NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A COMMAND AND CONTROLWARGAME TO TRAIN OFFICERS IN THE INTEGRATION OF TACTICS AND LOGISTICS IN A FIELD ARTILLERY BATTALION

by

Michael W. Schneider and Anthony R. Ferrara

March 1989

Thesis Advisor — Samuel H. Parry
Co-Advisor — William J. Walsh

Approved for public release; distribution is unlimited

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (If Applicable): 39	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (city, state, and ZIP code): Monterey, CA 93943-5000	7b Address (city, state, and ZIP code): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

11 Title (Include Security Classification): A Command and Control Wargame to Train Officers in the Integration of Tactics and Logistics in a Field Artillery Battalion

12 Personal Author(s): Michael W. Schneider, Anthony R. Ferrara

13a Type of Report	13b Time Covered	14 Date of Report (year, month,day)	15 Page Count
Master's Thesis	From To	March 1989	349

16 Supplementary Notation: The views expressed in this thesis 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	Subgroup	Wargame; Command and Control; Command Post Exercise (CPX); Field Artillery

19 Abstract (continue on reverse if necessary and identify by block number)

Due to peacetime training limitations, the integration of tactics and logistics as it relates to the command and control of a field artillery battalion cannot be easily practiced. This thesis presents a computer assisted wargame which will give battalion staff officers some experience in dealing with this shortcoming. The wargame emphasizes the decision maker in the command and control system. Specifically, this wargame forces the decision maker to consider numerous tactics / logistics interface issues and then make a series of command and control type decisions. At the end of each game, the player's performance is evaluated in terms of howitzer availability time, casualty rates, vulnerability rates, and ammunition optimization. The wargame itself is highly flexible and is capable of being played in support of a full scale battalion command post exercise or during weekly officer professional development time. Theses. (AW)

20 Distribution/Availability of Abstract	21 Abstract Security Classification	
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified	
22a Name of Responsible Individual: Samuel H.Parry	22b Telephone (Include Area code): (408) 646-2779	22c Office Symbol: 55Py

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted — All other editions are obsolete

security classification of this page: Unclassified

Approved for public release; distribution is unlimited.

A Command and Control Wargame to Train Officers in the Integration of Tactics and Logistics in a Field Artillery Battalion

Michael W. Schneider
Captain, United States Army
B.S., United States Military Academy, 1980

and

Anthony R. Ferrara
Captain, United States Army
B.S., United States Military Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTERS OF SCIENCE IN SYSTEMS TECHNOLOGY
(COMMAND, CONTROL AND COMMUNICATIONS)

from the

NAVAL POSTGRADUATE SCHOOL
March 1989

Authors: Michael W. Schneider
Michael W. Schneider

Courtney R. Ferrara
Anthony R. Ferrara

Approved By: S.H. Parry
Samuel H. Parry, Thesis Advisor

William J. Walsh
William J. Walsh, Co-Advisor

Carl R. Jones, Chairman,
Command, Control and Communications
Academic Group

Harrison Shull
Harrison Shull,
Provost and Academic Dean

ABSTRACT

Due to peacetime training limitations, the integration of tactics and logistics as it relates to the command and control of a field artillery battalion cannot be easily practiced. This thesis presents a computer assisted wargame which will give battalion staff officers some experience in dealing with this shortcoming. The wargame emphasizes the decision maker in the command and control system. Specifically, this wargame forces the decision maker to consider numerous tactics / logistics interface issues and then make a series of command and control type decisions. At the end of each game, the player's performance is evaluated in terms of howitzer availability time, casualty rates, vulnerability rates, and ammunition optimization. The wargame itself is highly flexible and is capable of being played in support of a full scale battalion command post exercise or during weekly officer professional development time.

TABLE OF CONTENTS

I. DESCRIPTION OF THE PROBLEM

A. INTRODUCTION

In the United States Army, combat-arms-type units have a very difficult time finding ways to conduct fully realistic training. In a peacetime environment there are many factors which inhibit realistic training. Budgetary constraints limit the scope and duration of training as well as minimize the amount of ammunition expended. Safety considerations limit intensity and realism in training. And, the lack of adequate training areas compresses the size of the "training battlefield" so that training does not take place over realistic distances. The implication to those concerned with the command and control of combat units is that there is very little "hands on" experience available in the integration of tactics and logistics as it relates to the command and control of these units. This lack of hands on experience by our officers is a potential deficiency which deserves some attention.

In order to address this deficiency, there are many options available. These options range from training through "field training exercises" (FTX) down to classroom instruction. An FTX would theoretically provide the best situation for training the tactics / logistics interface. However, FTXs rarely provide more than a start point for

understanding the tactics / logistics interface issues. This is true due to budget constraints, safety considerations, training area limitations, and low peace time ammunition consumption rates. In short, we are never faced with the problems of moving enough ammunition to keep up with tactics or slowing tactics to keep pace with ammunition resupply rates because we rarely fire ammunition on combined arms exercises (at least not at a realistic rate).

Classroom instruction can serve as an introduction to the issues related to the integration of tactics and logistics but it lacks the critical element of hands-on experience. An individual must confront and solve these type of problems for himself in order to be effective at the beginning of the next war.

The next best thing to an FTX is a command post exercise (CPX). In the CPX, all but the command and control elements are played notionally. Normally, some type of wargame is played at a remote location and then the various command posts must perform their wartime functions in accordance with the wargame scenario. In most cases, when CPX's are run by higher level headquarters, the resolution is such that very little training on the integration of tactics and logistics takes place at the battalion level. In fact, the nature of the tactics / logistics problem is such that it is very difficult for a wargame to provide an environment which

trains officers in the integration of these functions. Most wargames either emphasize tactics or logistics but not both. This is due to the fact that it requires much more detail than can be generated under normal conditions in a manually run wargame.

B. WHAT IS MEANT BY THE INTEGRATION OF TACTICS AND LOGISTICS?

According to Army Field Manual 100-5...

> Commanders must plan tactics and logistics concurrently to insure that the tactical scheme of maneuver and fire support are logistically supportable. They consider the constraints that Combat Service Support planners identify. They modify unsupportable plans or accept the risks involved. [ref. 1: p. 5-1]

This refers primarily to the integration of tactics and logistics in the planning of operations. However, the integration process does not stop there. During the conduct of operations, the integration of tactics and logistics is a continuous process which is driven by factors such as expenditure rates, casualty rates, supply availability, supply transportation capacity, and the tactical situation. All of these factors must be carefully managed in order to conduct a cohesive operation. Failure to integrate tactics and logistics can result in failure to accomplish the unit's overall mission.

C. SCOPE OF THE THESIS

This thesis describes the use of a battalion level, computer assisted wargame, played by the battalion

operations and logistics staff sections, to train officers in the integration of tactics and logistics as it applies to the command and control of a field artillery battalion. Limiting the discussion to the command and control of a field artillery battalion was done only to provide a manageable start point for exploring the larger issue of training officers in the integration of tactics and logistics. The field artillery battalion is a logical choice as a start point because of the field artillery's dependence on extremely large quantities of bulk ammunition. In wartime, this dependence will dictate the movement of firing batteries and the rate at which firing batteries can shoot. Ammunition trucks will become a scarce resource that needs to be carefully managed to ensure optimum ammunition resupply as well as survivability. Therefore, in addition to consideration of howitzer vulnerability thresholds, the tactical situation, movement plans, fire plans, and casualty projections, the commander and his staff will also have to consider unit distribution plans or point distribution plans, potential limitations on firing rates, ammunition truck movement plans, and truck vulnerability thresholds.

This thesis presents a wargame that attempts to capture these concepts in a form which will provide a learning environment for those tasked with the command and control of a field artillery battalion. The wargame is only computer assisted. It is important to note that the "essential

ingredients of any command and control system...are the commanders or decision makers themselves." [ref. 2: p. 618] The computer portion of the wargame will only provide relevant information to the decision maker, execute his decisions, and keep track of the game's statistics. At the end of the game, a measure of effectiveness (MOE) will be computed which will provide an assessment of the player's performance. The MOE is based on a computation of the howitzer availability time, the amount of time spent at a critical vulnerability level, the amount of time lost due to casualties, and the amount of time spent short of sufficient amounts of ammunition. This will force the decision maker to plan ahead and to consider all of the relevant concepts in order to maximize howitzer availability time, minimize vulnerability, minimize casualties, and optimize ammunition resupply. The intent of the wargame is to provide an environment in which officers can learn to consider the factors which are critical to the integration of tactics and logistics.

In order to build such a wargame, an analysis of the architecture of field artillery battalion is required. This analysis is the topic of Chapter II. Using this analysis, the approach taken to design a wargame which trains officers in the integration of tactics and logistics as it relates to the command and control of a field artillery battalion is discussed in Chapter III. Chapter IV describes the

processes used by the computer to derive some of the more important aspects of the wargame. A user's manual for the wargame is provided in Chapter V. Finally, Chapter VI discusses suggested enhancements and potential uses of the wargame.

II. THE FIELD ARTILLERY BATTALION SYSTEM / ARCHITECTURE

A. INTRODUCTION

Before building a wargame that simulates the operations of a field artillery battalion, it is critically important to analyze the battalion in order to determine how it works. More specifically, it is important to know not only the internal interactions which take place within the battalion, but also the interactions the battalion has with external elements. It is important to understand the battalion's environment and its mission. More importantly, it is important to understand how the environment affects the battalion's ability to achieve its goals and if there are any other constraints which may hinder the achievement of these goals.

One proven method of analysis is through the use of the "Systems Approach." A system, as defined by Professor James G. Taylor of the Naval Postgraduate School, is a "collection of elements which combine together to form a whole in order to accomplish a goal." [ref. 3] The systems approach provides a method of developing the system by considering the system's goals, the environment, the mission, the resources, and the management [ref. 4: p. 44].

To supplement the systems approach, the system architecture provides a depiction of the "interconnections

between elements of the system. Interconnections can be based on time, functional, informational or spatial relationships." [ref. 4: p. 126]

Through the use of both the systems approach and the systems architecture, the field artillery battalion can be thoroughly analyzed. A conscious effort has been made to separate the analysis using the systems approach and the systems architecture. It was felt that it would be better to first conduct a written analysis of the battalion using the systems approach and then fill in between the lines by letting each graphical portrayal of the system architecture say "a thousand words". This analysis will be helpful in determining which aspects of the battalion will be germane to the design of the wargame and which system characteristics can be ignored.

B. FIELD ARTILLERY MISSION

The mission of the field artillery is to "destroy, neutralize, or suppress the enemy by indirect fires and to integrate all fire support into combat operations. Successful execution of this mission demands effective integration of field artillery fires into the scheme of maneuver and swift, exact, execution from the time a target is acquired until ordnance is delivered on target." [ref. 5: p. 1-1] Field Manual 6-20-1, FIELD ARTILLERY CANNON BATTALION OPERATIONS, goes on to say that in order for the

battalion to support maneuver forces on the battlefield successfully, it must survive to perform the following 10 basic tasks:

1. Target Acquisition--Detecting and engaging targets that threaten maneuver elements of the supported brigade.

2. Meteorology, Survey, Technical Fire Direction--The battalion operations officer (S3) insures that fire direction centers (FDC) have the information they need to conduct effective fire direction.

3. Fire Planning--Field artillery fires are planned to achieve one of three effects on the target: suppression, neutralization or destruction. The desired effect will be determined by the supported unit commander, the fire support coordinator (FSCOORD), or the fire direction officer.

4. Tactical Fire Control--Tactical fire direction includes selection of rounds, shell/fuze combinations, and designation of units to fire.

5. Plans and Orders--Command and control of the field artillery cannon battalion is established through the assignment of tactical missions.

6. Positioning--The planning for the selection of any position must include consideration of communication requirements and combat service support in addition to the mission, terrain, and tactical situation.

7. Reconnaissance--Reconnaissance is performed to select the best battalion and battery positions, march routes, start and release points, command posts, observation posts, and communication sites, and to analyze the terrain where the battle will be fought. Prior to or concurrent with reconnaissance, the field artillery commander/S3 should coordinate with the maneuver commander/S3 to determine what areas maneuver units plan to occupy. Mutual agreement must be established to make the best use of the available terrain.

8. Displacement--Field artillery battalions must frequently displace to provide continuous fire support to maneuver units. There are three general ways that a field artillery battalion moves; by unit displacement,

by echelon, and by battery. In unit displacement, the entire battalion moves at once and no units are available to provide fire support. In displacement by echelon, some portion of the battalion moves and sets up in the next position and then the other portion of the battalion moves. In displacement by battery, one battery at a time displaces. This method provides the maximum amount of fire support at any given time but it is sometimes too slow to keep up with maneuver.

9. Communications--The cannon battalion commander must rely on communications to control elements of his command, gather information, distribute intelligence, and coordinate fire support. The primary means of communications in the cannon battalion are radio and wire.

10. Combat Service Support (CSS)--CSS is the process of keeping the maximum number of weapon systems operational. The support functions and operations in the battalion by the personnel officer (S1), logistics officer (S4), or any other supervisor must be closely coordinated with tactical operations. A continuous exchange of information among CSS coordinators, the S3, and battery commanders is essential to the success of both tactical and logistical plans. [ref. 5]

The mission of the field artillery and its implied tasks are critical to understanding how the field artillery battalion works and how to build a wargame which simulates the battalion. The mission can be broken into two parts. The first is the requirement to be capable of delivering timely, accurate, and effective fires and the second is to provide those fires in a manner which supports the overall scheme of maneuver. The ten basic tasks are the implied tasks which any field artillery unit must be able to perform in order to accomplish the field artillery mission. These tasks are important because they represent functions that the wargame will have to be able to perform to varying degrees depending on the mission and goals of the wargame.

The field artillery mission clearly implies that the field artillery battalion is a part of a larger system which, among other things, dictates the scheme of maneuver which must be supported. The mission also implies that in order to provide swift, exact indirect fires, much coordination and interaction with external elements must take place in order to ensure that the artillery can range the enemy, has enough ammunition on hand to support the operations, and has coordinated positions to which it can displace. Finally, the mission implies that the unit must be able to survive to perform all of these tasks.

C. GOALS

The mission and its implied tasks can be boiled down to several goals. These goals are necessary because they help to quantify the degree to which the unit is effective in the accomplishment of its mission. They also help to further describe the system and the positive and negative motivations which are inherent in the system. In this respect, an analysis of the goals will be very useful to the design of the wargame and particularly to the design of any MOEs which may be necessary.

The goals were chosen to address the mission and its implied tasks. They are stated in quantifiable terms. These goals are closely related and some could actually be considered as sub-goals to other goals. However, they have

been separated into different goals based upon the fact that there are specific command and control actions which can be taken to optimize each goal. In many cases these actions serve to oppose other goals and so a set of tradeoffs must be considered. The four goals are described below.

1. Maximize Availability Time

First and foremost, the battalion must be able to support the maneuver units. This means that the battalion must have the maximum number of cannons in a firing status for the maximum amount of time. It also means that the firing units must always have ammunition to fire and that they should minimize the number of tubes lost to maintenance or enemy fire.

In operational terms, it means that firing units must displace as little as possible without letting the enemy get out of range and without letting themselves get too vulnerable to enemy detection. This goal can be measured in tube hours, i.e., the sum of the total number of hours each tube is capable of providing fire support.

2. Minimize Vulnerability

A firing unit becomes more vulnerable to enemy detection in two ways. First, the longer a unit is in position, the more likely it is that it has been located by aerial reconnaissance. Second, the more rounds a firing battery fires out of a position, the more likely that unit

has been located by enemy counter-battery radar. These probabilities of detection can be determined from probability of detection versus time and versus "rounds fired" curves, respectively. These probabilities can be combined to derive a vulnerability factor.

In operational terms, a commander must be willing to accept some degree of vulnerability. Some positions are more vulnerable than others due to the amount of cover and concealment available. A vulnerability threshold can be specified by a commander in terms of rounds fired out of a position when time in the position is less than some specified maximum allowable time in that position. A unit should move when it reaches the commander's vulnerability threshold so that its vulnerability level is reset at zero. Many times, due to the tactical situation or due to a lack of prior planning, the commander can not move a unit when it reaches the commander's vulnerability threshold. It is the time spent above the vulnerability threshold that the commander wants to minimize. This goal can be measured in the number of tube hours spent above the vulnerability threshold.

3. Minimize Casualties

Casualties can be incurred both in positions and during displacement. In operational terms, a commander can minimize casualties by moving often thereby minimizing vulnerability. Or, he may take the safest but not

necessarily the shortest routes on displacements. This goal to conserve assets for both current and future operations must be balanced against the goal to maximize availability time. This goal can be measured in terms of the tube / truck / equipment / personnel time lost during the operation due to casualties.

4. Optimize Logistics Operations

Logistics operations also experience vulnerability and casualties and therefore goals two and three apply to logistics units. More importantly, logistics units are critical to the attainment of the first goal. Fire units which are low on ammunition or fuel can not adequately support maneuver forces and fire units which are out of those supplies are not available to support maneuver units.

In operational terms, there are many different forms of resupply operations. The battalion trains can deliver supplies to the unit (unit distribution) or it can deliver the supplies to a point where the fire units will have to go to pick up their supplies (point distribution). The trains can bundle their trucks into convoys to pick up and deliver supplies or they can use the "trickle" method to pick up and deliver supplies by sending as few as one truck at a time as the trucks become available. The method used depends on the tactical situation, the enemy situation, the terrain and the level of training of the units.

Optimization refers to making the most efficient use of the supplies and the resupply assets in order to ensure accomplishment of the field artillery mission. This goal can be measured in terms of the amount of time firing units spend critically short, or out of, critical supplies.

This analysis of the field artillery battalion's goals and especially the statements of the goals in operational terms tells us a lot about the internal operation of the battalion and about what motivates operational decisions. These motivating factors must also be present in the wargame in order to have any true training value.

D. ENVIRONMENT

The field artillery battalion exists in an environment consisting of both friendly and hostile elements. The friendly environment can also be divided. It consists of command relationships and support relationships.

Command relationships refer to the fact that the field artillery battalion is commanded by the division artillery which is commanded by the division and the fact that the maneuver battalions are commanded by the maneuver brigade which is in turn commanded by the division and finally that the support battalions are commanded by the division support command which is also commanded by the division. In short, there is no command relationship between the field artillery battalion and the maneuver units or the support units.

Support relationships refer to the relationship that does exist between the field artillery battalion and the maneuver units and the support units. The direct support field artillery battalion has a mission to provide close support to a maneuver brigade. With this mission, according to Field Manual 6-20-1, comes the requirement to provide first priority on calls for fire to the brigade and to provide fire support coordinating personnel to the brigade. The division artillery assigns missions to the field artillery battalion. By the same token, the division support command assigns missions to its subordinate units to provide service support to various divisional units. In this manner, the field artillery battalion will have a support unit assigned to provide supplies to support its mission.

The hostile side of the environment refers to the fact that the enemy exists to prevent accomplishment of the battalion's mission. It does this by producing casualties, by interdicting critical supplies, disrupting the plan, and disrupting command, control and communications. There are many forms of hostile acts performed by the enemy. Any wargame must consider this aspect of the environment.

Another aspect of the environment is the fact that all the elements of the system and the elements of the external environment exist on the battlefield. They are separated by considerable distances and must move frequently in order to

stay in the battle. They are positioned so as to best facilitate accomplishment of their mission and to assure survivability. They are positioned on terrain which provides some degree of cover and concealment and which may or may not facilitate communications with other elements of the system and environment.

Finally, a discussion of the environment would not be complete without considering what Clausewitz called "FRICTION." [ref. 6] This refers to the fact that operations never occur as planned. There are many reasons for plans going wrong which are beyond the control of the commander. All of them are lumped together and called friction. This requires commanders to be flexible. They must be able to modify plans quickly and issue effective fragmentary orders.

An understanding of the environment is necessary for a full understanding of the field artillery battalion system. It is also necessary to include the critical aspects of the environment in the wargame.

E. RESOURCES

The battalion's resources primarily include its people, its equipment, and its supplies. Its people include its leaders and its soldiers. They are trained and work together as a part of a team. They operate the equipment which helps to accomplish the mission. People are the

battalion's most important resource. It takes people to operate the equipment and to use the supplies. It takes trained people who can work together as a cohesive unit to accomplish the battalion's mission.

The field artillery battalion's most critical equipment includes its weapon systems, fire direction systems, and supply trucks. All are critical to accomplishment of its mission.

The field artillery battalion's most critical supplies are food, fuel, and ammunition. All are critical to accomplishment of its mission. Units establish required supply rates (RSR) which tell higher level units how much ammunition is required to accomplish the mission. The higher level unit then establishes the controlled supply rate (CSR) which is based upon how much ammunition is available. If the CSR is less than the RSR, then the unit must curtail operations to avoid exceeding the CSR.

Resources can be easily represented by computer simulation. However, the manipulation and expenditure and conservation of resources to accomplish the mission, which is the subject of the next paragraph, is best left to the human interaction portion of the wargame.

F. MANAGEMENT

In the military, the commander is responsible for all that his unit does or fails to do. However, he has a staff

to assist him in the two primary management functions of planning and controlling the battalion. The commander alone, however, must provide leadership for the battalion. His leadership serves to inspire and motivate his soldiers and it provides direction for his subordinate units and his staff. In addition to providing direction, the commander must approve all plans and issue all orders in his name. These aspects of command are very difficult to simulate using computers and are best represented in wargames by the human players themselves.

Staffs conduct planning under the commander's guidance. The S3 is responsible for incorporating all of the staff estimates and determining the recommended course of action. The commander then either approves or selects his own course of action. The staff then prepares the plans to execute that course of action. The plan becomes an order when issued to subordinates with an execution time. Planning is a complex process which requires much training. It is therefore suitable for the human element in a wargame.

Controlling refers to the process of monitoring operations to ensure that they are being carried out within the commander's intent. It also refers to monitoring the overall tactical situation to determine if the plan is still appropriate. If it is not appropriate, it involves modifying the plan and issuing fragmentary orders.

Controlling is a continuous process which requires the commander and his staff to monitor the battalion's environment and resources and to make the tradeoffs implied in the battalion's goals, all in order to accomplish the battalion's mission. This complex process is especially appropriate for the human interaction portion of the wargame.

G. ARCHITECTURAL LINK DEPICTIONS

The discussion of the field artillery battalion thus far using the systems approach has been very useful in understanding the battalion so that an effective wargame can be designed. Graphical portrayal of the discussion in the form of the battalion's system architecture will now be useful in clarifying and adding to that discussion.

Systems architectures can be depicted in several different ways depending on the perspective desired. For the purposes of this thesis, functional, spatial, informational, and time depictions will be used. A description of each can be found in this section.

1. Functional

"Functional architecture describes the technical structure of large systems." [ref. 7: p. 2.4] Figure 2.1 is a functional depiction of the divisional supersystem.

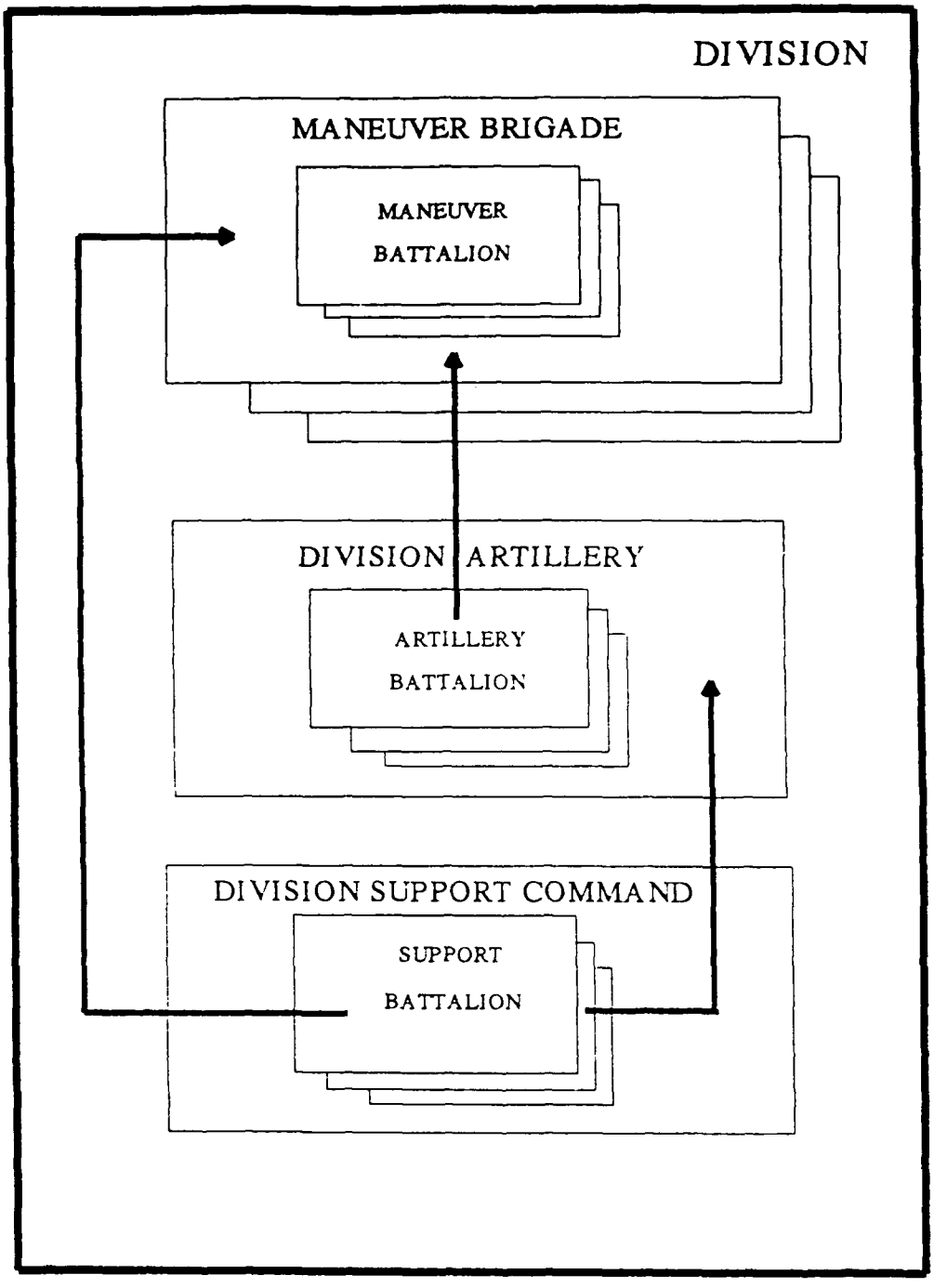

Figure 2.1 Divisional Super-System

Everything outside of the field artillery battalion itself is a part of the battalion's environment. Boxes within other boxes indicate command relationships. Arrows indicate support relationships.

Figure 2.2 is a functional depiction of the field artillery battalion. Again, boxes within other boxes indicate command relationships. However, in this case the primary function of each element is listed next to its box.

2. Spatial

Spatial architecture depicts the physical relationship of the elements in the system. Figure 2.3 is a spatial depiction of the field artillery battalion system. Since a direct support field artillery battalion normally supports a maneuver brigade, the brigade's battlefield geometry is depicted. The firing batteries must be able to project approximately two-thirds of their range into enemy territory. Therefore, the howitzers are found four to eight kilometers behind the front line of troops (FLOT). Normally, all three batteries support the entire brigade rather than one battery per maneuver battalion. Therefore, batteries are positioned where they can best support the entire brigade in accordance with the battalion's battery displacement plan.

The battalion tactical operations center (TOC) must be positioned farther back for survivability. But it must be in close proximity to the supported brigade headquarters

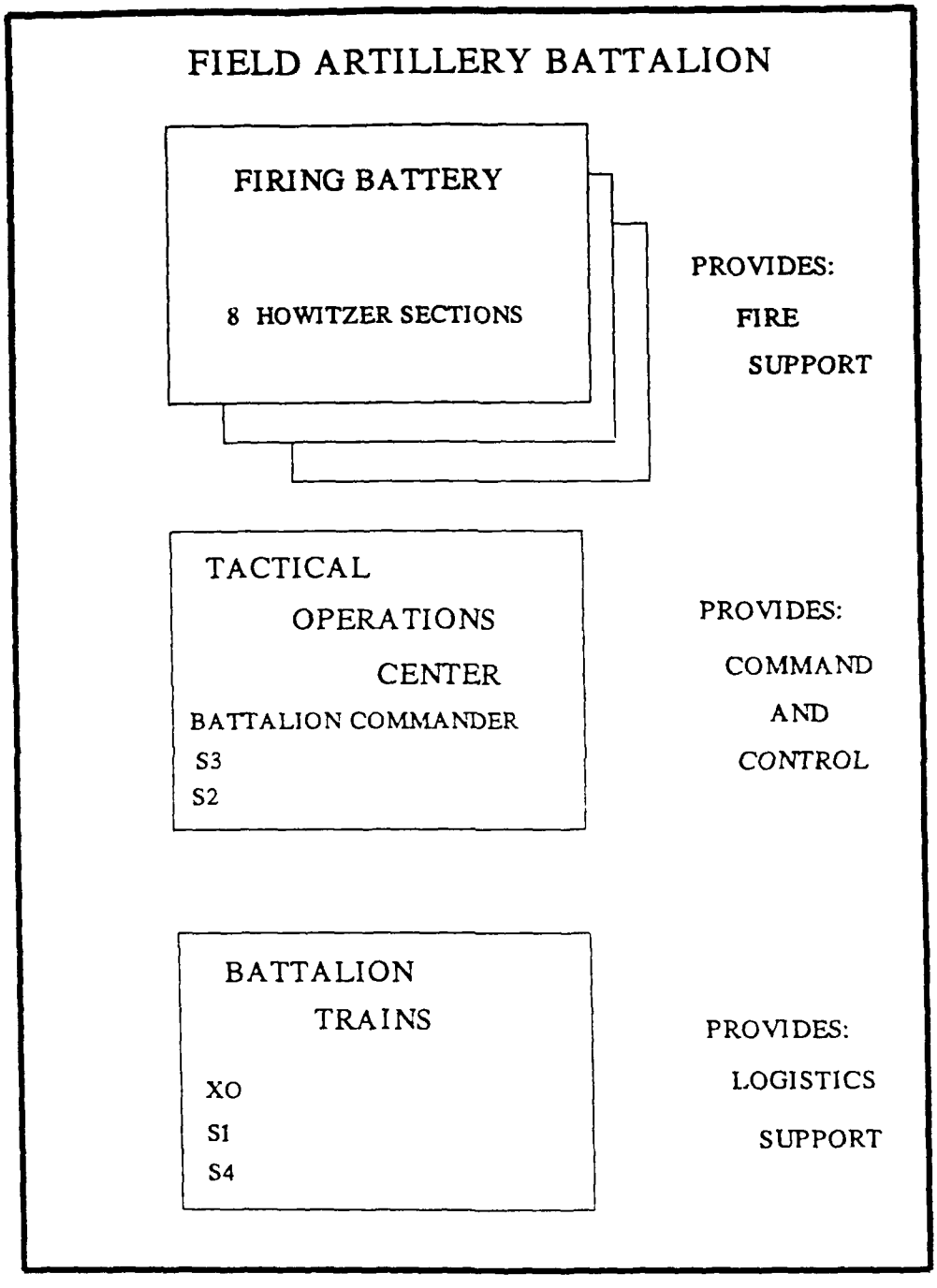

Figure 2.2 Field Artillery Battalion System

FIELD ARTILLERY BATTALION

SPACIAL ARCHITECTURE

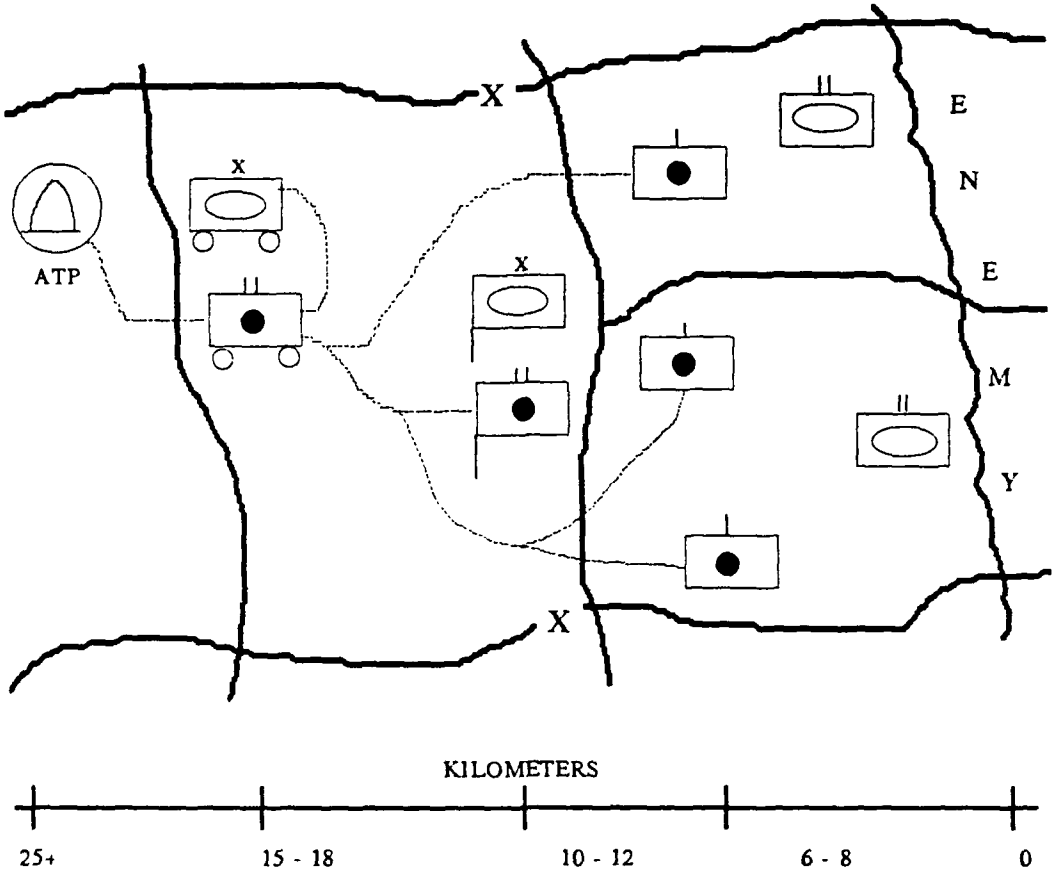

Figure 2.3 Field Artillery Battalion Spacial Architecture

and it must be able to communicate with the firing batteries and the brigade TOC.

The battalion trains are located even farther to the rear for survivability and for ease of coordination with support units. The dotted lines which emanate from the trains indicate that resupply convoys are constantly on the road.

The spatial architecture is important because it puts the physical relationships of the elements in perspective. An appreciation of the distances involved is necessary to the wargame to simulate the times required for events to transpire.

It is also important to note that Figure 2.3 is only a snapshot of the system at a particular instant in time. Elements of the system must continuously move in order to perform their missions and enhance survivability.

3. Information

Information architecture is especially important to the design of the wargame because it depicts the information flow both within the system and to external elements. Figure 2.4 depicts this flow for the battalion and its environment. Only the most critical elements of information have been depicted.

Implied in Figure 2.4 is the fact that if information flows between two elements, then a communications system must exist between them. This figure

FIELD ARTILLERY BATTALION

INFORMATION ARCHITECTURE

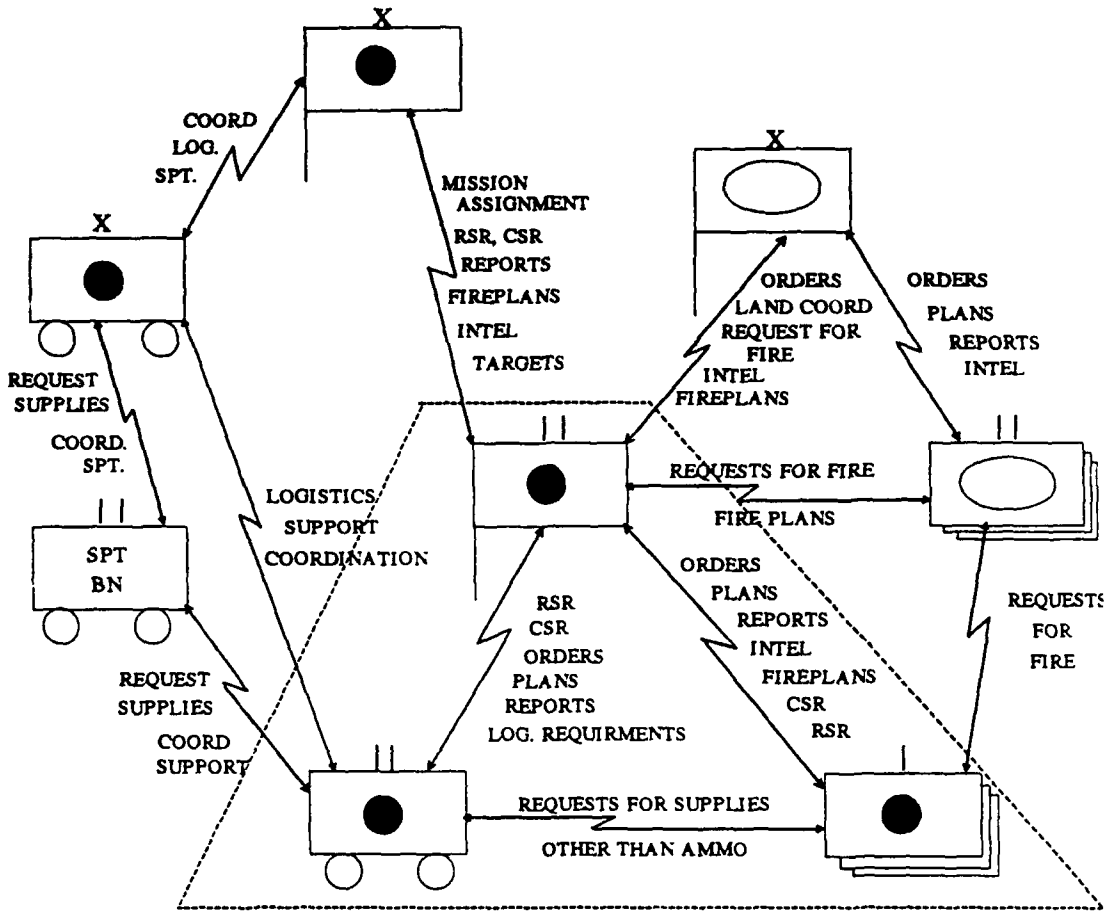

Figure 2.4 Field Artillery Battalion Information Architecture

does not show the communications nets. It only relates that communications exists between the elements. The habitual flow of information between elements leads to the use of various message formats for different types of information and it leads to standard operating procedures which dictate when and how the messages will be sent.

