Inputting ACCELERATION Data");
FOR i := 1 TO 6
 OUTPUT;
 OUTPUT("Input the number of seconds it takes to increase speed
 from");
 OUTPUT(i*5-5, "  TO  ",i*5);
 INPUT(ShipFile.accel[i]);
END FOR;

OUTPUT;
OUTPUT("Inputting DECELERATION Data");
FOR i := 6 DOWNTO 1
 OUTPUT;
 OUTPUT("Input the number of seconds it takes to decrease speed
 from");
 OUTPUT(i*5, "  TO  ",i*5-5);
 INPUT(ShipFile.decel[i]);
END FOR;

OUTPUT;
OUTPUT("Inputting Change of Heading Data");
FOR i := 1 TO 3
 FOR j := 1 TO 3
      FOR k := 1 TO 4
           OUTPUT;
           OUTPUT("Input the number of seconds it takes to change
           ships head from");
           OUTPUT(THead[k]-15, "    TO    ",THead[k],
           " degrees relative");
```

```
            OUTPUT("At Speed  ",TSpeed[i],  "  Using ",
            TRudder[j],"  degrees of rudder");
            INPUT(ShipFile.turnrate[i][j][k]);
        END FOR;
    END FOR;
END FOR;

NEW(ShipDataFile);
ASK ShipDataFile TO Open ("Ship.data",Output);

ASK ShipDataFile TO WriteString(ShipFile.name);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.desig);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.hullno);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.COname);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.COrank);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.XOname);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.XOrank);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(ShipFile.GFCS);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteInt(ShipFile.model, 4);
ASK ShipDataFile TO WriteLn;

CSbeach := "GS1";
CSship  := "NAV1";

ShipAccuracy.range       := 50.0;
ShipAccuracy.deflection := 25.0;
SpotterBias.range        := 10.0;
SpotterBias.deflection   := 5.0;

ASK ShipDataFile TO WriteString(CSbeach);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteString(CSship);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteReal(ShipAccuracy.range, 8, 4);
ASK ShipDataFile TO WriteLn;

ASK ShipDataFile TO WriteReal(ShipAccuracy.deflection, 8, 4);
ASK ShipDataFile TO WriteLn;
```

```
    ASK ShipDataFile TO WriteReal(SpotterBias.range, 8, 4);
    ASK ShipDataFile TO WriteLn;

    ASK ShipDataFile TO WriteReal(SpotterBias.deflection, 8, 4);
    ASK ShipDataFile TO WriteLn;

    FOR i := 1 TO 6
     ASK ShipDataFile TO WriteReal(ShipFile.accel[i], 8, 4);
     ASK ShipDataFile TO WriteLn;
    END FOR;

    FOR i := 1 TO 6
     ASK ShipDataFile TO WriteReal(ShipFile.decel[i], 8, 4);
     ASK ShipDataFile TO WriteLn;
    END FOR;

    FOR i := 1 TO 3          {speeds of 5, 15, 25 }
     FOR j:= 1 TO 3          {rudder angles of 5, 15, 25 }
          FOR k:= 1 TO 4     {head change 0-15,15-30,30-45,45-60}
               ASK ShipDataFile TO WriteReal(ShipFile.turnrate[i][j][k],
               8, 4);
               ASK ShipDataFile TO WriteLn;
          END FOR;
     END FOR;
    END FOR;

    ASK ShipDataFile TO Close;
    DISPOSE(ShipDataFile);

END PROCEDURE;

PROCEDURE GlobalInit;

    VAR
     seedfile,
     ShipDataFile   :    StreamObj;

     option,
     i,j,k,
     UpdateFile     :    INTEGER;

     FLAG           :    CHAR;

BEGIN

    NEW(ShipFile);
    NEW(ShipAccuracy);
    NEW(SpotterBias);
    NEW(Rand);
    NEW(ExData);

    ASK ExData TO Open("Excercise.data", Output);

    Xmax := 50000.0;
    Ymax := 50000.0;
```

```
TSpeed[0]  := -5;
TRudder[0] := -5;
FOR i := 1 TO 3
 TSpeed[i]      := TSpeed[i-1] + 10;
 TRudder[i]     := TRudder[i-1] + 10;
 THead[i]       := THead[i-1] + 15;
END FOR;

THead[4] := THead[3] + 15;

IF  FileExists("Ship.data") = FALSE
 CreateShipDataFile;

ELSE
 OUTPUT("Loading data, one moment please");

 NEW(ShipDataFile);
 ASK ShipDataFile TO Open ("Ship.data",Input);

 ASK ShipDataFile TO ReadString(ShipFile.name);
 ASK ShipDataFile TO ReadString(ShipFile.desig);
 ASK ShipDataFile TO ReadString(ShipFile.hullno);
 ASK ShipDataFile TO ReadString(ShipFile.COname);
 ASK ShipDataFile TO ReadString(ShipFile.COrank);
 ASK ShipDataFile TO ReadString(ShipFile.XOname);
 ASK ShipDataFile TO ReadString(ShipFile.XOrank);
 ASK ShipDataFile TO ReadString(ShipFile.GFCS);
 ASK ShipDataFile TO ReadInt(ShipFile.model);
 ASK ShipDataFile TO ReadString(CSbeach);
 ASK ShipDataFile TO ReadString(CSship);
 ASK ShipDataFile TO ReadReal(ShipAccuracy.range);
 ASK ShipDataFile TO ReadReal(ShipAccuracy.deflection);
 ASK ShipDataFile TO ReadReal(SpotterBias.range);
 ASK ShipDataFile TO ReadReal(SpotterBias.deflection);

 FOR i := 1 TO 6
      ASK ShipDataFile TO ReadReal(ShipFile.accel[i]);
 END FOR;

 FOR i := 1 TO 6
      ASK ShipDataFile TO ReadReal(ShipFile.decel[i]);
 END FOR;

 FOR i := 1 TO 3                {speeds of 5, 15, 25 }
      FOR j:= 1 TO 3            {rudder angles of 5, 15, 25}
           FOR k:= 1 TO 4       {heading change 0-15, 15-30,
                                30-45, 45-60}
                ASK ShipDataFile TO ReadReal
           (ShipFile.turnrate[i][j][k]);
           END FOR;
      END FOR;
 END FOR;

 ASK ShipDataFile TO Close;
 DISPOSE(ShipDataFile);
```

```
    END IF;
END PROCEDURE;
END MODULE.
```

```
DEFINITION MODULE gui;

(*-------------------------------------------------------------------------

MODULE NAME:        gui
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/7/91
LAST MODIFIED:      2/11/92

DESCRIPTION:

This is the definition module that creates and defines the graphic user
interface.

-------------------------------------------------------------------------*)


FROM Graphic    IMPORT GraphicLibObj;
FROM IOMod      IMPORT StreamObj, FileUseType(Input, Output), FileExists;
FROM GTypes     IMPORT PointType;
FROM Form       IMPORT DialogBoxObj;
FROM Menu       IMPORT MenuBarObj, MenuObj, MenuItemObj;
FROM Window     IMPORT WindowObj;
FROM Animate    IMPORT DynImageObj, DynAClockObj, DynDClockObj;
FROM Button     IMPORT ButtonObj;
FROM SimMod     IMPORT StartSimulation;
FROM Value      IMPORT ValueBoxObj;
FROM TextBox    IMPORT TextBoxObj;
FROM Check      IMPORT CheckBoxObj;
FROM Graph      IMPORT RDataPtMObj, RDataPt, IDataPt;
FROM Meter      IMPORT DigitalDisplayObj;

TYPE
    menutype = OBJECT(MenuBarObj)
     OVERRIDE
          ASK METHOD BeSelec-
    END OBJECT;

    loiterObj = OBJECT(DynImageObj)
     TELL METHOD CREEP;
    END OBJECT;

    ShipObj = OBJECT(DynImageObj)
     TELL METHOD ShowPosit;
     TELL METHOD ChangeCourse;
     TELL METHOD ChangeSpeed;
    END OBJECT;

PROCEDURE InitMenuBar;
PROCEDURE InitGUI;
PROCEDURE ShipInit;
PROCEDURE UpdateShip;
PROCEDURE SetTheClock;


VAR
```

```
mbar            :   menutype;

datawindow,
window          :   WindowObj;

lib,zlib        :   GraphicLibObj;

ship            :   ShipObj;

clock,
timer           :   DynDClockObj;

zbox,zscale,
zposit,
nudgebox,
dialogbox       :   DialogBoxObj;

zboxbutton,
zscalebutton,
zpositbutton,
nudgebutton,
tempbutton,
button          :   ButtonObj;

startitem,
stopitem,
pauseitem,
resumeitem,
paramitem,
zinitem,
nudgeitem,
zoutitem        :   MenuItemObj;
menu            :   MenuObj;

turtle          :   loiterObj;

checkbox        :   CheckBoxObj;

LatVal,
LonVal,
TimeVal,
SpeedVal,
CourseVal,
RudderVal       :   ValueBoxObj;

sys1txtbox,
sys2txtbox      :   TextBoxObj;

temp,dx, dy,
xposit,
yposit,
dlat,dlon,
dlon2,
scalefactor     :   REAL;

time,angle,
```

```
        tsec,deltatime,
        distance,
        fixlon,zlon,
        fixlat,zlat,
        zxlo,zxhi,
        zylo,zyhi,
        zoomfactor,
        nudgefactor     :    REAL;

    {monitored variables set by ship.Translation.x/y}
        latdeg,
        londeg,
        latmin,
        lonmin          :    IDataPt;

        latdecimalmin,
        londecimalmin,
        latsecs,
        lonsecs         :    RDataPt;

        speed,
        course,
        rudder          :    RDataPt;

        oldcourse,
        newcourse,
        newspeed,
        oldspeed        :    REAL;

        speeddial,
        coursedial,
        rudderdial,
        latdial,
        londial,
        latdecmindial,
        londecmindial,
        latmindial,
        lonmindial,
        latsecsdial,
        lonsecsdial     :    DigitalDisplayObj;

        posit           :    PointType;

        MapFamily       :    STRING;

        thr,tmin,
        it,
        system          :    INTEGER;

    END MODULE.
```

```
IMPLEMENTATION MODULE gui;

(*------------------------------------------------------------------------

MODULE NAME:        gui
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/7/91
LAST MODIFIED:      2/11/92

DESCRIPTION:

This is the module that creates and defines the graphic user interface.

------------------------------------------------------------------------*)


FROM UtilMod   IMPORT Delay, ClockRealSecs;
FROM GTypes    IMPORT PointArrayType, LineStyleType(SolidLine);
FROM Line      IMPORT PolylineObj;
FROM IOMod     IMPORT StreamObj,FileUseType(Input);
FROM MathMod   IMPORT pi, COS, SIN;
FROM StdScrn   IMPORT Reminder, Lines;
FROM Geo       IMPORT DConvert;
FROM SimMod    IMPORT StartSimulation, StopSimulation, Interrupt,
               SimTime;
FROM Graph     IMPORT RDataPtMObj, IDataPtMObj;
FROM Image     IMPORT ImageObj;
FROM Global    IMPORT ShipFile,TSpeed,TRudder,THead;

OBJECT menutype;

    ASK METHOD BeSelected;    (*Asynchronous menu handling routine*)

    VAR
     rotateto  :    REAL;

    BEGIN

     NEW(dialogbox);
     NEW(zbox);
     NEW(nudgebox);
     NEW(zscale);
     NEW(zposit);
     NEW(button);
     NEW(nudgebutton);
     NEW(tempbutton);
     NEW(zboxbutton);
     NEW(zscalebutton);
     NEW(zpositbutton);
     NEW(SpeedVal);
     NEW(CourseVal);
     NEW(RudderVal);
     NEW(LonVal);
     NEW(LatVal);
     NEW(sys1txtbox);
     NEW(sys2txtbox);
```

```
NEW(checkbox);
NEW(TimeVal);

CASE ASK LastPicked Id

     WHEN 1:   {COMMEX}

          NEW(turtle);

          ASK startitem       TO Deactivate;
          ASK stopitem        TO Activate;
          ASK pauseitem       TO Activate;
          ASK paramitem       TO Activate;
          ASK zinitem         TO Activate;
          ASK ship            TO StartMotion;
          ASK clock           TO StartMotion;

          TELL turtle TO CREEP;
          TELL ship TO ShowPosit;

          StartSimulation;
          StopSimulation;
          OUTPUT("execution has been halted");

     WHEN 2: {STOP}
          ASK pauseitem TO Deactivate;
          ASK paramitem TO Deactivate;
          HALT;

     WHEN 3: {PAUSE}
          ASK pauseitem       TO Deactivate;
          Interrupt(turtle,"CREEP");
          Interrupt(ship,"ShowPosit");
          ASK resumeitem      TO Activate;

     WHEN 4:
          ASK ship TO StartMotion;
          TELL turtle TO CREEP;
          TELL ship TO ShowPosit;
          ASK resumeitem      TO Deactivate;
          ASK pauseitem  TO Activate;
          StartSimulation;
          StopSimulation;
          OUTPUT("execution has been halted");


     WHEN 5:
          ASK dialogbox TO LoadFromLibrary(lib,"param");
          ASK window TO AddGraphic(dialogbox);

          SpeedVal := ASK dialogbox Child("speed",0);
          CourseVal := ASK dialogbox Child("course",0);
          RudderVal := ASK dialogbox Child("rudder",0);

          oldcourse := course;
          oldspeed  := speed;
```

```
            OUTPUT("old course is  ", oldcourse);

            ASK SpeedVal TO SetValue(speed);
            ASK CourseVal TO SetValue(course);
            ASK RudderVal TO SetValue(rudder);

            ASK dialogbox TO Draw;

            {get data from the dialog box}
            button := ASK dialogbox TO AcceptInput();

            IF ASK button ReferenceName = "okay"

                 SpeedVal := ASK dialogbox Child("speed",0);
                 CourseVal := ASK dialogbox Child("course",0);
                 RudderVal := ASK dialogbox Child("rudder",0);

                 speed := ASK SpeedVal Value();
                 course := ASK CourseVal Value();
                 rudder := ASK RudderVal Value();

                 TELL ship TO ChangeSpeed;
                 TELL ship TO ChangeCourse;

            END IF;

       WHEN 6:
{Zoom from a given location on the geo display.                    }
{ 2:1 zoom gives area 25,000 by 25,000 yards.                      }
{ 4:1 zoom gives area 12,500 by 12,500 yards.                      }
{ Does not zoom outside world size of 50,000 yards an axis.        }
{ If a boundry will be encroached, the zoom backs off and          }
{ picks up the full 25,000 yards from opposite direction.          }

            ASK zbox TO LoadFromLibrary(zlib,"zbox");
            ASK zscale TO LoadFromLibrary(zlib,"zscale");
            ASK zposit TO LoadFromLibrary(zlib,"zposit");

            ASK window TO AddGraphic(zbox);
            ASK zbox TO Draw;
            ASK window TO AddGraphic(zscale);
            ASK zscale TO Draw;

            zscalebutton := ASK zscale TO AcceptInput();

            IF ASK zscalebutton ReferenceName = "zoom2x"
                 zoomfactor := 12500.0;
            ELSE
                 zoomfactor := 6250.0;
            END IF;

            zboxbutton := ASK zbox TO AcceptInput();

            IF ASK zboxbutton ReferenceName = "PositZoom"
                 sysltxtbox := ASK zposit Child("line2",3);
```

```
sys2txtbox := ASK zposit Child("line3",3);

IF system =1
     ASK sys2txtbox TO SetHidden(TRUE);
ELSE
     ASK sys1txtbox TO SetHidden(TRUE);
END IF;

ASK window TO AddGraphic(zposit);
ASK zposit TO Draw;

zpositbutton := ASK zposit TO AcceptInput();

zxhi := 50000.0;
zyhi := 50000.0;

IF ASK zpositbutton ReferenceName="enter"
     LatVal := ASK zposit Child("lat",3);
     LonVal := ASK zposit Child("lon",3);
     zlat := ASK LatVal Value();
     zlon := ASK LonVal Value();
     zlon := ABS(DConvert(zlon,system) - dlon) *
     scalefactor;
     zlat := ABS(DConvert(zlat,system) - dlat) *
     120000.0;
     {resolve latitude, the y direction}

     IF (zlat - zoomfactor) < 0.0
          zylo := 0.0;
          zyhi := zoomfactor*2.0;
     ELSIF (zlat + zoomfactor) > 50000.0
          zyhi := 50000.0;
          zylo := zyhi - zoomfactor*2.0;
     ELSE
          zylo := zlat - zoomfactor;
          zyhi := zlat + zoomfactor;
     END IF;

     {resolve longitude}

     IF (zlon - zoomfactor) < 0.0
          zxlo := 0.0;
          zxhi := zoomfactor*2.0;
     ELSIF (zlon + zoomfactor) > 50000.0
          zxhi := 50000.0;
          zxlo := zyhi - zoomfactor*2.0;
     ELSE
          zxlo := zlon - zoomfactor;
          zxhi := zlon + zoomfactor;
     END IF;

END IF;

ELSE
     ASK window TO SetClickMonitoring(TRUE);
```

```
                    OUTPUT("Click Where You Want The Zoom
                    Centered");
                    OUTPUT("Press <Return> button when done");
                    INPUT(it);

                    zlat := window.ClickY;
                    zlat := zlat * 1.8311;
                    zlon := window.ClickX;
                    zlon := zlon * 1.5259;

                    {resolve latitude, the y direction}

                         IF (zlat - zoomfactor) < 0.0
                              zylo := 0.0;
                              zyhi := zoomfactor*2.0;
                         ELSIF (zlat + zoomfactor) > 50000.0
                              zyhi := 50000.0;
                              zylo := zyhi - zoomfactor*2.0;
                         ELSE
                              zylo := zlat - zoomfactor;
                              zyhi := zlat + zoomfactor;
                         END IF;

                         {resolve longitude}

                         IF (zlon - zoomfactor) < 0.0
                              zxlo := 0.0;
                              zxhi := zoomfactor*2.0;
                         ELSIF (zlon + zoomfactor) > 50000.0
                              zxhi := 50000.0;
                              zxlo := zyhi - zoomfactor*2.0;
                         ELSE
                              zxlo := zlon - zoomfactor;
                              zxhi := zlon + zoomfactor;
                         END IF;
               END IF;


               ASK window TO ZoomIn(zxlo,zylo,zxhi,zyhi);
               ASK zoutitem TO Activate;
               ASK zinitem TO Deactivate;
               ASK nudgeitem TO Activate;

          WHEN 7:
               ASK window TO ZoomIn(0.0,0.0,50000.0,60000.0);
               ASK zoutitem  TO Deactivate;
               ASK zinitem TO Activate;
               ASK nudgeitem TO Deactivate;

          WHEN 8:
               ASK nudgebox TO LoadFromLibrary(zlib,"nudge");
               ASK window TO AddGraphic(nudgebox);
               ASK nudgebox TO SetTranslation(10.0,10.0);
               ASK nudgebox TO Draw;

{get input from the check boxes to determine Fine or Course nudge}
```

```
{ Course nudge = zoomfactor * 0.2 ie. 2500 or 1250 yard nudge     }
{ Fine nudge   = zoomfactor * 0.1 ie. 1250 or 625 yard nudge      }

                checkbox := ASK nudgebox Child("fine",5);

                nudgebutton := ASK nudgebox TO AcceptInput();

                WHILE ASK nudgebutton ReferenceName <> "done"

                IF checkbox.Checked = TRUE
                     nudgefactor := 0.1;
                ELSE
                     nudgefactor := 0.2;
                END IF;

                     IF ASK nudgebutton ReferenceName = "left"
                          zxlo := zxlo-nudgefactor*zoomfactor;

                          IF zxlo < 0.0
                               zxlo := 0.0;
                               zxhi := zoomfactor*2.0;
                          ELSE
                               zxhi := zxhi - nudgefactor*zoomfactor;
                          END IF;

                     ELSIF ASK nudgebutton ReferenceName = "right"
                          zxhi := zxhi + nudgefactor * zoomfactor;

                          IF zxhi > 50000.0
                               zxlo := 50000.0 - zoomfactor*2.0;
                               zxhi := 50000.0;
                          ELSE
                               zxlo := zxlo + nudgefactor*zoomfactor;
                          END IF;

                     ELSIF ASK nudgebutton ReferenceName = "up"

                          zyhi := zyhi+nudgefactor*zoomfactor;

                          IF zyhi > 50000.0
                               zylo := 50000.0 - zoomfactor * 2.0;
                               zyhi := 50000.0;
                          ELSE
                               zylo := zylo+nudgefactor*zoomfactor;
                          END IF;

                     ELSIF ASK nudgebutton ReferenceName = "down"
                          zylo := zylo - nudgefactor * zoomfactor;

                          IF zylo < 0.0
                               zylo := 0.0;
                               zyhi := zoomfactor*2.0;
                          ELSE
                               zyhi := zyhi - nudgefactor *
                               zoomfactor;
                          END IF;
```

```
            END IF;

            ASK window TO ZoomIn(zxlo,zylo,zxhi,zyhi);
            ASK nudgebox TO Draw;

            nudgebutton := ASK nudgebox TO AcceptInput();
        END WHILE;


    WHEN 9:
        ASK dialogbox TO LoadFromLibrary(lib,"fix");
        ASK window TO AddGraphic(dialogbox);
        ASK dialogbox TO Draw;

        button := ASK dialogbox TO AcceptInput();

        IF ASK button ReferenceName = "enter"

            LatVal := ASK dialogbox Child("lat",6);
            LonVal := ASK dialogbox Child("lon",6);
            TimeVal := ASK dialogbox Child("time",6);

            fixlat := ASK LatVal Value();
            fixlon := ASK LonVal Value();
            time := ASK TimeVal Value();

            xposit := ABS(DConvert(fixlon,system) - dlon) *
            scalefactor;
            yposit := ABS(DConvert(fixlat,system) - dlat) *
            120000.0;

            thr := TRUNC(time/100.0);
            time := time - FLOAT(thr)*100.0;
            tmin := TRUNC(time);
            time := time - FLOAT(tmin);
            tsec := time * 100.0;
            tsec := tsec + FLOAT(tmin)*60.0 + FLOAT(thr)*3600.0;

            time := SimTime;
            deltatime := time - tsec;
            distance := deltatime * speed * 5.0/9.0;
            angle := -1.0 * (course - 90.0)*pi/180.0;
            dx := COS(angle)*distance;
            dy := SIN(angle)*distance;
            xposit := xposit + dx;
            yposit := yposit + dy;

            ASK ship TO SetTranslation(xposit,yposit);

        END IF;

    OTHERWISE

END CASE;
```

```
    END METHOD;
END OBJECT;
{*****************************************************************}
OBJECT loiterObj;
    TELL METHOD CREEP;
    BEGIN
     SetTheClock;
     WAIT DURATION 70000.0;   {wait 20 hours}
     ON INTERRUPT
     END WAIT;
    END METHOD;
END OBJECT;

{*****************************************************************}
OBJECT ShipObj;
    TELL METHOD ShowPosit;
    VAR
     templat,
     templon        :    REAL;

     i              :    INTEGER;
    BEGIN
    LOOP
    {monitored variables set by ship.Translation.x/y                }
    {latdeg,londeg,latmin,lonmin                    :    IDataPt    }
    {latdecimalmin,londecimalmin,latsecs,lonsecs    :    RDataPt    }

     templat := ship.Translation.y / 2000.0;
                  {convert yards to minutes of latitude}
     templat := templat/60.0;
                  {convert minutes to decimal degrees }
     templat := dlat + templat;
     latdeg := TRUNC(templat);
     latdecimalmin := (templat - FLOAT(latdeg)) * 60.0;

     templon := ship.Translation.x/scalefactor;
     templon := dlon - templon;
     londeg := TRUNC(templon);
     londecimalmin := (templon - FLOAT(londeg))*60.0;
```

```
 IF system = 1  {need to convert minutes into seconds}

      latmin := TRUNC(latdecimalmin);
      latsecs := (latdecimalmin - FLOAT(latmin)) * 60.0;
      lonmin := TRUNC(londecimalmin);
      lonsecs := (londecimalmin - FLOAT(lonmin)) * 60.0;

 END IF;

{updates ship posit every x number of seconds                    }
{chose 5 seconds as a reasonable number and not to bog down      }
{the computer with too many updates that wouldn't be used        }

 WAIT DURATION 5.0;

 ON INTERRUPT
      EXIT;

 END WAIT;

END LOOP;

END METHOD;

TELL METHOD ChangeSpeed;

VAR
 direction,
 tempspeed,
 incrspeed      :    REAL;

 rotateto       :    REAL;

BEGIN
 direction      := speed - oldspeed;
 tempspeed      := speed;
 speed          := oldspeed;
{speed is the monitored value, so to keep updated speed      }
{displayed, speed must be the changing variable in the method}

 IF direction > 0.0      {ship is accelerating}

 WAIT DURATION 5.0;
 ON INTERRUPT
 END WAIT;

      WHILE speed < tempspeed

           IF speed < 5.0
                incrspeed := 5.0/ShipFile.accel[1];
                {increased by kts/sec}
           ELSIF speed < 10.0
                incrspeed := 5.0/ShipFile.accel[2];
                {increased by kts/sec}
           ELSIF speed < 15.0
```

```
                incrspeed := 5.0/ShipFile.accel[3];
                {increased by kts/sec}
            ELSIF speed < 20.0
                incrspeed := 5.0/ShipFile.accel[4];
                {increased by kts/sec}
            ELSIF speed < 25.0
                incrspeed := 5.0/ShipFile.accel[5];
                {increased by kts/sec}
            ELSE
                incrspeed := 5.0/ShipFile.accel[6];
                {increased by kts/sec}
            END IF;

            speed := speed + incrspeed*5.0;
                {update for 5 second interval}
            ASK ship TO SetSpeed(speed*5.0/9.0);

            WAIT DURATION 5.0;
            ON INTERRUPT
            END WAIT;

        END WHILE;

        speed := tempspeed;
        ASK ship TO SetSpeed(speed*5.0/9.0);
    ELSE

    WAIT DURATION 5.0;
    ON INTERRUPT
    END WAIT;

        WHILE speed > tempspeed

            IF speed < 5.0
                incrspeed := 5.0/ShipFile.decel[1];
                {increased by kts/sec}
            ELSIF speed < 10.0
                incrspeed := 5.0/ShipFile.decel[2];
                {increased by kts/sec}
            ELSIF speed < 15.0
                incrspeed := 5.0/ShipFile.decel[3];
                {increased by kts/sec}
            ELSIF speed < 20.0
                incrspeed := 5.0/ShipFile.decel[4];
                {increased by kts/sec}
            ELSIF speed < 25.0
                incrspeed := 5.0/ShipFile.decel[5];
                {increased by kts/sec}
            ELSE
                incrspeed := 5.0/ShipFile.decel[6];
                {increased by kts/sec}
            END IF;

            speed := speed - incrspeed*5.0;
                {update for 5 second interval}
            ASK ship TO SetSpeed(speed*5.0/9.0);
```

```
            WAIT DURATION 5.0;
            ON INTERRUPT
            END WAIT;

       END WHILE;

       speed := tempspeed;
       ASK ship TO SetSpeed(speed*5.0/9.0);
  END IF;

IF direction = 0.0
ELSE
  ASK window TO Beep;
  OUTPUT("At ordered speed");
END IF;

END METHOD;

TELL METHOD ChangeCourse;

VAR
  i,j,k          :    INTEGER;

  rotateto,
  sumchange,
  tempcourse,
  incrchange,
  totchange      :    REAL;

BEGIN

  OUTPUT("old course is   ",oldcourse);
  OUTPUT("new course is   ",course);
  OUTPUT("x  ",ship.Translation.x);
  OUTPUT("y  ",ship.Translation.y);

  IF rudder < 0.0          {turning with left rudder}
       IF course < oldcourse
            totchange := oldcourse - course;
       ELSE
            totchange := oldcourse + (360.0 - course);
       END IF;
  ELSE                  {turning with right rudder}
       IF course < oldcourse
            totchange := course + (360.0 - oldcourse);
       ELSE
            totchange := course - oldcourse;
       END IF;
  END IF;

  tempcourse := course;          { final stop value }
  course := oldcourse;

WAIT DURATION 5.0;
ON INTERRUPT
```

```
END WAIT;

WHILE sumchange < totchange

 IF speed < 10.0
      i:=1;
 ELSIF speed < 20.0
      i := 2;
 ELSE
      i := 3;
 END IF;