Also depicted on Figure 2.4 with a dotted line is the system boundary. Everything within the dotted line is in the field artillery battalion system. Everything outside the dotted line is external to the system. The lines of communications that cross the boundaries are then external communications and the lines of communications within the boundaries are internal communications.

The information architecture is extremely important to the analysis as it relates to the design of the wargame. The primary problem will be which information is needed in the wargame and which information is not necessary to accomplish the goals of the wargame. Dividing the communications into internal and external communications also will be helpful in arriving at a solution to the problem.

4. Architectural

Architectural depictions based on time relationships help to place events in their natural order. In a complex system such as the field artillery battalion, this can be very difficult to do. Schedules of events are not

appropriate because events do not usually happen at a specific time. It is even difficult to attempt to put events into a chronological order without assigning specific times. The best that can be done is to use the concepts of time based depictions of architecture to analyze the combat process.

Figure 2.5 is the Conceptual Combat Operations Process Model [ref. 8: p. 27]. By avoiding specific events and analyzing the generic process instead, an understanding of the logical order of the component parts of any operation can be gained. It is important to note that as a generic model, it applies to tactical operations, logistical operations, and applies equally well to both current operations and future operations plans.

Figure 2.5 implies that combat operations consist of a cyclic process which is stimulated by the environment and leads to an attempt to understand the environment. This is followed by the formulation of alternative courses of action, guidance from higher headquarters, and ultimately a decision to take some action which relates to the perception of the environment. This decision is then sent to subordinate elements and is executed. This in turn has some impact on the environment and changes the stimulus which caused the action in the first place and the process repeats itself.

CONCEPTUAL COMBAT OPERATIONS PROCESS MODEL

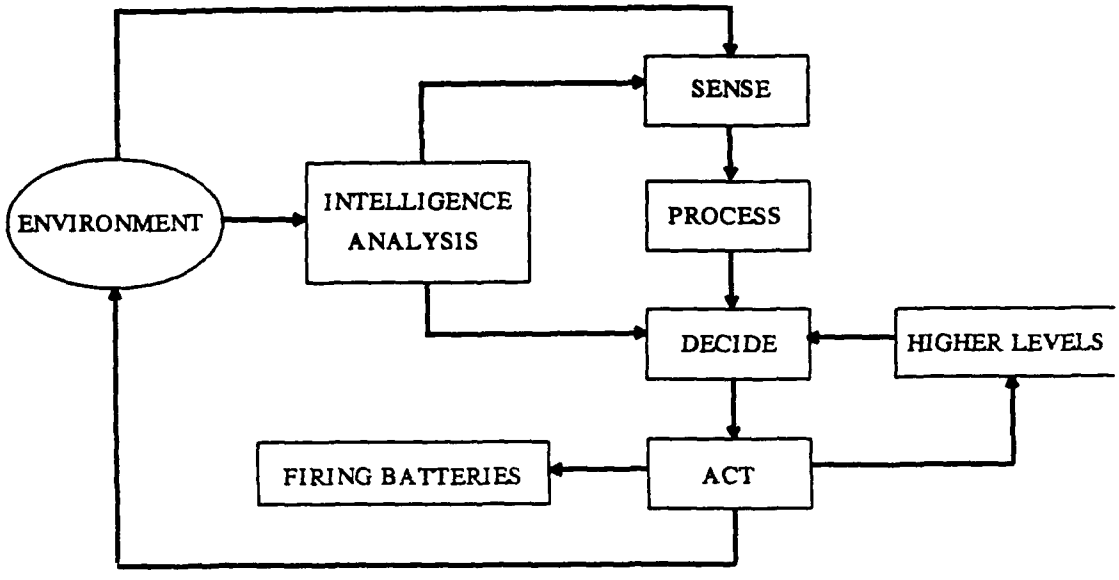

Figure 2.5 Conceptual Combat Operations Process Model

This model gives structure to the wargame by providing a logical order to the functions which must be performed both by the computer program and the human interaction portions of the game.

H. SUMMARY

In order to build a wargame that trains officers in the integration of tactics and logistics, an analysis of the field artillery battalion is necessary. Use of the systems approach provides a structure for this analysis so that all of the important aspects of the battalion can be considered. The understanding of the field artillery battalion which was gained through this analysis is critical to the design of the wargame. The next chapter will discuss the approach taken in the design of this wargame.

III. THE CONCEPTUAL APPROACH

A. INTRODUCTION

This section is not intended to provide detailed, technical design criteria for the wargame. Rather, it is intended to provide the approach taken to resolve many of the issues which arose while conceptualizing this wargame.

It is important to differentiate between the field artillery battalion system and the wargame. The wargame can be considered a system unto itself. Therefore, the best way to look at the conceptual approach is by using a system view of the wargame. In many cases, the best approach to take for certain system aspects of the wargame is to simply simulate the same system aspects from the field artillery battalion. In other cases, there are some important differences. Therefore, specific references will be made to the wargame system and to the battalion system in order to avoid confusion.

As with the battalion, a discussion of the mission, goals, environment, resources, and management follows. These aspects, along with a discussion of some of the architectural depictions, will provide the vehicle for a complete discussion of the approach taken for the design of this wargame.

B. MISSION OF THE WARGAME

The mission of this wargame is to train officers in the integration of tactics and logistics as related to the command and control of a field artillery battalion. There are several implied tasks which are necessary for the wargame to accomplish this mission. The wargame must be oriented toward the command and control elements of the battalion. Therefore, the players will consist of an operations and intelligence (O&I) staff section, a logistics (S4) staff section, and a battalion commander. It must require the players to weigh all of the tradeoffs involved in integration of tactics and logistics. Special emphasis must be given to those areas which are not normally experienced in a peacetime training environment. Since the mission is to train officers, the wargame must provide feedback to the officers so that good decisions are reinforced and poor decisions are penalized. This can be done through the use of a measure of effectiveness which evaluates the player's performance. Finally, the wargame must be able to provide a realistic representation of the field artillery battalion system. This means that the field artillery mission should be central to the wargame's design. Each of the artillery's ten implied tasks must be performed by the wargame to varying degrees.

1. Target Acquisition

This function will not be explicitly played in the first iteration of the wargame. Targets will be provided to the battalion TOC. All other firing will be done at some predetermined rate depending on the tactical situation.

2. Meteorology, Survey, Technical Fire Control

These functions also will not be explicitly played as a part of this wargame. This is due to the fact that these functions are not germane to the wargame's mission of training officers in the integration of tactics and logistics. These functions relate to the computation of actual firing data.

3. Fire Planning

This function must be performed as a part of the wargame at a tactical level since it occupies a significant portion of the operations and intelligence section's time. More importantly, it must be done with the logistics situation in mind. It requires an analysis of the CSR in order to avoid firing too many rounds.

4. Tactical Fire Control

Some battalion level fire missions will be useful for causing the O&I section to determine the number of rounds to fire on a given target and the number of fire units. However, the level of resolution need not include the various shells and fuzes in the wargame's first version. Once again, this will cause the O&I section to weigh the

desire for effects on target against the logistics situation.

5. Plans and Orders

These functions represent the essence of the command and control / battle management portion of the wargame. They will be performed by the players as they would actually perform them in a real wartime environment. The game will begin with a hard copy of the supported maneuver unit's operations order. The O&I section and the logistics section will then begin planning, develop an order, and then, using the computer, send the order to the units to execute. The computer program portion of the wargame is responsible for the execution of all orders.

6. Positioning

This function is also extremely important to the design of the wargame. All positions must be coordinated with the maneuver unit that owns the land. Positions must also be coordinated with the resupply plan.

7. Reconnaissance

Reconnaissance must be performed within the wargame so that realistic plans can be made. At the very least, map reconnaissance can be performed. Ultimately, a computer run network using Dykstra algorithms can be used to simulate route reconnaissance to determine the shortest or quickest routes.

8. Displacement

This function is central to the integration of tactics and logistics. Displacement routes must be well planned and coordinated with the resupply plan. The movement times must be minimized in order to maximize availability time. Routes which are under air defense umbrellas may be longer but they reduce casualties. Displacements must be timed so as to avoid falling too far behind maneuver forces or to avoid getting overrun by the enemy. Additionally, displacements should be timed to minimize the amount of time a unit spends above the commander's vulnerability threshold. All of these tradeoffs should exist within the wargame as well as the actual movement of the units along the paths and nodes of the previously mentioned network.

9. Communications

Some sense of communicating orders and receiving reports must be contained within the game. The computer assisted portion of the game will provide this service. The players will be able to establish certain standard operating procedures (SOP) to control such things as the frequency of standard reports like the unit situation reports.

10. Combat Service Support (CSS)

This function is obviously critical to accomplishment of the wargame's mission. In order to give proper emphasis to the importance of this function, each of

the ammunition resupply trucks will be explicitly played. Of the various types of supplies, only ammunition will be played in the first iteration of the wargame. This is because the other types of supplies are normally played through actual consumption even in peacetime exercises. It is ammunition which has the most bulk and the largest impact on tactics.

This analysis of the wargame's mission, along with its implied tasks, provides a discussion of the wargame's intended purpose, its scope, and a brief description of the functions it must be able to perform.

C. GOALS OF THE WARGAME

The goals of the wargame are defined on two levels. The first concerns the goals of the wargame's design and the second concerns the goals of the players of the wargame.

The goals of the wargame's design should simply be to design the wargame so that each of the battalion's goals are not only played, but also their degree of accomplishment is measured. Specifically, availability, vulnerability, casualties, and logistics optimization should all be played in quantifiable terms so that they can be measured by the computer portion of the wargame. Once they are measured, a measure of effectiveness (MOE) can be computed. This MOE provides feedback to the player so that they can reassess

their decisions and perhaps improve their MOE the next time the wargame is played.

The goals of the players should therefore be the same as for actual operations; to maximize fire support provided to maneuver units by maximizing availability time, to minimize the amount of time spent above the commander's vulnerability threshold, to minimize the time lost to casualties, and to minimize the amount of time units spend either critically short or out of ammunition.

These parallel goals (between the wargame and the players) serve as the thread of continuity which ties the wargame together. The wargame should be designed to accomplish the wargame's system goals and all components of the system should contribute to the furtherance of the goals within the intent given by the wargame's mission. In this way, the players will be able to use the same decision criteria in playing the wargame as they would in wartime. Their objective is to maximize the measure of effectiveness (MOE) by seeking to accomplish the player's goals which are the same goals which are found in the field artillery battalion system.

The actual MOE used for this wargame, named the Field Artillery Battalion Command and Control (FABCAC) Effectiveness Index by the authors, is directly related to each one of the goals. The MOE is as follows:

MOE = "tube hours available" ratio - "truck hours lost to casualties" ratio - "tube hours above the vulnerability threshold" ratio - (.5 * "tube hours critically short of ammunition" ratio).

Each of the ratios in the MOE is in terms of actual hours over maximum possible hours. Tube hours lost due to casualties are not explicitly listed in the MOE because they are accounted for under availability time. Tube hours lost due to ammunition outage are also accounted for under availability time. By accounting for ammunition outage under availability time, the players are afforded the opportunity to minimize the effect of an ammunition zero balance by moving a unit since it cannot shoot anyway. In this way, the players can get the unit into a better position tactically and set its vulnerability level to zero while out of ammunition rather than during a period in which the unit could be shooting.

D. WARGAME ENVIRONMENT

The wargame simulates many of the more important aspects of the field artillery battalion system environment. First and foremost is the fact that the battalion exists in a hostile environment. This is reflected in the wargame by inducing attrition of howitzers and ammunition trucks using Lanchester-type equations. Attrition takes place both in positions and along displacement routes. Attrition rates are based upon force postures and the tactical situation.

Command and support relationships should be understood by the players. The wargame is consistent with actual doctrine in this respect.

The facts that all of the units exist on the battlefield, are separated by considerable distances, are in positions which provide varying degrees of cover and concealment, and must move around the battlefield are all important to the design of the wargame. It is anticipated that the computer generated network will allow these aspects of the environment to be simulated. Many of the nodes will represent positions which afford varying degrees of cover and concealment. The paths indicate the distances between nodes and are used to control the movement of units between nodes so that realistic travel times are generated based upon path capacities. Additionally, paths have attrition characteristics which will determine the amount of attrition incurred by a unit while on that path.

Finally, "friction" is an important element to portray in any military wargame. Units will not always move through a route exactly as scheduled. Enemy artillery attack will cause unscheduled moves. These kinds of problems are also found in the wargame in order to cause the players to be flexible and issue fragmentary orders to rectify the situation.

The approach for the actual physical environment of the wargame involves a computer program which is written for

execution on a personal computer. It is important to restrict the wargame to execution on a PC because that is all that most battalions have readily accessible. This wargame can be played by the O&I section, the logistics section, and the battalion commander in the battalion headquarters or the PC can be set up in the battalion tactical operations center (TOC) in the motor pool. In either case, situation maps and status boards will be necessary not only to keep up with the battle, but also to help evaluate the effectiveness of the section's SOPs and information management procedures.

E. WARGAME RESOURCES

Other than the players themselves, personnel are not explicitly played as a part of this wargame. It is assumed that when equipment is lost, its personnel are also lost and vice versa.

The only equipment represented in the wargame are howitzers and ammunition trucks. These are the primary high density items of equipment found in a battalion. They are also the items of equipment which most significantly impact on the accomplishment of the wargame's mission.

The only supply represented in this version of the wargame is ammunition. Ammunition, by far, represents the most bulk in resupply operations and is directly involved in the integration of tactics and logistics.

In order to effectively run the wargame, the following resources will be necessary:

1. Operations players, logistics players, and commander.
2. One, IBM compatible, personal computer.
3. Maneuver unit operations order, overlays, and maps.
4. O&I section and logistics section status boards and map boards.
5. Wargame computer software and user's manual.

F. MANAGEMENT OF THE WARGAME

All of the management as defined for the field artillery battalion system will be handled within the human interaction portion of the wargame. The commander will provide direction and leadership to the staff and he will make final decisions. The staff (O&I and logistics) will prepare estimates and plans. They will also assist the commander in monitoring operations by interpreting reports and preparing and communicating orders and by assessing the situation to determine if the plan needs to be modified.

G. ARCHITECTURAL DEPICTIONS

Once again, an effort has been made to separate the architectural depictions into four categories which represent functional, spatial, informational and time relationships. The architectural depictions have been separated from the previous discussion because most of the depictions transcend any one aspect of a system and

therefore serve to tie together the previously mentioned ideas. Additionally, they reveal new aspects of the system which have not already been discussed.

1. Functional

"Functional architecture describes the technical structure of large systems." [ref. 7: p. 2.4] Figure 3.1 represents a decision taxonomy known as the "SHOR" paradigm [ref. 2: p. 626]. Within the context of this thesis, the "SHOR" paradigm is used to indicate, at a conceptual level, the boundary between the human portion of the wargame and the computer portion of the wargame. What is inside the dotted line is the human portion of the game and what is outside the dotted line is handled by the computer. An interpretation of the depiction indicates that the computer will provide some kind of trigger event which will provide the stimulus or data to the players (S). This will cause the players to attempt to interpret the data to determine what it means in relation to their mission accomplishment (H). They will then create alternative courses of action to counter the perceived impact of the data (O). Finally, they will take action by issuing an order (R). This order is executed by the computer and has some effect on the environment. The players monitor the environment and collect raw or preprocessed data and the cycle repeats itself. At any given time, multiple stimuli can be received by the players which can result in one or more responses.

THE SHOR MODEL

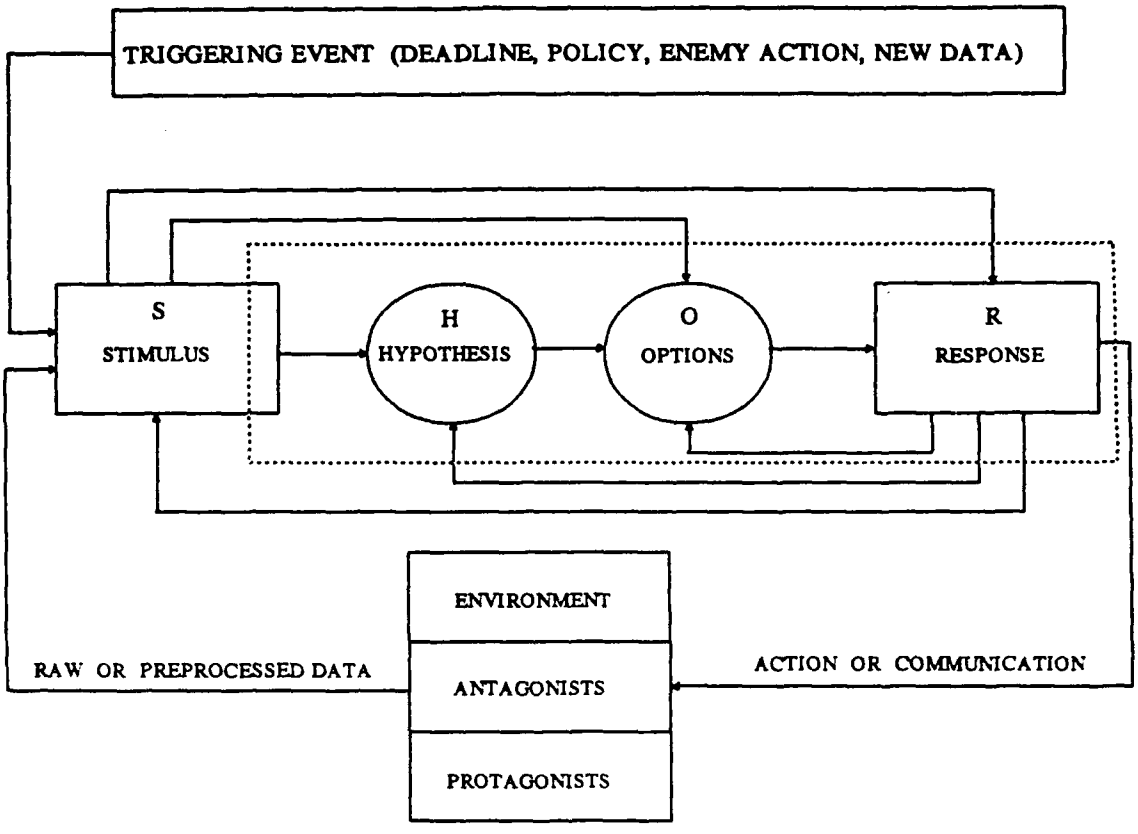

Figure 3.1 The SHOR Model

The entire process takes place continuously for the duration of the game.

Figure 3.2 is the computer architecture which will be used for the wargame. It has been divided into three major modules: the Pre-Processor, the Game, and the Post Processor. Each of these major modules has been divided into the lesser modules representing functions which are required to be performed to support the major module. Some of the more significant points which can be derived from this depiction are that the players will be able to input their own scenario if they do not want to use the one that will come with the game, the players will be able to input some of their own parameters which will represent their unit SOPs and commander's guidance, and finally, the players will be able to save a game which is in progress and then get back to it later.

2. Spatial

Spatial architecture depicts the physical relationship of the elements in the system. Figure 3.3 depicts a basic level configuration for game's set-up requirements. Figure 3.4 depicts the spatial relationship of the elements of the field artillery battalion within the wargame. In this figure, it is apparent that all of the elements are spatially related to each other through the use of the network architecture. Units are located at nodes and

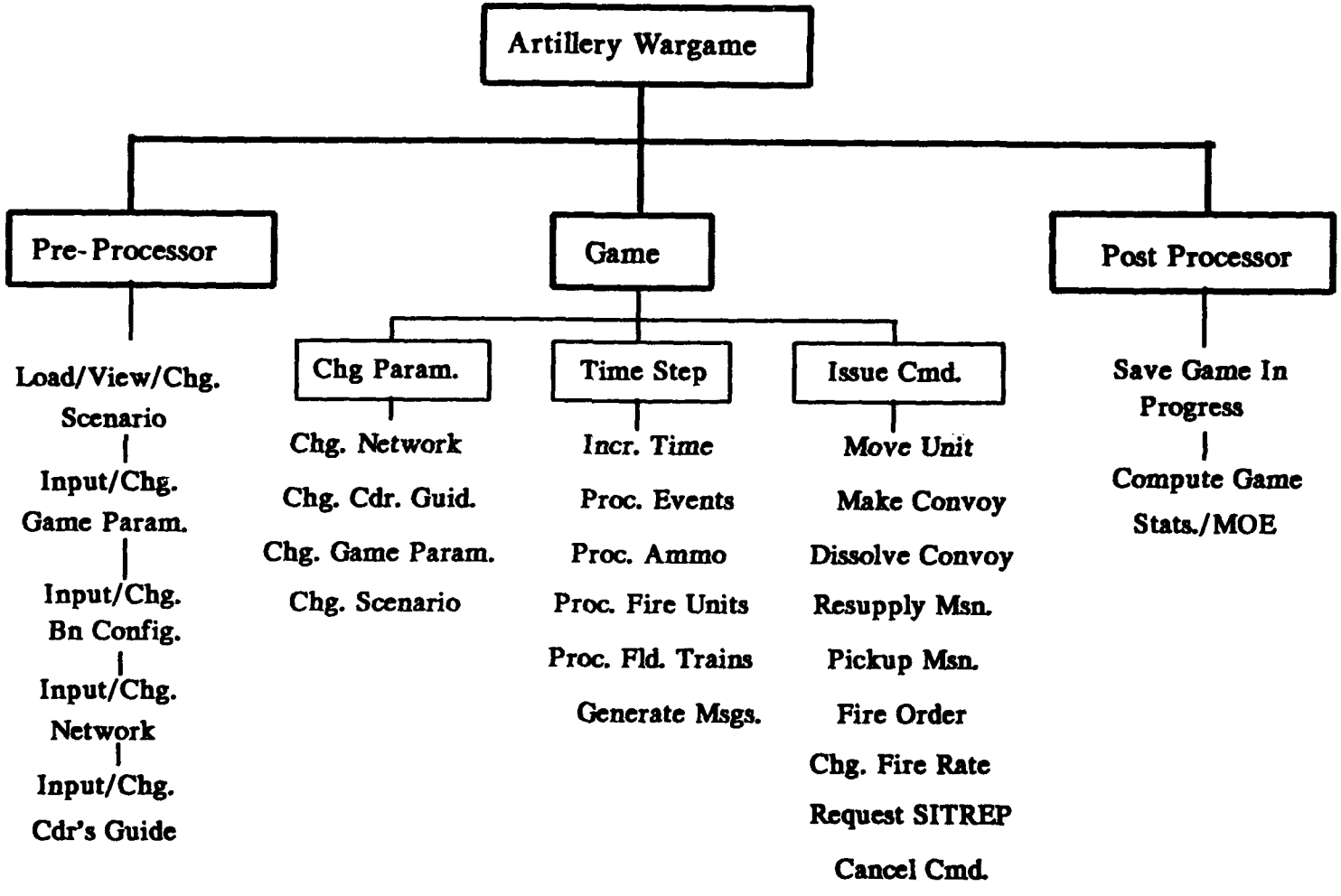

Figure 3.2 Computer Architecture

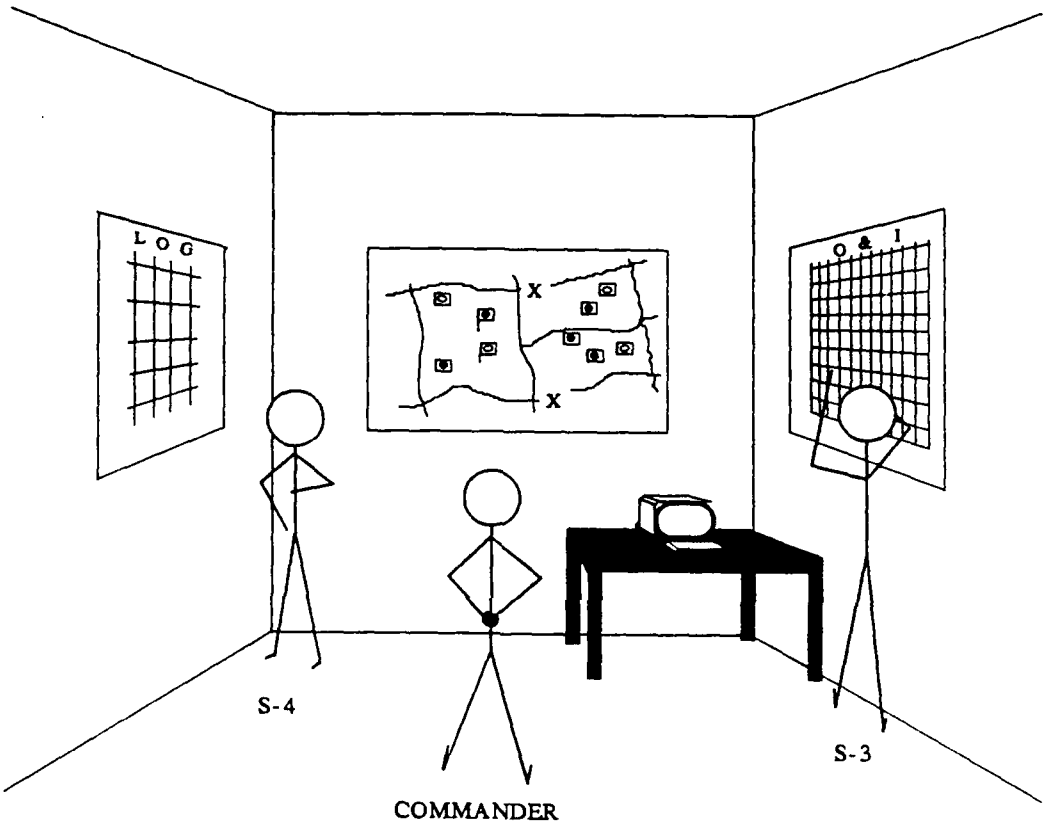

Figure 3.3 Wargame Setup Requirements

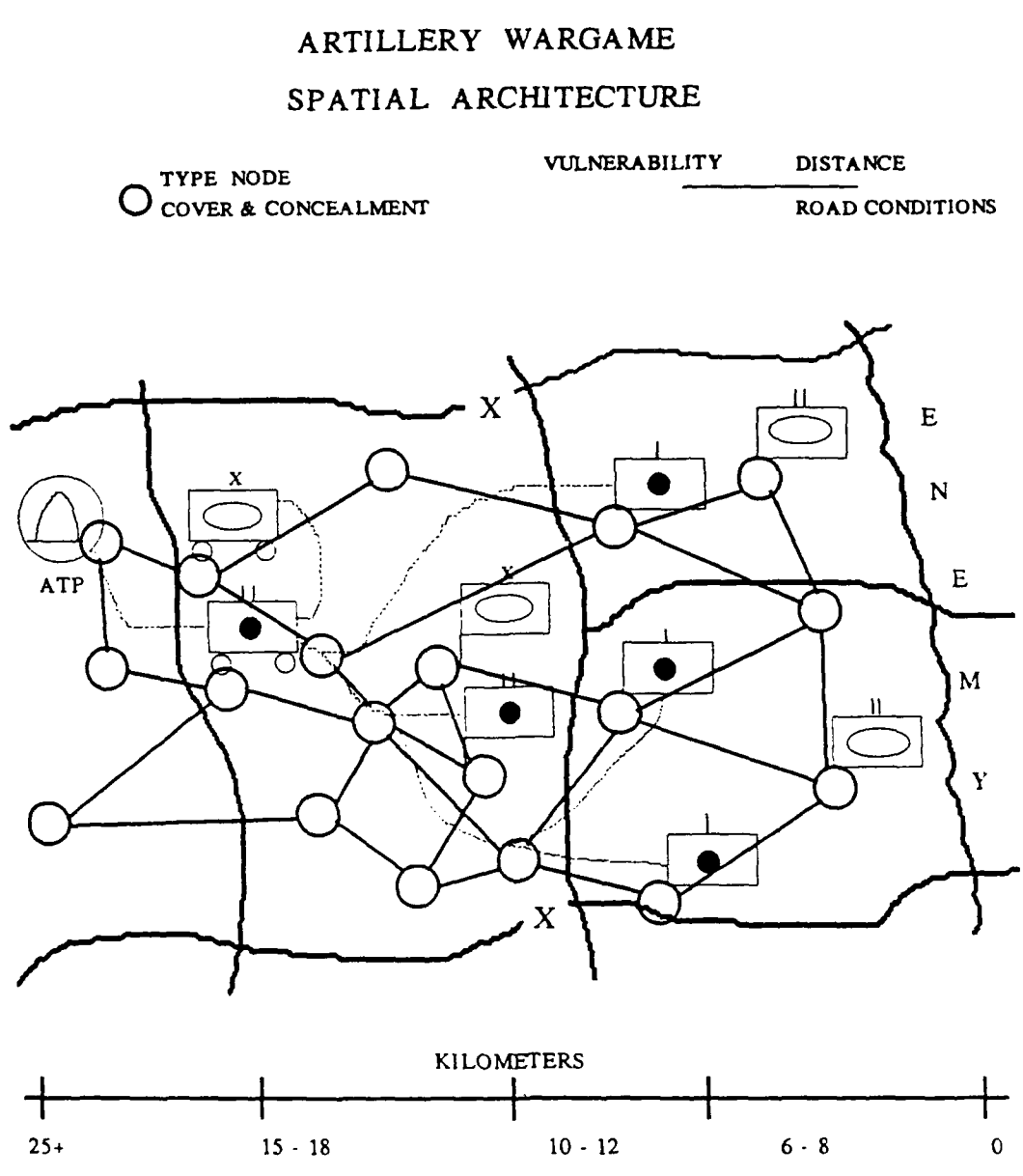

Figure 3.4 Artillery Wargame Spacial Architecture

displace along paths. The legend contains a listing of the various characteristics of nodes and paths.

3. Informational

Informational architecture depicts the flow of information within the system. Figure 3.5 is very similar to Figure 2.4 except that only those elements of information which will be used in the wargame have been listed on the diagram. Additionally, in Figure 3.5, the dotted line separates the game's players from the computer portion of the wargame.

4. Time

The time based architectural depiction for the wargame is the same as for the field artillery battalion system. In both cases, the process that the key people must use to make decisions remains the same. Therefore, Figure 2.5 applies equally well to the wargame.

H. SUMMARY

The basic conceptual approach taken for this wargame has been to analyze a field artillery battalion from a systems point of view. Then, using the knowledge gained from this analysis, lay out the conceptual framework for a wargame which emulates the pertinent aspects of the field artillery battalion. Once again, the systems approach is very useful in laying out this framework because it requires that every

ARTILLERY WARGAME

INFORMATION ARCHITECTURE

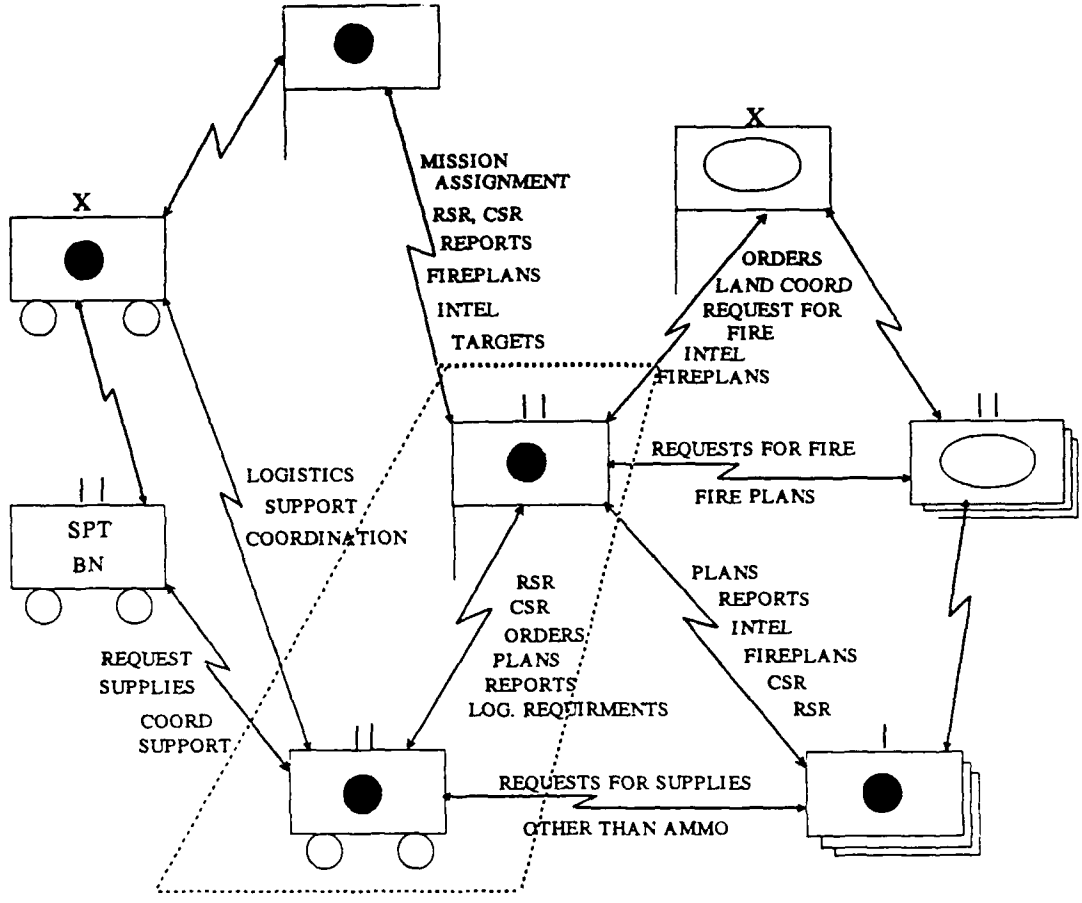

Figure 3.5 Artillery Wargame Information Architecture

facet of the system be considered under the categories of mission, goals, environment, resources, and management.

The wargame attempts to capture this framework in a form which provides a learning environment for those tasked with the command and control of a field artillery battalion. The wargame is only computer assisted since the essential element of any tactical command and control system is the human decision maker. The computer portion of the wargame only provides relevant information to the decision maker and then executes his decisions and keeps track of the game's statistics. At the end of the game, an MOE is calculated which provides an assessment of the player's performance. This forces the player to plan ahead and consider all the relevant concepts in order to maximize howitzer availability time, minimize casualties, minimize vulnerability time and optimize ammunition resupply. The intent of the wargame is to provide an environment in which officers can learn to consider the factors which are critical to the integration of tactics and logistics.

The computer program itself is complex. It utilizes a network architecture to facilitate the movement of firing batteries and ammunition convoys and it uses Lanchester attrition equations to decrement forces. Among other things, the program simulates firing, consumption of ammunition, movement of units, ammunition resupply, and changes in the tactical situation. Periodic reports as well

as emergency messages and warning reports are output to the decision maker. The decision maker has the ability to issue commands to the units. Among the commands are movement orders, firing rate orders, ammunition resupply orders and requests for situation reports from the units.

Finally, all this must be done in a format which is usable at the battalion level. This means that it will be executable on a personal computer, it will be well documented, and it will enable the players to use their normal unit / section SOPs.

IV. WARGAME DESIGN

A. INTRODUCTION

The purpose of this Chapter is to supplement the documented source code which can be found in the Appendix. A description of the code's structure and organization can be found in the documented source code. In this Chapter, a description of the overall design used in the coding of the computer portion of the wargame will be given.

Turbo Pascal version 5.0 and Turbo Pascal Database Toolbox version 4.0 were used to code the wargame. Turbo Pascal was used is because Turbo Pascal is one of the most popular programming languages on the market. By using Turbo Pascal, there is a reasonable expectation that users will be able to modify their own source code.

In order to implement the wargame as described in Chapter III, a "Time Step - Event Driven Hybrid" design was used. An "Event Driven" design was needed to enable the players to issue orders for future operations. A "Time Step" design was used to perform the routine functions which constantly occur over time in an artillery battalion. The hybrid design will become more apparent later in this Chapter.