 IF ABS(rudder) < 10.0
      j := 1;
 ELSIF ABS(rudder) < 20.0
      j := 2;
 ELSE
      j := 3;
 END IF;


 IF totchange < 15.0
      k := 1;
 ELSIF totchange < 30.0
      k := 2;
 ELSIF totchange < 45.0
      k := 3;
 ELSE
      k := 4;
 END IF;

 incrchange := 15.0/ShipFile.turnrate[i][j][k];
 sumchange := sumchange + incrchange;

 IF rudder < 0.0
      course := course - incrchange;
      IF course < 0.0
           course := 360.0 + course;
      END IF;
 ELSE
      course := course + incrchange;
      IF course >= 360.0;
           course := course - 360.0;
      END IF;
 END IF;

 rotateto := -1.0 * (course - 90.0) * pi / 180.0;

 ASK ship TO SetAutoRotation(TRUE);
 ASK ship TO SetCourse(rotateto);
 ASK ship TO SetRotation(rotateto);

 WAIT DURATION 1.0;
 ON INTERRUPT
 END WAIT;
```

```
    END WHILE;

    WAIT DURATION 4.0;
    ON INTERRUPT
    END WAIT;

    course := tempcourse;          { final stop value }
    rotateto := -1.0 * (course - 90.0) * pi / 180.0;

    ASK ship TO SetAutoRotation(TRUE);
    ASK ship TO SetCourse(rotateto);
    ASK ship TO SetRotation(rotateto);

    IF totchange = 0.0
    ELSE
     ASK window TO Beep;
     OUTPUT("On ordered course");
    END IF;
    OUTPUT("x  ",ship.Translation.x);
    OUTPUT("y  ",ship.Translation.y);
    rudder := 0.0;


    END METHOD;

END OBJECT;

{*********************************************************************}

PROCEDURE ShipInit;

VAR
    shoreline           :    ARRAY INTEGER OF PolylineObj;

    TheFile,
    shorefile           :    STRING;

    InFile              :    StreamObj;

    ctr,
    numberOfImages,
    dummy,
    i,j,
    numpoints           :    INTEGER;

    temp                :    REAL;

    shorepts            :    ARRAY INTEGER OF PointArrayType;

    shorectr            :    ARRAY INTEGER OF INTEGER;

    maps                :    ARRAY INTEGER OF STRING;

    mapfile,
    datafile            :    StreamObj;
```

```
    lattext,
    lontext              :    ImageObj;


BEGIN

    NEW(window);
    ASK window TO SetTitle("Cylon Spotter Geographical Display");
    ASK window TO SetSize(50.0, 50.0);
    ASK window TO SetTranslation(50.0,50.0);
    ASK window TO ShowWorld(0.0,0.0,50000.0,50000.0);
    ASK window TO Draw;

    NEW(datawindow);
    ASK datawindow TO SetTitle("Data Display");
    ASK datawindow TO SetSize(40.0, 40.0);
    ASK datawindow TO SetTranslation(10.0,10.0);
    ASK datawindow TO ShowWorld(0.0,0.0,100.0,100.0);
    ASK datawindow TO Draw;

    NEW(lib);
    ASK lib TO ReadFromFile("Gui.lib");

    NEW(zlib);
    ASK zlib TO ReadFromFile("zoom.lib");

    NEW(ship);
    ASK ship TO LoadFromLibrary(lib,"ship");

    NEW(clock);
    ASK clock TO LoadFromLibrary(zlib,"extime");
    ASK clock TO SetTimeScale(1.0/3600.0);
    ASK clock TO SetTranslation(10.0,85.0);

    NEW(timer);
    ASK timer TO LoadFromLibrary(zlib,"timer");
    ASK timer TO SetTranslation(55.0,85.0);

    ASK window TO AddGraphic(ship);
    ASK datawindow TO AddGraphic(clock);
    ASK datawindow TO AddGraphic(timer);
    ASK ship TO Scale(0.2,0.2);
    ASK clock TO Scale(0.35,0.15);
    ASK timer TO Scale(0.35,0.15);
    ASK clock TO Draw;
    ASK timer TO Draw;

    NEW(speeddial);
    NEW(coursedial);
    NEW(rudderdial);
    NEW(latdial);
    NEW(londial);
    NEW(lattext);
    NEW(lontext);

    ASK speeddial TO LoadFromLibrary(lib, "speeddisplay");
```

```
ASK coursedial TO LoadFromLibrary(lib, "coursedisplay");
ASK rudderdial TO LoadFromLibrary(lib, "rudderdisplay");
ASK latdial TO LoadFromLibrary(lib, "latdeg");
ASK londial TO LoadFromLibrary(lib, "londeg");
ASK lattext TO LoadFromLibrary(lib, "lattext");
ASK lontext TO LoadFromLibrary(lib, "lontext");

ASK speeddial TO SetTranslation(5.0, 65.0);
ASK datawindow TO AddGraphic(speeddial);
ASK speeddial TO Draw;

ASK coursedial TO SetTranslation(37.5, 65.0);
ASK datawindow TO AddGraphic(coursedial);
ASK coursedial TO Draw;

ASK rudderdial TO SetTranslation(70.0, 65.0);
ASK datawindow TO AddGraphic(rudderdial);
ASK rudderdial TO Draw;

ASK latdial TO SetTranslation(10.0, 32.5);
ASK datawindow TO AddGraphic(latdial);
ASK latdial TO Draw;

ASK londial TO SetTranslation(10.0, 1.0);
ASK datawindow TO AddGraphic(londial);
ASK londial TO Draw;

ASK lattext TO SetTranslation(45.0, 55.0);
ASK datawindow TO AddGraphic(lattext);
ASK lattext TO Draw;

ASK lontext TO SetTranslation(45.0, 23.5);
ASK datawindow TO AddGraphic(lontext);
ASK lontext TO Draw;


ASK GETMONITOR(speed, RDataPtMObj) TO SetGraph(speeddial);
ASK GETMONITOR(course, RDataPtMObj) TO SetGraph(coursedial);
ASK GETMONITOR(rudder, RDataPtMObj) TO SetGraph(rudderdial);
ASK GETMONITOR(londeg, IDataPtMObj) TO SetGraph(londial);
ASK GETMONITOR(latdeg, IDataPtMObj) TO SetGraph(latdial);



NEW(mapfile);
ASK mapfile TO Open("map.lis", Input);
ASK mapfile TO ReadInt(i);
NEW(maps, 1..i);
FOR j:= 1 TO i
 ASK mapfile TO ReadString(maps[j]);
END FOR;

Lines(2);
OUTPUT("Input the number of the Map Family for this   Excercise");
Lines(1);
```

```
FOR j := 1 TO i
 OUTPUT(j,".   ",maps[j]);
END FOR;
INPUT(j);
WHILE j>i
 Lines(2);
 OUTPUT("The number you have entered is Inconsistent with the Map
 listing.");
 OUTPUT("Try again");
 Lines(2);
 FOR j:= 1 TO i
      OUTPUT(j,".   ",maps[j]);
 END FOR;
 INPUT(j);
END WHILE;

MapFamily := maps[j];

TheFile := MapFamily + ".data";
NEW(datafile);
ASK datafile TO Open(TheFile,Input);
ASK datafile TO ReadInt(system);
ASK datafile TO Close;
DISPOSE(datafile);

IF system = 2
 NEW(latdecmindial);
 NEW(londecmindial);

 ASK latdecmindial TO LoadFromLibrary(lib, "latdecmin");
 ASK londecmindial TO LoadFromLibrary(lib, "londecmin");

 ASK latdecmindial TO SetTranslation(40.0, 32.5);
 ASK datawindow TO AddGraphic(latdecmindial);
 ASK latdecmindial TO Draw;

 ASK londecmindial TO SetTranslation(40.0, 1.0);
 ASK datawindow TO AddGraphic(londecmindial);
 ASK londecmindial TO Draw;

 ASK GETMONITOR(londecimalmin, RDataPtMObj) TO SetGraph
 (londecmindial);
 ASK GETMONITOR(latdecimalmin, RDataPtMObj) TO SetGraph
 (latdecmindial);

ELSE
 NEW(latmindial);
 NEW(lonmindial);
 NEW(latsecsdial);
 NEW(lonsecsdial);

 {minutes}
 ASK latmindial TO LoadFromLibrary(lib, "latmin");
 ASK lonmindial TO LoadFromLibrary(lib, "lonmin");

 ASK latmindial TO SetTranslation(40.0, 32.5);
```

```
 ASK datawindow TO AddGraphic(latmindial);
 ASK latmindial TO Draw;

 ASK lonmindial TO SetTranslation(40.0, 1.0);
 ASK datawindow TO AddGraphic(lonmindial);
 ASK lonmindial TO Draw;

 ASK GETMONITOR(lonmin, IDataPtMObj) TO SetGraph(lonmindial);
 ASK GETMONITOR(latmin, IDataPtMObj) TO SetGraph(latmindial);

 {seconds}
 ASK latsecsdial TO LoadFromLibrary(lib, "latsecs");
 ASK lonsecsdial TO LoadFromLibrary(lib, "lonsecs");

 ASK latsecsdial TO SetTranslation(80.0, 32.5);
 ASK datawindow TO AddGraphic(latsecsdial);
 ASK latsecsdial TO Draw;

 ASK lonsecsdial TO SetTranslation(80.0, 1.0);
 ASK datawindow TO AddGraphic(lonsecsdial);
 ASK lonsecsdial TO Draw;

 ASK GETMONITOR(lonsecs, RDataPtMObj) TO SetGraph(lonsecsdial);
 ASK GETMONITOR(latsecs, RDataPtMObj) TO SetGraph(latsecsdial);

END IF;

NEW(InFile);
shorefile := MapFamily + ".shore";
ASK InFile TO Open(shorefile,Input);
ASK InFile TO ReadInt(numberOfImages);
ASK InFile TO ReadInt(dummy);

NEW(shorectr, 0..numberOfImages);
i := 1;
WHILE i <= numberOfImages
 ASK InFile TO ReadInt(shorectr[i]);
 i := i + 1;
END WHILE;

NEW(shorepts, 0..numberOfImages);
i := 1;
WHILE i <= numberOfImages
 NEW(shorepts[i], 1..shorectr[i]);
 i := i + 1;
END WHILE;

i := 1;
WHILE i <= numberOfImages
 j := 1;
 WHILE j <= shorectr[i]
      ASK InFile TO ReadReal(shorepts[i][j].x);
      ASK InFile TO ReadReal(shorepts[i][j].y);
      ASK InFile TO ReadReal(temp);
      ASK InFile TO ReadReal(temp);
      j := j + 1;
```

```
        END WHILE;
        i := i +1;
      END WHILE;

      i := 1;
      NEW(shoreline, 0..numberOfImages);
      WHILE i <= numberOfImages
        NEW(shoreline[i]);
        i := i+1;
      END WHILE;

      i := 1;
      WHILE i <= numberOfImages
        OUTPUT("Adding image number ",i);
        ASK shoreline[i] TO SetPoints(shorepts[i]);
        ASK shoreline[i] TO SetWidth(50.0);
        ASK shoreline[i] TO SetStyle(SolidLine);

        ASK window TO AddGraphic(shoreline[i]);
        ASK shoreline[i] TO Draw;

        i := i+1;
      END WHILE;

      ASK InFile TO Close;
      DISPOSE(InFile);


END PROCEDURE;

{*************************************************************************}

PROCEDURE InitMenuBar;

BEGIN

      NEW(mbar);
      ASK mbar TO LoadFromLibrary(lib,"menubar");
      ASK window TO AddGraphic(mbar);
      ASK mbar TO Draw;

      startitem  := ASK mbar Descendant("start",1);
      stopitem   := ASK mbar Descendant("stop",2);
      pauseitem  := ASK mbar Descendant("pause",3);
      resumeitem := ASK mbar Descendant("resume",4);
      paramitem  := ASK mbar Descendant("shipparam",5);
      zinitem    := ASK mbar Descendant("zin",6);
      zoutitem   := ASK mbar Descendant("zout",7);
      nudgeitem  := ASK mbar Descendant("nudge",8);

      ASK stopitem        TO Deactivate;
      ASK pauseitem       TO Deactivate;
      ASK resumeitem      TO Deactivate;
      ASK paramitem       TO Deactivate;
      ASK zinitem         TO Deactivate;
      ASK zoutitem        TO Deactivate;
```

```
    ASK nudgeitem        TO Deactivate;

END PROCEDURE;

{*********************************************************************}

PROCEDURE InitGUI;

VAR
    datafile    :    StreamObj;

    infile      :    STRING;

BEGIN

    NEW(dialogbox);
    ASK dialogbox TO LoadFromLibrary(lib,"param");
    ASK window TO AddGraphic(dialogbox);

    ASK dialogbox TO SetLabel("Initial Parameters");
    ASK dialogbox TO Draw;

    {get data from the dialog box}
    button := ASK dialogbox TO AcceptInput();

    IF ASK button ReferenceName = "okay"

     SpeedVal := ASK dialogbox Child("speed",0);
     CourseVal := ASK dialogbox Child("course",0);
     RudderVal := ASK dialogbox Child("rudder",0);

     speed  := ASK SpeedVal  Value();
     course := ASK CourseVal Value();
     rudder := ASK RudderVal Value();
     oldspeed := speed;
     oldcourse := course;

    END IF;

    infile := MapFamily + ".data";
    NEW(datafile);
    ASK datafile TO Open(infile,Input);
    ASK datafile TO ReadInt(system);
    ASK datafile TO ReadReal(temp);
    ASK datafile TO ReadReal(temp);
    ASK datafile TO ReadReal(temp);
    ASK datafile TO ReadReal(dlat);
    ASK datafile TO ReadReal(dlon);
    ASK datafile TO ReadReal(dlon2);
    ASK datafile TO ReadReal(scalefactor);
    ASK datafile TO Close;
    DISPOSE(datafile);

    OUTPUT("Input the Ship's Starting Latitude");
    Reminder(system);
    INPUT(yposit);
```

```
    OUTPUT("Input the Ship's Starting Longitude");
    Reminder(system);
    INPUT(xposit);

    xposit := ABS(DConvert(xposit,system) - dlon)*scalefactor;
    yposit := ABS(DConvert(yposit,system) - dlat)*120000.0;