Finally, a great deal of the design considerations used for the wargame revolve around the desire to make the game as "user friendly" as possible. In order to achieve this

objective, no computer commands are used in the wargame. Everything is menu driven. All data entry fields are protected so that only the correct type of data can be entered. The terminology used by the game is as close to actual military terminology as possible.

B. OVERALL WARGAME DESIGN

The wargame's overall design can be described by referring to Figure 4.1. Note that upon loading the game there are three choices: "Play New Game", "Play Old Game", or "Quit." If "Quit" is chosen, the player is returned immediately to MS-DOS. If "Play Old Game" is chosen, the old game's files are loaded and the program goes straight to the Game Play Module. Finally, if "Play New Game" is chosen, the Pre-Processor Module is entered. The Pre-Processor is the portion of the computer program which allows the player to set up or initialize the game with such data as the battalion's configuration, the unit movement network, the commander's guidance, the game's scenario and the game's parameters. When the player is finished with the Pre-Processor Module, he enters the Game Play Module.

From within the Game Play Module, the entire game is played. The design of the game itself will be discussed later in this Chapter. From the Game Play Module, the player can quit the game at any time. Upon deciding to quit, the player has two options: either to quit with the

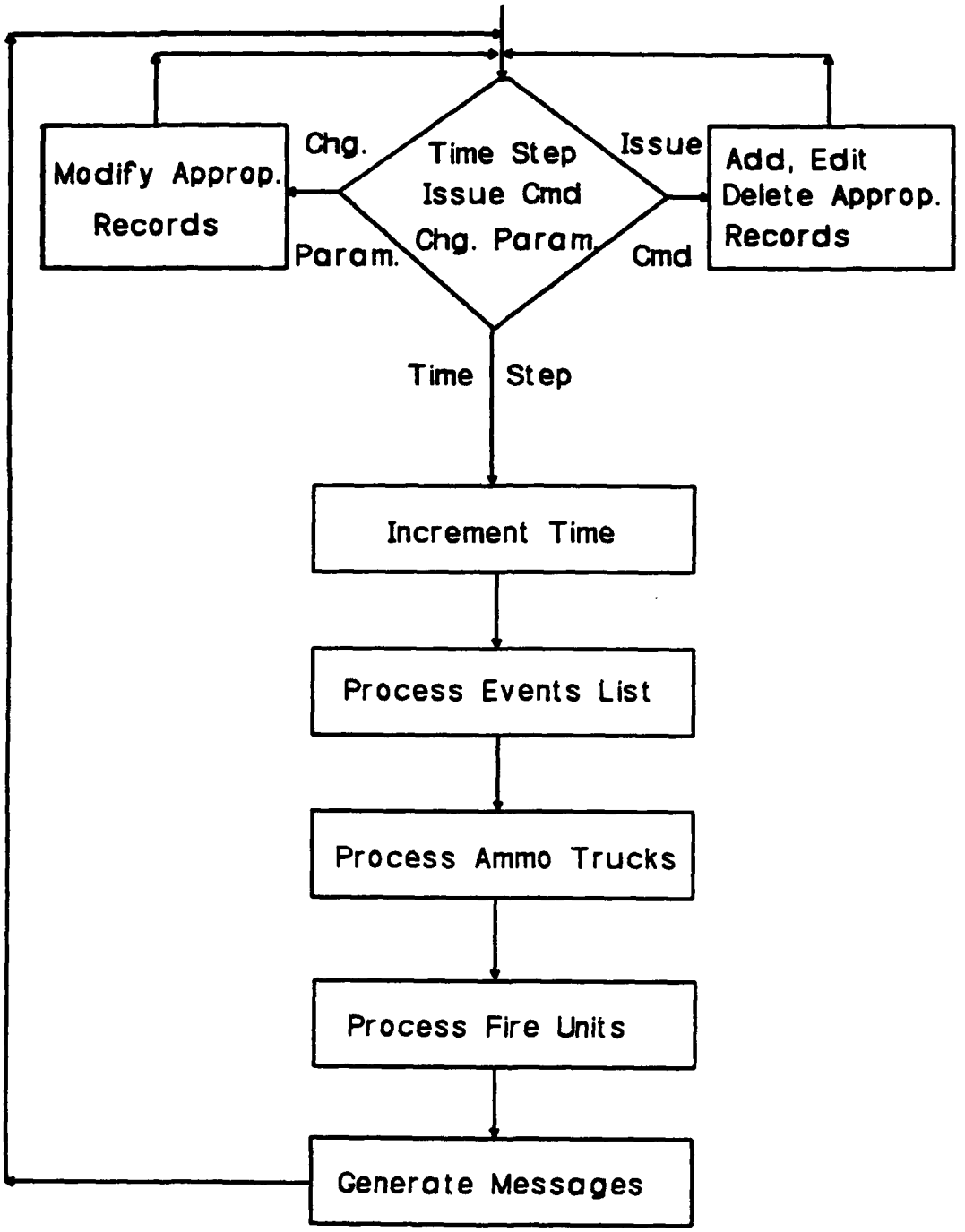

Figure 4.2 Game Flow Chart

intent to continue playing the game later or to permanently quit the game. When "Quit and Return" is selected, the game's files are saved to disk. In both cases, the program then enters the Post-Processor Module.

In the Post-Processor Module, the game's statistics are tabulated and the Field Artillery Battalion Command and Control Effectiveness Index is computed. After this information is printed, the program returns the player to MS-DOS.

C. RECORDS

The basic data structure used in the wargame is the record. The following records are used by this version of the wargame.

1. Scenario Record
2. Commander's Guidance Record
3. Game Parameters Record
4. Field Trains Record
5. Fire Unit Record
6. Ammunition Truck Record
7. Events List Record
8. Node Record
9. Path Record

The fields contained in each record can be found in the global declaration section of the source code. The first four types of records are all single records. The fire unit records and the ammunition truck records are maintained in

an array of records. Finally, the last three types of records are all stored in their own B+ lists in disk files created by the Turbo Pascal Toolbox. The events list records are stored in increasing order according to the event's date time group. The nodes and paths are each stored in their respective lists according to the node and path name.

The reason that the fire unit records and the ammunition truck records are stored in arrays is because there are only a finite number of those records and because every record must be accessed every time step. On the other hand, the event list, node, and path records are stored in database files because there are potentially an unlimited number of these records and because only selected records will be accessed in any given time step.

D. GAME PLAY DESIGN

The Game Play Module was designed in accordance with the flow chart depicted in Figure 4.2. Note that in this top level flow chart for the Game Play Module there is only one decision point. This was deliberately designed in order to focus all aspects of the game at one point. This point is referred to throughout the rest of the thesis as the "hub of the wargame." From this hub there are three possibilities: the Time Step Module, the Issue Commands Module, and the Change Game Initialization Module.

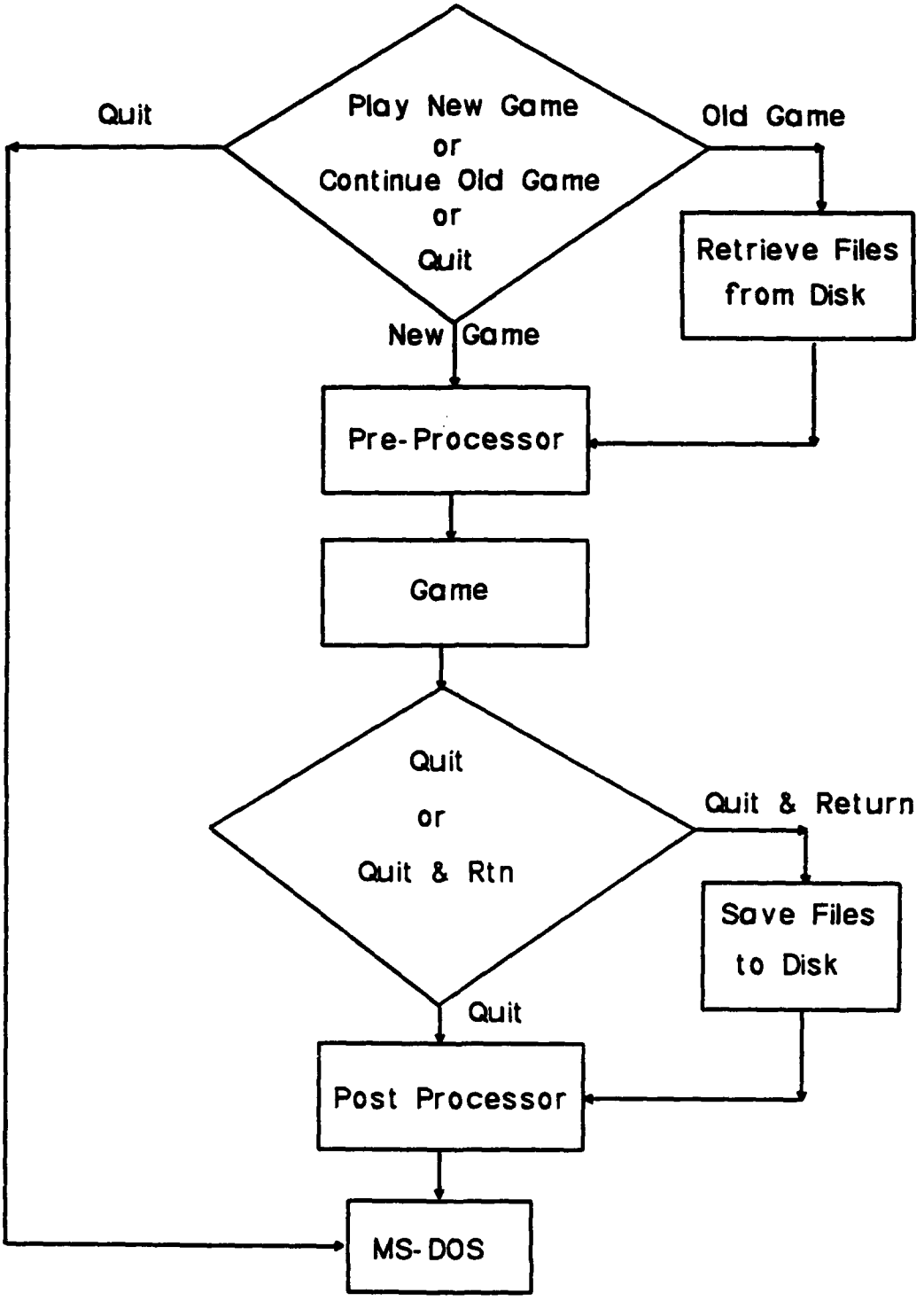

Figure 4.1 First Order Flow Chart

If the "Change Game Initialization" option is chosen, the program takes the player to many of the same procedures that were used in the Pre-Processor Module. The player is given the opportunity to change commander's guidance, add nodes and paths, and change some of the game's parameters.

If the "Issue Commands" Module is chosen, the player is given the option of issuing one of several commands. The list of possible commands follows:

1. Convoy Ammunition Trucks
2. Dissolve Ammunition Truck Convoy
3. Move a Unit
4. Request a Unit Situation Report
5. Change a Unit's Firing Rate
6. Assign Ammunition Resupply Mission
7. Assign Ammunition Pickup Mission
8. Issue a Fire Order
9. Cancel a Command

In most cases, the result of a command is to add or delete one or more events list records. In some cases, it involves direct modification of ammunition truck records or fire unit records.

If the time step is chosen, the program will increment the game time and then search the events list to retrieve any records with an execute time between the last time step's game time and the current game time. As each record is retrieved, it is executed based upon the record's key

field. The key field contains the "type action" which took place. This "type action" is what determines which procedure is called. After all applicable events are executed, the program enters the Field Trains Processing Module, followed by the Ammunition Truck Processing Module, then the Fire Unit Processing Module, and finally the Generate Messages Module.

In the Field Trains Processing Module the vulnerability level is determined, then the program checks to see if the unit receives an enemy artillery attack, and finally, casualties are computed. In each of the different type of unit modules, the vulnerability level is determined and then casualties are assessed based on the linear law form of the Lanchester attrition equations. More will be said about the assessment of casualties later is this Chapter.

Additional functions which take place in the Fire Unit Processing Module include the firing of ammunition by each fire unit, the determination of each unit's ammunition count / status, and the update of such fire unit statistics as availability time and ammunition critically low time.

In the Ammunition Truck Module, the program conducts the actual resupply operations, checks for sufficient crew rest to carry on with the mission, assesses casualties, and increments the truck's statistic. The only statistic which is tracked is the truck's casualty time.

The last module of the time step is the Generate Messages Module. There are two general categories of messages, those which are sent to the players by the battalion's subordinate units and those which are sent by other elements. The former category of messages is generated by checking for various flags in each subordinate unit's records. The later type of message is generated at random times throughout game. Examples of this type of random message include such messages as the "Bridge Destroyed" message and the "Road Mined" message.

E. PROGRAM ILLUSTRATION

In order to illustrate how all the different modules work together, an ammunition resupply operation will be described. It is important to note that an operation of this type may be only one of many operations taking place concurrently in the game.

The operation which will be used for the illustration involves ordering a convoy of five trucks to move from point "A" to point "B" to deliver their loads of ammunition. Multiple commands must be used to issue this order. First, if not already organized into a convoy, the "Convoy Ammunition Trucks" command must be issued. In order for this command to be executed, all trucks must be in the same location. Next, the "Assign Ammunition Resupply Mission" command must be issued. This command modifies all the truck records in the convoy to reflect the fact that a mission has

been received, and to reflect the fire unit to be resupplied and the node in which the resupply is to take place. All trucks in the convoy must have full loads of ammunition on board in order for this command to be executed. Finally, a "Move Unit" command must be issued. In this command, either a depart or a occupy time is specified and a route is specified in terms of nodes and paths.

When the "Move Unit" command is entered, a procedure is called which builds an itinerary for the unit. It takes either the depart time or the occupy time and works either forward or backward as appropriate to determine the itinerary. The time required to traverse each path is determined based upon the length of the path, whether it is day or night, the road conditions, and the type of convoy.

Using this information, event list records are created for the time that the unit departs its current location, the time that a unit moves through each node in the route, and the time that the unit occupies its new location. During each subsequent time step, the events list is checked. For each event relating to this move, the convoy's location is changed as well as other fields in the record. Also during each time step, the trucks in the convoy have casualties assessed based upon the vulnerability level of the path they are traversing and whether it is day or night.

Once the convoy arrives at its destination, each truck in the convoy is taken out of a moving status when the

occupation event is processed. In all subsequent time steps, until the resupply operation takes place, the program will check to determine if both the convoy and the unit to be resupplied are in the same location. If they are, the resupply operation will take place. After the resupply operation has taken place, the fire unit's ammunition count is incremented and each truck in the convoy's ammunition count is decremented. At the end of the time step a message is generated from the convoy to the battalion S4 notifying him that the mission has been accomplished and requesting further orders.

F. CASUALTIES

For the purposes of this wargame, it has been assumed that the artillery units would not receive any direct fire from enemy units. The only types of enemy fires in this wargame are from enemy air attack and from enemy artillery attack. Both of these types of attacks are produced by area fire weapons. Therefore, the Lanchester linear law differential model was used to assess casualties. Specifically, the following form was used:

$$\frac{dx}{dt} = - axy$$

Where "a" is the attrition coefficient, "x" is the number of friendly elements being attrited, and "y" is the number of firing elements.

For this wargame casualties are calculated based on three different possible unit postures: unit in a position, unit in a position and receiving a deliberate artillery attack, and unit moving. For each posture there are four different attrition coefficients: friendly cannons attacked by enemy artillery, friendly cannons attacked by enemy bombs, friendly trucks attacked by enemy artillery, and friendly trucks attacked by enemy bombs. The underlying unit of time for these coefficients is one minute.

The number of friendly elements being attrited, "x", is the percent of each truck which is remaining for ammunition trucks. For firing units it is the number of firing sections in operating condition.

The number of enemy firing elements, "y", is considered in two parts: enemy air bombs and enemy artillery rounds. This variable can be considered to be related to the intensity of the attack. For this wargame, the intensity of the attack varies according to several factors. Some of these factors are listed below:

1. Day or night
2. Rural or urban position
3. High Medium or low cover and concealment
4. Acceptable, high, or critical vulnerability level
5. Maneuver mission offense or defense

For each type of enemy weapon, air bomb or artillery round, the number of attacking elements is determined by summing

the number of bombs or rounds contributed by each factor. For example, if a fire unit is in an urban position, the time is later than end-evening-nautical-twilight (EENT), the position has good cover and concealment, the unit's vulnerability level is acceptable, and the supported maneuver unit's mission is offense then the value of "y" is determined by summing the number of bombs or the number of artillery rounds for those particular factors which are applicable to the unit's situation.

Next, the casualties assessed against the friendly truck or fire unit due to each type of enemy weapons system, air bomb or artillery round, are calculated and then summed. Once the casualties for one minute are determined, they are multiplied by the number of minutes in a time step and then assessed by subtracting the determined amount of casualties from the truck's effective percent figure or from the number of firing sections in operating condition for fire units.

By computing casualties in this manner, one procedure is used to determine casualties for any given situation by passing the appropriate parameters in the procedure call. The parameters are the type unit, the unit's posture, and factors which are applicable to the situation at hand.

V. PLAYER'S MANUAL

A. INTRODUCTION

This chapter is intended to provide a stand alone player's manual for the play of the wargame. For information concerning the technical aspects of the code, refer to Chapter IV or to the documented source code itself. In this chapter, the mechanics of actual game play will be discussed.

The wargame can be thought of as consisting of two parts, the computer assisted part and the manual part. In the manual portion, players should attempt to operate as closely as possible to the way they normally operate during combat or simulated combat conditions. The computer assisted portion of the game involves a computer program which provides the dynamic information which the players use to make command and control decisions. The computer assisted portion also keeps track of statistics and calculates a measure of effectiveness which can be used to assess player performance.

The wargame can be played at any level of intensity. That is, the players can play the manual portion of the game at a full scale with written orders, staff estimates, and detailed planning or they can play it relatively casually by just using "judgement" or "best guesses" to make decisions.

In either case, the computer assisted portion of the game remains the same, it simply takes the human input and processes it. The point is that the game can be played as a part of a full battalion CPX or during weekly officer professional development time. This is possible because the wargame is designed to be played at the player's pace. The pace is determined by the player by requiring him to physically initiate each time step.

In order to make the wargame easy to learn and remember, the wargame was designed to be completely menu driven. There are no commands to remember.

B. MEASURE OF EFFECTIVENESS (MOE)

It is important to discuss the wargame's measure of effectiveness up front in the player's manual because it tells a lot about what the game's objectives are and what the player's objectives should be. While playing this game, the player should always keep in mind the fact that the mission of the wargame is to train officers in the integration of tactics and logistics as it relates to the command and control of a field artillery battalion. As such, the game is oriented toward the integration of tactics and logistics and may not do justice to other important aspects of a field artillery battalion.

The player's objectives should be to play the game by taking the same actions they would take in a real situation while trying to maximize the game's MOE. The MOE is

calculated based upon statistics which are compiled throughout game play. The intent of the MOE is to reinforce favorable statistics and to penalize unfavorable statistics. In this way, players can judge how effective they were at the command and control of the field artillery battalion.

The statistics which are tracked in the wargame mirror the four goals of a field artillery battalion as discussed in Chapter II. The actual MOE, called the Field Artillery Battalion Command and Control Effectiveness Index, is directly related to each one of the goals. The MOE is as follows:

> MOE = "tube hours available" ratio - "truck hours lost to casualties" ratio - "tube hours above the vulnerability threshold" ratio - (.5 * "tube hours critically short of ammunition" ratio).

Tube hours lost due to casualties are not explicitly listed in the MOE because they are accounted for under availability time. Tube hours lost due to ammunition outage are also accounted for under availability time. By accounting for ammunition outage under availability time, the players are afforded the opportunity to minimize the effect of an ammunition zero balance by moving a unit since it can't shoot anyway. In this way, the players can get the unit into a bette.- position tactically and set its vulnerability level back to zero while out of ammunition rather than during a period in which the unit could be shooting.

C. GETTING STARTED

To start the computer assisted portion of the wargame, the players must first ensure the printer is turned on and is on-line. At the "A:" prompt, type WARGAME and press "return." This will take you to the Main Menu. Throughout the wargame, the "arrow" keys are used to move the cursor to the various choices, the "return" key is used to make a selection and the "F1" key is used to get help. Any data fields which require a choice to be made between such entries as "high", "medium", or "low" can be entered by just typing the first letter of the word, e.g., "h", "m" , or "l". At the main menu the player has three options; start a new game from the beginning, continue an old game from the point where the game was previously quit, or simply to quit the game and return to the MS-DOS "A:" prompt.

Selecting "Continue an Old Game" will take the player directly to the game play screen and will pick up with the one and only old game which is on file on that particular disk. Selecting "Start a New Game" will require the player to initialize the game. This can be done by loading a scenario file. The "Enter Scenario File Name" screen allows the player to do this. Refer to Figure 5.1 at the end of the Chapter. When the screen first is displayed, the cursor is after the "scn" on the directory line. The player has two options. He can hit "return" to get a directory listing of all the scenario files on the disk or he can press the

"escape" key to build a new scenario file. If the "escape" key is used, the program will require the player to enter a scenario name. A valid scenario file name is "*.SCN." If the player chooses the directory listing, then he must place the cursor over the file name desired and press "return." Either case will lead to the Game Setup Menu.

D. GAME SETUP MENU

The purpose of this menu is to allow the players to start the game or view, change, or print the game scenario / network, the game parameters, the commanders guidance, and the battalion configuration / status. All but the "Play Game" option will be discussed in this section. The "Play Game" option will be discussed in Section E of this chapter.

It is strongly recommended that players print all of the game initialization data before beginning the game. This serves two purposes. First, it forces the player to review the data to ensure it is correct. Secondly, it gives the players a hard copy of the data that can be used throughout the game as a reference. All printing is done by pressing the "F4" key. This prints the currently displayed screen.

Throughout the view and change options, use the arrow keys to move between fields, use "page up" and "page down" to move between pages where applicable, and use the "escape" key when finished with an option.

1. View or Change Scenario / Network

This option takes the players to another menu which allows them to view, change, or print the administrative information contained in the scenario file, the network node data, and the network path data. It also allows them to use the network utility to test the network.

The term "network" as used here refers to the unit movement network. Unfortunately, all unit movement in this game is represented by moving along this network. This will require the player to translate from grid coordinates and routes on the map to nodes and paths in the unit movement network.

In this network, nodes represent positions, or potential positions. Paths represent a route between two nodes. The nodes have a specific grid location on the map whereas the paths provide connectivity between two nodes. There may be two or more paths between the same two nodes. The conventional depiction of a network shows the paths between nodes as straight lines. For this wargame, it is recommended that an overlay be developed which shows all of the nodes as circles over their grid locations and all of the paths as route traces of the routes represented by each path. Each node and path will be labeled with a unique name. (Nodes are labeled with up to three numbers and paths with up to three letters.) In this way, the players can do their normal planning. When they give the movement order to

the computer assisted portion of the wargame, they use the overlay to dictate the route. (A future version of this game should just take the start and end nodes and determine the optimal route.)

Under the node data and path data options, the bulk of the network can be setup before the game. However, it is still possible to add nodes and paths during game play. The importance of setting up a good network cannot be over emphasized. The larger and more comprehensive the network, the more flexible and more realistic the game play will be for the players. The entire game is structured around the movement network. Units occupy nodes and move along paths. Point distribution of ammunition must also take place in a node.

The computer assisted portion of the game is interested in more than just location and length information for each node and each path. There are many other elements of information which are necessary to describe the nodes and paths. These will be covered in more detail in paragraphs b) and c) in this section.

a. Scenario Information (Figure 5.2)

This option allows the players to view, change, or print the administrative information contained in the scenario file. This screen contains the operations order number, the date of operations order, the map sheets used in the scenario, the game start time, and finally an

administrative note to be used at the discretion of the scenario builder.

b. Path Data (Figure 5.3)

Most of the fields in the path data screen are self explanatory. However, a few will require some explanation. A path in this game can have no more than one bridge. The player will have to pick a bridge if there is actually more than one bridge on the map.

Road conditions and path vulnerability each have three levels of quality. It is up to the players to make this assessment in accordance with the relative conditions in the game's area of operations. To get the first data screen, press "F5."

c. Node Data (Figure 5.4)

The node data screen is also fairly self explanatory, however, a few items require some discussion. A given node must have at least one but no more than six paths associated with it. A node can have an ammunition count independent of any fire units or ammunition trucks. This gives the players the ability to preposition ammunition at a node for eventual transfer to a fire unit or ammunition truck. The type position and cover and concealment fields are like the path vulnerability and road condition fields in that the entry is at the discretion of the players but should be based upon the relative conditions in the area of operations. All of these fields are important to the game

because they are some of the previously mentioned factors which help to determine the casualty rates.

d. Network Utility

This option has not yet been implemented. It will eventually allow the player to test the network by tracing routes through the network and determining route distances.

2. View or Change Game Parameters (Figure 5.5)

This option allows players to view or change the game parameters. The game parameters are generally expected values of such variables as convoy speeds. These expected values can be set by the players to reflect the average speeds based on a specific area of operations or opposing force. For example, an area with a good road network will have a higher expected value for the convoy speed. The actual values used by the game will be based on these expected values but will vary according to many factors. Examples of some of these factors are the unit's mission, the road conditions on the specific path, the intensity of conflict, etc.

3. View or Change Commander's Guidance (Figure 5.6)

This option allows the player to view or change elements of commander's guidance. The commander's guidance data consists of items which could be part of a unit's standard operating procedure (SOP), part of a specific operations order, or verbal guidance given by a battalion or

division artillery commander for a specific operation. The commander's guidance can also be changed while playing the game.

4. View or Change the Battalion Configuration / Status

This option allows the player to view or change data concerning the battalion's field trains, fire units and ammunition trucks. The data concerning these elements is the data that will be used to start the game.

Upon selecting this option, another menu is displayed. The first choice allows the player to view or change the trains location. The second option allows the players to view or change the number of fire units, and the number of ammunition trucks in the battalion at the beginning of the game. The third choice allows the player to view or change the starting firing unit data. And the last choice allows the player to view or change the starting ammunition truck data.

a. Location of Trains

This screen contains only the field trains location. Note that the correct entry for all location fields in this game are in terms of the unit's position node name.

b. Number of Fire Units / Trucks (Figure 5.7)

Whatever numbers are entered for the number of fire units and the number of ammunition trucks determines the number of screens that are created for the next two

options. The maximum number of fire units in this game is six and the maximum number of ammunition trucks is twenty-four. If there are three fire units currently on disk and six is entered for the number of fire units, then the third fire unit's data will be duplicated three more times for the additional three fire units. The next option can be used to edit the new fire unit records.

c. Firing Unit Data (Figure 5.8)

Use the "page-up" and "page-down" key to move between the various pages of fire unit data. Most of the fields on these screens are self explanatory. However, a few require additional explanation.

The location fields indicate the locations of the units at the beginning of the game. These locations are in terms of nodes. From a game play perspective, these positions can be pre-hostility assembly areas or they can be initial firing positions. At the beginning of the game, the player will be queried at the start of each time step as to whether or not hostilities have commenced. If the answer is no, the game continues but no casualties are assessed and no rounds are fired. If the answer is yes, the game continues with firing and casualties and the players will no longer be queried about the beginning of hostilities. This procedure gives the players the ability to exercise a pre-hostility deployment plan and tactically position their fire units and

their ammunition trucks prior to the commencement of hostilities.

The maximum rounds capacity per firing section field refers to the number of rounds the particular weapon system / organic ammunition carrier can carry combat loaded. The number of rounds on hand per fire unit can exceed the unit's total capacity while in a firing position. However, when the fire unit displaces, it can only move with its maximum total capacity. The excess rounds will be left as prepositioned ammunition on that node. Separate arrangements must be made with battalion ammunition trucks to transport that ammunition if necessary.

d. Ammo Truck Data (Figure 5.9)

As with the fire unit data, use the "page-up" and "page-down" keys to move between screens. All of the fields should be self explanatory.

E. GAME PLAY

After completing the initialization process, the player should select the game play option. This will lead to the game play control screen (Figure 5.10) which is the hub of the wargame. From this screen the players can initiate time steps, issue commands, change parameters, utilize the network utility, and they can end the game play by selecting the quit option. The game time will always be displayed on any screen which is generated out of the game play control screen.

In order to play the game, the following players are required as a minimum: A battalion commander, a battalion O&I section, and a battalion S-4 and / or Service Battery Commander. These players are required in order to integrate tactical and logistical planning and execution.

The game is actually played by initiating a time step, viewing and / or printing any messages which may be generated during the time step, and then issuing any commands which may be necessary for the battalion to execute its plans. The players themselves should act out their own roles as if the wargame were actually a real situation. However, rather than sending or receiving information / orders using a radio, the players will receive information from the computer and will send orders to the units represented by the computer.

The players should conduct all planning in accordance with their own unit standard operating procedures and Army doctrine. For the computer assisted portion of the game, special emphasis should be given to logistics plans since the computer will require detailed orders for ammunition resupply. Fire plans should also be made either manually or using TACFIRE. Finally, plans should be made well in advance for future fire unit moves so that routes can be planned thereby facilitating timely execution.

1. Time Step

This option will process all events which take place from the current time to the new current time. The difference between these times equates to the length of a time step. The time step size is set by the players under the game parameter option during game setup. A size of 30 minutes is recommended.

There are many functions which take place during the time step. The fire units shoot at some rate which is a function of the battalion's CSR but can not exceed the weapon system's sustained rate of fire. Resupply transactions take place between ammunition trucks and fire units in accordance with their orders. Ammunition trucks return from the ammunition transfer point with new truck loads of ammunition but the total ammunition drawn in a day can not exceed the battalion's CSR. Units / trucks move around the battlefield in accordance with their orders. Units may receive incoming artillery and must request permission for an emergency displacement. And finally, units / trucks may receive casualties where the degree of casualties is based upon numerous factors such as the type position, the position's cover and concealment, and the unit's vulnerability factor.

Casualties are determined each time step whether the units are in a position or are on the move. Casualties are assessed as fractional parts of firing sections and

ammunition trucks rather than whole sections and trucks. This is a result of the fact that Lanchesterian attrition was used. Therefore, casualties are determined as an expected value over time rather than discrete amounts. Due to this fact, it is possible to have less than a whole ammunition truck and less than a whole number of firing sections in a fire unit. In turn, all quantities which are related to these systems are calculated based on the effective percent of the system. For example, if a particular ammunition truck is at 74 effective percent, then it can only haul 74 percent of its full ammunition capacity.

During the time step, the computer not only executes previously issued commands, but also generates messages for the players to review and, if necessary, take some action. A discussion of each message follows:

a. Incoming Artillery

This message notifies the battalion staff that the named unit is receiving incoming and that they request permission for an emergency displacement. It requires the players to make an immediate response of either "y" or "n." If "y" is entered, the unit will displace to its alternate position within the same node. It will be in a moving status for approximately 30 minutes. Upon occupation, its vulnerability level will be set back to an acceptable level. If "n" is selected, the unit will stay in position and ride

the attack out. Casualties will be higher and its vulnerability level will continue to increase.

b. Unit Situation Reports

These messages will be displayed on a periodic basis in accordance with the battalion's standard operating procedure. The situation report frequency can be set using the "commander's guidance" option under the "Change Parameters" option of the Game Play Menu. Use the "page-up" and "page-down" keys to move from one unit situation report to the next. All fire units and the field trains will be displayed.

c. Bridge Out

This message will appear randomly. It notifies the players that a bridge on a certain path is out. It is up to the players to avoid the path once this message is received. If the players send a unit down a path which has a bridge out, then the time the unit spends on the path will be greatly increased.

d. Minefield

This message will appear randomly. It notifies the players that a minefield has been placed on a certain path. It is up to the players to avoid the path once this message is received. If the players send a unit down a path which has a minefield, then the time the unit spends on the path will be greatly increased and the casualties will be increased.

e. Vulnerability High / Critical

This message notifies the players that the vulnerability threshold for the named unit has become high or critical. To warrant a rating of high, a fire unit must exceed the commander's guidance for either the number of rounds fired out of a position or the amount of time spent in a position. The rating becomes critical when both items of the commander's guidance has been exceeded. For the field trains, the rating is high if too much time is spent in the position. And if two times the commander's guidance for time in position is exceeded, the field trains receives a vulnerability rating of critical. In either case, the expected value for the degree of casualties increases as the vulnerability rating increases.

f. Ammunition Low / Critical / Out

This message notifies the players, as a real fire unit would, when the named fire unit's ammunition levels decreases below certain thresholds. If the ammunition count is below 35 percent of the unit's maximum capacity, then the ammunition status is "low." If the ammunition count is below 10 percent of the unit's maximum capacity, then the status is "critical." And finally, if the ammunition count is 0, then the ammunition status is "out."

There is no penalty for an ammunition status of "low." It simply serves as a warning to the S4 to begin

planning for ammunition resupply operations to that unit. If the status is "critical", then the unit is not considered fully available. The amount of time a unit spends critically short of ammunition is incremented and, at the end of the game, a portion of the critically short time is subtracted from the availability time. Finally, if the unit is "out" of ammunition, it is placed in a cold status and is therefore not available to support the maneuver forces. The "Change Rate of Fire" command can be used to tell a firing unit to decrease or increase its rate of fire.

g. Resupply / Pick-up Complete, Request Orders

This message is generated when an ammunition truck has completed an ammunition resupply or pickup mission and needs further guidance. The players then must issue the appropriate commands to instruct the truck where to go and what to do. In most cases, since the truck can only go to the ammunition transfer point from the field trains location, the players will issue an order to get the truck back to the field trains. Another option would be to instruct the truck to go to a specific node to pick up prepositioned ammunition.

This approach was taken because most ammunition trucks / convoys do not have organic radios. In most cases they would call the field trains, utilizing the resupplied unit's radio, to give a situation report after accomplishing their resupply mission.

h. Return from Ammunition Transfer Point, Field Trains Gone, Request Orders

This message is generated when an ammunition truck returns to what was the field trains location and finds that the field trains is no longer there. The player's reaction should be to issue orders to the truck either to get to the field trains or to go somewhere else to deliver the ammunition.

i. Crew Rest Warning

This message is generated when an ammunition truck is given a movement order and the truck's crew has not had the amount of rest dictated by the commander's guidance in the last 24 hours. The truck will still accept the orders, but the casualty rate will be a little higher. If the crew has not had the amount of rest dictated by the commander's guidance in the last 48 hours, then the truck will not be able to accept the orders. This will force the battalion's logistics planners to use all of the trucks equally and to consider crew rest in their planning.

j. Front Line Trace Change

This message will be generated on a random basis. When the maneuver forces are in the offense, the front line trace will move some specified distance usually towards the enemy in the direction of the attack. When the maneuver forces are defending against an enemy attack, the front line trace will move some specified distance usually toward the friendly forces in the direction of the enemy

attack. The player's actions upon receipt of this message should be to redraw the front line trace on their maps and then check their firing unit's ranges to determine if they need to be moved.

2. Issue Commands

This option is chosen to give orders to any of the game's entities, i.e., firing units and / or ammunition trucks. Once chosen, the player is taken to another menu which lists each of the various types of commands. By selecting a command description from this menu, the player will be taken to the command screen. Most of the commands have a unique command number. It is recommended that the "F4" key be used to print the command screen so that the players can keep track of the commands given and their associated command number. The command number is needed in order to cancel a specific command.

a. Convoy Ammunition Trucks (Figure 5.11)

This command enables the players to organize the ammunition trucks into convoys. The convoy name must be unique. Up to 24 trucks can be placed in a convoy but a given truck can only be associated with one convoy. Furthermore, all trucks named in the command must be at the same location when the command is given. If the desired trucks are not all at the same location, then the "move" command must be used to get them all to the same place.

Once a truck is assigned to a convoy, it can only receive orders as a part of the convoy. If one of the trucks in the convoy can not accept a convoy order due to a critical lack of crew rest, then the truck will automatically be taken out of the convoy and will not execute the order.

b. Remove Ammunition Truck Convoy (Figure 5.12)

This command enables the players to break-up the named convoy. All of the trucks in the convoy will be disassociated from the convoy and will once again be able to receive orders as individual trucks.

c. Move a Unit (Figure 5.13a and 5.13b)

This command enables the players to order units to move around the battlefield. The players should first plan the move on a map of the battlefield. Once they determine where they want the unit to move, they should refer to the unit movement network overlay to determine if the appropriate nodes and paths exist to make the move. If they do not exist, the players should use the "change parameters" option to add the needed nodes and paths. Once this is completed or if the nodes and paths already existed, then this command screen can be used. All the fields on these screens should be self explanatory except the "node to resupply" field. If the unit making the move is a firing unit or ammo truck/convoy and it is supposed to pick up or deliver ammunition along the way, then the players should

enter the node where the ammunition resupply is to take place. This will cause the fire unit to stop at that node to pick up ammunition. If the ammunition resupply trucks are not there, then the ammunition resupply will not take place and the fire unit will continue on its way after one time step.

d. Request Situation Report

This command is used by the players to get an immediate situation report. The output of this command will be in the same format as the unit situation report which is generated on a periodic basis during the game. The situation reports for all units will be displayed.

e. Change Firing Rate (Figure 5.14)

This command is used to speed-up or slow-down the named fire unit's rate of fire. The number entered for the firing rate is a percent of the battalion's CSR. If the number exceeds 100, then the firing rate will be such that the unit, if in a firing status all day, will expend more than its "CSR worth" of ammunition. On the other hand, if the CSR is less than 100, then less than the "CSR worth" of ammunition will be expended. This command is intended to be used to manage the unit's ammunition expenditure. One of the statistics reported at the end of the game is the amount of ammunition the battalion has on hand in excess of the battalion's CSR. Since the CSR should never be higher than what the battalion requested as the required supply rate

(RSR), this statistic should be minimized. Therefore, the players should use this command both to ensure the fire units do not run out of ammunition and to ensure that too much ammunition is not on-hand at the end of the game.