    ASK ship TO SetTranslation(xposit,yposit);
    ASK ship TO Draw;
    UpdateShip;
    DISPOSE(dialogbox);

END PROCEDURE;

{*********************************************************************}

PROCEDURE UpdateShip;

VAR
    rotateto    :    REAL;

BEGIN

    ASK ship TO StopMotion;

    rotateto := -1.0 * (course - 90.0) * pi / 180.0;

    ASK ship TO SetAutoRotation(TRUE);
    ASK ship TO SetCourse(rotateto);
    ASK ship TO SetSpeed(speed*5.0/9.0);
    ASK ship TO SetRotation(rotateto);
    ASK ship TO StartMotion;

END PROCEDURE;

{*********************************************************************}

PROCEDURE SetTheClock;

VAR
    temp,
    thr,tmin,tsec        :    REAL;

    i,j,
    hr, min, sec         :    INTEGER;

BEGIN

    temp := ClockRealSecs;
    thr := temp/(3600.0*24.0);
    i := TRUNC(thr);
    thr := temp - FLOAT(i)*86400.0;
    hr := TRUNC(thr/3600.0);
    tsec := thr - FLOAT(hr)*3600.0;
    min := TRUNC(tsec/60.0);
    sec := TRUNC(tsec - FLOAT(min)*60.0);
```

```
    ASK clock TO SetTime(hr, min, sec);
    ASK clock TO Update;

END PROCEDURE;

END MODULE.
```

```
DEFINITION MODULE Plan;

(*-------------------------------------------------------------------------

MODULE NAME:        Plan
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      2/16/92

DESCRIPTION:

This is the definition module that plans future exercises and generates
target lists.
-------------------------------------------------------------------------*)


PROCEDURE PlanEx;
PROCEDURE CreateTgtList(IN map : STRING; IN ex : STRING);

END MODULE.
```

```
IMPLEMENTATION MODULE Plan;

{-----------------------------------------------------------------

MODULE NAME:        Plan
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      2/16/92

DESCRIPTION:

This is the module that plans future exercises and generates target
lists.
-----------------------------------------------------------------}


FROM StdScrn   IMPORT Screen, Lines;
FROM IOMod     IMPORT FileUseType(Input,Output),StreamObj, FileExists;
FROM Geo       IMPORT CreateGeoDisplay, MapFamily;
FROM UtilMod   IMPORT DateTime;
FROM Global    IMPORT TargetRecType;

VAR
    i,j,
    numex,
    temp,temp1,
    temp2,temp3,
    MoreFlag,
    scenario    :    INTEGER;

    StopFlag    :    STRING;

    overwrite,
    exname,
    commex      :    STRING;

    mapfile,
    ex,
    out1        :    StreamObj;

    drill,
    maps        :    ARRAY INTEGER OF STRING;

PROCEDURE PlanEx;

BEGIN
    { open listing of pre-existing excercise files }
    NEW(out1);
    OUTPUT("The Following Excercises Are Already Named");
    OUTPUT;
    IF FileExists("EX.list") = TRUE
     ASK out1 TO Open("EX.list", Input);
     ASK out1 TO ReadInt(numex);

     {read the list in to an array }
     IF numex > 0
```

```
      NEW(drill, 1..numex);
      FOR i:= 1 TO numex
           ASK out1 TO ReadString(drill[i]);
           OUTPUT("             ",drill[i]);
      END FOR;
 END IF;
ELSE
 OUTPUT("No Excercise Files Currently Exist");
 ASK out1 TO Open("EX.list", Output);
 numex := 0;
END IF;
ASK out1 TO Close;

{ get name of new excercise }
OUTPUT("Input The Name Of The Excercise You Wish To Create");
OUTPUT("Re-use of previously named file will cause that data to be
lost.");
OUTPUT;
OUTPUT("NOTE:    Keep the name to under 8 characters to prevent ");
OUTPUT("             conflicts with the operating system.");
INPUT(commex);

StopFlag := "N";
FOR i := 1 TO numex
 IF drill[i] = commex
      StopFlag := "Y";
 END IF;
END FOR;

{ storing the new excercise on the list, if not over-written }
IF StopFlag = "N"
 ASK out1 TO Open("EX.list", Output);
 ASK out1 TO WriteInt(numex+1,3);
 ASK out1 TO WriteLn;
 FOR i := 1 TO numex
      ASK out1 TO WriteString(drill[i]);
      ASK out1 TO WriteLn;
 END FOR;
 ASK out1 TO WriteString(commex);
 ASK out1 TO WriteLn;
 ASK out1 TO Close;
END IF;
DISPOSE(out1);

{ opening the file to contain the excercise script }
exname := commex;
NEW(ex);
commex := commex + ".ex";
ASK ex TO Open(commex, Output);

{ opening the file that contains the geographic maps }
NEW(mapfile);
ASK mapfile TO Open("map.lis", Input);
ASK mapfile TO ReadInt(i);
NEW(maps, 1..i);
FOR j:= 1 TO i
```

```
 ASK mapfile TO ReadString(maps[j]);
END FOR;

Lines(2);
OUTPUT("Input the number of the Map Family for this Excercise");
Lines(1);

FOR j := 1 TO i
 OUTPUT(j,".   ",maps[j]);
END FOR;
OUTPUT(i+1,".   Create New Geographic Display Chart.");
INPUT(j);
WHILE (j>i+1) OR (j < 0)
 Lines(2);
 OUTPUT("The number you have entered is Inconsistent with the
Map listing.");
 OUTPUT("Try again");
 Lines(2);
 FOR j:= 1 TO i
      OUTPUT(j,".   ",maps[j]);
 END FOR;
  OUTPUT(i+1,".   Create New Geographic Display Chart.");
 INPUT(j);
END WHILE;
ASK mapfile TO Close;

IF j = i+1
 CreateGeoDisplay;
 ASK mapfile TO Open("map.lis", Output);
 ASK mapfile TO WriteInt(i+1,3);
 ASK mapfile TO WriteLn;

 FOR j := 1 TO i
      ASK mapfile TO WriteString(maps[j]);
      ASK mapfile TO WriteLn;
 END FOR;
 ASK mapfile TO WriteString(MapFamily);
 ASK mapfile TO WriteLn;
 ASK mapfile TO Close;
 DISPOSE(mapfile);

ELSE
 MapFamily := maps[j];
END IF;


ASK ex TO WriteString(MapFamily);
ASK ex TO WriteLn;

MoreFlag := 1;

WHILE MoreFlag > 0

 OUTPUT("THESE OPTIONS ARE TO ASSST IN THE PREPARATION OF");
 OUTPUT("            AN NGFS EXCERCISE.");
 OUTPUT("");
```

```
OUTPUT("      1. Point Fire Excercise");
OUTPUT("      2. Area Fire Excercise");
OUTPUT("      3. Coordinated Illumination Excercise");
OUTPUT("      4. Z-40-G");
OUTPUT("      5. Z-42-G");
OUTPUT("      6. Z-43-G");
OUTPUT("      7. Z-44-G");
OUTPUT("      8. Z-45-G");
OUTPUT("      9. Exit");
OUTPUT("");
Screen(1);
INPUT(scenario);
Lines(2);

CASE scenario
     WHEN 1:
          OUTPUT("INPUT the number of salvos to be fired for this
          Point Fire excercise");
          INPUT(temp);
          ASK ex TO WriteInt(scenario, 3);
          ASK ex TO WriteLn;
          ASK ex TO WriteInt(temp, 3);
          ASK ex TO WriteLn;
          MoreFlag := 0;
          Lines(3);

     WHEN 2:
          OUTPUT("INPUT the number of salvos to be fired for this
          Area Fire excercise");
          INPUT(temp);
          ASK ex TO WriteInt(scenario, 3);
          ASK ex TO WriteLn;
          ASK ex TO WriteInt(temp, 3);
          ASK ex TO WriteLn;
          MoreFlag := 0;
          Lines(3);

     WHEN 3..8:
          MoreFlag := 0;

     OTHERWISE
          MoreFlag := 0;

END CASE;

MoreFlag := 0;
scenario := 9;
Lines(2);
OUTPUT("Another Menu Selection? < Y/N >");
OUTPUT;
INPUT(StopFlag);
IF (StopFlag = "Y") OR (StopFlag = "y")
     MoreFlag := 1;
END IF;
Lines(2);
END WHILE;
```

```
    Screen(5);
    INPUT(temp);
    Lines(5);
    OUTPUT("An excrecise target list will be generated and stored in a
    file ");
    OUTPUT("called ExTgt.list.  Print this list prior to commencing the
    excercise");
    OUTPUT("to allow the plots a chance to create an excercise
    chart.");
    OUTPUT("THIS IS REQUIRED BY THE CYLON SPOTTER");
    Lines(2);
    CreateTgtList(MapFamily, exname);
    Screen(5);
    INPUT(scenario);

    ASK ex TO Close;
    DISPOSE(ex);

END PROCEDURE;

PROCEDURE CreateTgtList(IN map : STRING; IN ex : STRING);

CONST
    format = " ********   ******.***    ******.***     ****   ***
    ******";
    format2 = " ********          ****************************************
    ************************";
    format3 = " ********   ******.***    ******.***            ******";

VAR
    infile,
    outfile         :    StreamObj;

    target          :    ARRAY INTEGER OF TargetRecType;

    str,
    mapfile         :    STRING;

    i,j,
    numtgts         :    INTEGER;

BEGIN
    mapfile := ex + "Tgt.lis";
    NEW(outfile);
    ASK outfile TO Open(mapfile, Output);

    ASK outfile TO WriteString("                    TARGET LIST FOR
    EXCERCISE ");
    ASK outfile TO WriteString(ex);
    ASK outfile TO WriteLn;
    ASK outfile TO WriteLn;
    ASK outfile TO WriteString("Created on ");
    DateTime(str);
    ASK outfile TO WriteString(str);
    ASK outfile TO WriteString("   Using Map File: ");
```

```
ASK outfile TO WriteString(map);
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;

{*********************************************************}

ASK outfile TO WriteString("                          AREA TARGET   LIST
DATA");
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;

NEW(infile);
mapfile := map + ".atgt";
ASK infile TO Open(mapfile,Input);
ASK infile TO ReadInt(numtgts);
ASK outfile TO WriteString(" Tgt ID       Latitude          Longitude
Width  Length     Quantity");
ASK outfile TO WriteLn;
NEW(target, 1..numtgts);
FOR i := 1 TO numtgts
 NEW(target[i]);
 ASK infile TO ReadString(target[i].tgtID);
 ASK infile TO ReadReal(target[i].x);
 ASK infile TO ReadReal(target[i].y);
 ASK infile TO ReadReal(target[i].lat);
 ASK infile TO ReadReal(target[i].lon);
 ASK infile TO ReadString(target[i].type);
 ASK infile TO ReadString(target[i].protection);
 ASK infile TO ReadInt(target[i].width);
 ASK infile TO ReadInt(target[i].length);
 ASK infile TO ReadString(target[i].quantity);
END FOR;
ASK infile TO Close;

FOR i := 1 TO numtgts
 str := SPRINT (target[i].tgtID, target[i].lat, target[i].lon,
target[i].width, target[i].length, target[i].quantity) WITH
format;
 ASK outfile TO WriteString(str);
 ASK outfile TO WriteLn;
END FOR;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteString("                          AREA TARGET   LIST
DESCRIPTION");
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteString(" Tgt ID       Target Type
Protection");
ASK outfile TO WriteLn;

FOR i := 1 TO numtgts
```

```
 str := SPRINT (target[i].tgtID, target[i].type,
target[i].protection) WITH format2;
 ASK outfile TO WriteString(str);
 ASK outfile TO WriteLn;
END FOR;

ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;

{************************************************************}

ASK outfile TO WriteString("                POINT TARGET LIST
DATA");
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;

NEW(infile);
mapfile := map + ".ptgt";
ASK infile TO Open(mapfile,Input);
ASK infile TO ReadInt(numtgts);
ASK outfile TO WriteString(" Tgt ID       Latitude          Longitude
Quantity");
ASK outfile TO WriteLn;
NEW(target, 1..numtgts);
FOR i := 1 TO numtgts
 NEW(target[i]);
 ASK infile TO ReadString(target[i].tgtID);
 ASK infile TO ReadReal(target[i].x);
 ASK infile TO ReadReal(target[i].y);
 ASK infile TO ReadReal(target[i].lat);
 ASK infile TO ReadReal(target[i].lon);
 ASK infile TO ReadString(target[i].type);
 ASK infile TO ReadString(target[i].protection);
 ASK infile TO ReadString(target[i].quantity);
END FOR;
ASK infile TO Close;

FOR i := 1 TO numtgts
 str := SPRINT (target[i].tgtID, target[i].lat, target[i].lon,
target[i].quantity) WITH format3;
 ASK outfile TO WriteString(str);
 ASK outfile TO WriteLn;
END FOR;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteString("                        POINT TARGET  LIST
DESCRIPTION");
ASK outfile TO WriteLn;
ASK outfile TO WriteLn;
ASK outfile TO WriteString(" Tgt ID       Target Type
Protection");
ASK outfile TO WriteLn;
```

```
    FOR i := 1 TO numtgts
     str := SPRINT (target[i].tgtID, target[i].type,
    target[i].protection) WITH format2;
     ASK outfile TO WriteString(str);
     ASK outfile TO WriteLn;
    END FOR;

    ASK outfile TO WriteLn;
    ASK outfile TO WriteLn;
    ASK outfile TO WriteLn;
    ASK outfile TO WriteLn;

    ASK outfile TO Close;
    DISPOSE(outfile);
END PROCEDURE;

END MODULE.
```

```
DEFINITION MODULE PointSpot;

{-------------------------------------------------------------------------

MODULE NAME:        PointSpot
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the definition module that controls the spotting for point
targets.
-------------------------------------------------------------------------}


FROM Spotter   IMPORT SpotterObj;
FROM Global    IMPORT TargetRecType, AimPt;

TYPE
    PointSpotObj = OBJECT(SpotterObj);
     ASK METHOD Prosecute(IN SALVOS : INTEGER; IN mapfamily :
    STRING);
     ASK METHOD GetPointTgt(IN mapfamily : STRING);
    END OBJECT;

VAR
    PSpotter    :    PointSpotObj;
    tgtl        :    ARRAY INTEGER OF TargetRecType;
    numtgts     :    INTEGER;


END MODULE.
```

```
IMPLEMENTATION MODULE PointSpot;

(*-------------------------------------------------------------------------

MODULE NAME:        PointSpot
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      2/17/92

DESCRIPTION:

This is the module that controls the spotting for point targets.

-------------------------------------------------------------------------*)


FROM IOMod      IMPORT StreamObj, FileUseType(Input, Output);
FROM Global     IMPORT TargetRecType, FirstTarget, AimType, Rand, AimPt,
                ExData;
FROM Global     IMPORT ShipAccuracy;
FROM Spotter    IMPORT SpotterObj;
FROM StdScrn    IMPORT Screen, Lines;
FROM UtilMod    IMPORT ClockTimeSecs, ClockRealSecs, Delay;

OBJECT PointSpotObj;

    ASK METHOD GetPointTgt(IN mapfamily : STRING);

    VAR
     strm1          :    StreamObj;

     i              :    INTEGER;

     tgtlist        :    STRING;

    BEGIN
     tgtlist := mapfamily + ".ptgt";

     NEW(strm1);
     ASK strm1 TO Open (tgtlist, Input);
     ASK strm1 TO ReadInt(numtgts);
     NEW(tgt1,1..numtgts);

     FOR i := 1 TO numtgts
          NEW(tgt1[i]);
          ASK strm1 TO ReadString(tgt1[i].tgtID);
          ASK strm1 TO ReadReal(tgt1[i].x);
          ASK strm1 TO ReadReal(tgt1[i].y);
          ASK strm1 TO ReadReal(tgt1[i].lat);
          ASK strm1 TO ReadReal(tgt1[i].lon);
          ASK strm1 TO ReadString(tgt1[i].type);
          ASK strm1 TO ReadString(tgt1[i].protection);
          ASK strm1 TO ReadString(tgt1[i].quantity);
     END FOR;

     ASK strm1 TO Close;
```

```
 DISPOSE(strm1);

END METHOD;

ASK METHOD Prosecute(IN salvos : INTEGER; IN mapfamily :   STRING);

VAR
 dum2       :    CHAR;

 Spotter    :    SpotterObj;

 roundsfired,
 tgt,
 FFEflag    :    INTEGER;

 tempR,
 LR, AD     :    REAL;

BEGIN
 NEW(PSpotter);

 ASK PSpotter TO Greeting;
 ASK PSpotter TO GetPointTgt(mapfamily);
 tgt := ASK Rand UniformInt(1,numtgts);
 ASK PSpotter TO GetAimPt;

 OUTPUT("FIRE MISSION  TARGET NUMBER  ", tgt1[tgt].tgtID, "  OVER");
 Screen(8);
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(3);
 OUTPUT("FIRE MISSION  TARGET NUMBER  ", tgt1[tgt].tgtID, "  OUT");
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(6);
 OUTPUT("FROM TARGET NUMBER ",tgt1[tgt].tgtID);
 OUTPUT("DIRECTION ", AimPt.bearing, "  MAGNETIC");
 OUTPUT(AimPt.rangedir," ",AimPt.range," ",AimPt.elevdir," ",
 AimPt.elev);
 OUTPUT(tgt1[tgt].type);
 OUTPUT(salvos," SALVOS IN EFFECT SHORE ADJUST OVER");
 Screen(8);
 INPUT(dum2);
 ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
 ASK ExData TO WriteLn;

 Lines(2);
 Screen(3);
 OUTPUT("FROM TARGET NUMBER ",tgt1[tgt].tgtID);
 OUTPUT("DIRECTION ", AimPt.bearing, "  MAGNETIC");
```

```
OUTPUT(AimPt.rangedir," ",AimPt.range," ",AimPt.elevdir," ",
AimPt.elev);
OUTPUT(tgt1[tgt].type);
OUTPUT(salvos," SALVOS IN EFFECT SHORE ADJUST OUT");
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
ASK ExData TO WriteLn;

Lines(2);
Screen(3);
OUTPUT("GUN TARGET LINE _ _ _ DEGREES MAGNETIC");
OUTPUT("READY _____  (TIME OF FLIGHT IN SECONDS)");
OUTPUT("OVER");
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
ASK ExData TO WriteLn;

Lines(2);
Screen(6);
OUTPUT("GUN TARGET LINE _ _ _ DEGREES MAGNETIC");
OUTPUT("READY _____  (TIME OF FLIGHT IN SECONDS)");
OUTPUT("BREAK , FIRE , OVER");
Screen(8);
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
ASK ExData TO WriteLn;
Lines(2);

Screen(3);
OUTPUT("FIRE , OUT");
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
ASK ExData TO WriteLn;

FFEflag := 0;
roundsfired    := 0;

WHILE (roundsfired < salvos) AND (FFEflag = 0)

     Screen(3);
     OUTPUT("SHOT");
     INPUT(dum2);
     ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
     ASK ExData TO WriteLn;
     Delay(2);

     Screen(3);
     IF (roundsfired + 1) = salvos
          OUTPUT("SPLASH , ROUNDS COMPLETE , OVER");

     ELSE
          OUTPUT("SPLASH , OUT");

     END IF;
     INPUT(dum2);
     ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
```

```
        ASK ExData TO WriteLn;

        IF (roundsfired + 1) < salvos

             Screen(6);
             LR := ASK Rand TO Normal(0.0,ShipAccuracy.deflection);
             IF LR < 0.0
                  OUTPUT("LEFT ", ABS(TRUNC(LR)));
             ELSE
                  OUTPUT("RIGHT ", ABS(TRUNC(LR)));
             END IF;

             tempR := FLOAT(AimPt.range);
             AD := ASK Rand TO Normal(tempR, ShipAccuracy.range);
             IF AD < 0.0
                  OUTPUT("DROP ", ABS(TRUNC(AD)));
             ELSE
                  OUTPUT("ADD ", ABS(TRUNC(AD)));
             END IF;

             OUTPUT("OUT");
             Screen(8);
             INPUT(dum2);
             ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
             ASK ExData TO WriteLn;
             Lines(2);

             Screen(3);
             IF LR < 0.0
                  OUTPUT("LEFT ", ABS(TRUNC(LR)));
             ELSE
                  OUTPUT("RIGHT ", ABS(TRUNC(LR)));
             END IF;

             IF AD < 0.0
                  OUTPUT("DROP ", ABS(TRUNC(AD)));
             ELSE
                  OUTPUT("ADD ", ABS(TRUNC(AD)));
             END IF;
             OUTPUT("OUT");
             INPUT(dum2);
             ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
             ASK ExData TO WriteLn;

        END IF;

        roundsfired := roundsfired + 1;

END WHILE;

Screen(6);
OUTPUT("ROUNDS COMPLETE , END OF MISSION");
OUTPUT("TARGET NUMBER  ", tgt1[tgt].tgtID, "  DESTROYED , OVER");
Screen(8);
INPUT(dum2);
ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
```

```
      ASK ExData TO WriteLn;

      Screen(3);
      OUTPUT("ROUNDS COMPLETE , END OF MISSION");
      OUTPUT("TARGET NUMBER  ", tgt1[tgt].tgtID, "  DESTROYED , OUT");
      INPUT(dum2);
      ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
      ASK ExData TO WriteLn;

    END METHOD;

END OBJECT;

END MODULE.
```

```
DEFINITION MODULE ShipData;

(*-------------------------------------------------------------------------

MODULE NAME:        ShipData
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/7/91
LAST MODIFIED:      2/14/92

DESCRIPTION:

This is the definition module that controls editing the data file
containing pertinent ship's data.

-------------------------------------------------------------------------*)


PROCEDURE VerifyShip;
PROCEDURE ChangeShip;
PROCEDURE ChangeAccel;
PROCEDURE ChangeDecel;
PROCEDURE ChangeTRate;
PROCEDURE ShipHardCopy;

END MODULE.
```

```
IMPLEMENTATION MODULE ShipData;

(*-------------------------------------------------------------------------

MODULE NAME:        ShipData
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/7/91
LAST MODIFIED:      2/14/92

DESCRIPTION:

This is the module that controls editing the data file containing
pertinent ship's data.
-------------------------------------------------------------------------*)


FROM CRTMod    IMPORT ClearScreen;
FROM StdScrn   IMPORT Screen, Lines;
FROM Global    IMPORT ShipFile,CSbeach,CSship,ShipAccuracy,SpotterBias;
FROM Global    IMPORT THead, TRudder, TSpeed;
FROM IOMod     IMPORT StreamObj, FileUseType(Output);
FROM UtilMod   IMPORT DateTime;

VAR
    dummy :    CHAR;

(*************************************************************************)

PROCEDURE VerifyShip;

CONST
    format = "   ***          ***.**      ***.**      ***.**      ***.**";
    format2 = "         **   **   **       ***.**";
    tt = "TO";

VAR
    rdum                    REAL;

    i,j,k       :       INTEGER;

    head            :    ARRAY INTEGER OF STRING;


BEGIN

    Lines(3);
    OUTPUT("The following is current ships data,");
    OUTPUT("Please hit <return> when ready to continue");
    Lines(2);
    OUTPUT("SHIP:         USS ", ShipFile.name," ",ShipFile.desig,    " ",
    ShipFile.hullno);
    OUTPUT("CO:           ", ShipFile.COrank," ", ShipFile.COname);
    OUTPUT("XO:           ", ShipFile.XOrank, " ",     ShipFile.XOname);
    OUTPUT("Ship's GFCS: ", ShipFile.GFCS, "  mod     ",ShipFile.model);

    OUTPUT(" ");
```

```
OUTPUT("Spotter's Call Sign:     ",CSbeach);
OUTPUT("Ship's Call Sign:        ",CSship);
OUTPUT(" ");
OUTPUT("Gun Battery Accuracy, (error is standard deviation):     ");
OUTPUT("     range:       ",ShipAccuracy.range);
OUTPUT("     deflection: ",ShipAccuracy.deflection);
INPUT(dummy);
Lines(3);

OUTPUT("              Acceleration Data");
OUTPUT("Time in Seconds it takes To Increase Speed From the");
OUTPUT("     Low End To the High End of the Speed Range");
Lines(1);
OUTPUT("           KNOTS          SECONDS");
FOR i := 1 TO 6
 PRINT(i*5-5,tt,i*5,ShipFile.accel[i]) WITH format2;
END FOR;
Lines(1);
Screen(5);
INPUT(dummy);

OUTPUT("              Deceleration Data");
OUTPUT("Time in Seconds it takes To Decrease Speed From the");
OUTPUT("     High End To the Low End of the Speed Range");
Lines(1);
OUTPUT("            KNOTS         SECONDS");
FOR i := 6 DOWNTO 1
 PRINT(i*5,tt,i*5-5,ShipFile.decel[i]) WITH format2;
END FOR;
Lines(1);
Screen(5);
INPUT(dummy);

NEW(head, 1..4);
head[1] := " 0-15";
head[2] := "15-30";
head[3] := "30-45";
head[4] := "45-60";

OUTPUT("              Turn Rate Data");
OUTPUT("Time in Seconds it takes to turn in Relative  Degrees,");
OUTPUT("     At a given Speed and Rudder Angle");
Lines(2);

FOR i := 1 TO 3
 OUTPUT("                TABLE FOR SPEED  ",TSpeed[i],"Knots");
 OUTPUT;
 OUTPUT("              Amount Of Turn In Relative Degrees");
 OUTPUT("Rudder Angle    ",head[1],"      ",head[2],"      ",
head[3],"      ",head[4]);

 FOR j := 1 TO 3
      PRINT(TRudder[j],ShipFile.turnrate[i][j][1],
      ShipFile.turnrate[i][j][2], ShipFile.turnrate[i][j][3],
      ShipFile.turnrate[i][j][4]) WITH format;
 END FOR;
```

```
      Screen(5);
      INPUT(dummy);
      Lines(2);
    END FOR;

    OUTPUT("Would You Like A Hard Copy of This Data Made? < Y/N     >");
    INPUT(dummy);
    IF (dummy = 'y') OR (dummy = 'Y')
      Lines(2);
      OUTPUT("It will take a moment.....");
      ShipHardCopy;
    END IF;

END PROCEDURE;

{*****************************************************************}

PROCEDURE ChangeShip;

CONST
    format = "   ***          ***.**      ***.**      ***.**      ***.**";
    format2 = "  *       **  **  **       ***.**";
    tt = "TO";

VAR
    rdum            :    REAL;

    UpdateFile,
    option          :    INTEGER;

    flag,
    again,
    more            :    CHAR;

    ShipDataFile    :    StreamObj;

    dummy           :    CHAR;

    entry,
    i,j,k           :    INTEGER;

    head            :    ARRAY INTEGER OF STRING;

BEGIN

more := 'y';
UpdateFile := 0;


WHILE (more = "y") OR (more = "Y")

    more := 'n';

    VerifyShip;
    OUTPUT(" ");
    OUTPUT("Please Input the number of the option you wish to  alter:");
```

```
OUTPUT(" 1.  Ship ");
OUTPUT(" 2.  CO ");
OUTPUT(" 3.  XO ");
OUTPUT(" 4.  Gun Fire Control System ");
OUTPUT(" 5.  Call Signs ");
OUTPUT(" 6.  Gun Battery Accuracy ");
OUTPUT(" 7.      Acceleration Data");
OUTPUT(" 8.      Deceleration Data");
OUTPUT(" 9.  Turning Rate Data");
OUTPUT("10.  Do Not Wish To Alter Any Data ");
INPUT(option);

flag := 'n';

CASE option
 WHEN 1:
      OUTPUT(" ");
      OUTPUT("Answer either <Y> folowed by <return>");
      OUTPUT("  or <return> for NO Change Desired");
      OUTPUT("    to select an entry for change");
      OUTPUT(" ");

      OUTPUT(" ");
      OUTPUT("Ship's Name: USS ",ShipFile.name," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Ship's Name ?");
           INPUT(ShipFile.name);
           flag := 'n';
           UpdateFile := 1;
      END IF;
      OUTPUT(" ");
      OUTPUT("Ship's Designater: ",ShipFile.desig," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Ship's Designator ?");
           INPUT(ShipFile.desig);
           flag := 'n';
           UpdateFile := 1;
      END IF;
      OUTPUT(" ");
      OUTPUT("Ship's Hull Number: ",ShipFile.hullno," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Ship's Hull Number ?");
           INPUT(ShipFile.hullno);
           flag := 'n';
           UpdateFile := 1;
      END IF;