If the firing rate entered is less than 50 percent of the battalion's CSR, then the unit will be considered less than 100 percent available. The percentage of time considered available will be the same percentage as the firing rate.

f. Ammunition Resupply Mission (Figure 5.15)

This command is used to assign a resupply mission to an ammunition truck or convoy. Legal entries for the "Unit to Resupply" field include any of the fire unit names and the word "prepo." By entering preposition, the players are telling the trucks to deliver the ammunition to a node where the ammunition will be unloaded. The players can then use the next command to tell a fire unit to pick up the ammunition.

For any resupply mission to actually take place, the trucks must be full and located at the same node as the fire unit. If a preposition mission, then the truck must be at the node specified in the resupply command. In order to get the truck and the fire units into the same location, one or more move command must be used. Therefore, to cause a resupply mission to be executed, the players must issue at least two commands, a resupply mission command and a move

command to the truck / convoy or to the fire unit or to both the fire unit and the truck / convoy.

It is important to note that since the wargame uses Lanchesterian attrition, the trucks are attrited by fractions of a truck. Therefore, as the game progresses, each truck will have a different effective strength. Since a truck can only carry its "effective strength" worth of a full truck's hauling capacity, less rounds than full capacity of a truck will actually be delivered to the fire unit or node. This will require the players to keep track of each truck's hauling capacity. This approach may not be 100 percent realistic but it does account for the expected value of casualties over time.

g. Ammunition Pickup Mission

Selection of this command will take the players to another menu. This menu gives them two options, "Fire Units" or "Ammo Trucks".

If "Fire Units" is chosen, the command screen (Figure 5.16) is used to tell a fire unit to go to a node to pick-up prepositioned ammunition. Like the resupply command, this command will require the players to give a move command to the fire unit if it is not already in the appropriate node.

If "Ammo Trucks" is chosen, the command screen (Figure 5.17) is used to tell an ammunition truck or convoy to go to the ammunition transfer point (ATP) to pick-up

ammunition. The trucks must be in the field trains location to receive this command. This constraint was imposed because it was assumed that the trucks would have to at least pass through the field trains on the way to the ATP to get paperwork, and / or fuel, food, maintenance, and rest.

The amount of time the trucks take to go to the ATP is randomly chosen based on a normal distribution with a mean of the "ATP turn-around time" as specified under the game parameters. It is important to update the ATP turn-around time whenever the field trains moves or the ATP moves. Unlike the resupply command and the fire unit pickup command, a move command is not necessary with this command.

The battalion can only pick-up its "CSR worth" of ammunition during any given day. Even if the day's allotment of ammunition has already been picked-up from the ATP, the wargame will allow the players to send trucks to the ATP. However, the trucks will return empty. The point is that it is up to the players to manage both the pick-up and delivery of ammunition.

h. Issue Fire Order (Figure 5.18)

This command enables the players to issue battalion level fire orders. It can be used to direct the fire units to shoot battalion level targets of opportunity or planned fires. Planned fires can be planned either manually or using TACFIRE to fulfil the requirements of the fire support annex of the maneuver unit's operations order.

Once the fire plan is finished, it can be sent to the fire units using this command screen. All that is necessary is the unit name, time to fire, and the number of volleys.

The intent of this command is to ensure that the S3 players are able to perform all of their responsibilities while trying to integrate operations and logistics in the command and control of the battalion. They must be careful not to shoot so much ammunition that the fire unit's rate of fire must be reduced to too low a level to be considered 100 percent available. On the other hand, they must conduct normal fire planning in support of the maneuver forces.

i. Cancel Command (Figure 5.19)

This command enables the players to delete the following commands: "Move a Unit", "Ammo Resupply Mission", "Ammo Pickup Mission", and "Issue a Fire Order." Each of these command screens has a "command number" field which contains a computer generated number. It is this number which is entered into the "Cancel Command" screen in order to cancel a specific command. For all but the "move" command, the command must not have been executed in order for the "cancel" command to have any effect. In the case of the "move" command, if the move has already begun but is not yet completed, the "cancel" command will cause the unit to stop at the nearest node. If the unit is a firing unit and it has ammunition, it will go into a firing status unless it is immediately given another movement order.

If a fire unit is on the move, and the players want to hip-shoot the fire unit, then they must cancel the move command and issue a fire order. If any type unit is on the move and the players want to change the route or destination, then they must cancel the "move" command and then issue a new "move" command from the unit's new location.

3. Change Parameters

Selection of this option from the game control screen takes the players to more menus which give them the ability to change all the entries which were made during game setup except for the battalion configuration / status records. For example, the time step size can be changed, nodes and paths can be added to the network, and the situation report frequency can be changed. The battalion configuration records can not be accessed because they contain the status of the battalion at the start of the game. To give the players the ability to edit these records would be the same as giving them the ability to circumvent the commands. If the players want to check a units status, they can use the "Request a Situation Report" command.

4. Network Utility

This option is not yet implemented. It is envisioned that this option will be used by the players to conduct movement planning. This option will eventually be capable of determining the shortest route by time or

distance and determining the safest route according to the path vulnerability levels.

5. Output Game Statistics

This option gives the players the ability to request the game's statistics, MOE, and the number of rounds on-hand in excess of the battalion's CSR at any point during the game. It is recommended that this listing be printed at least once at the end of every 24 hours of game play. This will provide a means of comparing statistics between games to judge if the players are improving in their ability to command and control the battalion. More will be said on the meaning of the statistics in Section F of this chapter.

5. Quit

Upon selecting the "Quit" option, the players are given two options. The first option is to quit with the intention of continuing the game later. This gives them the ability to continue the game where they left off at a later date. All of the appropriate files are saved to disk and are recalled the next time the game is played by selecting the "Continue an Old Game" option from the main menu. The second choice is to quit the game without the intention of ever continuing the game. In this case, the game's data is not saved to disk.

In either case, the computer will provide a listing of the wargame's statistics, the MOE, and the number of

rounds remaining in excess of the battalion's CSR before returning to the MS-DOS "A:" prompt.

F. INTERPRETATION OF THE RESULTS

The "results" referred to in this section is the listing which is printed whenever the "Output Game Statistics" option or the "Quit" option is selected from the game play control screen (Figure 5.20). The listing itself should be self explanatory. It is the interpretation of the listing which requires some explanation.

Each time the listing is obtained, it is based on the cumulative statistics from the start of the game to the current time rather than since the last time the listing was obtained. Since this wargame does not allow replacements of casualties, the availability ratio will gradually decrease as time goes on due to casualties alone regardless of how well the players exercise command and control over the battalion. The same is true of the truck casualty ratio except that it will gradually increase. Since Lanchesterian attrition was used, this amount of change in the statistics should be consistent for the same wargame setup data and the same C2 decisions at any given amount of time into a particular game. Therefore, in order to be able to compare the performance between different groups of players playing the same game or between the same group of players playing the same game at two different times, they must obtain a printout of the statistics at the same amount of time into

the game during each game. For example, if the players request output of the statistics after every 24 hours of game play each time they play the game, they will be able to compare their performance from game to game. This is because the differences in the statistics for the same amount of time into each game will be due to command and control decisions made by the players. There will be no dependence on the amount of casualties which occur based simply on the amount of time the game has been played.

The ratios provide a measure of how well the players have performed with respect to the specific statistics involved in each ratio. For a better idea of how well the players have performed with respect to the overall command and control of the battalion, the Field Artillery Battalion Command and Control Effectiveness Index is used to determine a number which is based on all of the ratios. The number itself has no meaning other than as a means of comparing performance from one game to the next, assuming the same game setup data is used for each game. It can also be used to compare performance between two different groups of players each playing their own game when each game had the same setup data.

The final statistic is the number of rounds the battalion has on-hand in excess of the battalion's CSR. This statistic first counts all of the ammunition each firing unit has and then adds the amount of ammunition the

battalion has prepositioned in nodes. Even prepositioned ammunition which is currently behind enemy lines is included in the count because it is assumed that the loss of ammunition was a result of a poor command and control decision on the part of the players. The product of the battalion's CSR and the number of tubes in operating condition is then subtracted from the total count. The player's objective is to minimize this number. If the number is negative, the battalion has less than the CSR on-hand. The greater the number, the more rounds the battalion has on hand in excess of the CSR. Since it is assumed that the RSR equals the CSR in this game, large numbers would tend to indicate that the battalion did not properly project their required ammunition supply rate.