 WHEN 2:
      OUTPUT(" ");
      OUTPUT("Answer either <Y> folowed by <return>");
      OUTPUT("  or <return> for NO Change Desired");
      OUTPUT("    to select an entry for change");
      OUTPUT(" ");
```

```
     OUTPUT(" ");
     OUTPUT("CO's Rank: ",ShipFile.COrank," ?");
     INPUT(flag);
     IF (flag = "Y") OR (flag = "y")
          OUTPUT("CO's Rank ?");
          INPUT(ShipFile.COrank);
          flag := 'n';
          UpdateFile := 1;
     END IF;
     OUTPUT(" ");
     OUTPUT("CO's Name: ",ShipFile.COname," ?");
     INPUT(flag);
     IF (flag = "Y") OR (flag = "y")
          OUTPUT("CO's Name ?");
          INPUT(ShipFile.COname);
          flag := 'n';
          UpdateFile := 1;
     END IF;

WHEN 3:
     OUTPUT(" ");
     OUTPUT("Answer either <Y> folowed by <return>");
     OUTPUT("  or <return> for NO Change Desired");
     OUTPUT("    to select an entry for change");
     OUTPUT(" ");

     OUTPUT(" ");
     OUTPUT("XO's Rank: ",ShipFile.XOrank," ?");
     INPUT(flag);
     IF (flag = "Y") OR (flag = "y")
          OUTPUT("XO's Rank ?");
          INPUT(ShipFile.XOrank);
          flag := 'n';
          UpdateFile := 1;
     END IF;
     OUTPUT(" ");
     OUTPUT("XO's Name: ",ShipFile.XO[illegible]?");
     INPUT(flag);
     IF (flag = "Y") OR (flag = "y"
          OUTPUT("XO's Name ?");
          INPUT(ShipFile.XOname);
          flag := 'n';
          UpdateFile := 1;
     END IF;

WHEN 4:
     OUTPUT(" ");
     OUTPUT("Answer either <Y> folowed by <return>");
     OUTPUT("  or <return> for NO Change Desired");
     OUTPUT("    to select an entry for change");
     OUTPUT(" ");

     OUTPUT(" ");
     OUTPUT("Ship's GFCS: ",ShipFile.GFCS," ?");
     INPUT(flag);
```

```
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Ship's GFCS ?");
           INPUT(ShipFile.GFCS);
           flag := 'n';
           UpdateFile := 1;
      END IF;
      OUTPUT(" ");
      OUTPUT("Ship's GFCS mod: ",ShipFile.model," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Ship's GFCS mod ?");
           INPUT(ShipFile.model);
           flag := 'n';
           UpdateFile := 1;
      END IF;

 WHEN 5:
      OUTPUT(" ");
      OUTPUT("Answer either <Y> folowed by <return>");
      OUTPUT("  or <return> for NO Change Desired");
      OUTPUT("    to select an entry for change");
      OUTPUT(" ");

      OUTPUT(" ");
      OUTPUT("Spotter's Call Sign: ",CSbeach," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Spotter's Call Sign ?");
           INPUT(CSbeach);
           flag := 'n';
           UpdateFile := 1;
      END IF;
      OUTPUT(" ");
      OUTPUT("Ship's Call Sign: ",CSship," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Ship's Call Sign ?");
           INPUT(CSship);
           flag := 'n';
           UpdateFile := 1;
      END IF;

 WHEN 6:
      OUTPUT(" ");
      OUTPUT("Answer either <Y> folowed by <return>");
      OUTPUT("  or <return> for NO Change Desired");
      OUTPUT("    to select an entry for change");
      OUTPUT(" ");

      OUTPUT(" ");
      OUTPUT("Gun Battery Accuracy, (error is standard
deviation) ");
      OUTPUT("Range: ",ShipAccuracy.range," ?");
      INPUT(flag);
      IF (flag = "Y") OR (flag = "y")
           OUTPUT("Gun Battery Accuracy Range ?");
```

```
                    INPUT(ShipAccuracy.range);
                    flag := 'n';
                    UpdateFile := 1;
               END IF;
               OUTPUT(" ");
               OUTPUT("Deflection: ",ShipAccuracy.deflection," ?");
               INPUT(flag);
               IF (flag = "Y") OR (flag = "y")
                    OUTPUT("Gun Battery Accuracy Deflection ?");
                    INPUT(ShipAccuracy.deflection);
                    flag := 'n';
                    UpdateFile := 1;
               END IF;

          WHEN 7:
               ChangeAccel;
               UpdateFile := 1;

          WHEN 8:
               ChangeDecel;
               UpdateFile := 1;

          WHEN 9:
               ChangeTRate;
               UpdateFile := 1;

          OTHERWISE

        END CASE;

        OUTPUT("Change Any Other Ship's Data ?, INPUT < Y/N > <return>");
        INPUT(more);

    END WHILE;

        IF UpdateFile = 1
         NEW(ShipDataFile);
         ASK ShipDataFile TO Open ("Ship.data",Output);
         ASK ShipDataFile TO WriteString(ShipFile.name);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.desig);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.hullno);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.COname);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.COrank);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.XOname);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.XOrank);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteString(ShipFile.GFCS);
         ASK ShipDataFile TO WriteLn;
         ASK ShipDataFile TO WriteInt(ShipFile.model, 4);
         ASK ShipDataFile TO WriteLn;
```

```
     ASK ShipDataFile TO WriteString(CSbeach);
     ASK ShipDataFile TO WriteLn;
     ASK ShipDataFile TO WriteString(CSship);
     ASK ShipDataFile TO WriteLn;
     ASK ShipDataFile TO WriteReal(ShipAccuracy.range, 8, 4);
     ASK ShipDataFile TO WriteLn;
     ASK ShipDataFile TO WriteReal(ShipAccuracy.deflection, 8, 4);
     ASK ShipDataFile TO WriteLn;
     ASK ShipDataFile TO WriteReal(SpotterBias.range, 8, 4);
     ASK ShipDataFile TO WriteLn;
     ASK ShipDataFile TO WriteReal(SpotterBias.deflection, 8, 4);
     ASK ShipDataFile TO WriteLn;
     FOR i := 1 TO 6
          ASK ShipDataFile TO WriteReal(ShipFile.accel[i], 8, 4);
          ASK ShipDataFile TO WriteLn;
     END FOR;

     FOR i := 1 TO 6
          ASK ShipDataFile TO WriteReal(ShipFile.decel[i], 8, 4);
          ASK ShipDataFile TO WriteLn;
     END FOR;


     FOR i := 1 TO 3                {speeds of 5, 15, 25 }
          FOR j:= 1 TO 3            {rudder angles of 5, 15,25}
               FOR k:= 1 TO 4       {heading change 0-15, 15-30,
                                30-45, 45-60}
                    ASK ShipDataFile TO
     WriteReal(ShipFile.turnrate[i][j][k], 8, 4);
                    ASK ShipDataFile TO WriteLn;
               END FOR;
          END FOR;
     END FOR;
     ASK ShipDataFile TO Close;
     DISPOSE(ShipDataFile);
    END IF;

    OUTPUT("A Hard Copy of This Data Is Being Made ");
    OUTPUT("It will take a moment.....");
    ShipHardCopy;


END PROCEDURE;

{*********************************************************************}

PROCEDURE ChangeAccel;

CONST
    format2 = "  *      **  **  **      ***.**";
    tt = "TO";

VAR
    again           :    CHAR;

    dummy           :    CHAR;
```

```
    entry,
    i,j,k           :    INTEGER;

BEGIN
    again := 'y';
    WHILE (again = "y") OR (again = "Y")
     OUTPUT("              Acceleration Data");
     OUTPUT("Time in Seconds it takes To Increase Speed From the");
     OUTPUT("    Low End To the High End of the Speed Range");
     Lines(1);
     OUTPUT("ENTRY      KNOTS          SECONDS");
     FOR i := 1 TO 6
          PRINT(i,i*5-5,tt,i*5,ShipFile.accel[i]) WITH format2;
     END FOR;

     Lines(1);
     OUTPUT("Input the number corresponding to the entry you wish to
     change");
     INPUT(i);
     WHILE (i < 1) OR (i > 6)
          OUTPUT("Input is out of bounds, must be between 1 and 6,
          TRY AGAIN");
          OUTPUT("Input ENTRY");
          INPUT(i);
     END WHILE;

     Lines(1);
     OUTPUT("ENTRY      KNOTS          SECONDS");
     PRINT(i,i*5-5,tt,i*5,ShipFile.accel[i]) WITH format2;
     OUTPUT("Enter New Time in Seconds");
     INPUT(ShipFile.accel[i]);
     Lines(2);

     OUTPUT("              Acceleration Data");
     OUTPUT("Time in Seconds it takes To Increase Speed From the");
     OUTPUT("    Low End To the High End of the Speed Range");
     Lines(1);
     OUTPUT("ENTRY      KNOTS          SECONDS");
     FOR i := 1 TO 6
          PRINT(i,i*5-5,tt,i*5,ShipFile.accel[i]) WITH format2;
     END FOR;
     Lines(2);
     OUTPUT("Change more Acceleration Data? < Y/N >");
     INPUT(again);
    END WHILE;

END PROCEDURE;

{*****************************************************************}

PROCEDURE ChangeDecel;

CONST
    format2 = "  *       **  **  **       ***.**";
    tt = "TO";
```

```
VAR
    again           :    CHAR;

    dummy           :    CHAR;

    entry,
    i,j,k           :    INTEGER;

BEGIN
    again := 'y';
    WHILE (again = "y") OR (again = "Y")
     OUTPUT("              Deceleration Data");
     OUTPUT("Time in Seconds it takes To Decrease Speed From the");
     OUTPUT("    High End To the Low End of the Speed Range");
     Lines(1);
     OUTPUT("ENTRY        KNOTS         SECONDS");
     FOR i := 1 TO 6
          PRINT(i,i*5,tt,i*5-5,ShipFile.decel[i]) WITH format2;
     END FOR;

     Lines(1);
     OUTPUT("Input the number corresponding to the entry you wish to
     change");
     INPUT(i);
     WHILE (i < 1) OR (i > 6)
          OUTPUT("Input is out of bounds, must be between 1 and 6,
          TRY AGAIN");
          OUTPUT("Input ENTRY");
          INPUT(i);
     END WHILE;

     Lines(1);
     OUTPUT("ENTRY         KNOTS        SECONDS");
     PRINT(i,i*5,tt,i*5-5,ShipFile.decel[i]) WITH format2;
     OUTPUT("Enter New Time in Seconds");
     INPUT(ShipFile.decel[i]);
     Lines(2);

     OUTPUT("              Deceleration Data");
     OUTPUT("Time in Seconds it takes To Decrease Speed From the");
     OUTPUT("    High End To the Low End of the Speed Range");
     Lines(1);
     OUTPUT("ENTRY         KNOTS        SECONDS");
     FOR i := 1 TO 6
          PRINT(i,i*5,tt,i*5-5,ShipFile.decel[i]) WITH format2;
     END FOR;
      Lines(2);
      OUTPUT("Change more Deceleration Data? < Y/N >");
      INPUT(again);
    END WHILE;

END PROCEDURE;

{*********************************************************************}
```

```
PROCEDURE ChangeTRate;

CONST
    format = "  *          ***          ***.**      ***.**      ***.**
    ***.**";

VAR
    again           :    CHAR;

    entry,
    i,j,k           :    INTEGER;

    head            :    ARRAY INTEGER OF STRING;


BEGIN
    NEW(head, 1..4);
    head[1] := " 0-15";
    head[2] := "15-30";
    head[3] := "30-45";
    head[4] := "45-60";

    again := 'y';
    WHILE (again = "y") OR (again = "Y")

     OUTPUT("      Select The Speed You Wish To Edit");
     OUTPUT("For Rudder Angle and Heading Change Combinations");
     OUTPUT;
     OUTPUT("              1.  5 KNOTS");
     OUTPUT("              2. 15 KNOTS");
     OUTPUT("              3. 25 KNOTS");
     OUTPUT;
     OUTPUT("Input The Number Corresponding To The Correct Speed");
     INPUT(i);

     WHILE (i<1) OR (i>3)
          OUTPUT("The entry was not 1,2, or 3 ; TRY AGAIN");
          OUTPUT;
          OUTPUT("              1.  5 KNOTS");
          OUTPUT("              2. 15 KNOTS");
          OUTPUT("              3. 25 KNOTS");
          OUTPUT;
          OUTPUT("Input The Number Corresponding To The Correct Speed");
          INPUT(i);
     END WHILE;



     OUTPUT("                    Turn Rate Data");
     OUTPUT("     Time in Seconds it takes to turn in Relative
     Degrees,");
     OUTPUT("          At a given Speed and Rudder Angle");
     Lines(2);

     OUTPUT("                TABLE FOR SPEED  ",TSpeed[i],"Knots");
     OUTPUT;
```

```
      OUTPUT("              Amount Of Turn In Relative Degrees");
      OUTPUT("ENTRY   Rudder Angle     ",head[1],"      ",head[2],
      "      ",head[3],"      ",head[4]);

      FOR j := 1 TO 3
           PRINT(j,TRudder[j],ShipFile.turnrate[i][j][1],
           ShipFile.turnrate[i][j][2], ShipFile.turnrate[i][j][3],
           ShipFile.turnrate[i][j][4]) WITH format;
      END FOR;

      Lines(1);
      OUTPUT("Input the number corresponding to the entry you wish
      to change");
      INPUT(j);
      WHILE (j < 1) OR (j > 3)
           OUTPUT("The entry was not 1,2, or 3 ; TRY AGAIN");
           OUTPUT("Input ENTRY");
           INPUT(j);
      END WHILE;

      OUTPUT("ENTRY   Rudder Angle     ",head[1],"      ",head[2],
      "      ",head[3],"      ",head[4]);
      PRINT(j,TRudder[j],ShipFile.turnrate[i][j][1],
      ShipFile.turnrate[i][j][2], ShipFile.turnrate[i][j][3],
      ShipFile.turnrate[i][j][4]) WITH format;
      Lines(2);
      FOR k := 1 TO 4
           OUTPUT("Input the new time in Seconds for  ",head[k],
           "  using ",TRudder[j]," Degrees of Rudder");
           INPUT(ShipFile.turnrate[i][j][k]);
      END FOR;
      Lines(2);
      OUTPUT("ENTRY   Rudder Angle     ",head[1],"      ",head[2],
      "      ",head[3],"      ",head[4]);
      PRINT(j,TRudder[j],ShipFile.turnrate[i][j][1],
      ShipFile.turnrate[i][j][2], ShipFile.turnrate[i][j][3],
      ShipFile.turnrate[i][j][4]) WITH format;