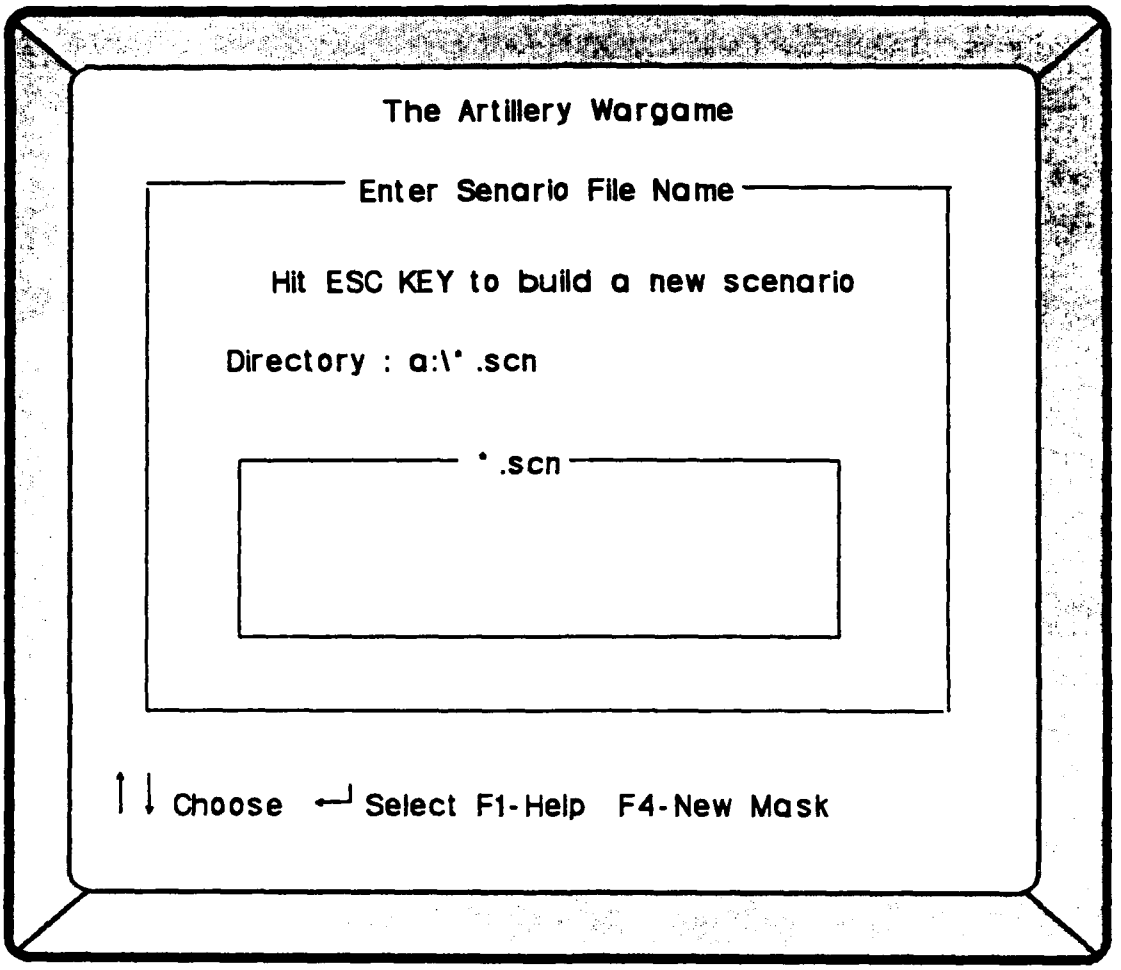

Figure 5.1 Build Scenario Screen

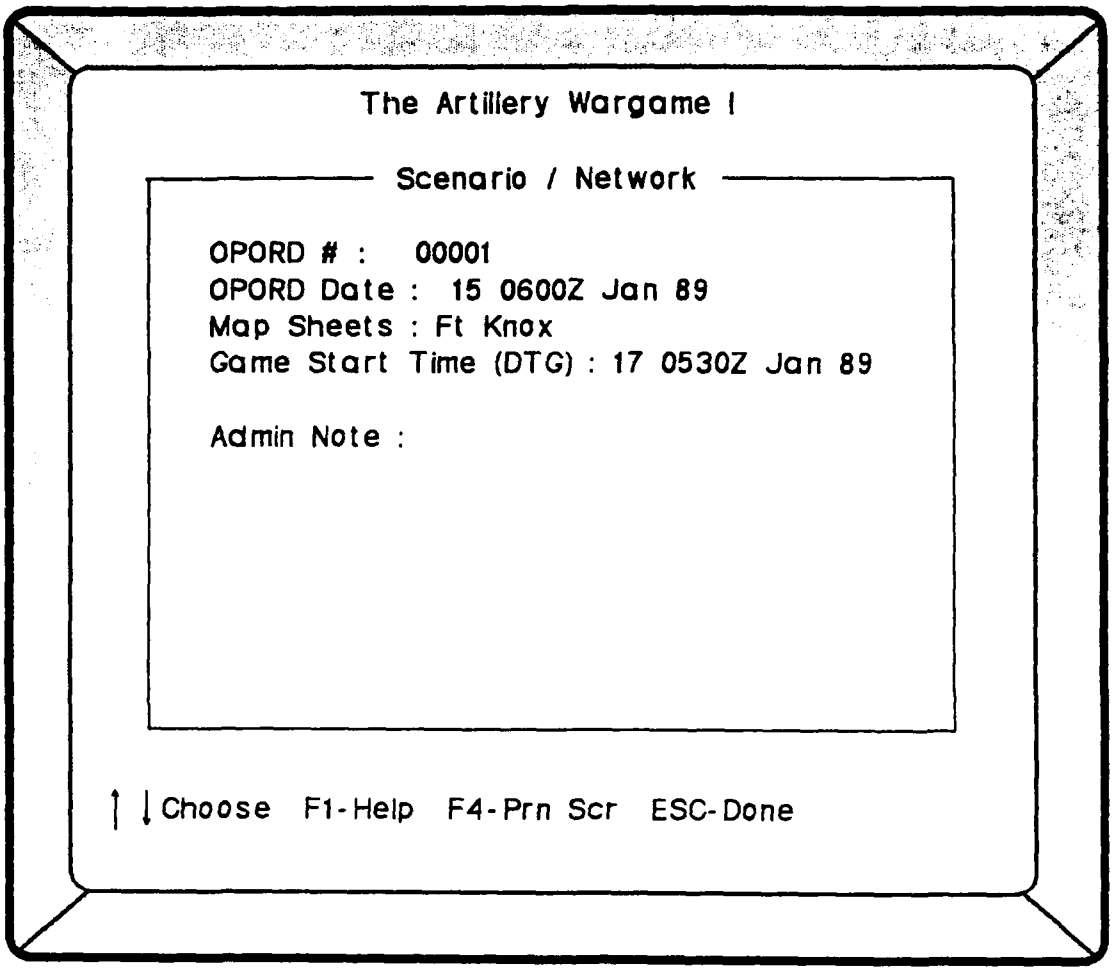

Figure 5.2 Scenario Data Screen

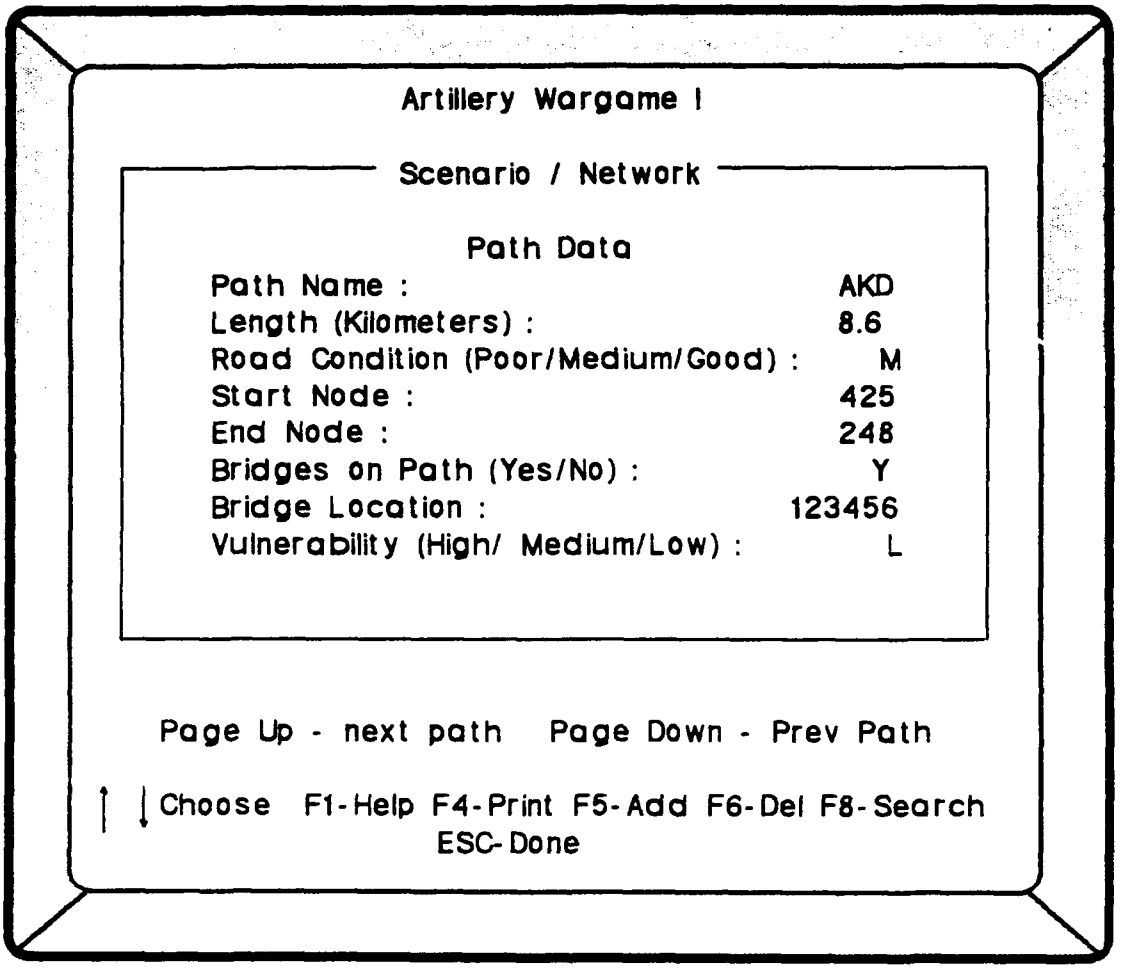

Figure 5.3 Path Data Screen

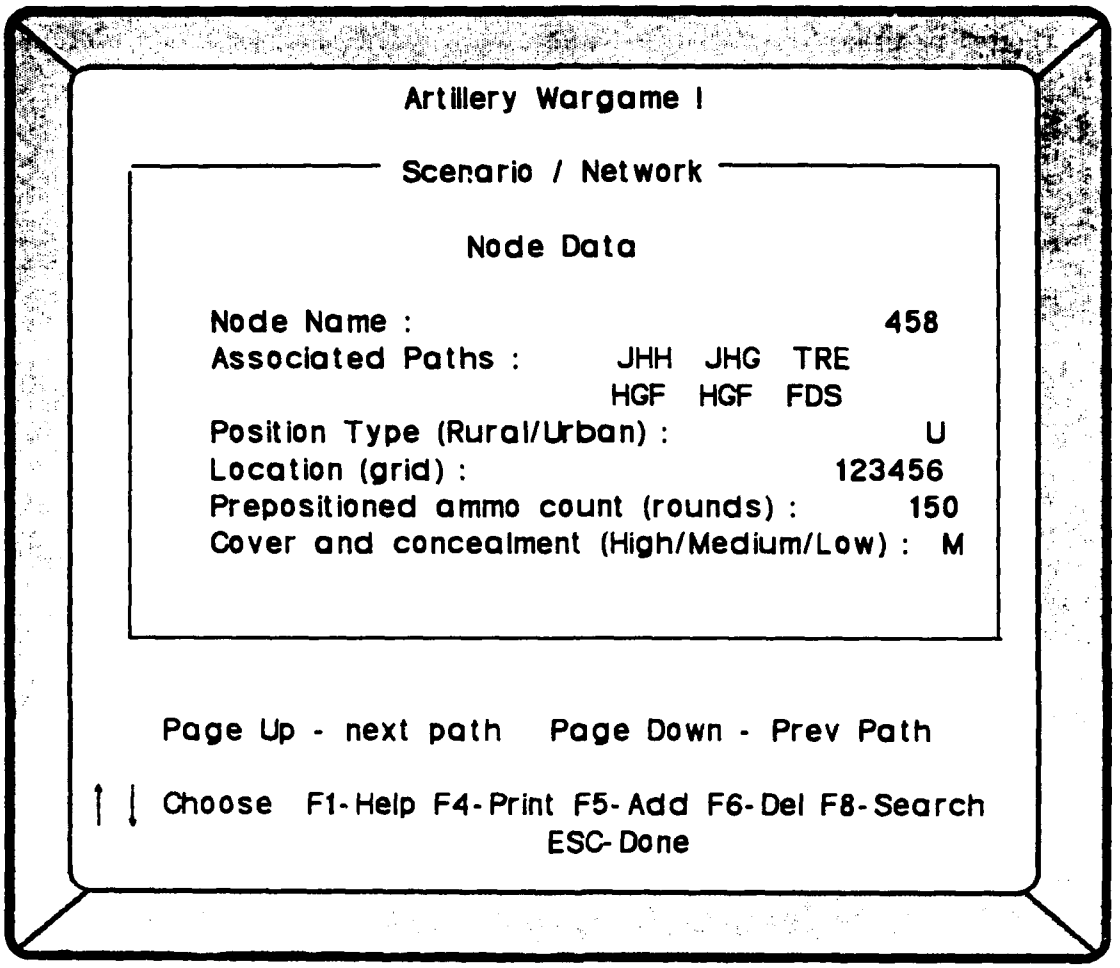

Figure 5.4 Node Data Screen

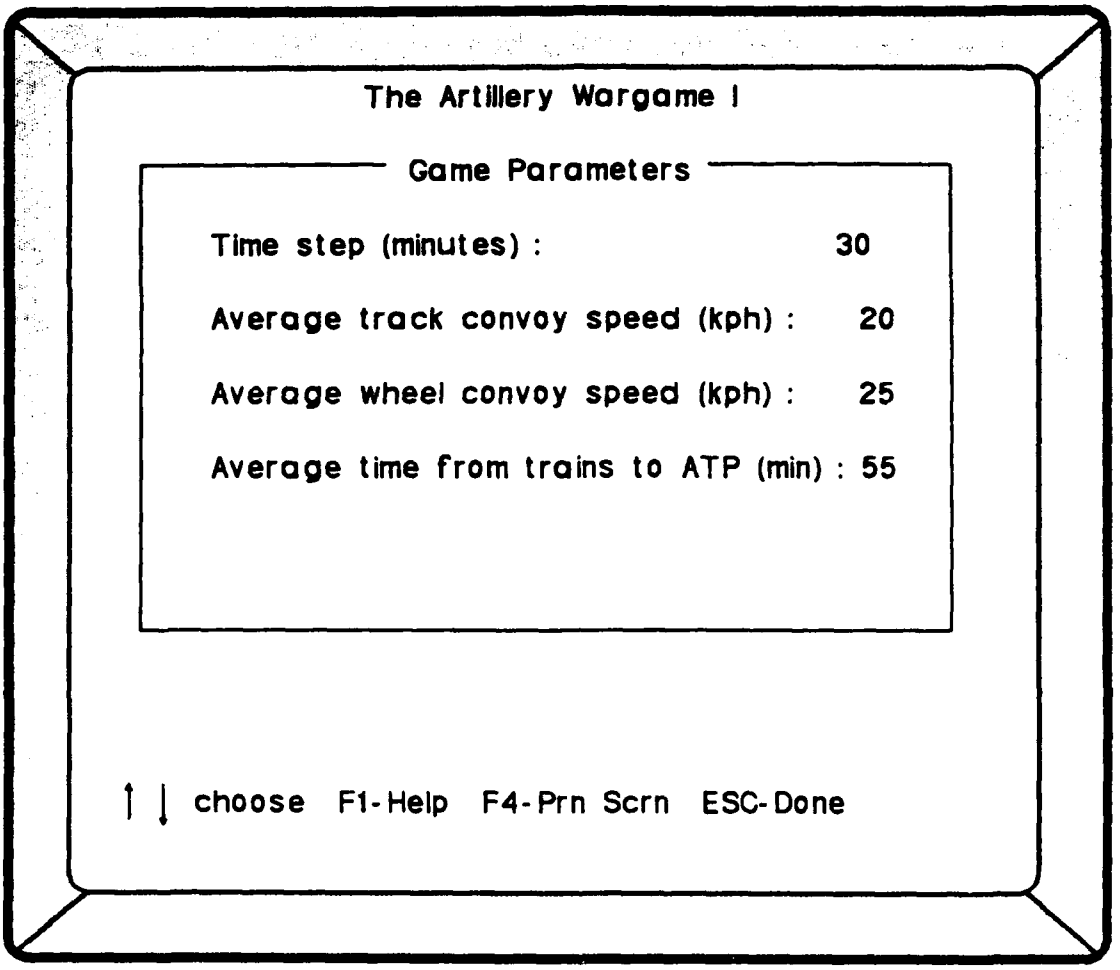

Figure 5.5 Game Parameter Screen

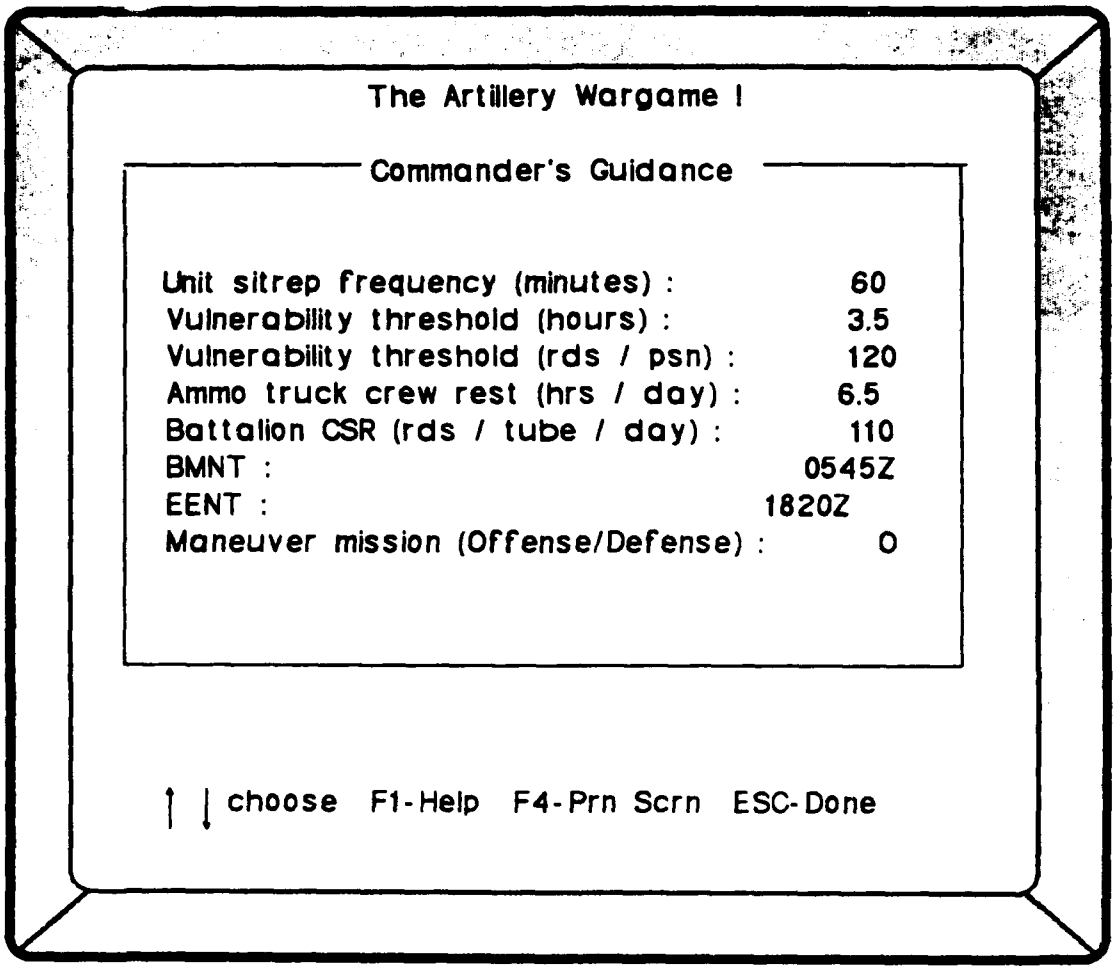

Figure 5.6 Commander's Guidance Screen

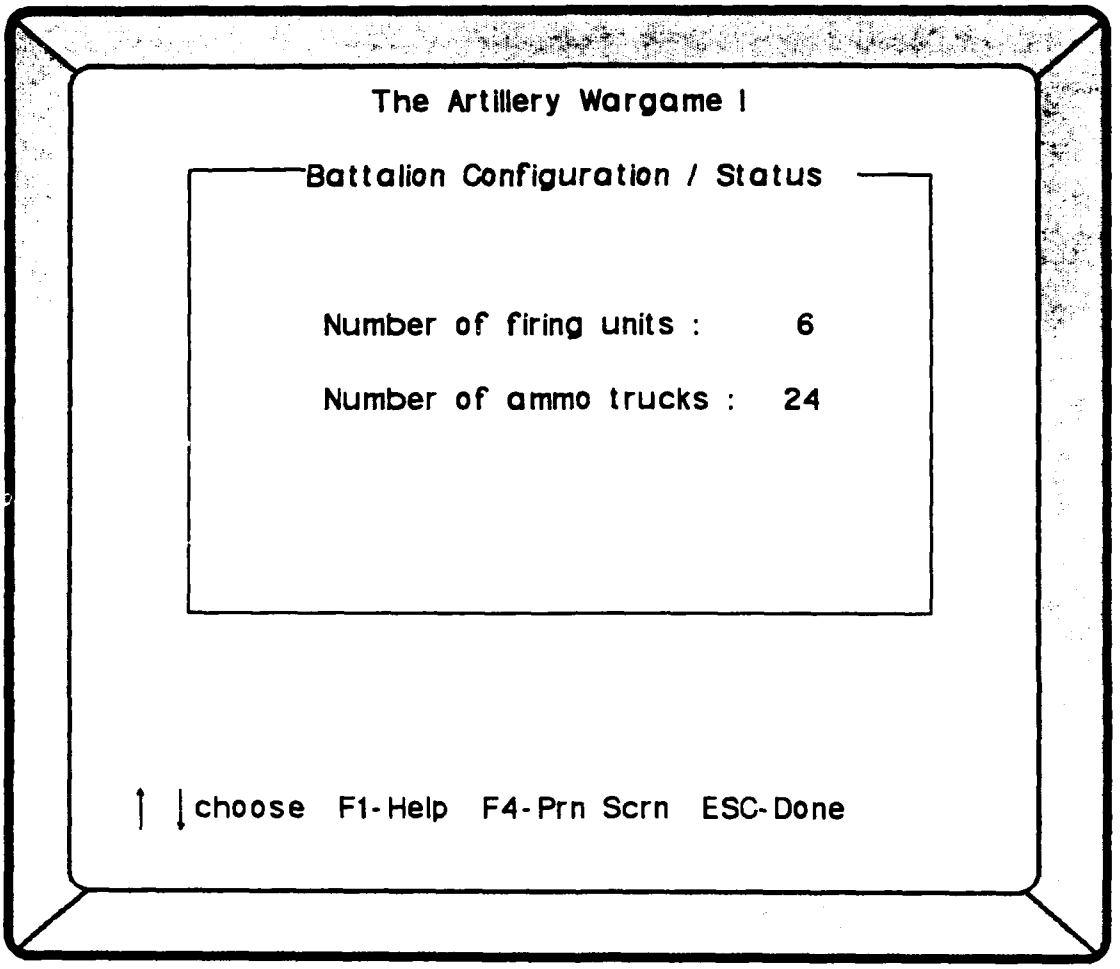

Figure 5.7 Bn Configuration / Status Screen

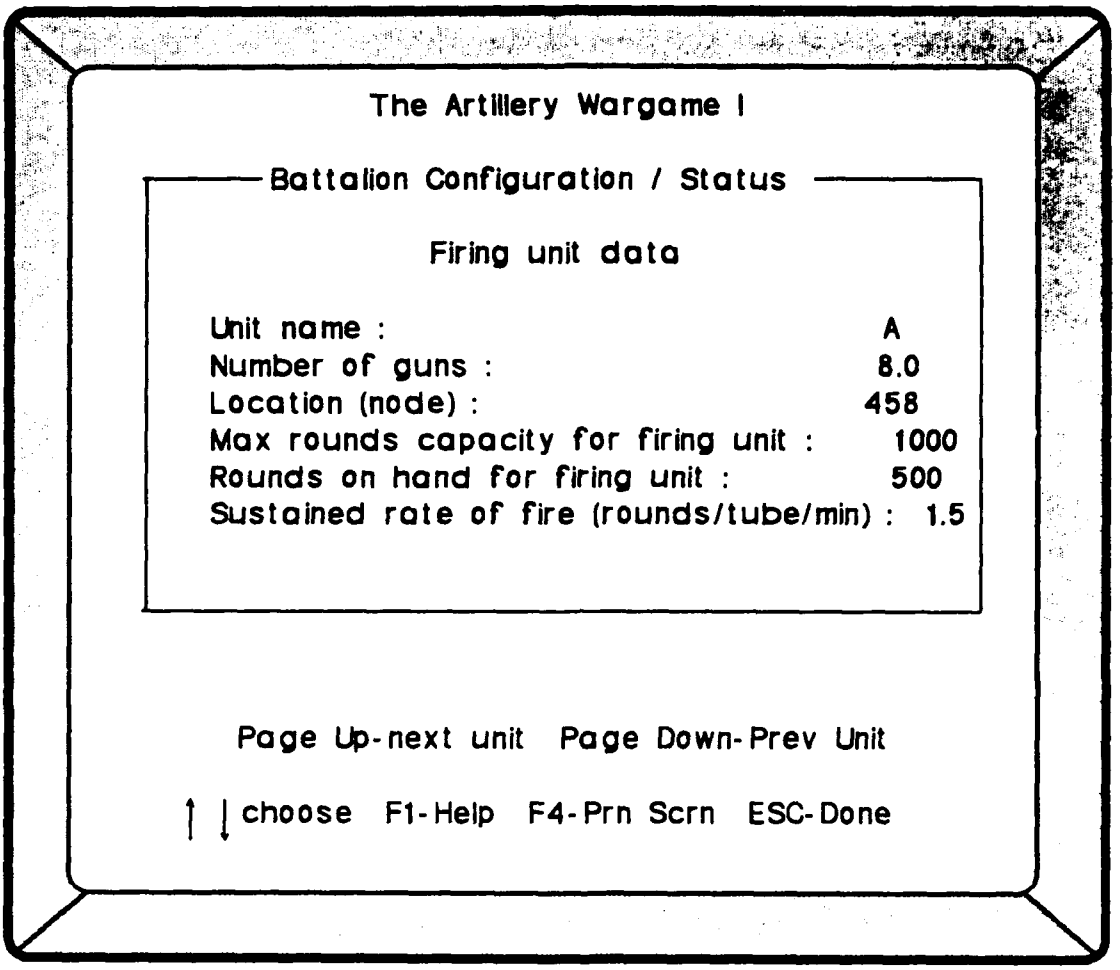

Figure 5.8 Firing Unit Data Screen

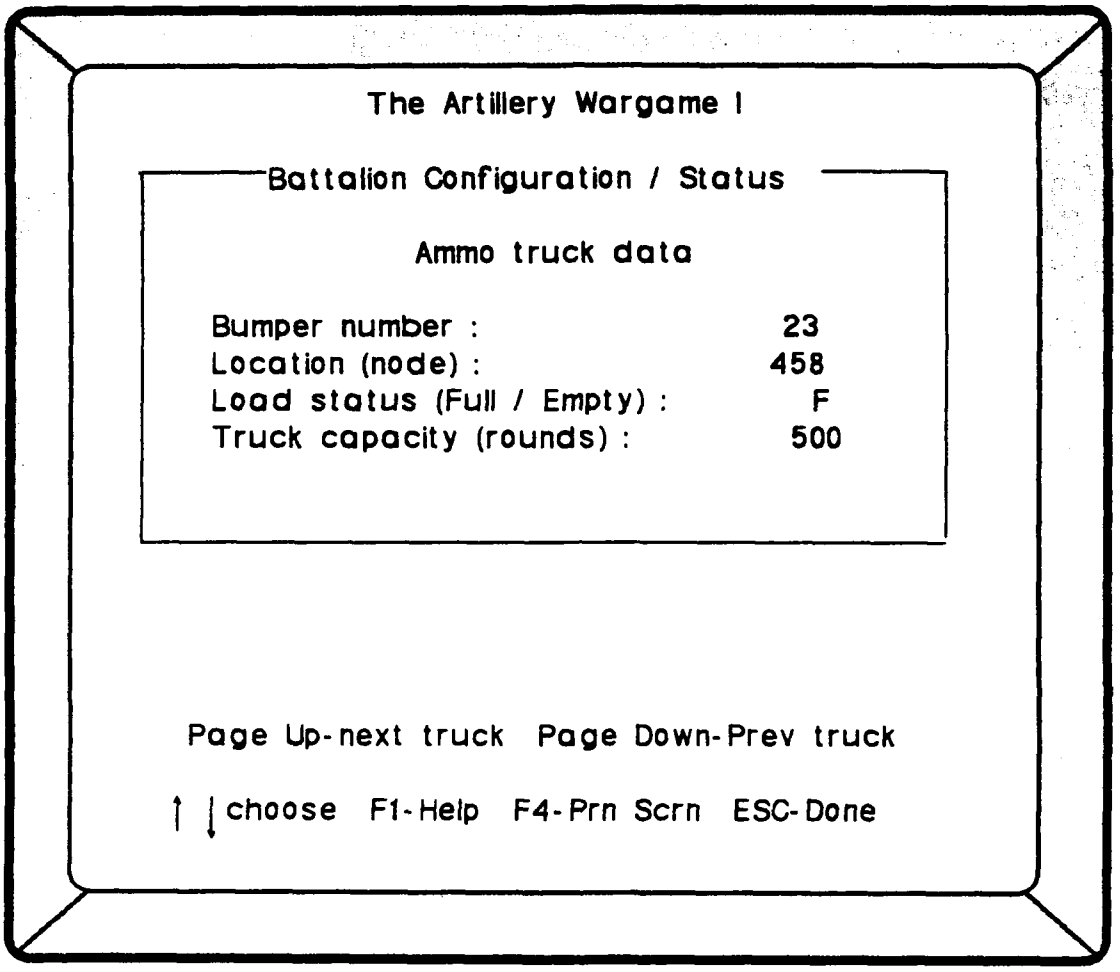

Figure 5.9 Ammo Truck Data Screen

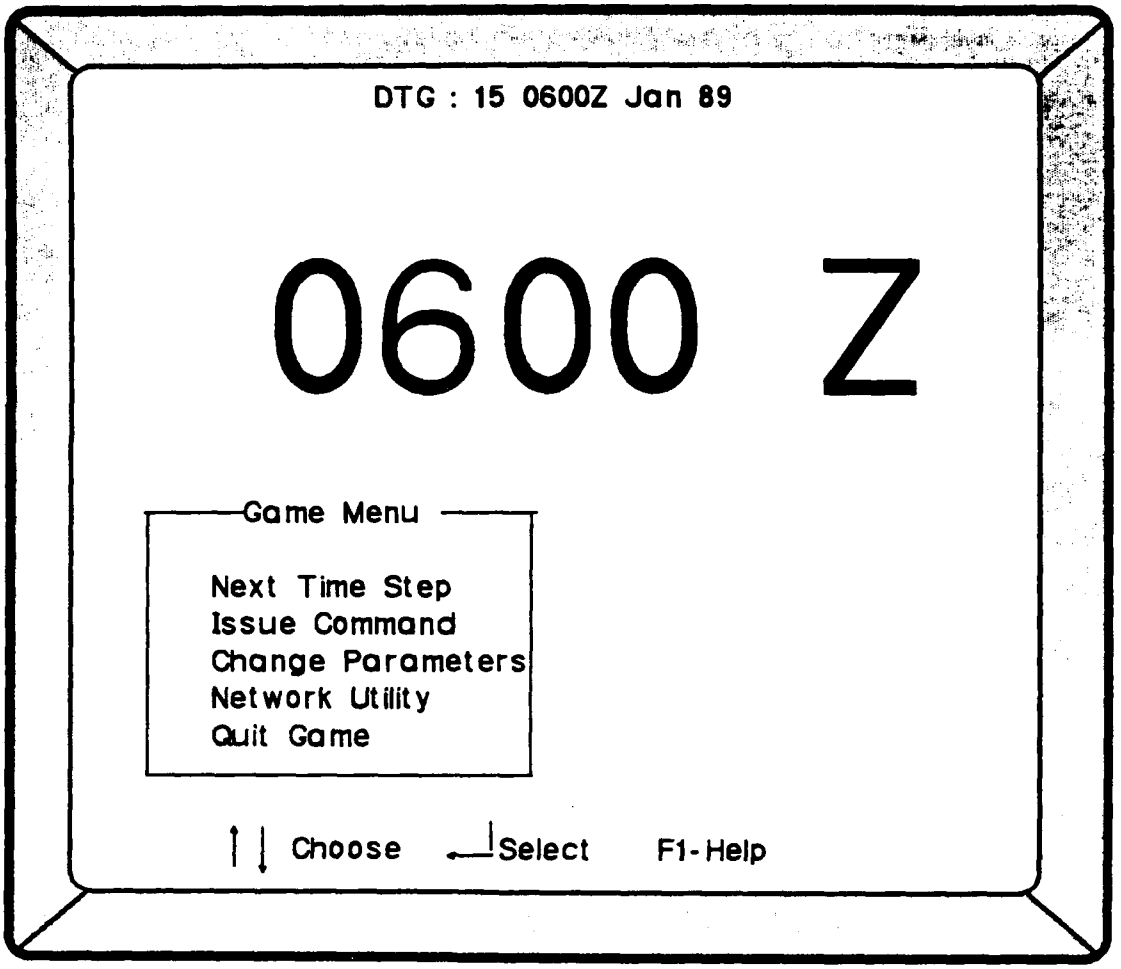

Figure 5.10 Game Play Screen

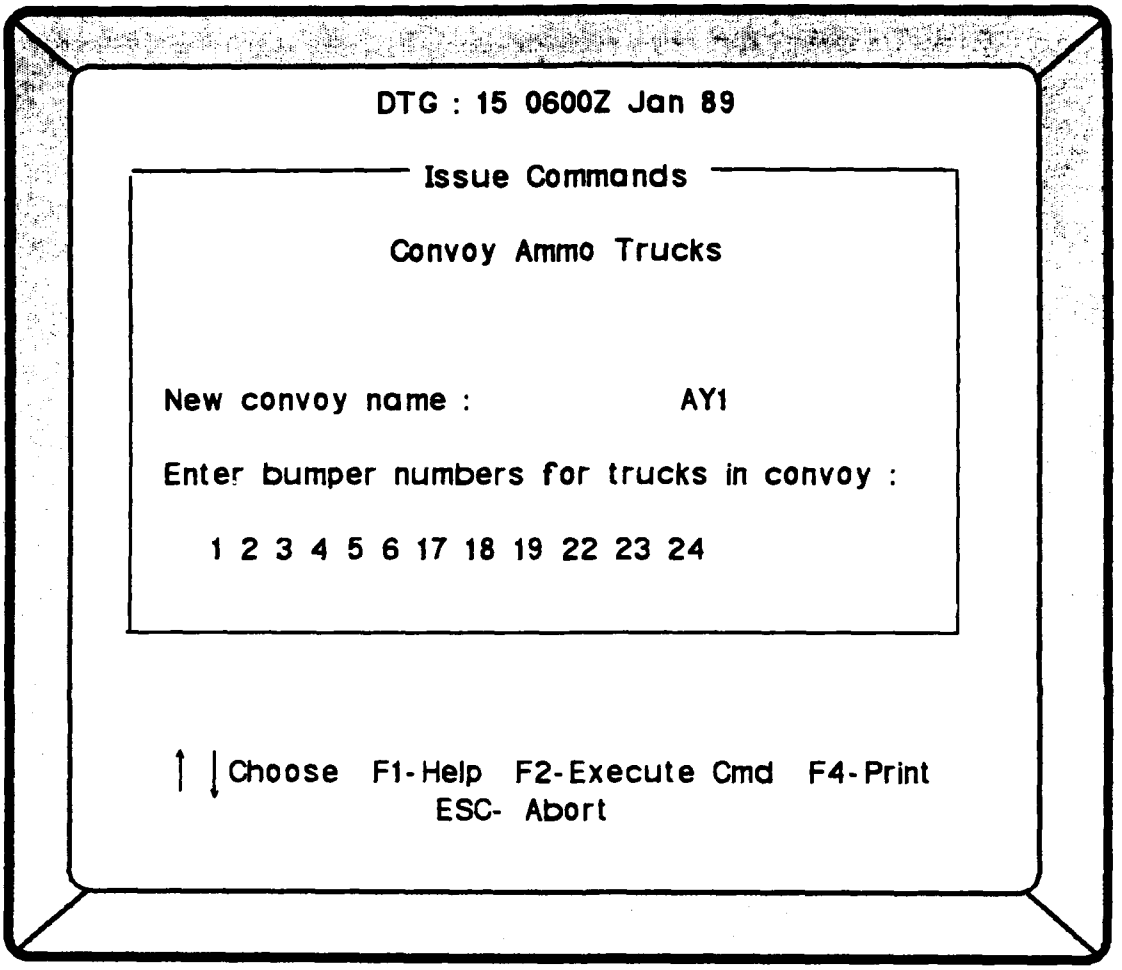

Figure 5.11 Create Truck Convoy Screen

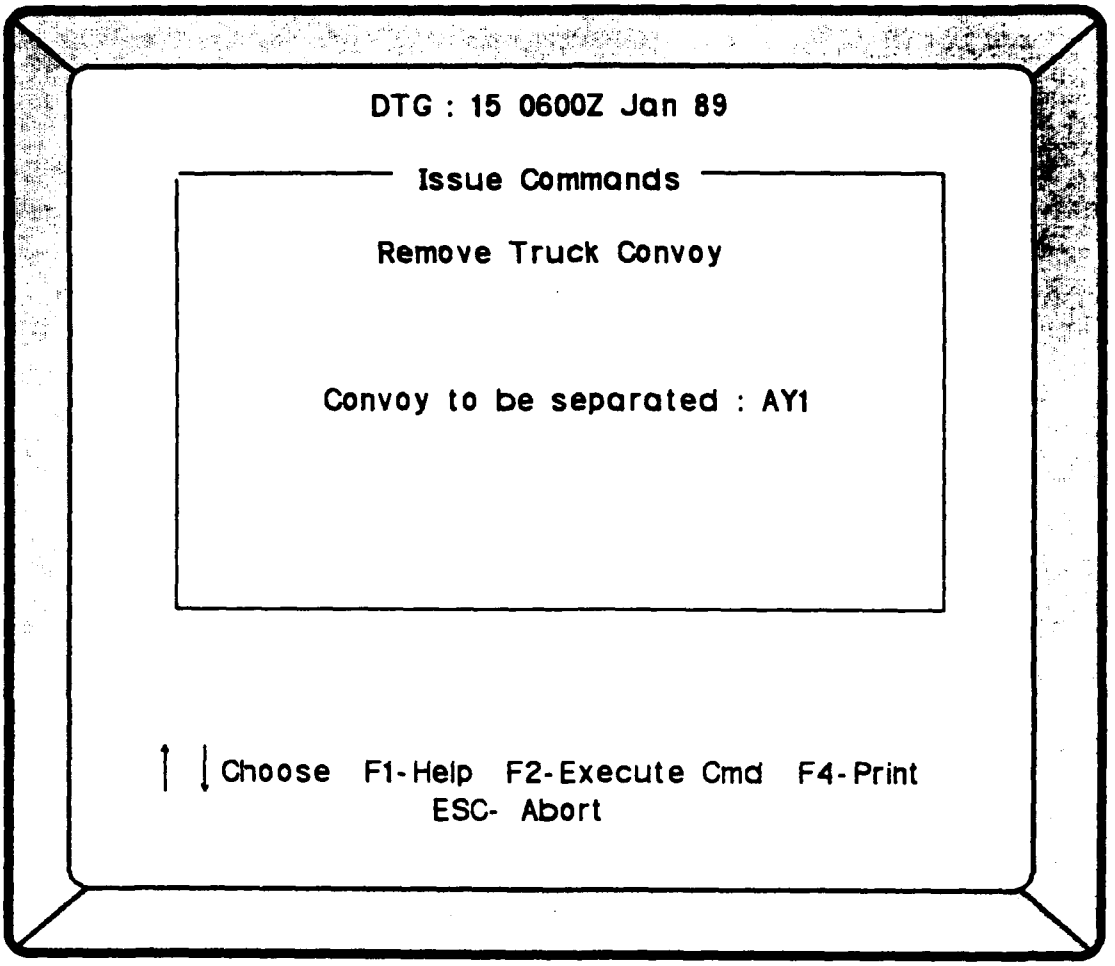

Figure 5.12 Remove Truck Convoy Screen

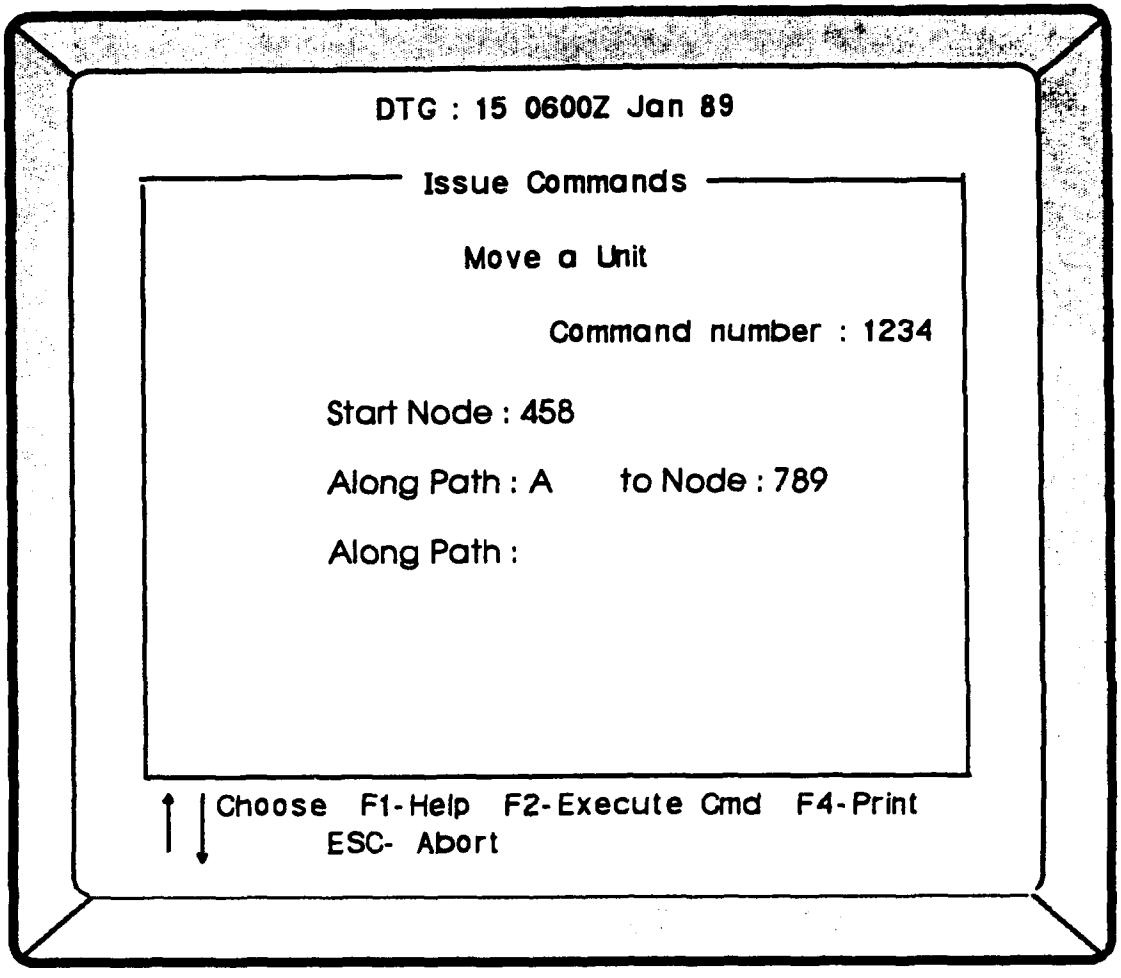

Figure 5.13a Move a Unit Screen (Part 1)

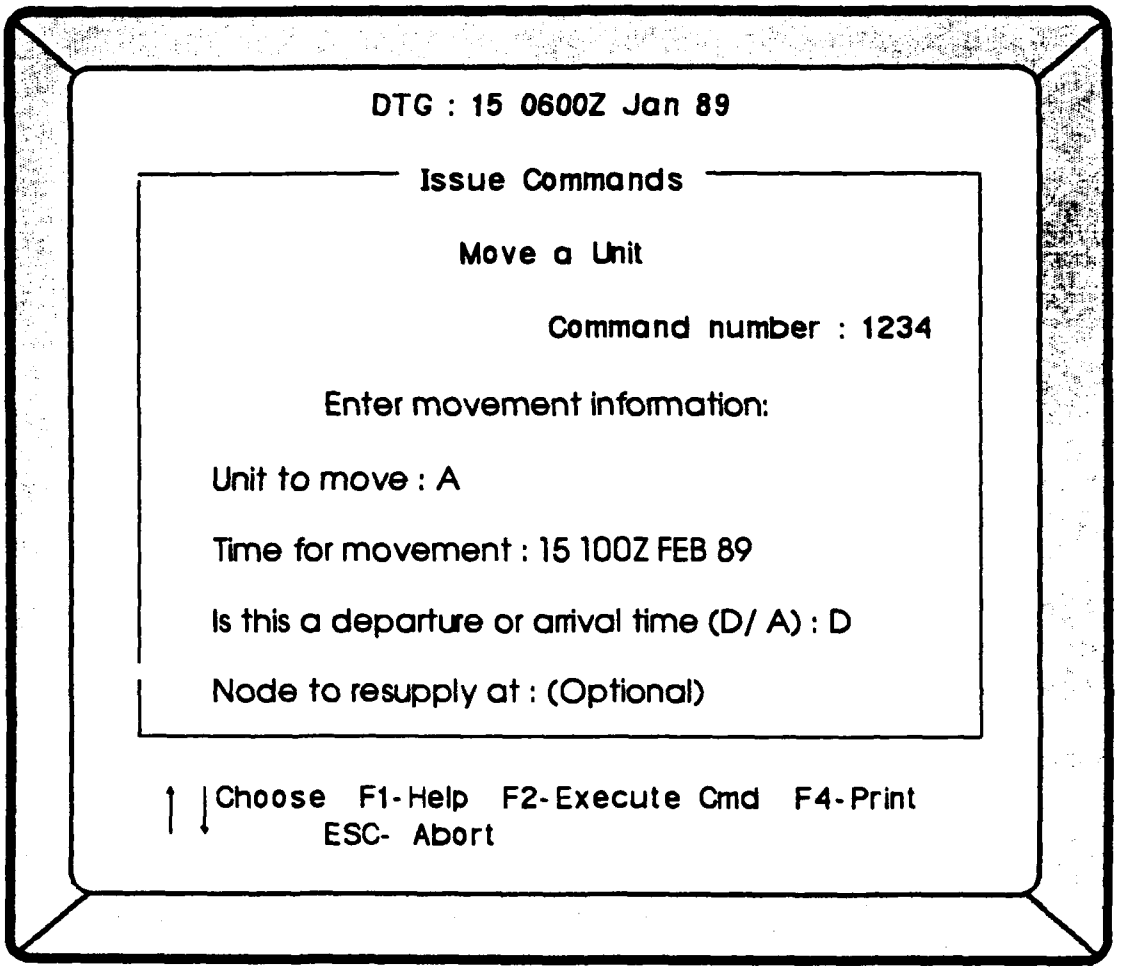

Figure 5.13b Move a Unit Screen (Part 2)

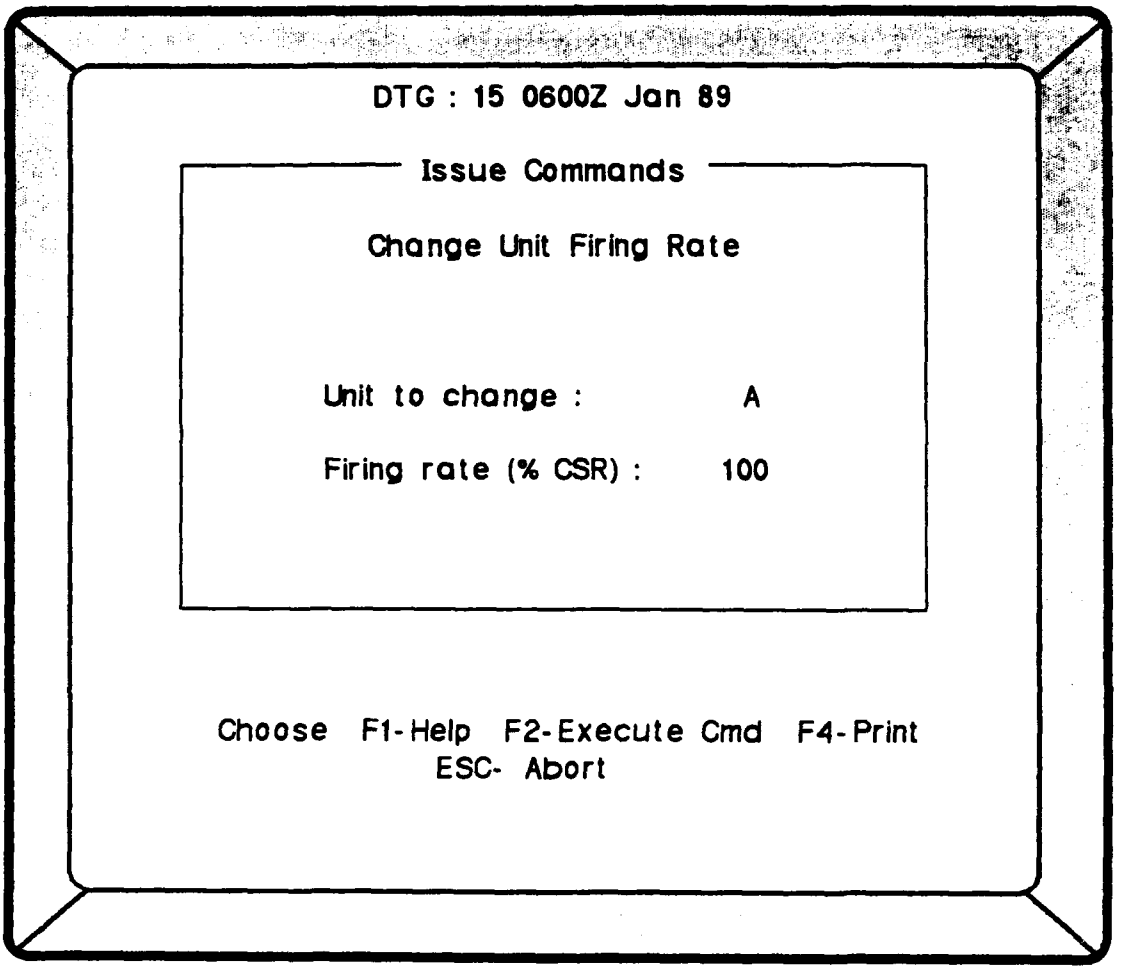

Figure 5.14 Change Unit Firing Rate Screen

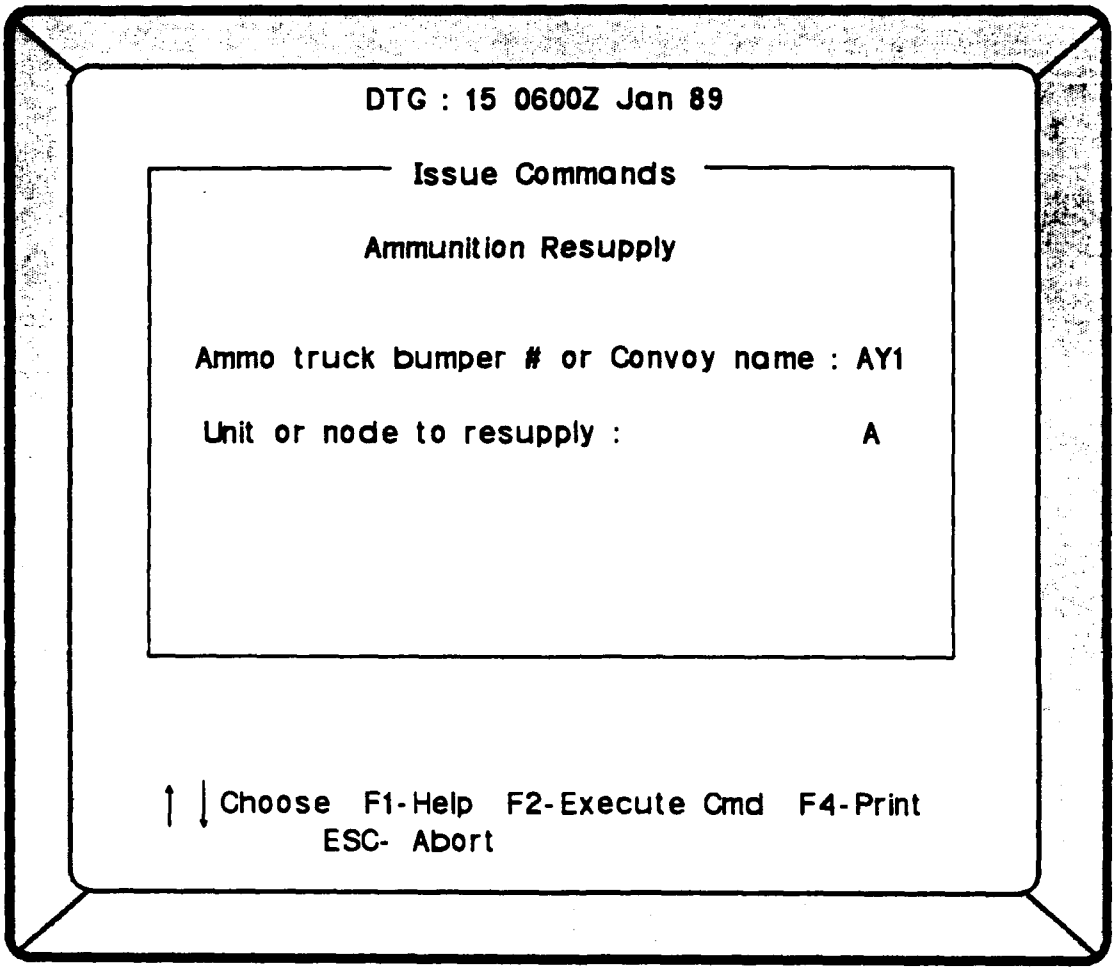

Figure 5.15 Ammo Resupply Mission Screen

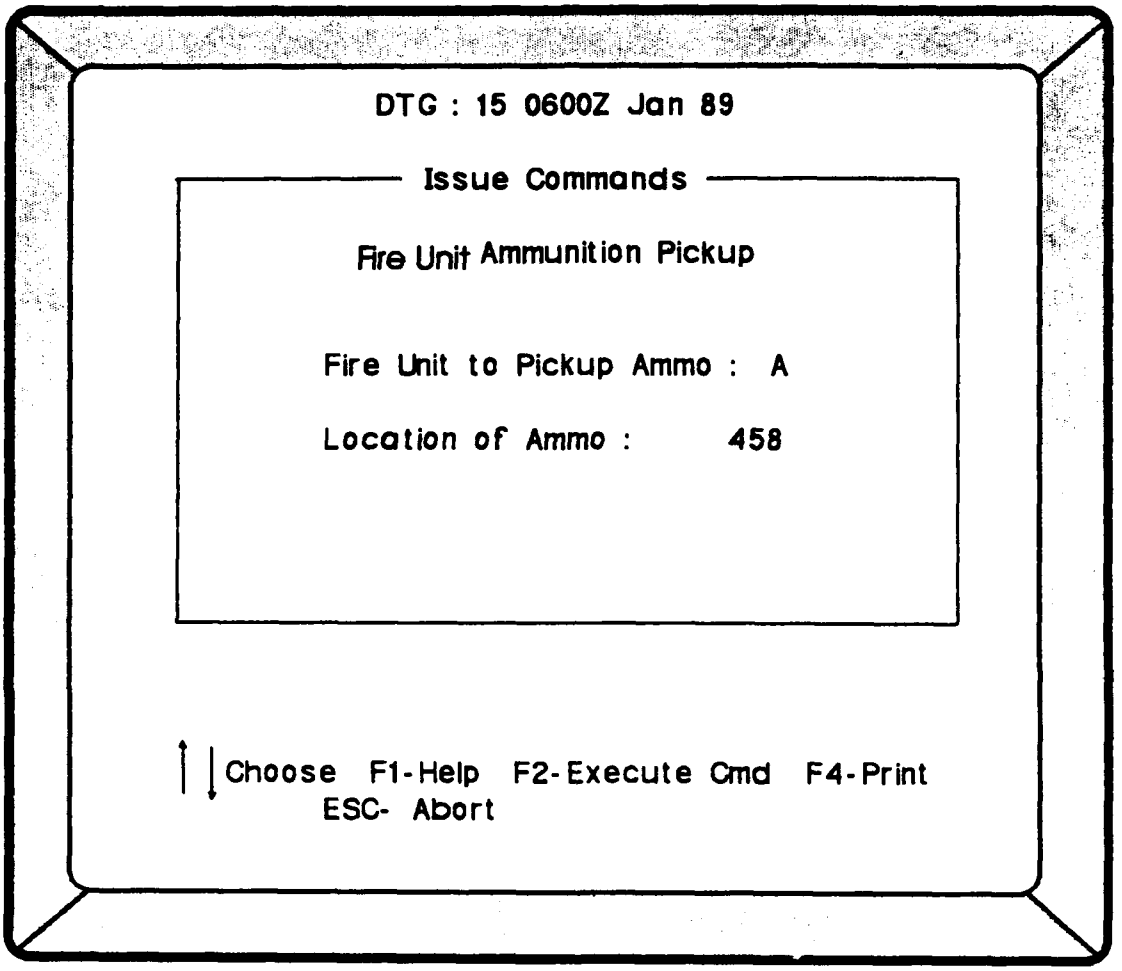

Figure 5.16 Ammo Pickup Mission Screen

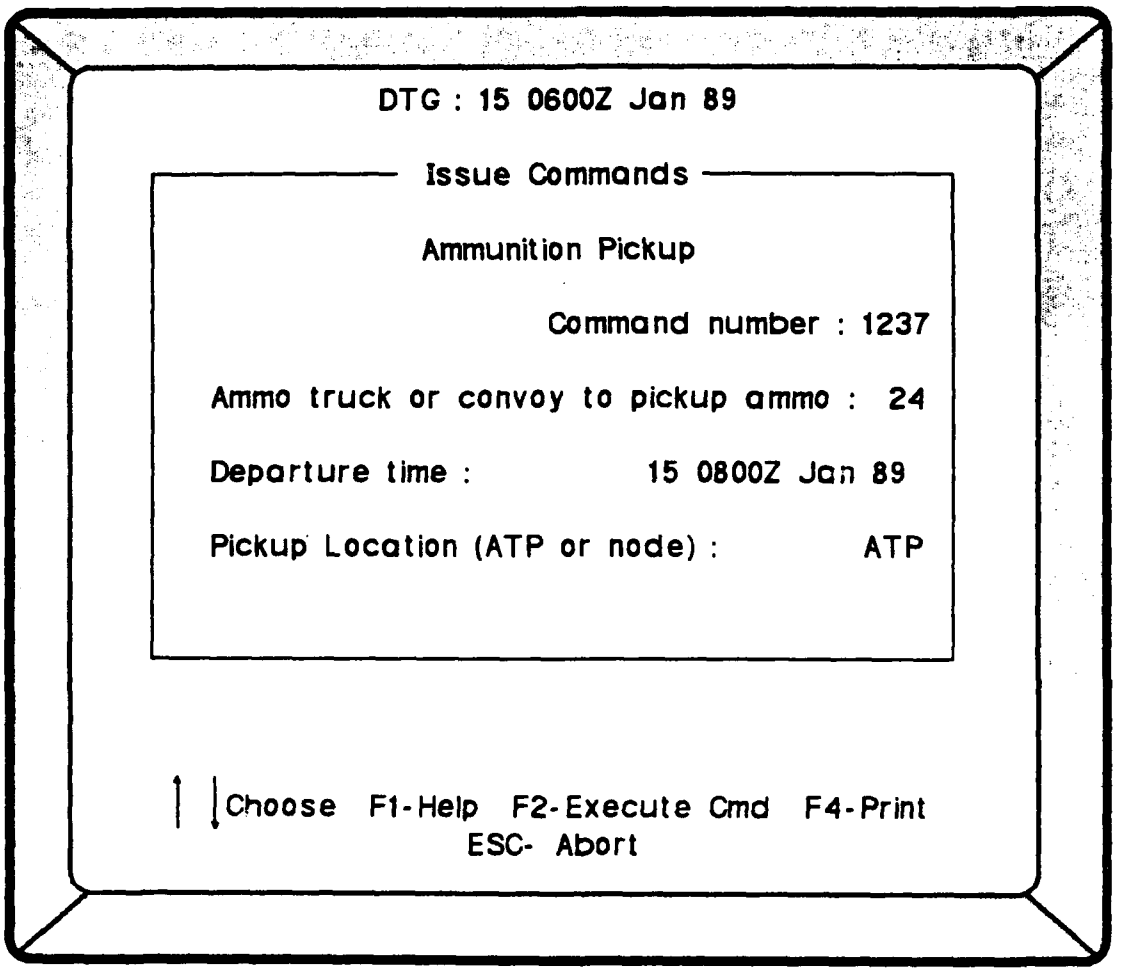

Figure 5.17 Ammo Truck / Convoy Pickup Screen

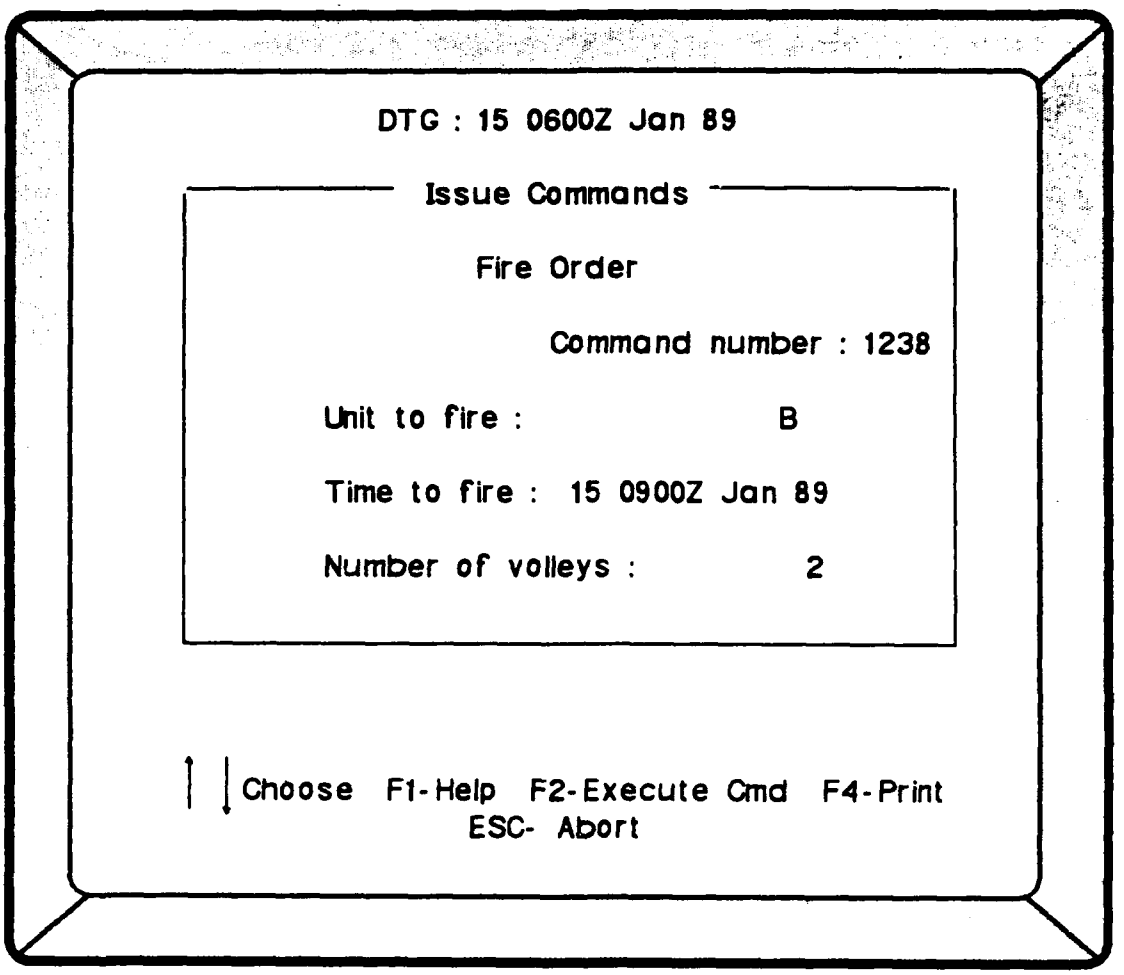

Figure 5.18 Fire Order Screen

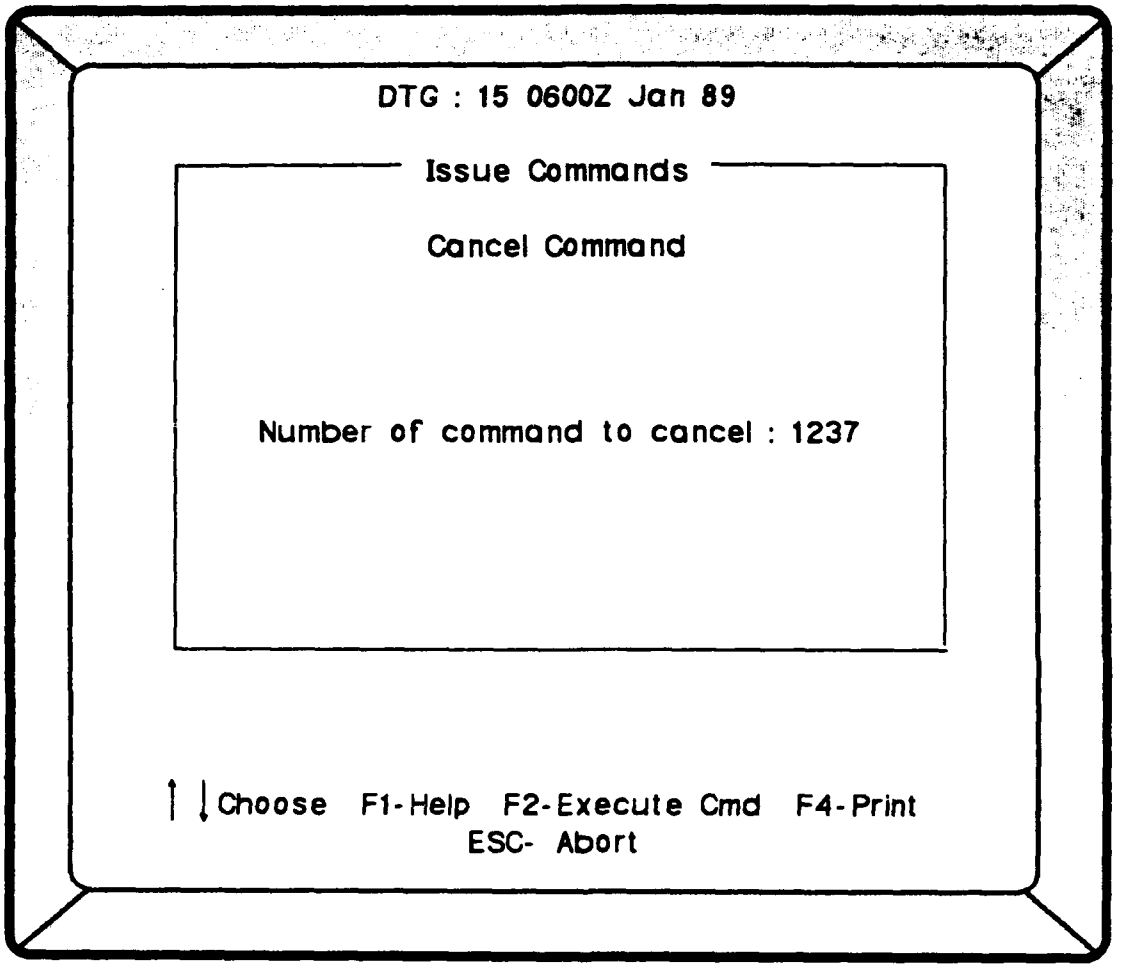

Figure 5.19 Cancel Command Screen

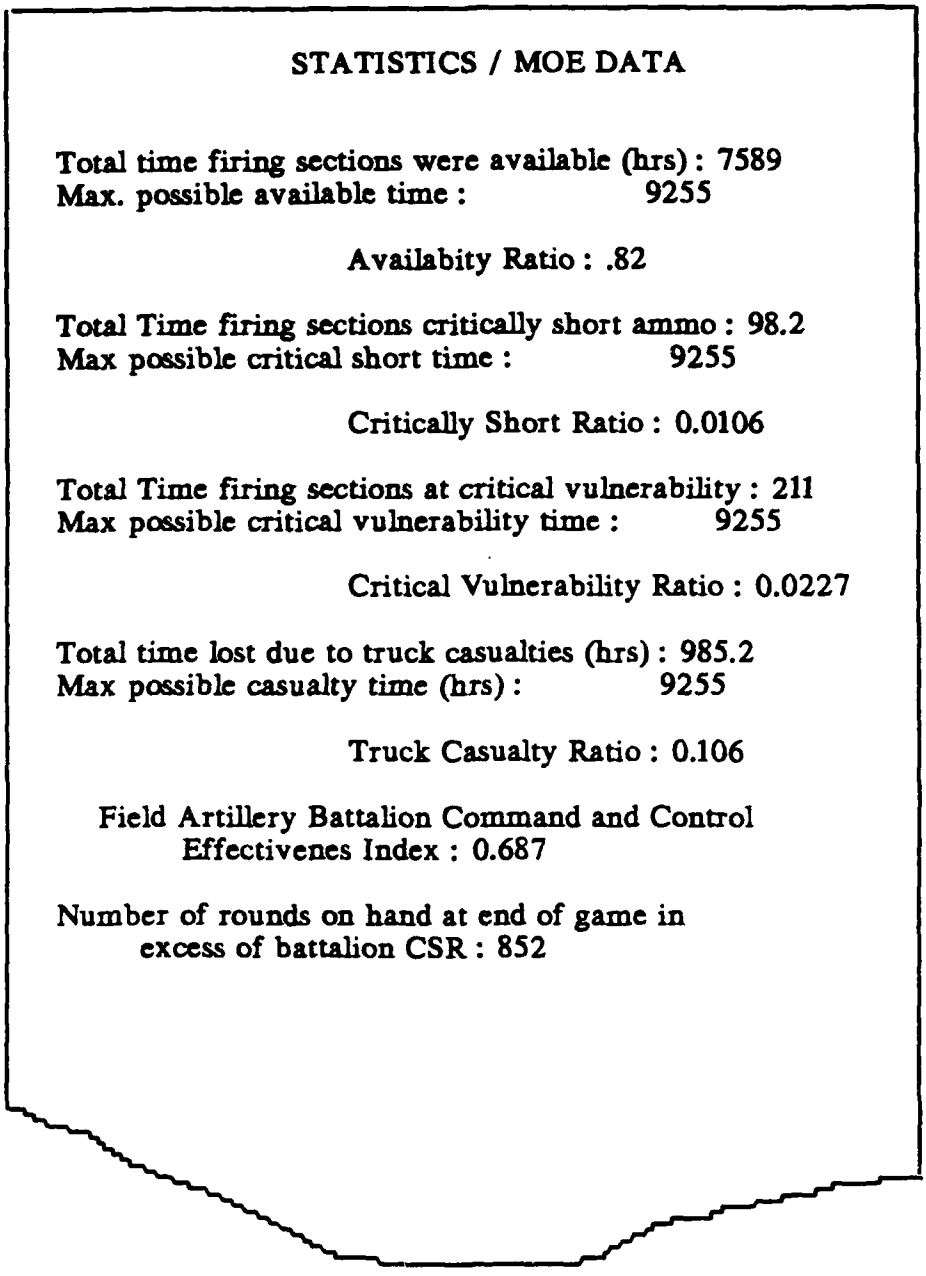

Figure 5.20 Statistics / MOE Printout

VI. SUGGESTED ENHANCEMENTS AND POTENTIAL USES

A. INTRODUCTION

The primary objective of this wargame has been achieved; to provide a tool for training officers in the integration of tactics and logistics as it relates to the command and control of a field artillery battalion. As with any undertaking of this kind, there is still plenty of room for improvements which would enhance the wargame's effectiveness in training these officers. In spite of the need for improvement, the wargame, as it currently exists, is still very useful.

B. POTENTIAL WARGAME USES

It is very important to note, before discussing any of the wargame's uses, that this game is <u>not</u> to be used as a tool to <u>validate</u> a unit's war plans. This point cannot be emphasized enough. The game, while it was intended to be as realistic as possible, does not claim to accurately model casualty rates. It is primarily for this reason that the wargame should not be used to validate a unit's war plans.

It is possible, however, to use the wargame as a tool to develop familiarity with a unit's area of operations and its plans. By loading a unit movement network for a unit's area of operations, the unit's configuration into the battalion configuration file, the commander's guidance into the

commander's guidance file, and SOP / operations order items into the game parameter file, the players can exercise and become familiar with the unit's general defense plans. Pre-hostility deployment plans, to include occupation of assembly areas followed by occupation of initial positions and pickup and delivery of the unit's basic load of ammunition can easily be practiced. Just as easily, the unit's post hostility plans can also be practiced. This type of training, using the unit's actual wartime area of operations and battalion configuration, can pay great dividends if the plans are ever called upon to be used in time of war.

One of the strengths of the wargame is the ability for the computer assisted portion of the game to be integrated into any level of CPX. For example, the game can be played as a part of a battalion's weekly officer professional development time, in conjunction with a battalion level CPX, or it can be integrated into a full scale, higher level, CPX. In every case, the utility of the game is that it allows the players to fully integrate tactical decisions with the logistics considerations. Additionally, it measures their performance using the Field Artillery Battalion Command and Control Effectiveness Index. By providing this feedback, the players can assess their decisions, and perhaps vary their method of command and control in order to maximize howitzer availability time,

minimize casualty time, vulnerability time, and time spent critically short of ammunition.

Another training strategy is to play the game competitively. For example, at the division artillery level, the game's setup files can be established so that they are the same for each battalion. Then, each battalion can play the game for a specified period of game time. At the end of the game, the game's statistics are assessed to determine which battalion more effectively command and controlled the wargame's units. A similar technique can be used for training within a battalion by playing the game competitively between the battalion's day and night shift personnel.

Finally, the game can be used by a battalion to study the effect on their tactics of increasing enemy lethality. By changing the value of "a" in the Lanchester equations found in the source code, the casualty rates can be changed. For each level of casualty rates, a set of tactics can be developed to more effectively command and control the battalion. The types of things which can be varied within the set of tactics include the size of ammunition convoys, the frequency and distance of moving firing units, the rate of firing for fire units, the use of point or unit distribution for ammunition resupply, the types of positions which are occupied (rural or urban), and the positioning of

the field trains in relation to the fire units and the ammunition transfer point.

C. IMPROVEMENTS / ENHANCEMENTS

The wargame, as it currently exists, is fully functional. However, there are some areas which can be expanded in order to make the wargame more useful. The enhancements described in the following paragraphs are listed in order of priority from highest to lowest.

1. Different Ammunition Types

Currently, the wargame allows the players to manage ammunition of only one type. While there are many things that can be learned about the integration of tactics and logistics when only using one ammunition type, the issues become much more complex when the myriad of possible ammunition types are included in the game. It is not only a logistics management problem, but also it has a profound impact on tactical planning.

In order to implement this enhancement, the fire unit, truck, and node records all will have to be modified to include each of the various ammunition types. Additionally, everywhere in the time step that ammunition is either transferred or expended, it must be done for each ammunition type. Finally, new criteria must be used to determine what constitutes low or critical ammunition levels.

2. Explicitly Play FLOT / Ranges

Currently, there is no cartesian coordinate system implemented within the wargame. This would not be difficult to implement and would enable certain useful functions. For example, it would enable the howitzer's range to be played. Currently, it is up to the players to check the unit's range fan against the battlefield geometry to determine if the unit needs to move due to range considerations. If the cartesian coordinate system were implemented, this would enable the computer to also check the range. If the players failed to move the unit as the FLOT moved out of range, then the unit would no longer be considered available. Other functions would be enabled as well by implementing this enhancement, range checks could be made for fire missions, unit positions could be overrun by the FLOT, and prepositioned ammunition could also be overrun by the FLOT.

In order to implement this enhancement, the node records need to be modified to include an X and Y coordinate for the UTM grid location of the node. Additionally, a record needs to be created for the coordinates of the FLOT, and procedures need to be written to calculate the distance between two points and to determine if a point lies above, below, left, or right of a line.

3. Requests for Fires

The way the wargame is now written, the players can fire a fire mission but the target is not explicitly named.

It is up to the players to ensure the target is within range before firing. Furthermore, the only impetus for firing comes from the players themselves, either to fire a fire plan directed by the operations order or to fire targets of opportunity generated by the players themselves. Since the players are assumed to be the battalion staff, the only types of fire missions which are played are battalion level missions or battery missions which are originated at the battalion level. All other firing is done by establishing a firing rate in terms of the CSR in order to expend the ammunition.

The first problem can be resolved by implementing the cartesian coordinate system. The second problem can be resolved by randomly generating requests for fires on randomly generated coordinates located on the far side of the FLOT. This process can be written to simulate FIREFINDER radar output or target lists from the division artillery target intelligence files. In either case, it would be up to the players to decide which targets to engage, which fire units to use, and with which ammunition types.

4. Play Fuel

Fuel was not played in this version of the wargame because fuel is already played in real training through actual consumption. Nevertheless, fuel is a very important

consideration in the integration of tactics and logistics. As such, it should be added to the wargame.

In order to implement this improvement, a similar approach to the one used for the ammunition resupply can be used. Rather than rounds of ammunition, resupply will be in gallons of fuel. Rather than a firing rate, consumption will be in terms of a rate based upon the unit's posture, e.g. in position or moving. Like the ammunition trucks, fuel can be delivered by the combination of a movement command to the trucks or fire units and a resupply mission command.

Implementation of this command will ensure that the players do not move the units so much that they run out of fuel. It also forces them to plan for fuel resupply using either unit or point distribution thereby integrating the requirement for fuel into the battalion's overall operational plan.

5. Play Maintenance

Like fuel, maintenance was not included in the first version of the wargame because it is regularly practiced in real training. It is, however, a real problem and therefore should be included in future versions of the game.

It can be implemented by forcing a certain amount of "down" time or maintenance time for every kilometer traveled by the battalion's vehicles. This "down" time would then be subtracted from a fire unit's availability time. For

ammunition trucks the "down" time could be handled similarly to the way that crew rest requirements are currently handled. In short, the truck would not be available for ammunition resupply missions while in a "down" status.

6. Network Utility

The network utility was assigned the lowest priority because it provides a service which is normally done manually by the players and it may be construed as detracting from the player's training. The network utility is intended to determine the shortest route (time or distance) between two nodes in the unit movement network or determine the safest route in terms of vulnerability factors. It can be used as a staff planning aid in determining unit movement routes or it can be used to automate the movement orders by allowing the players to specify that a move between two nodes be done using the shortest or safest route.

The node and path records needed for this enhancement are already in place. All that is needed are procedures containing the appropriate algorithms to implement the functions.

D. FINAL COMMENTS

At the very least, this wargame can serve as a working prototype for a wargame to train officers in the integration of tactics and logistics as related to the command and control of a field artillery battalion. The word "working"

is emphasized because this wargame, as currently implemented, is fully operational and is capable of use by units in the field. The wargame's greatest strengths are that is requires very little training on the use of the wargame itself, it can be played on an IBM compatible personal computer, it uses the Field Artillery Battalion Command and Control Effectiveness Index to provide feedback, and it forces the players to consider aspects of field artillery battalion command and control which are very difficult if not impossible to practice through actual training.

Finally, it should be pointed out that this wargame was originally conceived to satisfy a perceived deficiency in training which is common to all units that require large quantities of bulk supplies, such as ammunition, to sustain them in combat. The point is that this wargame, with relatively minor changes, could be used for air defense artillery units, armor units, and infantry units. In short, the concept can be expanded into a whole class of wargames which can be used by battalion level officers for training in the integration of tactics and logistics.