      Lines(2);
      OUTPUT("Change more Deceleration Data? < Y/N >");
      INPUT(again);
    END WHILE;

END PROCEDURE;

{*********************************************************************}

PROCEDURE ShipHardCopy;

CONST
    format = "   ***          ***.**      ***.**      ***.**      ***.**";
    format2 = "         **  **  **       ***.**";
    tt = "TO";

VAR
    rdum            :    REAL;
```

```
    i,j,k       :    INTEGER;

    head        :    ARRAY INTEGER OF STRING;

    str         :    STRING;

    copy        :    StreamObj;


BEGIN
    NEW(copy);
    ASK copy TO Open("ShipData.prt",Output);

    ASK copy TO WriteString("The following is current ships data as
    of:");
    ASK copy TO WriteLn;
    DateTime(str);
    ASK copy TO WriteString(str);
    ASK copy TO WriteLn;
    ASK copy TO WriteLn;
    ASK copy TO WriteLn;
    ASK copy TO WriteString("SHIP:        USS ");
    ASK copy TO WriteString(ShipFile.name);
    ASK copy TO WriteString(" ");
    ASK copy TO WriteString(ShipFile.desig);
    ASK copy TO WriteString(" ");
    ASK copy TO WriteString(ShipFile.hullno);
    ASK copy TO WriteLn;
    ASK copy TO WriteString("CO:          ");
    ASK copy TO WriteString(ShipFile.COrank);
    ASK copy TO WriteString(" ");
    ASK copy TO WriteString(ShipFile.COname);
    ASK copy TO WriteLn;
    ASK copy TO WriteString("XO:          ");
    ASK copy TO WriteString(ShipFile.XOrank);
    ASK copy TO WriteString(" ");
    ASK copy TO WriteString(ShipFile.XOname);
    ASK copy TO WriteLn;
    ASK copy TO WriteString("Ship's GFCS: ");
    ASK copy TO WriteString(Ship[illegible].GFCS);
    ASK copy TO WriteString("  mo[illegible]
    ASK copy TO WriteInt(ShipFile.model,3);
    ASK copy TO WriteLn;

    ASK copy TO WriteLn;
    ASK copy TO WriteString("Spotter's Call Sign:    ");
    ASK copy TO WriteString(CSbeach);
    ASK copy TO WriteLn;
    ASK copy TO WriteString("Ship's Call Sign:       ");
    ASK copy TO WriteString(CSship);
    ASK copy TO WriteLn;
    ASK copy TO WriteLn;
    ASK copy TO WriteString("Gun Battery Accuracy, (error is standard
    deviation): ");
    ASK copy TO WriteLn;
```

```
ASK copy TO WriteS ing("    range:      ");
ASK copy TO WriteReal(ShipAccuracy.range,6,2);
ASK copy TO WriteLn;
ASK copy TO WriteString("    deflection: ");
ASK copy TO WriteReal(ShipAccuracy.deflection,6,2);
ASK copy TO WriteLn;
ASK copy TO WriteLn;
ASK copy TO WriteLn;
ASK copy TO WriteLn;

ASK copy TO WriteString("             Acceleration Data");
ASK copy TO WriteLn;
ASK copy TO WriteString("Time in Seconds it takes To Increase
Speed From the");
ASK copy TO WriteLn;
ASK copy TO WriteString("    Low End To the High End of the
Speed Range");
ASK copy TO WriteLn;
ASK copy TO WriteLn;
ASK copy TO WriteString("           KNOTS        SECONDS");
ASK copy TO WriteLn;
FOR i := 1 TO 6
 str := SPRINT(i*5-5,tt,i*5,ShipFile.accel[i]) WITH format2;
 ASK copy TO WriteString(str);
 ASK copy TO WriteLn;
END FOR;
ASK copy TO WriteLn;
ASK copy TO WriteLn;

ASK copy TO WriteString("             Deceleration Data");
ASK copy TO WriteLn;
ASK copy TO WriteString("Time in Seconds it takes To Decrease
Speed From the");
ASK copy TO WriteLn;
ASK copy TO WriteString("    High End To the Low End of the
Speed Range");
ASK copy TO WriteLn;
ASK copy TO WriteLn;
ASK copy TO WriteString("           KNOTS        SECONDS");
ASK copy TO WriteLn;
FOR i := 6 DOWNTO 1
 str := SPRINT(i*5,tt,i*5-5,ShipFile.decel[i]) WITH format2;
 ASK copy TO WriteString(str);
 ASK copy TO WriteLn;
END FOR;
ASK copy TO WriteLn;
ASK copy TO WriteLn;
ASK copy TO WriteLn;

NEW(head, 1..4);
head[1] := " 0-15";
head[2] := "15-30";
head[3] := "30-45";
head[4] := "45-60";

ASK copy TO WriteString("                    Turn Rate Data");
```

```
    ASK copy TO WriteLn;
    ASK copy TO WriteString("     Time in Seconds it takes to turn  in
    Relative Degrees,");
    ASK copy TO WriteLn;
    ASK copy TO WriteString("            At a given Speed and  Rudder
    Angle");
    ASK copy TO WriteLn;
    ASK copy TO WriteLn;
    ASK copy TO WriteLn;

    FOR i := 1 TO 3
     ASK copy TO WriteString("                TABLE FOR SPEED  ");
     ASK copy TO WriteInt(TSpeed[i],2);
     ASK copy TO WriteString("  Knots");
     ASK copy TO WriteLn;
     ASK copy TO WriteLn;
     ASK copy TO WriteString("              Amount Of Turn In
     Relative Degrees");
     ASK copy TO WriteLn;
     ASK copy TO WriteString("Rudder Angle    ");
     ASK copy TO WriteString(head[1]);
     ASK copy TO WriteString("      ");
     ASK copy TO WriteString(head[2]);
     ASK copy TO WriteString("      ");
     ASK copy TO WriteString(head[3]);
     ASK copy TO WriteString("      ");
     ASK copy TO WriteString(head[4]);
     ASK copy TO WriteLn;

     FOR j := 1 TO 3
          str := SPRINT(TRudder[j], ShipFile.turnrate[i][j][1],
          ShipFile.turnrate[i][j][2], ShipFile.turnrate[i][j][3],
          ShipFile.turnrate[i][j][4]) WITH format;
          ASK copy TO WriteString(str);
          ASK copy TO WriteLn;
     END FOR;
    ASK copy TO WriteLn;
    ASK copy TO WriteLn;
    END FOR;

    ASK copy TO Close;
    DISPOSE(copy);
    Lines(2);
    OUTPUT("A hard copy of the Ship's Data File Has Been Saved ");
    OUTPUT("A File Named ShipData.prt");
    Lines(2);

END PROCEDURE;

END MODULE.
```

```
DEFINITION MODULE Spotter;

{-------------------------------------------------------------------------

MODULE NAME:        Spotter
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      12/2/91

DESCRIPTION:

This is the definition module that controls spotting targets with
methods common to all types of spotting.

-------------------------------------------------------------------------}


TYPE
    SpotterObj = OBJECT
     ASK METHOD Greeting;
     ASK METHOD GetAimPt;

     {
     ASK METHOD SelectTarget(IN Aim:STRING; OUT Target:TargetType);
     ASK METHOD ConvertXYtoMILS(IN Xcoord,Ycoord : REAL;  OUT Xmil,
     Ymil : REAL);
     ASK METHOD ConvertXYtoRangeBearing(IN Xcoord,Ycoord : REAL;
     OUT Range, Bearing : REAL);
     ASK METHOD ConvertMILStoXY(IN Xmil, Ymil : REAL;
     OUT Xcoord,Ycoord : REAL);
     ASK METHOD ConvertRangeBearingtoXY(IN Range, Bearing : REAL;
     OUT Xcoord,Ycoord : REAL);
     }

    END OBJECT;

END MODULE.
```

```
IMPLEMENTATION MODULE Spotter;

{-------------------------------------------------------------------------

MODULE NAME:        Spotter
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      12/2/91

DESCRIPTION:

This is the module that controls spotting targets with methods common to
all types of spotting.

-------------------------------------------------------------------------}


FROM Global    IMPORT ShipFile,CSbeach,CSship, AimPt, Rand;
FROM StdScrn   IMPORT Screen, Lines;
FROM Debug     IMPORT TraceStream;
FROM IOMod     IMPORT FileUseType(Output);
FROM Global    IMPORT AimType, ExData;
FROM MathMod   IMPORT SQRT,ACOS,ASIN;
FROM UtilMod   IMPORT ClockTimeSecs, ClockRealSecs, Delay;


OBJECT SpotterObj;

    ASK METHOD Greeting;

    VAR
     dum1 :    CHAR;

    BEGIN

     OUTPUT("This is an NGFS Excercise for the USS",ShipFile.name);
     OUTPUT(" ");
     Screen(4);
     INPUT(dum1);
     OUTPUT(" ");
     OUTPUT(" ");
     Screen(3);
     OUTPUT(CSbeach,"  THIS IS  ", CSship);
     OUTPUT("ON STATION AND READY FOR CALL FOR FIRE OVER");
     INPUT(dum1);
     ASK ExData TO WriteReal(ClockRealSecs, 8, 4);
     ASK ExData TO WriteLn;
     OUTPUT(" ");
     OUTPUT(" ");
     OUTPUT(" ");
     Screen(6);
     OUTPUT(CSship,"  THIS IS  ",CSbeach);
    END METHOD;

    ASK METHOD GetAimPt;
```

```
    VAR

     a,                  { temp X value      }
     b,                  { temp Y value      }
     c    :    REAL;     { hypotenuse   }

    BEGIN

     NEW(AimPt);

     a := ASK Rand UniformReal(-500.0, 500.0);
     b := ASK Rand UniformReal(-500.0, 500.0);
     c := SQRT(a*a + b*b);

     AimPt.range := ROUND(c);
     AimPt.rangedir := "ADD";

     AimPt.elev := ASK Rand TO UniformInt(-50,1500);

          IF AimPt.elev < 0
               AimPt.elevdir := "DOWN";
          ELSE
               AimPt.elevdir := "UP";
          END IF;

     AimPt.Xcoord := a;
     AimPt.Ycoord := b;

     IF b >= 0.0
          IF a >= 0.0
               AimPt.bearing := ROUND(ACOS(a/c));
          ELSE
               AimPt.bearing := 180 - ROUND(ASIN(b/c));
          END IF;

     ELSE
          IF a < 0.0
               a := ABS(a);
               AimPt.bearing := 180 + ROUND(ACOS(a/c));
          ELSE
               AimPt.bearing := 360 - ROUND(ACOS(a/c));
          END IF;
     END IF;

     IF AimPt.bearing = 360
          AimPt.bearing := 0;
     END IF;
    END METHOD;

END OBJECT;

END MODULE.
```

```
DEFINITION MODULE StdScrn;

{-------------------------------------------------------------------------

MODULE NAME:        StdScrn
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      1/22/92

DESCRIPTION:

This is the definition module that controls outputting standard
screen messages.  This will reduce the number of OUTPUT statements
found in the dialog sections of the scenarios.
-------------------------------------------------------------------------}


PROCEDURE Screen(IN Phrase    :    INTEGER);
PROCEDURE Lines(IN numlines   :    INTEGER);
PROCEDURE Reminder(IN system : INTEGER);

END MODULE.
```

```
IMPLEMENTATION MODULE StdScrn;

(*-------------------------------------------------------------------------

MODULE NAME:        StdScrn
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       11/27/91
LAST MODIFIED:      1/25/92

DESCRIPTION:

This is the module that controls outputting standard screen messages.
This will reduce the number of OUTPUT statements found in the dialog
sections of the scenarios.
-------------------------------------------------------------------------*)


PROCEDURE Screen(IN phrase    :    INTEGER);

BEGIN

CASE phrase
    WHEN 1:
     OUTPUT("Input the number of the option: ");
     OUTPUT(" ");

    WHEN 2:
     OUTPUT("INPUT Y to make another menu selection or <return> to
    continue");

    WHEN 3:
     OUTPUT("Strike the <return> key when the ship responds
    correctly as follows: ");
     OUTPUT(" ");

    WHEN 4:
     OUTPUT("Strike the <return> key when WHEN THE SHIP IS READY TO
     COMMENCE THE EXCERCISE ");
     OUTPUT(" ");

    WHEN 5:
     OUTPUT("Strike the <return> key when WHEN READY TO CONTINUE");
     OUTPUT(" ");

    WHEN 6:
     OUTPUT("SPOTTER READS THE FOLLOWING INTO THE VOICE CIRCUIT:");
     OUTPUT(" ");

    WHEN 7:
     OUTPUT("Input the new information or <return> to keep data the
     same. ");
     OUTPUT(" ");

    WHEN 8:
     Lines(1);
```

```
      OUTPUT("Hit the <return> key when the Spotter has read the
     message. ");
      Lines(2);

     OTHERWISE
      OUTPUT("BOGUS INPUT FOR PROCEDURE Screen in module StdScrn ");
      OUTPUT(" ");

END CASE;

END PROCEDURE;

PROCEDURE Lines(IN numlines  :    INTEGER);

BEGIN
    CASE numlines
      WHEN 1:
           OUTPUT(" ");

      WHEN 2:
           OUTPUT(" ");
           OUTPUT(" ");

      WHEN 3:
           OUTPUT(" ");
           OUTPUT(" ");
           OUTPUT(" ");

      WHEN 4:
           OUTPUT(" ");
           OUTPUT(" ");
           OUTPUT(" ");
           OUTPUT(" ");

      WHEN 5:
           OUTPUT(" ");
           OUTPUT(" ");
           OUTPUT(" ");
           OUTPUT(" ");
           OUTPUT(" ");

      OTHERWISE
           OUTPUT(" ");
           OUTPUT(" ");

    END CASE;
END PROCEDURE;

PROCEDURE Reminder(IN system : INTEGER);

BEGIN
    CASE system

    WHEN 1:
      OUTPUT("41-09-45.0 as 410945.0 <DDMMSS.S>");
```

```
    WHEN 2:
     OUTPUT("41-09.75    as 4109.75 <DDMM.mm>");