APPENDIX

DOCUMENTED SOURCE CODE

```
(************************************************)
{
  Program name : WARGAME
  Purpose : this program is a computer assisted
     wargame designed to exercise the command and
     control functions in a field artillery
     battalion as they relate to the integration
     of tactics and logistics.
  Written by : Anthony R. Ferrara
               Michael W. Schneider
  Program completed : 23 March 1989
  Language used : Turbo Pascal 5.0
  System : IBM Personal System II Model 50
     running DOS 3.30
  Environment required : IBM compatible system
     with color monitor.

  Program structure :
     this program makes extensive use of the unit
     structure defined by Turbo pascal.  All of
     the procedures, functions, and variables of
     the program are contained in the units that
     are part of this program.

  Units :
    GLOBAL - this unit contains the declarations
      for all of the files, records and variables
      used for the game itself.
    GAME - this is the unit responsible for the
      overall control of the game: its
      initialization, modification of parameters,
      execution of the game, and issuing of
      commands.
    INIT - contains the procedures that initial-
      ize the game and allow the user to change
      any of the games parameters.
    SCENARIO - contains the procedures that
      initialize, modify, and create the scenario
      for the game.
    TIMESTEP - this unit contains the procedures
      that comprise the "heart" of the game, it
      actually executes each time step and
      processes the event records for the game.
    COMMANDS - this unit contains the procedures
```

```
      that allow the user to issue commands that
      allow the command and control of an
      artillery battalion to take place.
    GAMEUTIL - this unit contains a number of
      general purpose utility procedures and
      functions that are used by more than one
      unit and are specific to the game.
    UTILITY - this unit contains a library of
      general purpose procedures that are
      specifically for the generation of the user
      interface, both input and output.
    TACCESS - this a Borland written unit that
      contains a number of low level procedures
      that provide database related functions
      used throughout the game.
    TAHIGH - this a Borland written unit that
      contains a number of high level procedures
      that provide database related functions
      used throughout the game.
                                                  }
(************************************************)

program wargame;
{$m 65520,0,655360}

uses crt, utility, game;

var choice : integer;

begin
initialize_screen;
remove_cursor;
repeat
   center_text (1, 'The Artillery Wargame !', blue);
   choice := menu_selection ( 'Main  Menu',
                              'Start a new game|Continue an
old game|Quit\' );
   case choice of
     1 : play_new_game;
     2 : play_old_game
   end
until choice = 3;
restore_cursor;
initialize_screen
end.

^Z
```

```
(************************************************)
{
 Unit name : GLOBAL

 Purpose : this unit contains all of the declara-
   tions necessary for the game itself.  It con-
   tains declarations for all of the files used
   to save the game variables from game to game
   on disk.  It also contains default values for
   records wherever they were possible.  The
   specific purpose of each variable is clearly
   indicated by its name.
                                                 }
(************************************************)
unit global;


interface

uses dos, utility, taccess, tahigh;

const
   scenario_file_header = 'This is a valid wargame scenario
file !';
   max_firing_units = 6;
   max_ammo_trucks  = 24;
   trains_filename              = 'trains.dat';
   firing_unit_filename         = 'fireunit.dat';
   ammo_truck_filename          = 'ammotrck.dat';
   cdrs_guidance_filename       = 'cdrguide.dat';
   game_parameter_filename      = 'gamparam.dat';
   node_data_filename_ext       = '.sc1';
   path_data_filename_ext       = '.sc2';
   node_index_filename_ext      = '.sc3';
   path_index_filename_ext      = '.sc4';
   event_data_filename          = 'evnt$lst.dat';
   event_time_index_filename    = 'evnt$lst.ix1';
   event_serial_index_filename = 'evnt$lst.ix2';
   message_data_filename        = 'msg$list.dat';
   message_type_index_filename = 'msg$list.idx';

{$I global.typ}
type
   firing_unit_record = record
      record_status                  : longint;
      firing_unit_name               : string5;
      number_of_guns                 : integer;
      location                       : string10;
      section_max_rounds_capacity    : integer;
      rounds_on_hand                 : integer;
      sustained_rate_of_fire         : integer;
```

```
    time_in_position                : longint;
    rounds_fired_from_position      : integer;
    vulnerability_status            : char;
    firing_status                   : char;
    ammo_status                     : char;
    support_mission                 : string3;
    rate_percent_csr                : real;
    ammo_pickup_mission             : boolean;
    ammo_pickup_location            : string10;
    traverse_minefield              : boolean;
    minefield_location              : string10;
    pending_movement                : boolean;
    ammo_low                        : boolean;
    ammo_critical                   : boolean;
    ammo_out                        : boolean;
    vulnerability_high              : boolean;
    vulnerability_critical          : boolean;
    sections_in_operating_condition : real;
    total_availability_time         : longint;
    critically_short_time           : longint;
    critically_vulnerable_time      : longint
    end;
ammo_truck_record = record
    record_status           : longint;
    bumper_number           : string5;
    location                : string10;
    load_status             : char;
    ammo_capacity           : integer;
    convoy_name             : string5;
    mission_assigned        : char;
    moving                  : boolean;
    firing_unit_to_resupply : string5;
    node_to_resupply        : string5;
    vulnerability_status    : char;
    amount_of_rest          : longint;
    time_since_rest_began   : longint;
    traverse_minefield      : boolean;
    minefield_location      : string10;
    pending_movement        : boolean;
    effective_percent       : real;
    killed                  : boolean;
    casualty_time           : longint
    end;
battalion_trains_record = record
    record_status           : longint;
    location                : string10;
    moving                  : boolean;
    pending_movement        : boolean;
    time_in_position        : longint;
    vulnerability_status    : char;
    vulnerability_high      : boolean;
```

```
      vulnerability_critical : boolean
      end;
   cdr_guidance_record = record
      record_status                    : longint;
      unit_sitrep_frequency            : integer;
      vulnerability_threshold_time     : real;
      vulnerability_threshold_rounds   : integer;
      crew_rest_per_day                : real;
      bn_csr                           : integer;
      bmnt                             : string5;
      eent                             : string5;
      maneuver_mission                 : char;
      axis                             : integer
      end;
   game_param_record = record
      record_status          : longint;
      time_step_size         : integer;
      avg_track_convoy_speed : integer;
      avg_wheel_convoy_speed : integer;
      avg_time_trains_to_atp : integer
      end;
   scenario_info_record = record
      record_status  : longint;
      opord_number   : string10;
      opord_date     : string15;
      map_sheets     : string80;
      start_dtg      : string15;
      admin_notes    : array [1..10] of string60
      end;
{  node_record = record
      record_status     : longint;
      node_name         : string5;
      paths             : array [1..6] of string5;
      grid              : string10;
      position_type     : char;
      cover_concealment : char;
      ammo_count        : integer
      end;
   path_record = record
      record_status  : longint;
      path_name      : string5;
      start_node     : string5;
      end_node       : string5;
      length         : real;
      road_condition : char;
      bridge         : char;
      bridge_grid    : string10;
      vulnerability  : char
      end;
   event_record = record
      record_status : longint;
```

```
      event_type     : char;
      serial_number  : integer;
      time_key       : string10;
      unit_type      : char;
      unit_name      : string5;
      volleys        : integer;
      grid           : string10;
      node           : string5;
      path           : string5
      end;
   message_record = record
      record_status  : longint;
      message_type   : char;
      unit_type      : char;
      unit_name      : string5;
      location       : string10;
      ammo_status    : char;
      vulnerability  : char
      end; }
   stat_record = record
      availability            : real;
      max_availability        : real;
      availability_ratio      : real;
      ammo_short_time         : real;
      max_ammo_short_time     : real;
      ammo_short_time_ratio   : real;
      vulnerability           : real;
      max_vulnerability       : real;
      vulnerability_ratio     : real;
      truck_casualties        : real;
      max_truck_casualties    : real;
      truck_casualties_ratio  : real;
      moe                     : real;
      rounds_on_hand          : integer;
      total_guns_at_start     : integer
      end;
firing_unit_array   =   array   [1..max_firing_units]   of
firing_unit_record;
ammo_truck_array    =   array   [1..max_ammo_trucks]    of
ammo_truck_record;

var
   game_stats : stat_record;
   scenario_file_name    : string80;
   scenario_file         : text;
   node_data_filename    : string80;
   path_data_filename    : string80;
   node_index_filename   : string80;
   path_index_filename   : string80;
   trains_file    : file of battalion_trains_record;
   fireunit_file  : file of firing_unit_record;
```

```
   ammotrck_file : file of ammo_truck_record;
   cdrguide_file : file of cdr_guidance_record;
   gamparam_file : file of game_param_record;

   number_of_firing_units : integer;
   number_of_ammo_trucks  : integer;
   firing_units           : firing_unit_array;
   ammo_trucks            : ammo_truck_array;
   battalion_trains       : battalion_trains_record;
   commanders_guidance    : cdr_guidance_record;
   game_parameters        : game_param_record;
   scenario_info          : scenario_info_record;
   nodes                  : dataset;
   paths                  : dataset;
   event_list             : datafile;
   time_index             : indexfile;
   serial_number_index    : indexfile;
   messages               : datafile;
   message_type_index     : indexfile;

   game_start_time : datetime;
   game_start_dtg  : string15;
   game_time : datetime;
   game_dtg  : string15;
   total_game_time : longint;
   new_day : boolean;
   day_time : boolean;
   time_since_last_sitrep : integer;
   atp_rounds_on_hand : integer;
   hostilities_started : boolean;
   command_serial_number : integer;

const
   default_firing_unit_data : firing_unit_record
      = ( record_status                    : 0;
          firing_unit_name                 :
null_string;
          number_of_guns                   : 8;
          location                         : '1';
          section_max_rounds_capacity      : 250;
          rounds_on_hand                   : 500;
          sustained_rate_of_fire           : 1;
          time_in_position                 : 0;
          rounds_fired_from_position       : 0;
          vulnerability_status             : 'A';
          firing_status                    : 'H';
          ammo_status                      : 'S';
          support_mission                  : 'DS';
          rate_percent_csr                 : 1.0;
          ammo_pickup_mission              : false;
```

```
            ammo_pickup_location                                :
null_string;
            traverse_minefield                      : false;
            minefield_location                                  :
null_string;
            pending_movement                        : false;
            ammo_low                                : false;
            ammo_critical                           : false;
            ammo_out                                : false;
            vulnerability_high                      : false;
            vulnerability_critical                  : false;
            sections_in_operating_condition         : 8.0;
            total_availability_time                 : 0;
            critically_short_time                   : 0;
            critically_vulnerable_time              : 0
          );
   default_ammo_truck_data  : ammo_truck_record
      = ( record_status             : 0;
          bumper_number             : null_string;
          location                  : '1';
          load_status               : 'F';
          ammo_capacity             : 300;
          convoy_name               : null_string;
          mission_assigned          : 'N';
          moving                    : false;
          firing_unit_to_resupply   : null_string;
          node_to_resupply          : null_string;
          vulnerability_status      : 'A';
          amount_of_rest            : 0;
          time_since_rest_began     : 0;
          traverse_minefield        : false;
          minefield_location        : null_string;
          pending_movement          : false;
          effective_percent         : 1.0;
          killed                    : false;
          casualty_time             : 0
        );
   default_battalion_trains : battalion_trains_record
      = ( record_status             : 0;
          location                  : '1';
          moving                    : false;
          pending_movement          : false;
          time_in_position          : 0;
          vulnerability_status      : 'A';
          vulnerability_high        : false;
          vulnerability_critical    : false
        );
   default_commanders_guidance : cdr_guidance_record
      = ( record_status                     : 0;
          unit_sitrep_frequency             : 60;
          vulnerability_threshold_time      : 6.5;
```

```
            vulnerability_threshold_rounds : 200;
            crew_rest_per_day              : 6.5;
            bn_csr                         : 100;
            bmnt                           : '0545Z';
            eent                           : '1820Z';
            maneuver_mission               : 'O';
            axis                           : 090
           );
   default_game_parameters : game_param_record
      = ( record_status            : 0;
          time_step_size           : 30;
          avg_track_convoy_speed   : 20;
          avg_wheel_convoy_speed   : 35;
          avg_time_trains_to_atp   : 180
        );
   default_scenario_info : scenario_info_record
      = ( record_status  : 0;
          opord_number   : '0001';
          opord_date     : '  ';
          map_sheets     : 'None applicable';
          start_dtg      : '15 0600Z NOV 88';
          admin_notes    : ('  ','  ','  ','  ','  ','  ',
                            '  ','  ','  ','  ','  ','  ')

        );
   default_node : node_record
      = ( record_status     : 0;
          node_name         : null_string;
          paths                : (null_string, null_string,
null_string,
                                  null_string,     null_string,
null_string);
          grid              : null_string;
          position_type     : 'R';
          cover_concealment : 'M';
          ammo_count        : 0
        );
   default_path : path_record
      = ( record_status  : 0;
          path_name      : null_string;
          start_node     : null_string;
          end_node       : null_string;
          length         : 0;
          road_condition : 'M';
          bridge         : 'N';
          bridge_grid    : null_string;
          vulnerability  : 'M'
        );
   default_scenario_file_name              :   string80   =
'new$scen.scn';
   default_number_of_firing_units : integer  = 6;
   default_number_of_ammo_trucks  : integer  = 24;
```

```
implementation

begin
end.

^Z
```

```
(************************************************)
{
 Unit name : GAME

 Purpose : this unit contains the procedures that
   actually comprise the heart of the wargame.
   It is responsible for initializing either a
   new or old game, running the game, and taking
   care of the functions necessary when quitting.
   As part of the game itself it also controls
   execution of the time step, the issuing of
   commands, and the changing of allowable game
   parameters for the execution of the game.
   This unit can be thought of as having four
   subordinate units that contains the procedures
   that carry out its functions.  These are INIT,
   SCENARIO, TIMESTEP, and COMMANDS.
                                                 }
(************************************************)
unit game;

interface
{$I-}

uses crt, printer, utility, gameutil, global, init,
     scenario, timestep, commands, taccess, tahigh;

procedure play_new_game;
procedure play_old_game;

implementation
(************************************************)
{
 Procedure name : PLAY_WARGAME

 Purpose : this procedure controls the play of
   wargame once it has been initialized as
   either a new or old game, as appropriate.
   It displays the game screen, and provides the
   game menu.  It relies on four units: INIT,
   SCENARIO, TIMESTEP, and COMMANDS.  These are
   called as is necessary to handle the selection
   made from the game menu by the player.

 Parameters : none.

 Called by : PLAY_NEW_GAME
             PLAY_OLD_GAME
                                                 }
(************************************************)
procedure play_wargame;
```

```
var
   choice : integer;

(************************************************)
{
 Procedure name : DISPLAY_GAME_MAIN_SCREEN

 Purpose : this procedure displays the current
   game time in large block letters at the top
   of the screen and the current dtg in normal
   size characters on the top line of the screen.

 Parameters : none.

 Called by : PLAY_WARGAME
             EXECUTE_NEXT_TIME_STEP
                                                }
(************************************************)
procedure display_game_main_screen;

var
   dtg : string15;

begin
initialize_screen;
dtg := remove_blanks (game_dtg);
put_font_string (17, 3, copy (dtg, 3, 4) + blank + copy
(dtg, 7, 1), cyan);
center_text (1, 'DTG :  '+ game_dtg, yellow)
end;

procedure execute_next_time_step;

begin
inc_time (game_time, game_parameters.time_step_size);
datetime_to_dtg (game_time, game_dtg);
total_game_time        :=        total_game_time        +
game_parameters.time_step_size;
new_day := ((game_time.hour * 60) + game_time.min) <
   game_parameters.time_step_size;
determine_day_or_night;
display_game_main_screen;
process_events_list;
stock_atp;
process_field_trains;
process_ammo_trucks;
process_fire_units;
generate_messages
end;
```

```
(************************************************)
{
 Procedure name : ISSUE_COMMAND

 Purpose : this procedure presents the command
   menu to the player.  It is responsible for
   calling the appropriate procedure to handle
   the command selected by the user.

 Parameters : none.

 Called by : PLAY_WARGAME
                                                 }
(************************************************)
procedure issue_command;

begin
clear_area (1,2,80,25);
draw_window   (1,2,80,23,   menu_forecolor,   menu_backcolor,
'Issue Commands');
repeat
   menu_x1 := 12;  menu_y1 := 5;
   menu_x2 := 68;  menu_y2 := 19;
   choice  :=  menu_selection  (null_string,  'Create  truck
convoy|'+
                                          'Remove      truck
convoy|'+
                                          'Move a unit|'+
                                          'Request sitrep|'+

                                          'Change     firing
rate|'+
                                          'Issue        fire
order|'+
                                          'Cancel command|'+

                                          'Ammo     resupply
mission|'+
                                          'Cancel   resupply
mission|'+
                                          'Fire  unit  ammo
pickup|'+
                                          'Cancel fire unit
pickup|'+
                                          'Ammo  truck  ammo
pickup|'+
                                          'RETURN        TO
GAME\');
   menu_x1 := menu_x1_default;  menu_y1 := menu_y1_default;
   menu_x2 := menu_x2_default;  menu_y2 := menu_y2_default;
   case choice of
```

```
            1 : create_truck_convoy;
            2 : remove_truck_convoy;
            3 : move_unit;
            4 : display_sitrep;
            5 : change_firing_rate;
            6 : issue_fire_order;
            7 : cancel_command;
            8 : ammo_resupply_mission;
            9 : cancel_resupply_mission;
           10 : fire_unit_ammo_pickup;
           11 : cancel_fire_unit_pickup;
           12 : ammo_truck_ammo_pickup
      end
   until choice = 13;
   initialize_screen
   end;

   (************************************************)
   {
    Procedure name : CHANGE_PARAMETERS

    Purpose : this procedure presents the menu for
      allowing the player to change the commander's
      guidance, the game parameters, modify the
      network, or access the network utility.

    Parameters : none.

    Called by : PLAY_WARGAME
                                                    }
   (************************************************)
   procedure change_parameters;

   var
      choice : integer;

   begin
   clear_area (1,2,80,25);
   repeat
      choice := menu_selection ('Scenario / Parameter Menu',
                                'View or change scenario /
   network|'+
                                'View or change game
   parameters|'+
                                'View or change commander''s
   guidance|'+
                                'Return to game\');
      case choice of
         1 : view_scenario;
         2 : view_game_parameters;
         3 : view_commanders_guidance
```

```
   end
until choice = 4;
initialize_screen
end;

(*************************************************)
{
 Procedure name : QUIT_GAME

 Purpose : this procedure handles the actions
   neccessary for quiting the game.  It presents
   the player with the option to save the game in
   progress or abandon it.  It then calculates
   and outputs the statistics for the game.

 Parameters : none.

 Called by : PLAY_WARGAME
                                                 }
(*************************************************)
procedure quit_game;

var
   choice : integer;

(*************************************************)
{
 Procedure name : SAVE_GAME

 Purpose : this procedure saves the current game
   so that it can be continued at a later time.
   It saves all variables in files on disk so
   that can be retrieved by INITIALIZE_OLD_GAME.

 Parameters : none.

 Called by : QUIT_GAME
                                                 }
(*************************************************)
procedure save_game;

begin
end;

(*************************************************)
{
 Procedure name : OUTPUT_STATISTICS

 Purpose : this procedure calculates and prints
   on a printer the statistics for the play of
   the game.
```

```
 Parameters : none.

 Called by : QUIT_GAME
                                                    }
(*************************************************)
procedure output_statistics;

(*************************************************)
{
 Procedure name : CALCULATE_STATS

 Purpose : this procedure calculates the
   statistics for the game.

 Parameters : none.

 Called by : OUTPUT_STATISTICS
                                                    }
(*************************************************)
procedure calculate_stats;

var
    [illegible]eger;
    [illegible]ode : node_record;
    [illegible]name : string10;

begin
with game_stats do
   begin
   availability := 0.0;
   ammo_short_time := 0.0;
   vulnerability := 0.0;
   rounds_on_hand := 0;
   total_guns_at_start := 0;
   for i := 1 to number_of_firing_units do
      begin
      availability := availability +
         (firing_units[i].total_availability_time / 60);
      ammo_short_time := ammo_short_time +
         (firing_units[i].critically_short_time / 60);
      vulnerability := vulnerability +
         (firing_units[i].critically_vulnerable_time / 60);
      rounds_on_hand := rounds_on_hand +
         firing_units[i].rounds_on_hand;
      total_guns_at_start := total_guns_at_start +
         firing_units[i].number_of_guns
      end;
   max_availability := total_game_time * total_guns_at_start
/ 60;
   availability_ratio := availability / max_availability;
```

```
   max_ammo_short_time       :=       total_game_time       *
total_guns_at_start / 60;
   ammo_short_time_ratio      :=      ammo_short_time       /
max_ammo_short_time;
   max_vulnerability       :=       total_game_time       *
total_guns_at_start / 60;
   vulnerability_ratio := vulnerability / max_vulnerability;

   truck_casualties := 0.0;
   for i := 1 to number_of_ammo_trucks do
      begin
      truck_casualties := truck_casualties +
         (ammo_trucks[i].casualty_time / 60);
      if ammo_trucks[i].load_status = 'F' then
         rounds_on_hand := rounds_on_hand +
            round (ammo_trucks[i].ammo_capacity *
                   ammo_trucks[i].effective_percent)
      end;
   max_truck_casualties      :=      total_game_time       *
number_of_ammo_trucks / 60;
   truck_casualties_ratio     :=     truck_casualties      /
max_truck_casualties;
   moe := availability_ratio - ammo_short_time_ratio -
          vulnerability_ratio - truck_casualties_ratio;
   tareset (nodes);
   tanext (nodes, temp_node, node_name);
   while ok do
      begin
      rounds_on_hand       :=       rounds_on_hand       +
temp_node.ammo_count;
      tanext (nodes, temp_node, node_name)
      end;
   rounds_on_hand        :=        rounds_on_hand        -
commanders_guidance.bn_csr;
   if rounds_on_hand < 0 then
      rounds_on_hand := 0
   end
end;

(************************************************)
{
 Procedure name : PRINT_STATS

 Purpose : this procedure prints the statistics
   for the game on the printer.

 Parameters : none.

 Called by : OUTPUT_STATISTICS
                                                 }
(************************************************)
```

```
procedure print_stats;

const
   ff = #12;

begin
with game_stats do
begin
writeln (lst, ff);
writeln (lst);
writeln (lst);
writeln (lst);
writeln (lst);
writeln (lst);
writeln (lst);
write   (lst, 'Game start time : ');
writeln (lst, game_start_dtg);
write   (lst, 'Game end time   : ');
writeln (lst, game_dtg);
writeln (lst);
write    (lst, 'Total time firing sections were available
(hours) : ');
writeln (lst, availability:4:2);
write   (lst, 'Maximum possible availability time (hours) :
');
writeln (lst, max_availability:4:2);
write   (lst, 'Availability ratio (maximize) : ');
writeln (lst, availability_ratio:4:2);
writeln (lst);
write    (lst, 'Total time firing sections were critically
short ammo (hours) : ');
writeln (lst, ammo_short_time:4:2);
write      (lst, 'Maximum possible critically_short_time
(hours) : ');
writeln (lst, max_ammo_short_time:4:2);
write   (lst, 'Critically short ratio (minimize) : ');
writeln (lst, ammo_short_time_ratio:4:2);
writeln (lst);
write    (lst, 'Total time firing sections were critically
vulnerable (hours) : ');
writeln (lst, vulnerability:4:2);
write    (lst, 'Maximum possible critically vulnerable time
(hours) : ');
writeln (lst, max_vulnerability:4:2);
write   (lst, 'Critically vulnerable ratio (minimize) : ');
writeln (lst, vulnerability_ratio:4:2);
writeln (lst);
write    (lst, 'Total time lost due to truck casualties
(hours) : ');
writeln (lst, truck_casualties:4:2);
write   (lst, 'Maximum possible casualty time (hours) : ');
```

```
writeln (lst, max_truck_casualties:4:2);
write   (lst, 'Truck casualty time ratio (minimize) : ');
writeln (lst, truck_casualties_ratio:4:2);
writeln (lst);
writeln (lst, 'MOE (availability ratio - critically short
ratio -');
write   (lst, '     vulnerability ratio - casualty ratio)
   : ');
writeln (lst, moe:4:2);
writeln (lst);
write    (lst, 'Number of rounds on hand in excess of CSR
(minimize) : ');
writeln (lst, rounds_on_hand);
writeln (lst, ff)
end
end;

begin
if not printer_ready then
   begin
   save_screen;
   draw_window (21,12,60,15, yellow, red, null_string);
   center_text (13, 'Printer not ready', yellow);
   center_text (14, 'press any key', white);
   key := get_key;
   key := null;
   restore_screen
   end;
if total_game_time > 0 then
   begin
   calculate_stats;
   if printer_ready then
      print_stats
   end
end;

begin
choice  :=  menu_selection  ('Quit  Game',  'Save  game  in
progress|Abandon game\');
if choice = 1 then
   save_game;
close_all_files;
output_statistics
end;

begin
initialize_screen;
repeat
   display_game_main_screen;
   if not hostilities_started then
      check_for_start_of_hostilities;
```

```
      menu_x1 := 1;
      menu_y1 := 11;
      menu_x2 := 23;
      menu_y2 := 23;
      choice := menu_selection ('Game Menu', 'Next time step|'+
                                             'Issue command|'+
                                             ' C h a n g e
parameters|'+
                                             ' N e t w o r k
utility|'+
                                             'Quit game\');
      menu_x1 := menu_x1_default;
      menu_y1 := menu_y1_default;
      menu_x2 := menu_x2_default;
      menu_y2 := menu_y2_default;
      case choice of
         1 : execute_next_time_step;
         2 : issue_command;
         3 : change_parameters;
         4 : network_utility
      end
   until choice = 5;
   initialize_screen;
   quit_game
   end;

   (************************************************)
   {
    Procedure name : PLAY_NEW_GAME

    Purpose : this procedure initializes and plays
      a new game.

    Parameters : none.

    Called by : WARGAME
                                                    }
   (************************************************)
   procedure play_new_game;

   begin
   if initialize_new_game then
      play_wargame
   end;

   (************************************************)
   {
    Procedure name : PLAY_OLD_GAME

    Purpose : this procedure initializes and plays
```

```
   an old game.

 Parameters : none.

 Called by : WARGAME
                                                    }
(************************************************)
procedure play_old_game;

begin
if initialize_old_game then
   play_wargame
end;

begin
end.
^Z
```

```
(************************************************)
{
 Unit name : INIT

 Purpose : this unit contains the procedures that
   initialize all of the variables and files for
   both a new game or an old game to be continued
   if desired.  It also contains the procedures
   that allow the user to create or modify his
   battalion for the game.  This includes the
   trains, fire units, and ammo trucks.  The
   player can also modify the game parameters and
   commanders guidance with procedures in this
   unit.  It calls procedures in unit SCENARIO
   to also allow the user to create and modify a
   scenario complete with nodes and paths.
                                                 }
(************************************************)
unit init;

interface
{$I-}

uses dos, crt, utility, gameutil, global, scenario, taccess,
tahigh;

procedure view_game_parameters;
procedure view_commanders_guidance;
procedure view_battalion_configuration;
function  initialize_new_game : boolean;
function  initialize_old_game : boolean;

implementation

var
   firing_unit_number : integer;
   ammo_truck_number  : integer;

(************************************************)
{
 Procedure name : UNIQUE_FIRING_UNIT_NAME

 Purpose : this procedure insures that a firing
   unit that is being created or modified is
   assigned a unique name.  It checks the
   names of the other firing units, the ammo
   trucks, and the trains.

 Parameters : STRING_VALUE - name to be checked

 Called by : passed as a parameter to
```

```
   EDIT_SCREEN by UPDATE_FIRING_UNIT_DATA
                                                    }
(************************************************)
{$F+}
function unique_firing_unit_name (string_value : string80):
boolean;

var
   unique_name : boolean;
   unit_number : integer;

begin
unique_name := true;
if string_value <> null_string then
   begin
   for unit_number := 1 to number_of_firing_units do
      if (string_value =
firing_units[unit_number].firing_unit_name) and
         (unit_number <> firing_unit_number) then
            unique_name := false;
   if unique_name then
      unique_name := (truck_number (string_value) = 0) and
                     (string_value <> 'TRAIN')
   end
else
   unique_name := false;
if not unique_name then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'unit must have a unique name',
null_string);
unique_firing_unit_name := unique_name
end;
{$F-}

(************************************************)
{
 Procedure name : UNIQUE_BUMPER_NUMBER

 Purpose : this procedure insures that an ammo
   truck that is being created or modified is
   assigned a unique bumper number.  It checks
   the bumper numbers of the other trucks, the
   names of the firing units, and the trains.

 Parameters : STRING_VALUE - bumper number to be
   checked.

 Called by : passed as a parameter to
   EDIT_SCREEN by UPDATE_AMMO_TRUCK_DATA
                                                    }
```

```
(************************************************)
{$F+}
function unique_bumper_number (string_value : string80):
boolean;

var
   unique_name : boolean;
   truck_number : integer;

begin
unique_name := true;
if string_value <> null_string then
   begin
   for truck_number := 1 to number_of_ammo_trucks do
      if (string_value =
ammo_trucks[truck_number].bumper_number) and
         (truck_number <> ammo_truck_number) then
            unique_name := false;
   if unique_name then
      unique_name := (unit_number (string_value) = 0) and
                     (string_value <> 'TRAIN')
   end
else
   unique_name := false;
if not unique_name then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'truck must have a unique name',
null_string);
unique_bumper_number := unique_name
end;
{$F-}

(************************************************)
{
 Procedure name : SAVE_GAME_PARAMETERS_TO_DISK

 Purpose . this procedure saves the game
   parameter to disk so that it can be used in
   successive games to be played by the user
   to save him the trouble of having to tailor
   it each time.

 Parameters : none.

 Called by : VIEW_GAME_PARAMTERS
             READY_TO_PLAY
                                                 }
(************************************************)
procedure save_game_parameters_to_disk;
```

```
begin
save_screen;
draw_window (27,11,54,15, yellow, red, null_string);
shade_window (27,11,54,15, black);
center_text (13, 'updating files...', yellow);
assign (gamparam_file, game_parameter_filename);
rewrite (gamparam_file);
if ioresult = 0 then
   write (gamparam_file, game_parameters);
close (gamparam_file);
restore_screen
end;

(************************************************)
{
 Procedure name : SAVE_COMMANDERS_GUIDANCE_TO_DISK

 Purpose : this procedure saves the commander's
   guidance to disk so that it can be used in
   successive games to be played by the user
   to save him the trouble of having to tailor
   it each time.

 Parameters : none.

 Called by : VIEW_COMMANDERS_GUIDANCE
             READY_TO_PLAY
                                                 }
(************************************************)
procedure save_commanders_guidance_to_disk;

begin
save_screen;
draw_window (27,11,54,15, yellow, red, null_string);
shade_window (27,11,54,15, black);
center_text (13, 'updating files...', yellow);
assign (cdrguide_file, cdrs_guidance_filename);
rewrite (cdrguide_file);
if ioresult = 0 then
   write (cdrguide_file, commanders_guidance);
close (cdrguide_file);
restore_screen
end;

(************************************************)
{
 Procedure name : SAVE_UNIT_DATA_TO_DISK

 Purpose : this procedure saves the unit data
   to disk so that it can be used in
   successive games to be played by the user
```

```
   to save him the trouble of having to tailor
   it each time. It saves the firing unit data,
   ammo truck data, and battalion trains data.

 Parameters : none.

 Called by : VIEW_BATTALION_CONFIGURATION
             READY_TO_PLAY
                                                    }
(************************************************)
procedure save_unit_data_to_disk;

begin
save_screen;
draw_window (27,11,54,15, yellow, red, null_string);
shade_window (27,11,54,15, black);
center_text (13, 'updating files...', yellow);
assign (trains_file, trains_filename);
rewrite (trains_file);
if ioresult = 0 then
   write (trains_file, battalion_trains);
close (trains_file);
assign (fireunit_file, firing_unit_filename);
rewrite (fireunit_file);
if ioresult = 0 then
   for firing_unit_number := 1 to number_of_firing_units do
      write (fireunit_file,
firing_units[firing_unit_number]);
close (fireunit_file);
assign (ammotrck_file, ammo_truck_filename);
rewrite (ammotrck_file);
if ioresult = 0 then
   for ammo_truck_number := 1 to number_of_ammo_trucks do
      write (ammotrck_file, ammo_trucks[ammo_truck_number]);

close (ammotrck_file);
restore_screen
end;

(************************************************)
{
 Procedure name : VIEW_GAME_PARAMETERS

 Purpose : this procedure allows the player to
   view, change, or print the game parameters for
   the game.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME
             CHANGE_PARAMETERS
```

```
                                                              }
(*************************************************)
procedure view_game_parameters;

(*************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   battalion configuration data.

 Parameters : none.

 Called by : VIEW_GAME_PARAMETERS
                                                              }
(*************************************************)
procedure set_up_fields;

begin
number_of_fields := 4;
with field_list[1] do
   begin
   label_string := 'Time step (minutes) :';
   label_x := 15;  label_y := 10;
   int_value := game_parameters.time_step_size;
   str (int_value, str_val);
   x1 := 63;  y1 := 10;  x2 := 67;  y2 := 10;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end;
with field_list[2] do
   begin
   label_string := 'Average track convoy speed (kph) :';
   label_x := 15;  label_y := 12;
   int_value := game_parameters.avg_track_convoy_speed;
   str (int_value, str_val);
   x1 := 63;  y1 := 12;  x2 := 65;  y2 := 12;
   field_type := int;  int_min_value := 1;  int_max_value :=
120
   end;
with field_list[3] do
   begin
   label_string := 'Average wheel convoy speed (kph) :';
   label_x := 15;  label_y := 14;
   int_value := game_parameters.avg_wheel_convoy_speed;
   str (int_value, str_val);
   x1 := 63;  y1 := 14;  x2 := 65;  y2 := 14;
   field_type := int;  int_min_value := 1;  int_max_value :=
120
   end;
```

```
with field_list[4] do
   begin
   label_string := 'Average time from trains to ATP
(minutes) :';
   label_x := 15;  label_y := 16;
   int_value := game_parameters.avg_time_trains_to_atp;
   str (int_value, str_val);
   x1 := 63;  y1 := 16;  x2 := 65;  y2 := 16;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end
end;

begin
set_up_fields;
save_screen;
clear_area (1,2,80,25);
draw_window (1,3,80,23, white, blue, 'Game Parameters');
display_edit_screen_help_line;
edit_screen (number_of_fields, field_list, not
abort_allowed);
with game_parameters do
   begin
   time_step_size           := field_list[1].int_value;
   avg_track_convoy_speed   := field_list[2].int_value;
   avg_wheel_convoy_speed   := field_list[3].int_value;
   avg_time_trains_to_atp   := field_list[4].int_value
   end;
save_game_parameters_to_disk;
restore_screen
end;

(************************************************)
{
 Procedure name : VIEW_COMMANDERS_GUIDANCE

 Purpose : this procedure allows the player to
   view, change, or print the commanders guidance
   for the game.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME
             CHANGE_PARAMETERS
                                                  }
(************************************************)
procedure view_commanders_guidance;

(************************************************)
{
 Procedure name : SET_UP_FIELDS
```

```
 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   commanders guidance data.

 Parameters : none.

 Called by : VIEW_COMMANDERS_GUIDANCE
                                                    }
(*************************************************)
procedure set_up_fields;

begin
number_of_fields := 9;
with field_list[1] do
   begin
   label_string := 'Unit sitrep frequency (minutes) :';
   label_x := 14;  label_y := 5;
   int_value := commanders_guidance.unit_sitrep_frequency;
   str (int_value, str_val);
   x1 := 63;  y1 := 5;  x2 := 65;  y2 := 5;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end;
with field_list[2] do
   begin
   label_string := 'Vulnerability threshold (hours) :';
   label_x := 14;  label_y := 7;
   float_value :=
commanders_guidance.vulnerability_threshold_time;
   str (float_value:6:3, str_val);
   x1 := 63;  y1 := 7;  x2 := 68;  y2 := 7;
   field_type := float;  float_min_value := 0.0;
float_max_value := 100.0
   end;
with field_list[3] do
   begin
   label_string := 'Vulnerability threshold (rds / position)
:';
   label_x := 14;  label_y := 9;
   int_value :=
commanders_guidance.vulnerability_threshold_rounds;
   str (int_value, str_val);
   x1 := 63;  y1 := 9;  x2 := 65;  y2 := 9;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end;
with field_list[4] do
   begin
   label_string := 'Ammo truck crew rest (hrs / day) :';
   label_x := 14;  label_y := 11;
```

```
      float_value := commanders_guidance.crew_rest_per_day;
      str (float_value:6:3, str_val);
      x1 := 63;  y1 := 11;  x2 := 68;  y2 := 11;
      field_type := float;  float_min_value := 0.0;
   float_max_value := 24.0
      end;
   with field_list[5] do
      begin
      label_string := 'Battalion CSR :';
      label_x := 14;  label_y := 13;
      int_value := commanders_guidance.bn_csr;
      str (int_value, str_val);
      x1 := 63;  y1 := 13;  x2 := 66;  y2 := 13;
      field_type := int;  int_min_value := 1;  int_max_value :=
   maxint
      end;
   with field_list[6] do
      begin
      label_string := 'BMNT :';
      label_x := 14;  label_y := 15;
      str_val := commanders_guidance.bmnt;
      x1 := 63;  y1 := 15;  x2 := 67;  y2 := 15;
      field_type := time
      end;
   with field_list[7] do
      begin
      label_string := 'EENT :';
      label_x := 14;  label_y := 17;
      str_val := commanders_guidance.eent;
      x1 := 63;  y1 := 17;  x2 := 67;  y2 := 17;
      field_type := time
      end;
   with field_list[8] do
      begin
      label_string := 'Maneuver mission (Offense/Defense/No
   contact) :';
      label_x := 14;  label_y := 19;
      str_val := commanders_guidance.maneuver_mission;
      x1 := 63;  y1 := 19;  x2 := 63;  y2 := 19;
      field_type := ch;  valid_char_set := ['O','D','N'];
      end;
   with field_list[9] do
      begin
      label_string := 'Axis (bearing in degrees) :';
      label_x := 14;  label_y := 21;
      int_value := commanders_guidance.axis;
      str (int_value, str_val);
      x1 := 63;  y1 := 21;  x2 := 66;  y2 := 21;
      field_type := int;  int_min_value := 1;  int_max_value :=
   360
      end
```

```
end;

begin
set_up_fields;
save_screen;
clear_area (1,2,80,25);
draw_window (1,3,80,23, white, blue, 'Commander''s
Guidance');
display_edit_screen_help_line;
edit_screen (number_of_fields, field_list, not
abort_allowed);
with commanders_guidance do
   begin
   unit_sitrep_frequency          :=
field_list[1].int_value;
   vulnerability_threshold_time   :=
field_list[2].float_value;
   vulnerability_threshold_rounds :=
field_list[3].int_value;
   crew_rest_per_day              :=
field_list[4].float_value;
   bn_csr                         :=
field_list[5].int_value;
   bmnt                           := field_list[6].str_val;
   eent                           := field_list[7].str_val;
   maneuver_mission               :=
field_list[8].str_val[1];
   axis                           := field_list[9].int_value

   end;
save_commanders_guidance_to_disk;
restore_screen
end;

(************************************************)
{
 Procedure name : VIEW_BATTALION_CONFIGURATION

 Purpose : this procedure allows the player to
   view, change, or print the battalion
   configuration for the game.  It presents the
   player with a menu to select the trains,
   fire units, or ammo trucks.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME
                                                }
(************************************************)
procedure view_battalion_configuration;
```

```
var
   choice : integer;

(************************************************)
{
 Procedure name : UPDATE_TRAINS_LOCATION

 Purpose : this procedure allows the player to
   view, change, or print the location of the
   battalion trains for the start of the game.

 Parameters : none.

 Called by : VIEW_BATTALION_CONFIGURATION
                                                }
(************************************************)
procedure update_trains_location;

(************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   location of the battalion trains data.

 Parameters : none.

 Called by : UPDATE_TRAINS_LOCATION
                                                }
(************************************************)
procedure set_up_fields;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Trains location (node) :';
   label_x := 27;  label_y := 13;
   str_val := battalion_trains.location;
   x1 := 53;  y1 := 13;  x2 := 57;  y2 := 13;
   field_type := strg
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,4,79,22);
display_edit_screen_help_line;
```

```
edit_screen (number_of_fields, field_list, not
abort_allowed);
battalion_trains.location := upper_case
(field_list[1].str_val);
restore_screen
end;

(************************************************)
{
 Procedure name : UPDATE_NUMBER_OF_UNITS

 Purpose : this procedure allows the player to
   view, change, or print the number of ammo
   trucks and fire units to use for the play of
   the game.

 Parameters : none.

 Called by : VIEW_BATTALION_CONFIGURATION
                                                }
(************************************************)
procedure update_number_of_units;

(************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   the number of fire units and ammo trucks.

 Parameters : none.

 Called by : UPDATE_NUMBER_OF_UNITS
                                                }
(************************************************)
procedure set_up_fields;

begin
number_of_fields := 2;
with field_list[1] do
   begin
   label_string := 'Number of firing units :';
   label_x := 25;  label_y := 12;
   int_value := number_of_firing_units;
   str (int_value, str_val);
   x1 := 53;  y1 := 12;  x2 := 55;  y2 := 12;
   field_type := int;  int_min_value := 1;  int_max_value :=
max_firing_units
   end;
with field_list[2] do
```

```
   begin
   label_string := 'Number of ammo trucks :';
   label_x := 25;  label_y := 14;
   int_value := number_of_ammo_trucks;
   str (int_value, str_val);
   x1 := 53;  y1 := 14;  x2 := 55;  y2 := 14;
   field_type := int;  int_min_value := 1;  int_max_value :=
max_ammo_trucks
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,4,79,22);
display_edit_screen_help_line;
edit_screen (number_of_fields, field_list, not
abort_allowed);
number_of_firing_units := field_list[1].int_value;
number_of_ammo_trucks  := field_list[2].int_value;
restore_screen
end;

(************************************************)
{
 Procedure name : UPDATE_FIRING_UNIT_DATA

 Purpose : this procedure allows the player to
   view, change, or print the data that pertains
   to the fire units for the play of the game.

 Parameters : none.

 Called by : VIEW_BATTALION_CONFIGURATION
                                                 }
(************************************************)
procedure update_firing_unit_data;

(************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   data for the fire units.

 Parameters : none.

 Called by : UPDATE_FIRING_UNIT_DATA
                                                 }
(************************************************)
```

```
procedure set_up_fields;

begin
number_of_fields := 6;
with field_list[1] do
   begin
   label_string := 'Unit name :';
   label_x := 17;  label_y := 8;
   x1 := 57;  y1 := 8;  x2 := 61;  y2 := 8;
   field_type := eval;  eval_function :=
unique_firing_unit_name
   end;
with field_list[2] do
   begin
   label_string := 'Number of guns :';
   label_x := 17;  label_y := 10;
   x1 := 57;  y1 := 10;  x2 := 61;  y2 := 10;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end;
with field_list[3] do
   begin
   label_string := 'Location (node) :';
   label_x := 17;  label_y := 12;
   x1 := 57;  y1 := 12;  x2 := 61;  y2 := 12;
   field_type := strg
   end;
with field_list[4] do
   begin
   label_string := 'Firing section capacity (rounds) :';
   label_x := 17;  label_y := 14;
   x1 := 57;  y1 := 14;  x2 := 60;  y2 := 14;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end;
with field_list[5] do
   begin
   label_string := 'Rounds on hand for firing unit :';
   label_x := 17;  label_y := 16;
   x1 := 57;  y1 := 16;  x2 := 60;  y2 := 16;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
   end;
with field_list[6] do
   begin
   label_string := 'Sustained rate of fire (rounds/tube) :';

   label_x := 17;  label_y := 18;
   x1 := 57;  y1 := 18;  x2 := 60;  y2 := 18;
   field_type := int;  int_min_value := 1;  int_max_value :=
maxint
```

```
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,4,79,22);
display_edit_screen_help_line;
center_text (24, 'Page Up - next firing unit    '+
                 'Page Down - previous firing unit', red);
center_text (5, 'Firing unit data', cyan);
firing_unit_number := 1;
repeat
   save_screen;
   clear_area (2,6,79,22);
   with firing_units[firing_unit_number] do
      begin
      field_list[1].str_val := firing_unit_name;
      field_list[2].int_value := number_of_guns;
      str (field_list[2].int_value, field_list[2].str_val);
      field_list[3].str_val := location;
      field_list[4].int_value :=
section_max_rounds_capacity;
      str (field_list[4].int_value, field_list[4].str_val);
      field_list[5].int_value := rounds_on_hand;
      str (field_list[5].int_value, field_list[5].str_val);
      field_list[6].int_value := sustained_rate_of_fire;
      str (field_list[6].int_value, field_list[6].str_val);
      edit_screen (number_of_fields, field_list, not
abort_allowed);
      firing_unit_name                 := upper_case
(field_list[1].str_val);
      number_of_guns                   :=
field_list[2].int_value;
      sections_in_operating_condition := number_of_guns;
      location                         := upper_case
(field_list[3].str_val);
      section_max_rounds_capacity      :=
field_list[4].int_value;
      rounds_on_hand                   :=
field_list[5].int_value;
      sustained_rate_of_fire           :=
field_list[6].int_value
      end;
   if key = page_down then
      begin
      if firing_unit_number = 1 then
         firing_unit_number := number_of_firing_units
      else
         decr (firing_unit_number)
      end
```

```
   else if key = page_up then
      begin
      if firing_unit_number = number_of_firing_units then
         firing_unit_number := 1
      else
         incr (firing_unit_number)
      end;
   restore_screen
until key = escape;
restore_screen
end;

(************************************************)
{
 Procedure name : UPDATE_AMMO_TRUCK_DATA

 Purpose : this procedure allows the player to
   view, change, or print the data that pertains
   to the ammo trucks for the play of the game.

 Parameters : none.

 Called by : VIEW_BATTALION_CONFIGURATION
                                                 }
(************************************************)
procedure update_ammo_truck_data;

(************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   data for the ammo trucks.

 Parameters : none.

 Called by : UPDATE_AMMO_TRUCK_DATA
                                                 }
(************************************************)
procedure set_up_fields;

begin
number_of_fields := 4;
with field_list[1] do
   begin
   label_string := 'Bumper number :';
   label_x := 24;  label_y := 10;
   x1 := 54;  y1 := 10;  x2 := 58;  y2 := 10;
   field_type := eval;  eval_function :=
unique_bumper_number
```

```
      end;
   with field_list[2] do
      begin
      label_string := 'Location (node) :';
      label_x := 24;  label_y := 12;
      x1 := 54;  y1 := 12;  x2 := 58;  y2 := 12;
      field_type := strg
      end;
   with field_list[3] do
      begin
      label_string := 'Load status (Full / Empty) :';
      label_x := 24;  label_y := 14;
      x1 := 54;  y1 := 14;  x2 := 54;  y2 := 14;
      field_type := ch;  valid_char_set := ['F','E']
      end;
   with field_list[4] do
      begin
      label_string := 'Truck capacity (rounds) :';
      label_x := 24;  label_y := 16;
      x1 := 54;  y1 := 16;  x2 := 57;  y2 := 16;
      field_type := int;  int_min_value := 1;  int_max_value :=
maxint
      end
end;

begin
set_up_fields;
save_screen;
clear_area (2,4,79,22);
display_edit_screen_help_line;
center_text (24, 'Page Up - next ammo truck    '+
                 'Page Down - previous ammo truck', red);
center_text (5, 'Ammo truck data', cyan);
ammo_truck_number := 1;