    OTHERWISE

    END CASE;
END PROCEDURE;


END MODULE.
```

```
DEFINITION MODULE TgtMkr;

{-------------------------------------------------------------------------

MODULE NAME:        TgtMkr
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/20/91
LAST MODIFIED:      2/15/92

DESCRIPTION:

This is the definition module that controls the creation of target
lists.
-------------------------------------------------------------------------}


PROCEDURE CreateAreaTgt;
PROCEDURE CreatePointTgt;

END MODULE.
```

```
IMPLEMENTATION MODULE TgtMkr;

{-------------------------------------------------------------------------

MODULE NAME:        TgtMkr
AUTHOR:             LT. WARREN A. MAZANEC
DATE WRITTEN:       12/20/91
LAST MODIFIED:      2/15/92

DESCRIPTION:

This is the definition module that controls the creation of target
lists.
-------------------------------------------------------------------------}


FROM Geo        IMPORT sys, dlat, dlon, scale, DConvert, MapFamily;
FROM Global     IMPORT TargetRecType;
FROM StdScrn    IMPORT Reminder;
FROM IOMod      IMPORT StreamObj, FileUseType(Output);

VAR

    changeFlag,
    areaFlag,
    ptFlag,
    shoreFlag       :    STRING;

    edit,
    areactr,
    ptctr,
    shorectr,
    i               :    INTEGER;

    tempptgt,ptgt,
    atgt,tempatgt,
    cbat,tempctgt,
    firstpt,
    firstarea,
    firstcounter    :    TargetRecType;

    PtFile,
    AFile           :    StreamObj;

    outfile         :    STRING;

{*****************  AREA TARGET POINTS  ************************}

PROCEDURE CreateAreaTgt;

BEGIN

OUTPUT;
OUTPUT;
OUTPUT("Do you wish to enter AREA Target points? <Y/N>");
INPUT(areaFlag);
```

```
areactr := 0;

IF (areaFlag = "Y") OR (areaFlag = "y")
    changeFlag := "Y";
ELSE
    changeFlag := "N";
END IF;

NEW(atgt);
firstarea := atgt;
WHILE (areaFlag = "Y") OR (areaFlag = "y")
    areactr := areactr +1;
    OUTPUT;
    OUTPUT;
    OUTPUT("Input the Target's Alpha-Numeric Designation");
    INPUT(atgt.tgtID);
    OUTPUT;
    OUTPUT("Input the Latitude of Area Target's center ",areactr," ");
    Reminder(sys);
    INPUT(atgt.lat);
    atgt.y := ABS(DConvert(atgt.lat, sys) - dlat) * 120000.0;
    OUTPUT("Input the Longitude of Area Target's center   ",areactr,
    " ");
    Reminder(sys);
    INPUT(atgt.lon);
    atgt.x := ABS(DConvert(atgt.lon, sys) - dlon) * scale;
    atgt.next := NILREC;

    OUTPUT;
    OUTPUT("Input Target Description - ie. Men In The Field");
    INPUT(atgt.type);
    OUTPUT;
    OUTPUT("Input Target Protection OR <None>");
    INPUT(atgt.protection);
    OUTPUT;
    OUTPUT("Input the target width in yards");
    INPUT(atgt.width);
    OUTPUT;
    OUTPUT("Input target length in yards");
    INPUT(atgt.length);
    OUTPUT;
    OUTPUT("Input target quantity");
    INPUT(atgt.quantity);

    OUTPUT("Another Area Target ? <Y/N>");
    INPUT(areaFlag);

    IF (areaFlag = "Y") OR (areaFlag = "y")
     tempatgt := atgt;
     NEW(atgt);
     tempatgt.next := atgt;
    END IF;

END WHILE;

WHILE (changeFlag = "Y") OR (changeFlag = "y")
```

```
OUTPUT;
OUTPUT;
OUTPUT("You have input ", areactr,  " Area Targets and they are as
follows:");
OUTPUT;
OUTPUT("Tgt no.     lat           lon");
i := 1;
atgt := firstarea;
WHILE i <= areactr
 OUTPUT("  ",i,"  ",atgt.lat,"  ",atgt.lon,"  ", atgt.type," ",
 atgt.protection,"  ",atgt.width,"  ",atgt.length,"  ",
 atgt.quantity);
 atgt := atgt.next;
 i := i + 1;
END WHILE;

OUTPUT;
OUTPUT;
OUTPUT("Would You like to Change An Area Target? <Y/N>");
INPUT(changeFlag);
OUTPUT;

IF (changeFlag = "Y") OR (changeFlag = "y")
 OUTPUT;
 OUTPUT;
 OUTPUT("INPUT the number of the record you wish to edit");
 INPUT(edit);
 OUTPUT;

 i := 1;
 atgt := firstarea;
 WHILE i < edit
      i := i + 1;
      atgt := atgt.next;
 END WHILE;

 OUTPUT("  ",i,"  ",atgt.lat,"  ",atgt.lon,"  ",atgt.type,"  ",
 atgt.protection,"  ",atgt.width,"  ",atgt.length,"  ",
 atgt.quantity);
 OUTPUT("Input the new Latitude ");
 Reminder(sys);
 INPUT(atgt.lat);
 OUTPUT("Input the new Longitude ");
 Reminder(sys);
 INPUT(atgt.lon);
 atgt.y := ABS(DConvert(atgt.lat, sys) - dlat) * 120000.0;
 atgt.x := ABS(DConvert(atgt.lon, sys) - dlon) * scale;
 OUTPUT;
 OUTPUT("Input Target Description - ie. Men_In_The_Field");
 INPUT(atgt.type);
 OUTPUT;
 OUTPUT("Input Target Protection OR <None>");
 INPUT(atgt.protection);
 OUTPUT;
 OUTPUT("Input the target width in yards");
```

```
      INPUT(atgt.width);
      OUTPUT;
      OUTPUT("Input target length in yards");
      INPUT(atgt.length);
      OUTPUT;
      OUTPUT("Input target quantity");
      INPUT(atgt.quantity);

    END IF;

END WHILE;

IF areactr > 0
    OUTPUT("Here in creating map shore files");
    outfile := MapFamily + ".atgt";
    OUTPUT("the outfile name is  ", outfile);
    NEW(AFile);
    ASK AFile TO Open(outfile,Output);

    ASK AFile TO WriteInt(areactr,4);
    ASK AFile TO WriteLn;

    i := 0;
    atgt := firstarea;

    WHILE i < areactr

      ASK AFile TO WriteString(atgt.tgtID);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteReal(atgt.x,12,6);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteReal(atgt.y,12,6);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteReal(atgt.lat,12,6);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteReal(atgt.lon,12,6);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteString(atgt.type);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteString(atgt.protection);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteInt(atgt.width,4);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteInt(atgt.length,4);
      ASK AFile TO WriteLn;

      ASK AFile TO WriteString(atgt.quantity);
      ASK AFile TO WriteLn;
```

```
     atgt := atgt.next;
     i := i +1;

    END WHILE;

    ASK AFile TO Close;
    DISPOSE(AFile);

END IF;

END PROCEDURE;

{*****************  POINT TARGET POINTS  ************************}

PROCEDURE CreatePointTgt;

BEGIN

OUTPUT;
OUTPUT;
OUTPUT("Do you wish to enter POINT Target points? <Y/N>");
INPUT(ptFlag);
ptctr := 0;

IF (ptFlag = "Y") OR (ptFlag = "y")
    changeFlag := "Y";
ELSE
    changeFlag := "N";
END IF;

NEW(ptgt);
firstpt := ptgt;
WHILE (ptFlag = "Y") OR (ptFlag = "y")
    ptctr := ptctr +1;
    OUTPUT;
    OUTPUT;
    OUTPUT("Input the Target's Alpha-Numeric Designation");
    INPUT(ptgt.tgtID);
    OUTPUT;
    OUTPUT("Input the Latitude of Point Target's center ",ptctr,"  ");
    Reminder(sys);
    INPUT(ptgt.lat);
    ptgt.y := ABS(DConvert(ptgt.lat, sys) - dlat) * 120000.0;
    OUTPUT("Input the Longitude of Area Target's center ",ptctr,"  ");
    Reminder(sys);
    INPUT(ptgt.lon);
    ptgt.x := ABS(DConvert(ptgt.lon, sys) - dlon) * scale;
    ptgt.next := NILREC;

    OUTPUT;
    OUTPUT("Input Target Description - ie. Microwave_Tower");
    INPUT(ptgt.type);
    OUTPUT;
    OUTPUT("Input Target Protection OR <None>");
    INPUT(ptgt.protection);
```

```
    ptgt.width := 1;
    ptgt.length := 1;
    OUTPUT;
    OUTPUT("Input target quantity");
    INPUT(ptgt.quantity);

    OUTPUT("Another Point Target ? <Y/N>");
    INPUT(ptFlag);

    IF (ptFlag = "Y") OR (ptFlag = "y")
     tempptgt := ptgt;
     NEW(ptgt);
     tempptgt.next := ptgt;
    END IF;

END WHILE;

WHILE (changeFlag = "Y") OR (changeFlag = "y")

    OUTPUT;
    OUTPUT;
    OUTPUT("You have input ", ptctr,  " Point Targets and they are as
    follows:");
    OUTPUT;
    OUTPUT("Tgt no.   lat       lon");
    i := 1;
    ptgt := firstpt;
    WHILE i <= ptctr
     OUTPUT("  ",i,"  ",ptgt.lat,"  ",ptgt.lon,"  ",ptgt.type,"  ",
     ptgt.protection,"  ",ptgt.quantity);
     ptgt := ptgt.next;
     i := i + 1;
    END WHILE;

    OUTPUT;
    OUTPUT;
    OUTPUT("Would You like to Change An Point Target? <Y/N>");
    INPUT(changeFlag);
    OUTPUT;

    IF (changeFlag = "Y") OR (changeFlag = "y")
     OUTPUT;
     OUTPUT;
     OUTPUT("INPUT the number of the record you wish to edit");
     INPUT(edit);
     OUTPUT;

     i := 1;
     ptgt := firstpt;
     WHILE i < edit
          i := i + 1;
          ptgt := ptgt.next;
     END WHILE;

     OUTPUT("  ",i,"  ",ptgt.lat,"  ",ptgt.lon,"  ",ptgt.type,"  ",
     ptgt.protection,"  ",ptgt.quantity);
```

```
      OUTPUT("Input the new Latitude ");
      Reminder(sys);
      INPUT(ptgt.lat);
      OUTPUT("Input the new Longitude ");
      Reminder(sys);
      INPUT(ptgt.lon);
      ptgt.y := ABS(DConvert(ptgt.lat, sys) - dlat) * 120000.0;
      ptgt.x := ABS(DConvert(ptgt.lon, sys) - dlon) * scale;
      OUTPUT;
      OUTPUT("Input Target Description - ie. Men In The Field");
      INPUT(ptgt.type);
      OUTPUT;
      OUTPUT("Input Target Protection OR <None>");
      INPUT(ptgt.protection);
      OUTPUT;
      OUTPUT("Input target quantity");
      INPUT(ptgt.quantity);

    END IF;

END WHILE;

IF ptctr > 0
    OUTPUT("Here in creating map shore files");
    outfile := MapFamily + ".ptgt";
    OUTPUT("the outfile name is  ", outfile);
    NEW(PtFile);
    ASK PtFile TO Open(outfile,Output);

    ASK PtFile TO WriteInt(ptctr,4);
    ASK PtFile TO WriteLn;

    i := 0;
    ptgt := firstpt;

    WHILE i < ptctr

      ASK PtFile TO WriteString(ptgt.tgtID);
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteReal(ptgt.x,12,6);
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteReal(ptgt.y,12,6);
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteReal(ptgt.lat,12,6);
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteReal(ptgt.lon,12,6);
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteString(ptgt.type);
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteString(ptgt.protection);
```

```
      ASK PtFile TO WriteLn;

      ASK PtFile TO WriteString(ptgt.quantity);
      ASK PtFile TO WriteLn;

      ptgt := ptgt.next;
      i := i +1;

    END WHILE;

    ASK PtFile TO Close;
    DISPOSE(PtFile);

END IF;

END PROCEDURE;

END MODULE.
```

REFERENCES

1. Commander Naval Surface Force United States Atlantic Fleet Instruction 3570.2D, *Training and Qualification of Ships in Naval Gunfire Support (NGFS) and Shore Bombardment (Gunsmoke Manual)* , 7 August 1989.

2. *MODSIM II, The Language for Object-Oriented Programmin, Reference Manual,* CACI Products Comapny, 1991.

3. *SIMGRAPHICS II Reference Manual,* CACI Products Comapny, 1991.

4. Hobbs, R.R, *Marine Navigation 1: Piloting,* United States Naval Institute, 1977.

BIBLIOGRAPHY

Brantley, P., Fox, B. L., and Schrage, L. E., *A Guide to Simulation, Second Edition*, Springer-Verlag, New York, Inc., 1987.

Gibson, E., "Objects—Born and Bred," *Byte*, v. 15, n. 10, pp. 245-254, October, 1990.

Hughes, W. P., *Fleet Tactics*, Naval Institute Press, 1986.

Naval Warfare Publication, *Supporting Arms in Amphibious Operations, NWP 22-2 (Rev. B)*.

Rice, J. A., *Mathematical Statistics and Data Analysis*, Wadsworth & Brooks/Cole Advanced Books & Software, 1988.

Richardson, D., *Naval Armament*, Jane's Publishing Incorporated, 1982.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	LT Warren A. Mazanec Submarine Officer Advance Course Naval Submarine Base New London Groton, CT 06349	2
4.	Professor Michael P. Bailey, Code OR/Ba Naval Postgraduate School Monterey, CA 93943-5000	5
5.	Professor Wayne P. Hughes, Code OR/Hl Naval Postgraduate School Monterey, CA 93943-5000	2
6.	LCDR Roger Stemp, Code OR/St Naval Postgraduate School Monterey, CA 93943-5000	1
7.	John Bowden Naval Weapons Support Center Crane, IN 47522	1
8.	Hal Duncan CACI Products Company 3344 North Torrey Pines Court La Jolla, CA 92037	1
9.	CAPT Henry Bress Naval Research Laboratory 4555 Overlook Ave., SW Code 1505 Washington, DC 20375-5000	1