repeat
   save_screen;
   clear_area (2,6,79,22);
   with ammo_trucks[ammo_truck_number] do
      begin
      field_list[1].str_val := bumper_number;
      field_list[2].str_val := location;
      field_list[3].str_val := load_status;
      field_list[4].int_value := ammo_capacity;
      str (field_list[4].int_value, field_list[4].str_val);
      edit_screen (number_of_fields, field_list, not
abort_allowed);
      bumper_number  :=  upper_case (field_list[1].str_val);

      location       :=  upper_case (field_list[2].str_val);

      load_status    :=  field_list[3].str_val[1];
```

```
      ammo_capacity  :=  field_list[4].int_value;
      end;
   if key = page_down then
      begin
      if ammo_truck_number = 1 then
         ammo_truck_number := number_of_ammo_trucks
      else
         decr (ammo_truck_number)
      end
   else if key = page_up then
      begin
      if ammo_truck_number = number_of_ammo_trucks then
         ammo_truck_number := 1
      else
         incr (ammo_truck_number)
      end;
   restore_screen
until key = escape;
restore_screen
end;

begin
save_screen;
clear_area (1,2,80,25);
draw_window (1,3,80,23, white, blue, 'Battalion
Configuration / Status');
menu_x1 := 20;
menu_y1 := 7;
menu_x2 := 60;
menu_y2 := 19;
repeat
   choice := menu_selection (null_string, 'Location of
trains|'+
                                          'Number of firing
units / trucks|'+
                                          'Firing unit
data|'+
                                          'Ammo truck
data|'+
                                          'Return\');
    case choice of
       1 : update_trains_location;
       2 : update_number_of_units;
       3 : update_firing_unit_data;
       4 : update_ammo_truck_data
    end
until choice = 5;
menu_x1 := menu_x1_default;
menu_y1 := menu_y1_default;
menu_x2 := menu_x2_default;
menu_y2 := menu_y2_default;
```

```
save_unit_data_to_disk;
restore_screen
end;

(************************************************)
{
 Procedure name : INTIALIZE_NEW_GAME

 Purpose : this procedure initializes all
   variables and files for the play of a new game.
   It also provides a menu for allowing the player
   to view, modify, or print any of the parameters,
   commanders guidance, unit data, or network.  It
   returns a false if the player elects to return
   to the game's main menu.

 Parameters : none.

 Called by : PLAY_NEW_GAME
                                                 }
(************************************************)
function initialize_new_game : boolean;

var
   choice : integer;

(************************************************)
{
 Procedure name : INTIALIZE_NEW_GAME_VARIABLES

 Purpose : this procedure initializes all
   variables and files for the play of a new game.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME
                                                 }
(************************************************)
procedure initialize_new_game_variables;

(************************************************)
{
 Procedure name : INTIALIZE_GAME_PARAMETERS

 Purpose : this procedure initializes the game
   parameters by reading them from disk if they
   exist from a previous game or by assigning
   default values from unit GLOBAL if a file
   does not exist from a previous game.

 Parameters : none.
```

```
 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                    }
(************************************************)
procedure initialize_game_parameters;

begin
game_parameters := default_game_parameters;
assign (gamparam_file, game_parameter_filename);
reset  (gamparam_file);
if ioresult = 0 then
   read (gamparam_file, game_parameters);
close (gamparam_file)
end;

(************************************************)
{
 Procedure name : INITIALIZE_COMMANDERS_GUIDANCE

 Purpose : this procedure initializes the
   commanders guidance by reading them from disk
   if they exist from a previous game or by
   assigning default values from unit GLOBAL if a
   file does not exist from a previous game.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                    }
(************************************************)
procedure initialize_commanders_guidance;

begin
commanders_guidance := default_commanders_guidance;
assign (cdrguide_file, cdrs_guidance_filename);
reset  (cdrguide_file);
if ioresult = 0 then
   read (cdrguide_file, commanders_guidance);
close (cdrguide_file)
end;

(************************************************)
{
 Procedure name : INITIALIZE_UNIT_DATA

 Purpose : this procedure initializes the unit
   data by reading them from disk if they exist
   from a previous game or by assigning default
   values from unit GLOBAL if a file does not
   exist from a previous game.
```

```
 Parameters : none.

 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                    }
(************************************************)
procedure initialize_unit_data;

begin
battalion_trains := default_battalion_trains;
for firing_unit_number := 1 to max_firing_units do
   firing_units[firing_unit_number] :=
default_firing_unit_data;
for ammo_truck_number := 1 to max_ammo_trucks do
   ammo_trucks[ammo_truck_number] :=
default_ammo_truck_data;
assign (trains_file, trains_filename);
reset (trains_file);
assign (fireunit_file, firing_unit_filename);
reset (fireunit_file);
assign (ammotrck_file, ammo_truck_filename);
reset (ammotrck_file);
if ioresult = 0 then
   begin
   read (trains_file, battalion_trains);
   if ioresult <> 0 then
      battalion_trains := default_battalion_trains;
   firing_unit_number := 0;
   while not (eof (fireunit_file)) and (ioresult = 0) and
         (firing_unit_number < max_firing_units) do
      begin
      incr (firing_unit_number);
      read (fireunit_file, firing_units[firing_unit_number])

      end;
   if (ioresult <> 0) and (firing_unit_number <> 0) then
      firing_units[firing_unit_number] :=
default_firing_unit_data;
   ammo_truck_number := 0;
   while not eof (ammotrck_file) and (ioresult = 0) and
         (ammo_truck_number < max_ammo_trucks) do
      begin
      incr (ammo_truck_number);
      read (ammotrck_file, ammo_trucks[ammo_truck_number])
      end;
   if (ioresult <> 0) and (ammo_truck_number <> 0) then
      ammo_trucks[ammo_truck_number] :=
default_ammo_truck_data;
   end;
close (trains_file);
close (fireunit_file);
close (ammotrck_file)
```

```
end;

(************************************************)
{
 Procedure name : INITIALIZE_EVENTS_LIST

 Purpose : this procedure initializes the files
   that will be used to maintain the events list
   throughout the game.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                }
(************************************************)
procedure initialize_event_list;

begin
makefile  (event_list, event_data_filename,  sizeof
(event_record));
makeindex (time_index, event_time_index_filename,
           sizeof (string10) - 1, duplicates);
makeindex (serial_number_index, event_serial_index_filename,

           sizeof (string5) - 1, duplicates)
end;

(************************************************)
{
 Procedure name : INITIALIZE_MESSAGE_LIST

 Purpose : this procedure initializes the files
   that will be used to maintain the message
   buffer throughout the game.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                }
(************************************************)
procedure initialize_message_list;

begin
makefile  (messages, message_data_filename,  sizeof
(message_record));
makeindex (message_type_index, message_type_index_filename,
           sizeof (char), duplicates)
end;

begin
save_screen;
```

```
draw_window  (26,11,55,15, yellow, red, null_string);
center_text (13, 'initializing game...', yellow);
repeat
   scenario_file_name :=
      get_file ('Enter scenario file name',
                'Hit ESC key to build a new scenario',
                '*.scn')
until (scenario_file_name = null_string) or
(scenario_file_valid);
if scenario_file_name = null_string then
   begin
   scenario_file_name := get_new_scenario_filename;
   initialize_new_scenario
   end;
game_start_dtg  := scenario_info.start_dtg;
dtg_to_datetime (game_start_dtg, game_start_time);
game_dtg  := game_start_dtg;
game_time := game_start_time;
number_of_firing_units := default_number_of_firing_units;
number_of_ammo_trucks  := default_number_of_ammo_trucks;
initialize_game_parameters;
initialize_commanders_guidance;
initialize_unit_data;
initialize_event_list;
initialize_message_list;
total_game_time := 0;
new_day := false;
determine_day_or_night;
time_since_last_sitrep := 0;
hostilities_started := false;
command_serial_number := 0;
randomize;
restore_screen
end;

(************************************************)
{
 Procedure name : READY_TO_PLAY

 Purpose : this procedure insures that the game
   is ready to be played after initialization by
   checking to make sure that all units are
   placed at an existing node for game start.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME
                                                 }
(************************************************)
function  ready_to_play : boolean;
```

```
var
   ready : boolean;
   temp_node : node_record;

begin
atp_rounds_on_hand := 0;
for firing_unit_number := 1 to number_of_firing_units do
   atp_rounds_on_hand := atp_rounds_on_hand +
      firing_units[firing_unit_number].number_of_guns;
atp_rounds_on_hand := atp_rounds_on_hand *
commanders_guidance.bn_csr;
save_game_parameters_to_disk;
save_commanders_guidance_to_disk;
save_unit_data_to_disk;
ready := true;
for firing_unit_number := 1 to number_of_firing_units do
   begin
   taread (nodes, temp_node,
           firing_units[firing_unit_number].location,
exactmatch);
   if not ok then ready := false
   end;
if ready then
   for ammo_truck_number := 1 to number_of_ammo_trucks do
      begin
      taread (nodes, temp_node,
              ammo_trucks[ammo_truck_number].location,
exactmatch);
      if not ok then ready := false
      end;
if ready then
   begin
   taread (nodes, temp_node, battalion_trains.location,
exactmatch);
   if not ok then ready := false
   end;
if not ready then
   display_error_message ('INITIALIZATION ERROR',
                          'firing units, trucks, and
trains',
                          'must start game at valid
locations',
                           null_string,
                          'correct this before continuing');

ready_to_play := ready
end;

begin
initialize_new_game_variables;
repeat
```

```
      choice := menu_selection ('Scenario / Parameter Menu',
                                'View or change scenario /
network|'+
                                'View or change game
parameters|'+
                                'View or change commander''s
guidance|'+
                                'View or change battalion
configuration / status|'+
                                'Play game|'+
                                'Main menu\');
      case choice of
         1 : view_scenario;
         2 : view_game_parameters;
         3 : view_commanders_guidance;
         4 : view_battalion_configuration
      end
   until ((choice = 5) and ready_to_play) or (choice = 6);
   if choice = 6 then
      close_all_files;
   initialize_new_game := choice = 5
   end;

   (************************************************)
   {
    Procedure name : INTIALIZE_OLD_GAME

    Purpose : this procedure initializes all
      variables and files for the play of an old game
      that was saved in progress.  The procedure
      automatically loads the data from disk if
      there is a game file there to be loaded.  It
      returns a false value if it could not load the
      game successfully.

    Parameters : none.

    Called by : PLAY_OLD_GAME
                                                    }
   (************************************************)
   function initialize_old_game : boolean;

   begin
   initialize_old_game := false
   end;

   begin
   end.
   ^Z
```

```
(************************************************)
{
 Unit name : SCENARIO

 Purpose : this unit contains the procedures that
   allow a player to create, modify, view, or
   print a scenario file.  This includes the
   scenario header, the node data, and the path
   data.  It also reads and saves scenario files
   from and to disk.
                                                 }
(************************************************)
unit scenario;

interface
{$I-}

uses dos, crt, utility, gameutil, global, taccess, tahigh;

function  scenario_file_valid : boolean;
function  get_new_scenario_filename : string;
procedure initialize_new_scenario;
procedure view_scenario;
procedure network_utility;

implementation

const
   add_record = true;
   edit_record = false;

(************************************************)
{
 Procedure name : SAVE_SCENARIO_TO_DISK

 Purpose : this procedure saves the scenario
 header information to disk.

 Parameters : none.

 Called by : VIEW_SCENARIO
                                                 }
(************************************************)
procedure save_scenario_to_disk;

var
   i : integer;

begin
save_screen;
draw_window (26,11,55,15, yellow, red, null_string);
```

```
shade_window (26,11,55,15, black);
center_text (13, 'updating files...', yellow);
assign (scenario_file, scenario_file_name);
rewrite (scenario_file);
if ioresult = 0 then
   with scenario_info do
      begin
      writeln (scenario_file, scenario_file_header);
      writeln (scenario_file, opord_number);
      writeln (scenario_file, opord_date);
      writeln (scenario_file, map_sheets);
      writeln (scenario_file, start_dtg);
      for i := 1 to 10 do
         writeln (scenario_file, admin_notes[i])
      end;
close (scenario_file);
restore_screen
end;

(************************************************)
{
 Procedure name : VALID_SCENARIO_FILENAME

 Purpose : this procedure insures that a
   file name entered for the new scenario is a
   valid and not previously used name.

 Parameters : STRING_VALUE - filen name to be
   checked.

 Called by : passed as a parameter to
   EDIT_SCREEN by GET_NEW_SCENARIO_FILENAME
                                                 }
(************************************************)
{$F+}
function  valid_scenario_filename (string_value : string80)
: boolean;

var
   temp_file : file;
   valid_filename : boolean;

begin
valid_filename := false;
assign (temp_file, string_value);
reset (temp_file);
if ioresult = 2 then
   begin
   rewrite (temp_file);
   if ioresult <> 2 then
      begin
```

```
      valid_filename := true;
      erase (temp_file)
      end
   end;
if not valid_filename then
   display_error_message ('FILE ERROR', null_string,
                          'filename entered is not valid',
                          'or already exists',
                           null_string);
close (temp_file);
valid_scenario_filename := valid_filename
end;
{$F-}

(************************************************)
{
 Procedure name : SCENARIO_FILE_VALID

 Purpose : this procedure checks a selected
 scenario file name to insure that the file
 contains a valid scenario that can be loaded.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                 }
(************************************************)
function scenario_file_valid : boolean;

var
   file_header : string80;
   i : integer;
   scenario_valid : boolean;
   dir  : dirstr;
   name : namestr;
   ext  : extstr;

begin
scenario_valid := false;
assign (scenario_file, scenario_file_name);
reset  (scenario_file);
if ioresult = 0 then
   begin
   readln (scenario_file, file_header);
   if file_header = scenario_file_header then
      begin
      with scenario_info do
         begin
         readln (scenario_file, opord_number);
         readln (scenario_file, opord_date);
         readln (scenario_file, map_sheets);
```

```
            readln (scenario_file, start_dtg);
            for i := 1 to 10 do
               readln (scenario_file, admin_notes[i])
            end;
         if (ioresult = 0) and (valid_dtg
(scenario_info.start_dtg)) then
            begin
            fsplit (scenario_file_name, dir, name, ext);
            node_data_filename  := dir + name +
node_data_filename_ext;
            path_data_filename  := dir + name +
path_data_filename_ext;
            node_index_filename := dir + name +
node_index_filename_ext;
            path_index_filename := dir + name +
path_index_filename_ext;
            taopen (nodes, node_data_filename,  sizeof
(node_record),
                           node_index_filename, sizeof
(string_5)    );
            if ok then
               begin
               taopen (paths, path_data_filename,  sizeof
(path_record),
                              path_index_filename, sizeof
(string_5)    );
               if ok then
                  scenario_valid := true
               end
            end
         end
      end;
   if not scenario_valid then
      display_error_message (null_string, scenario_file_name,
                             'INVALID SCENARIO FILE',
                             'check it before continuing',
                             'or try another');
   scenario_file_valid := scenario_valid;
   close  (scenario_file)
   end;

   (************************************************)
   {
    Procedure name : GET_NEW_SCENARIO_FILENAME

    Purpose : this procedure prompts the player to
    enter a name for the new scenario that he would
    like to create.

    Parameters : none.
```

```
 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                    }
(*************************************************)
function  get_new_scenario_filename : string;

(*************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   prompt for entering a file name.

 Parameters : none.

 Called by : GET_NEW_SCENARIO_FILENAME
                                                    }
(*************************************************)
procedure set_up_fields;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'File name for new scenario :';
   label_x := 10;  label_y := 13;
   str_val := default_scenario_file_name;
   x1 := 40;  y1 := 13;  x2 := 70;  y2 := 13;
   field_type := eval; eval_function :=
valid_scenario_filename;
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,4,79,22);
draw_window (1,3,80,23, white, blue, 'New Scenario');
display_edit_screen_help_line;
edit_screen (number_of_fields, field_list, not
abort_allowed);
get_new_scenario_filename := upper_case
(field_list[1].str_val);
restore_screen
end;

(*************************************************)
{
 Procedure name : INITIALIZE_NEW_SCENARIO

 Purpose : this procedure initializes the new
```

```
   scenario by assigning a default header to the
   scenario and creating the files to hold the
   node and path data.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME_VARIABLES
                                                    }
(*************************************************)
procedure initialize_new_scenario;

var
   dir  : dirstr;
   name : namestr;
   ext  : extstr;

begin
scenario_info := default_scenario_info;
save_scenario_to_disk;
fsplit (scenario_file_name, dir, name, ext);
node_data_filename  := dir + name + node_data_filename_ext;
path_data_filename  := dir + name + path_data_filename_ext;
node_index_filename := dir + name + node_index_filename_ext;

path_index_filename := dir + name + path_index_filename_ext;

tacreate (nodes, node_data_filename,  sizeof (node_record),
                 node_index_filename, sizeof (string_5)
);
tacreate (paths, path_data_filename,  sizeof (path_record),
                 path_index_filename, sizeof (string_5)    )

end;

(*************************************************)
{
 Procedure name : NETWORK_UTILITY

 Purpose : this procedure allows the user to
   get information about the network he has
   created.  It includes determining min time and
   min distance routes between selected nodes.

 Parameters : none.

 Called by : VIEW_SCENARIO
             PLAY_WARGAME
                                                    }
(*************************************************)
procedure network_utility;
```

```
begin
end;

(************************************************)
{
 Procedure name : VIEW_SCENARIO

 Purpose : this procedure allows the player to
   view, change, or print the game scenario for
   the game.  It presents the player with a menu
   to select the scenario header, the node data,
   or the path data.  It also allows him to use
   the network utility from here.

 Parameters : none.

 Called by : INITIALIZE_NEW_GAME
             CHANGE_PARAMETERS
                                                 }
(************************************************)
procedure view_scenario;

var
   choice : integer;

(************************************************)
{
 Procedure name : UPDATE_SCENARIO_INFORMATION

 Purpose : this procedure allows the player to
   view, change, or print the scenario header
   information for the current scenario.

 Parameters : none.

 Called by : VIEW_SCENARIO
                                                 }
(************************************************)
procedure update_scenario_information;

(************************************************)
{
 Procedure name : SET_UP_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   scenario header information.

 Parameters : none.

 Called by : UPDATE_SCENARIO_INFORMATION
```

```
                                                    )
(*************************************************)
procedure set_up_fields;

begin
number_of_fields := 14;
with field_list[1] do
   begin
   label_string := 'OPORD # :';
   label_x := 11;  label_y := 6;
   str_val := scenario_info.opord_number;
   x1 := 22;  y1 := 6;  x2 := 31;  y2 := 6;
   field_type := strg
   end;
with field_list[2] do
   begin
   label_string := 'OPORD Date :';
   label_x := 11;  label_y := 7;
   str_val := scenario_info.opord_date;
   x1 := 25;  y1 := 7;  x2 := 39;  y2 := 7;
   field_type := strg
   end;
with field_list[3] do
   begin
   label_string := 'Map sheets :';
   label_x := 11;  label_y := 8;
   str_val := scenario_info.map_sheets;
   x1 := 25;  y1 := 8;  x2 := 70;  y2 := 8;
   field_type := strg
   end;
with field_list[4] do
   begin
   label_string := 'Game start time (DTG) :';
   label_x := 11;  label_y := 9;
   str_val := scenario_info.start_dtg;
   x1 := 36;  y1 := 9;  x2 := 50;  y2 := 9;
   field_type := dtg
   end;
with field_list[5] do
   begin
   label_string := 'Admin note :';
   label_x := 11;  label_y := 11;
   str_val := scenario_info.admin_notes[1];
   x1 := 25;  y1 := 11;  x2 := 70;  y2 := 11;
   field_type := strg
   end;
with field_list[6] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[2];
```

```
   x1 := 11;  y1 := 12;  x2 := 70;  y2 := 12;
   field_type := strg
   end;
with field_list[7] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[3];
   x1 := 11;  y1 := 13;  x2 := 70;  y2 := 13;
   field_type := strg
   end;
with field_list[8] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[4];
   x1 := 11;  y1 := 14;  x2 := 70;  y2 := 14;
   field_type := strg
   end;
with field_list[9] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[5];
   x1 := 11;  y1 := 15;  x2 := 70;  y2 := 15;
   field_type := strg
   end;
with field_list[10] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[6];
   x1 := 11;  y1 := 16;  x2 := 70;  y2 := 16;
   field_type := strg
   end;
with field_list[11] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[7];
   x1 := 11;  y1 := 17;  x2 := 70;  y2 := 17;
   field_type := strg
   end;
with field_list[12] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[8];
   x1 := 11;  y1 := 18;  x2 := 70;  y2 := 18;
   field_type := strg
   end;
```

```
with field_list[13] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[9];
   x1 := 11;  y1 := 19;  x2 := 70;  y2 := 19;
   field_type := strg
   end;
with field_list[14] do
   begin
   label_string := null_string;
   label_x := 1;  label_y := 1;
   str_val := scenario_info.admin_notes[10];
   x1 := 11;  y1 := 20;  x2 := 70;  y2 := 20;
   field_type := strg
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,4,79,22);
display_edit_screen_help_line;
edit_screen (number_of_fields, field_list, not
abort_allowed);
with scenario_info do
   begin
   opord_number     := field_list[1].str_val;
   opord_date       := field_list[2].str_val;
   map_sheets       := field_list[3].str_val;
   start_dtg        := field_list[4].str_val;
   game_start_dtg  := start_dtg;
   dtg_to_datetime (game_start_dtg, game_start_time);
   game_dtg  := game_start_dtg;
   game_time := game_start_time;
   admin_notes[1]  := field_list[5].str_val;
   admin_notes[2]  := field_list[6].str_val;
   admin_notes[3]  := field_list[7].str_val;
   admin_notes[4]  := field_list[8].str_val;
   admin_notes[5]  := field_list[9].str_val;
   admin_notes[6]  := field_list[10].str_val;
   admin_notes[7]  := field_list[11].str_val;
   admin_notes[8]  := field_list[12].str_val;
   admin_notes[9]  := field_list[13].str_val;
   admin_notes[10] := field_list[14].str_val
   end;
save_scenario_to_disk;
restore_screen
end;

(************************************************)
```

```
{
 Procedure name : UPDATE_NODE_DATA

 Purpose : this procedure allows the player to
   view, change, or print the node data that is
   part of the current scenario.

 Parameters : none.

 Called by : VIEW_SCENARIO
                                                  }
(************************************************)
procedure update_node_data;

var
   current_node : node_record;
   current_node_name : string5;

(************************************************)
{
 Procedure name : INITIALIZE_NODE_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   node data.

 Parameters : none.

 Called by : UPDATE_NODE_DATA
                                                  }
(************************************************)
procedure initialize_node_fields;

begin
number_of_fields := 11;
with field_list[1] do
   begin
   label_string := 'Node name :';
   label_x := 15;  label_y := 7;
   x1 := 28;  y1 := 7;  x2 := 32;  y2 := 7;
   field_type := strg
   end;
with field_list[2] do
   begin
   label_string := 'Associated paths :';
   label_x := 15;  label_y := 9;
   x1 := 36;  y1 := 9;  x2 := 40;  y2 := 9;
   field_type := strg
   end;
with field_list[3] do
   begin
```

```
      label_string := null_string;
      label_x := 15;  label_y := 9;
      x1 := 43;  y1 := 9;  x2 := 47;  y2 := 9;
      field_type := strg
      end;
   with field_list[4] do
      begin
      label_string := null_string;
      label_x := 15;  label_y := 9;
      x1 := 50;  y1 := 9;  x2 := 54;  y2 := 9;
      field_type := strg
      end;
   with field_list[5] do
      begin
      label_string := null_string;
      label_x := 15;  label_y := 10;
      x1 := 36;  y1 := 10;  x2 := 40;  y2 := 10;
      field_type := strg
      end;
   with field_list[6] do
      begin
      label_string := null_string;
      label_x := 15;  label_y := 10;
      x1 := 43;  y1 := 10;  x2 := 47;  y2 := 10;
      field_type := strg
      end;
   with field_list[7] do
      begin
      label_string := null_string;
      label_x := 15;  label_y := 10;
      x1 := 50;  y1 := 10;  x2 := 54;  y2 := 10;
      field_type := strg
      end;
   with field_list[8] do
      begin
      label_string := 'Location (grid) :';
      label_x := 15;  label_y := 12;
      x1 := 50;  y1 := 12;  x2 := 59;  y2 := 12;
      field_type := strg
      end;
   with field_list[9] do
      begin
      label_string := 'Position type (Rural/Urban) :';
      label_x := 15;  label_y := 14;
      x1 := 58;  y1 := 14;  x2 := 58;  y2 := 14;
      field_type := ch;  valid_char_set := ['R','U']
      end;
   with field_list[10] do
      begin
      label_string := 'Cover and concealment (High/Medium/Low)
   :';
```

```
    label_x := 15;  label_y := 16;
    x1 := 58;  y1 := 16;  x2 := 58;  y2 := 16;
    field_type := ch;  valid_char_set := ['H','M','L']
    end;
 with field_list[11] do
    begin
    label_string := 'Prepositioned ammo count (rounds) :';
    label_x := 15;  label_y := 18;
    x1 := 58;  y1 := 18;  x2 := 61;  y2 := 18;
    field_type := int;  int_min_value := 0;  int_max_value :=
 maxint
    end
 end;

 (************************************************)
 {
  Procedure name : DISPLAY_NODE_DATA

  Purpose : this procedure uses the data that
    describes the fields for displaying the
    node data and uses it to display a node's data.

  Parameters : NODE_DATA - record containing the
    data for the node to be displayed.

  Called by : UPDATE_NODE_DATA
                                                  }
 (************************************************)
 procedure display_node_data (node_data : node_record);

 var
    i : integer;

 begin
 with node_data do
    begin
    field_list[1].str_val := node_name;
    field_list[2].str_val := paths[1];
    field_list[3].str_val := paths[2];
    field_list[4].str_val := paths[3];
    field_list[5].str_val := paths[4];
    field_list[6].str_val := paths[5];
    field_list[7].str_val := paths[6];
    field_list[8].str_val := grid;
    field_list[9].str_val := position_type;
    field_list[10].str_val := cover_concealment;
    str (ammo_count, field_list[11].str_val)
    end;
 for i := 1 to number_of_fields do
    with field_list[i] do
       begin
```

```
        put_string (label_x, label_y, label_string);
        put_string (x1, y1, str_val)
        end
    end;

    (************************************************)
    {
     Procedure name : EDIT_NODE_SCREEN

     Purpose : this procedure uses the data that
       describes the fields for displaying the
       node data and uses it to display a node's data
       and allow it to be changed.

     Parameters : NODE_DATA - record containing the
       data for the node to be displayed.
       NEW_RECORD - boolean record indicating whether
       a new node is being entered or an existing one
       is being modified.

     Called by : EDIT_NODE
                                                    }
    (************************************************)
    procedure edit_node_screen (var node_data : node_record;
    new_record : boolean);

    var
       temporary_field : field_record;
       error_code : integer;
       i : integer;

    begin
    save_screen;
    if new_record then
       display_add_record_help_line
    else
       display_edit_screen_help_line;
    clear_area (2,6,79,22);
    clear_area (1,24,80,24);
    with node_data do
       begin
       field_list[1].str_val := node_name;
       field_list[2].str_val := paths[1];
       field_list[3].str_val := paths[2];
       field_list[4].str_val := paths[3];
       field_list[5].str_val := paths[4];
       field_list[6].str_val := paths[5];
       field_list[7].str_val := paths[6];
       field_list[8].str_val := grid;
       field_list[9].str_val := position_type;
       field_list[10].str_val := cover_concealment;
```

```
   str (ammo_count, field_list[11].str_val)
   end;
if new_record then
   edit_screen (number_of_fields, field_list, abort_allowed)

else
   begin
   with field_list[1] do
      begin
      put_string (label_x, label_y, label_string);
      put_string (x1, y1, str_val)
      end;
   temporary_field := field_list[1];
   for i := 2 to number_of_fields do
      field_list[i - 1] := field_list[i];
   decr (number_of_fields);
   edit_screen (number_of_fields, field_list, not
abort_allowed);
   incr (number_of_fields);
   for i := number_of_fields downto 2 do
      field_list[i] := field_list[i - 1];
   field_list[1] := temporary_field
   end;
with node_data do
   begin
   node_name := upper_case (field_list[1].str_val);
   paths[1] := upper_case (field_list[2].str_val);
   paths[2] := upper_case (field_list[3].str_val);
   paths[3] := upper_case (field_list[4].str_val);
   paths[4] := upper_case (field_list[5].str_val);
   paths[5] := upper_case (field_list[6].str_val);
   paths[6] := upper_case (field_list[7].str_val);
   grid := field_list[8].str_val;
   position_type := field_list[9].str_val[1];
   cover_concealment := field_list[10].str_val[1];
   val (field_list[11].str_val, ammo_count, error_code)
   end;
restore_screen
end;

(************************************************)
{
 Procedure name : GET_FIRST_NODE

 Purpose : this procedure retrieves the first node
   from the current scenario node file.

 Parameters : NODE_DATA - record containing the
   data for the node being retrieved.
   CURRENT_NODE_NAME - string containing the name
   of the node being retrieved.
```

```
 Called by : UPDATE_NODE_DATA
                                                    }
(*************************************************)
procedure get_first_node (var current_node : node_record;
                          var current_node_name : string5);

begin
tareset (nodes);
tanext (nodes, current_node, current_node_name);
if not ok then
   current_node_name := null_string
end;

(*************************************************)
{
 Procedure name : GET_NEXT_NODE

 Purpose : this procedure retrieves the next node
   from the current scenario node file.

 Parameters : NODE_DATA - record containing the
   data for the node being retrieved.
   CURRENT_NODE_NAME - string containing the name
   of the node being retrieved.

 Called by : UPDATE_NODE_DATA
                                                    }
(*************************************************)
procedure get_next_node (var current_node : node_record;
                         var current_node_name : string5);

begin
tanext (nodes, current_node, current_node_name);
if not ok then
   begin
   tanext (nodes, current_node, current_node_name);
   if not ok then
      current_node_name := null_string
   end
end;

(*************************************************)
{
 Procedure name : GET_PREV_NODE

 Purpose : this procedure retrieves the previous
   node from the current scenario node file.

 Parameters : NODE_DATA - record containing the
   data for the node being retrieved.
```

```
   CURRENT_NODE_NAME - string containing the name
   of the node being retrieved.

 Called by : UPDATE_NODE_DATA
                                                    }
(************************************************)
procedure get_prev_node (var current_node : node_record;
                         var current_node_name : string5);

begin
taprev (nodes, current_node, current_node_name);
if not ok then
   begin
   taprev (nodes, current_node, current_node_name);
   if not ok then
      current_node_name := null_string
   end
end;

(************************************************)
{
 Procedure name : ADD_NODE

 Purpose : this procedure adds new node record to
   the current scenario node file.

 Parameters : CURRENT_NODE - record containing the
   data for the node being added.
   CURRENT_NODE_NAME - string containing the name
   of the node being added.

 Called by : UPDATE_NODE_DATA
                                                    }
(************************************************)
procedure add_node (var current_node : node_record;
                    var current_node_name : string5);

var
   node_data : node_record;
   done : boolean;

begin
done := false;
node_data := default_node;
repeat
   edit_node_screen (node_data, add_record);
   if (key = f2) and (node_data.node_name <> null_string)
then
      begin
      tainsert (nodes, node_data, node_data.node_name);
      if not ok then
```

```
            display_error_message ('RECORD ERROR',
                                    null_string, 'a node
already exists',
                                   'with this name',
null_string)
         else
            begin
            taread (nodes, node_data, node_data.node_name,
exactmatch);
            current_node := node_data;
            current_node_name := current_node.node_name;
            taflush (nodes);
            done := true
            end
         end
      else if (key = f2) and (node_data.node_name =
null_string) then
         display_error_message ('RECORD ERROR',
                                 null_string, 'node must have a
name',
                                'before it can be saved',
null_string )
      else
         done := true
until done;
key := null
end;

(************************************************)
{
 Procedure name : DELETE_NODE

 Purpose : this procedure deletes a node record
   from the current scenario node file.

 Parameters : CURRENT_NODE - record containing the
   data for the node being deleted.
   CURRENT_NODE_NAME - string containing the name
   of the node being deleted.

 Called by : UPDATE_NODE_DATA
                                                  }
(************************************************)
procedure delete_node (var current_node : node_record;
                       var current_node_name : string5);

var
   next_node : node_record;
   next_node_name : string5;

(************************************************)
```

```
{
 Procedure name : VERIFY

 Purpose : this procedure prompts the player for
   a yes or no to verify the action that he last
   selected.

 Parameters : none.

 Called by : DELETE_NODE
                                                    }
(************************************************)
function  verify : boolean;

var
   number_of_fields : integer;
   field_list : field_array;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Are you sure (Y/N) ?';
   label_x := 28;  label_y := 12;
   str_val := 'N';
   x1 := 50;  y1 := 12;  x2 := 50;  y2 := 12;
   field_type := ch; valid_char_set := ['Y','N'];
   end;
save_screen;
draw_window (21,10,60,14, blue, lightgray, null_string);
shade_window (21,10,60,14, black);
edit_screen (number_of_fields, field_list, not
abort_allowed);
verify := field_list[1].str_val[1] = 'Y';
key := null;
restore_screen
end;

begin
if (current_node_name <> null_string) and verify then
   begin
   get_next_node (next_node, next_node_name);
   tadelete (nodes, current_node_name);
   taread (nodes, current_node, next_node_name, exactmatch);

   if ok then
      current_node_name := current_node.node_name
   else
      current_node_name := null_string;
   taflush (nodes)
   end
```

```
end;

(*************************************************)
{
 Procedure name : EDIT_NODE

 Purpose : this procedure allows the player to
   edit a node in the current scenario node file.

 Parameters : CURRENT_NODE - record containing the
   data for the node being edited.
   CURRENT_NODE_NAME - string containing the name
   of the node being edited.

 Called by : UPDATE_NODE_DATA
                                                  }
(*************************************************)
procedure edit_node (var current_node : node_record;
                     var current_node_name : string5);

begin
edit_node_screen (current_node, edit_record);
taupdate (nodes, current_node, current_node.node_name);
taflush (nodes);
key := null
end;

(*************************************************)
{
 Procedure name : FIND_NODE

 Purpose : this procedure allows the player to
   find a node in the current scenario node file.

 Parameters : CURRENT_NODE - record containing the
   data for the node being searched for.
   CURRENT_NODE_NAME - string containing the name
   of the node being searched for.

 Called by : UPDATE_NODE_DATA
                                                  }
(*************************************************)
procedure find_node (var current_node : node_record;
                     var current_node_name : string5);

var
   node_data : node_record;
   search_string : string5;

(*************************************************)
{
```

```
 Procedure name : GET_SEARCH_STRING

 Purpose : this procedure prompts the player for
   the name of the node that he wants to find.

 Parameters : none.

 Called by : FIND_NODE
                                                    }
(*************************************************)
function  get_search_string : string5;

var
   number_of_fields : integer;
   field_list : field_array;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Node to search for :';
   label_x := 26;  label_y := 12;
   str_val := null_string;
   x1 := 48;  y1 := 12;  x2 := 52;  y2 := 12;
   field_type := strg;
   end;
save_screen;
draw_window (21,10,60,14, blue, lightgray, null_string);
shade_window (21,10,60,14, black);
edit_screen (number_of_fields, field_list, not
abort_allowed);
get_search_string := upper_case (field_list[1].str_val);
key := null;
restore_screen
end;

begin
search_string := get_search_string;
if search_string <> null_string then
   begin
   taread (nodes, node_data, search_string, partialmatch);
   if ok then
      begin
      current_node := node_data;
      current_node_name := current_node.node_name
      end
   end
end;

begin
initialize_node_fields;
```

```
save_screen;
clear_area (2,4,79,22);
display_edit_list_help_line;
center_text (24, 'Page Up - next node     Page Down -
previous node', red);
center_text (5, 'Node data', cyan);
get_first_node (current_node, current_node_name);
repeat
   save_screen;
   clear_area (2,6,79,22);
   if current_node_name <> null_string then
      display_node_data (current_node)
   else
      center_text (14, 'There are currently no nodes in the
database', white);
   key := get_key;
   case key of
      home_key  : get_first_node (current_node,
current_node_name);
      page_up   : get_next_node  (current_node,
current_node_name);
      page_down : get_prev_node  (current_node,
current_node_name);
      f4        : print_screen;
      f5        : add_node       (current_node,
current_node_name);
      f6        : delete_node    (current_node,
current_node_name);
      f7        : edit_node      (current_node,
current_node_name);
      f8        : find_node      (current_node,
current_node_name)
   end;
   restore_screen
until key = escape;
restore_screen
end;

(************************************************)
{
 Procedure name : UPDATE_PATH_DATA

 Purpose : this procedure allows the player to
   view, change, or print the path data that is
   part of the current scenario.

 Parameters : none.

 Called by : VIEW_SCENARIO
                                                }
(************************************************)
```

```
procedure update_path_data;

var
   current_path : path_record;
   current_path_name : string5;

(************************************************)
{
 Procedure name : INITIALIZE_PATH_FIELDS

 Purpose : this procedure initializes the data
   that describes the fields for displaying the
   path data.

 Parameters : none.

 Called by : UPDATE_PATH_DATA
                                                 }
(************************************************)
procedure initialize_path_fields;

begin
number_of_fields := 8;
with field_list[1] do
   begin
   label_string := 'Path name :';
   label_x := 15;  label_y := 6;
   x1 := 58;  y1 := 6;  x2 := 62;  y2 := 6;
   field_type := strg
   end;
with field_list[2] do
   begin
   label_string := 'Start node :';
   label_x := 15;  label_y := 8;
   x1 := 58;  y1 := 8;  x2 := 62;  y2 := 8;
   field_type := strg
   end;
with field_list[3] do
   begin
   label_string := 'End node :';
   label_x := 15;  label_y := 10;
   x1 := 58;  y1 := 10;  x2 := 62;  y2 := 10;
   field_type := strg
   end;
with field_list[4] do
   begin
   label_string := 'Length (kilometers) :';
   label_x := 15;  label_y := 12;
   x1 := 58;  y1 := 12;  x2 := 63;  y2 := 12;
   field_type := float;  float_min_value := 0.0;
float_max_value := 500.0
```

```
      end;
   with field_list[5] do
      begin
      label_string := 'Road condition (Poor/Medium/Good) :';
      label_x := 15;  label_y := 14;
      x1 := 58;  y1 := 14;  x2 := 58;  y2 := 14;
      field_type := ch;  valid_char_set := ['P','M','G']
      end;
   with field_list[6] do
      begin
      label_string := 'Bridges on path (Yes/No) :';
      label_x := 15;  label_y := 16;
      x1 := 58;  y1 := 16;  x2 := 58;  y2 := 16;
      field_type := ch;  valid_char_set := ['Y','N']
      end;
   with field_list[7] do
      begin
      label_string := 'Bridge location (grid) :';
      label_x := 15;  label_y := 18;
      x1 := 58;  y1 := 18;  x2 := 67;  y2 := 18;
      field_type := strg
      end;
   with field_list[8] do
      begin
      label_string := 'Vulnerability (High/Medium/Low) :';
      label_x := 15;  label_y := 20;
      x1 := 58;  y1 := 20;  x2 := 58;  y2 := 20;
      field_type := ch;  valid_char_set := ['H','M','L']
      end
   end;

   (************************************************)
   {
    Procedure name : DISPLAY_PATH_DATA

    Purpose : this procedure uses the data that
      describes the fields for displaying the
      path data and uses it to display a path's data.

    Parameters : PATH_DATA - record containing the
      data for the path to be displayed.

    Called by : UPDATE_PATH_DATA
                                                    }
   (************************************************)
   procedure display_path_data (path_data : path_record);

   var
      i : integer;

   begin
```

```
with path_data do
   begin
   field_list[1].str_val := path_name;
   field_list[2].str_val := start_node;
   field_list[3].str_val := end_node;
   str (length:3:1, field_list[4].str_val);
   field_list[5].str_val := road_condition;
   field_list[6].str_val := bridge;
   field_list[7].str_val := bridge_grid;
   field_list[8].str_val := vulnerability
   end;
for i := 1 to number_of_fields do
   with field_list[i] do
      begin
      put_string (label_x, label_y, label_string);
      put_string (x1, y1, str_val)
      end
end;

(************************************************)
{
 Procedure name : EDIT_PATH_SCREEN

 Purpose : this procedure uses the data that
   describes the fields for displaying the
   path data and uses it to display a path's data
   and allow it to be changed.

 Parameters : PATH_DATA - record containing the
   data for the path to be displayed.
   NEW_RECORD - boolean record indicating whether
   a new path is being entered or an existing one
   is being modified.

 Called by : EDIT_PATH
                                                 }
(************************************************)
procedure edit_path_screen (var path_data : path_record;
new_record : boolean);

var
   temporary_field : field_record;
   error_code : integer;
   i : integer;

begin
save_screen;
if new_record then
   display_add_record_help_line
else
   display_edit_screen_help_line;
```

```
clear_area (2,6,79,22);
clear_area (1,24,80,24);
with path_data do
   begin
   field_list[1].str_val := path_name;
   field_list[2].str_val := start_node;
   field_list[3].str_val := end_node;
   str (length:3:1, field_list[4].str_val);
   field_list[5].str_val := road_condition;
   field_list[6].str_val := bridge;
   field_list[7].str_val := bridge_grid;
   field_list[8].str_val := vulnerability
   end;
if new_record then
   edit_screen (number_of_fields, field_list, abort_allowed)

else
   begin
   with field_list[1] do
      begin
      put_string (label_x, label_y, label_string);
      put_string (x1, y1, str_val)
      end;
   temporary_field := field_list[1];
   for i := 2 to number_of_fields do
      field_list[i - 1] := field_list[i];
   decr (number_of_fields);
   edit_screen (number_of_fields, field_list, not
abort_allowed);
   incr (number_of_fields);
   for i := number_of_fields downto 2 do
      field_list[i] := field_list[i - 1];
   field_list[1] := temporary_field
   end;
with path_data do
   begin
   path_name      := upper_case (field_list[1].str_val);
   start_node     := upper_case (field_list[2].str_val);
   end_node       := upper_case (field_list[3].str_val);
   val (field_list[4].str_val, length, error_code);
   road_condition := field_list[5].str_val[1];
   bridge         := field_list[6].str_val[1];
   bridge_grid    := field_list[7].str_val;
   vulnerability  := field_list[8].str_val[1]
   end;
restore_screen
end;

(************************************************)
{
 Procedure name : GET_FIRST_PATH
```

```
 Purpose : this procedure retrieves the first path
   from the current scenario path file.

 Parameters : PATH_DATA - record containing the
   data for the path being retrieved.
   CURRENT_PATH_NAME - string containing the name
   of the path being retrieved.

 Called by : UPDATE_PATH_DATA
                                                    }
(************************************************)
procedure get_first_path (var current_path : path_record;
                          var current_path_name : string5);

begin
tareset (paths);
tanext (paths, current_path, current_path_name);
if not ok then
   current_path_name := null_string
end;

(************************************************)
{
 Procedure name : GET_NEXT_PATH

 Purpose : this procedure retrieves the next path
   from the current scenario path file.

 Parameters : PATH_DATA - record containing the
   data for the path being retrieved.
   CURRENT_PATH_NAME - string containing the name
   of the path being retrieved.

 Called by : UPDATE_PATH_DATA
                                                    }
(************************************************)
procedure get_next_path (var current_path : path_record;
                         var current_path_name : string5);

begin
tanext (paths, current_path, current_path_name);
if not ok then
   begin
   tanext (paths, current_path, current_path_name);
   if not ok then
      current_path_name := null_string
   end
end;

(************************************************)
```

```
{
 Procedure name : GET_PREV_PATH

 Purpose : this procedure retrieves the previous
   path from the current scenario path file.

 Parameters : PATH_DATA - record containing the
   data for the path being retrieved.
   CURRENT_PATH_NAME - string containing the name
   of the path being retrieved.

 Called by : UPDATE_PATH_DATA
                                                  }
(*************************************************)
procedure get_prev_path (var current_path : path_record;
                         var current_path_name : string5);

begin
taprev (paths, current_path, current_path_name);
if not ok then
   begin
   taprev (paths, current_path, current_path_name);
   if not ok then
      current_path_name := null_string
   end
end;

(*************************************************)
{
 Procedure name : ADD_PATH

 Purpose : this procedure adds new path record to
   the current scenario path file.

 Parameters : CURRENT_PATH - record containing the
   data for the path being added.
   CURRENT_PATH_NAME - string containing the name
   of the path being added.

 Called by : UPDATE_PATH_DATA
                                                  }
(*************************************************)
procedure add_path (var current_path : path_record;
                    var current_path_name : string5);

var
   path_data : path_record;
   done : boolean;

begin
done := false;
```

```
path_data := default_path;
repeat
   edit_path_screen (path_data, add_record);
   if (key = f2) and (path_data.path_name <> null_string)
then
      begin
      tainsert (paths, path_data, path_data.path_name);
      if not ok then
         display_error_message ('RECORD ERROR',
                                null_string, 'a path
already exists',
                               'with this name',
null_string)
      else
         begin
         taread (paths, path_data, path_data.path_name,
exactmatch);
         current_path := path_data;
         current_path_name := current_path.path_name;
         taflush (paths);
         done := true
         end
      end
   else if (key = f2) and (path_data.path_name =
null_string) then
      display_error_message ('RECORD ERROR',
                             null_string, 'path must have a
name',
                            'before it can be saved',
null_string )
   else
      done := true
until done;
key := null
end;

(************************************************)
{
 Procedure name : DELETE_PATH

 Purpose : this procedure deletes a path record
   from the current scenario path file.

 Parameters : CURRENT_PATH - record containing the
   data for the path being deleted.
   CURRENT_PATH_NAME - string containing the name
   of the path being deleted.

 Called by : UPDATE_PATH_DATA
                                                 }
(************************************************)
```

```
procedure delete_path (var current_path : path_record;
                       var current_path_name : string5);

var
   next_path : path_record;
   next_path_name : string5;

(************************************************)
{
 Procedure name : VERIFY

 Purpose : this procedure prompts the player for
   a yes or no to verify the action that he last
   selected.

 Parameters : none.

 Called by : DELETE_PATH
                                                 }
(************************************************)
function  verify : boolean;

var
   number_of_fields : integer;
   field_list : field_array;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Are you sure (Y/N) ?';
   label_x := 28;  label_y := 12;
   str_val := 'N';
   x1 := 50;  y1 := 12;  x2 := 50;  y2 := 12;
   field_type := ch; valid_char_set := ['Y','N'];
   end;
save_screen;
draw_window (21,10,60,14, blue, lightgray, null_string);
shade_window (21,10,60,14, black);
edit_screen (number_of_fields, field_list, not
abort_allowed);
verify := field_list[1].str_val[1] = 'Y';
key := null;
restore_screen
end;

begin
if (current_path_name <> null_string) and verify then
   begin
   get_next_path (next_path, next_path_name);
   tadelete (paths, current_path_name);
```

```
   taread (paths, current_path, next_path_name, exactmatch);

   if ok then
      current_path_name := current_path.path_name
   else
      current_path_name := null_string;
   taflush (paths)
   end
end;

(*************************************************)
{
 Procedure name : EDIT_PATH

 Purpose : this procedure allows the player to
   edit a path in the current scenario path file.

 Parameters : CURRENT_PATH - record containing the
   data for the path being edited.
   CURRENT_PATH_NAME - string containing the name
   of the path being edited.

 Called by : UPDATE_PATH_DATA
                                                 }
(*************************************************)
procedure edit_path (var current_path : path_record;
                     var current_path_name : string5);

begin
edit_path_screen (current_path, edit_record);
taupdate (paths, current_path, current_path.path_name);
taflush (paths);
key := null
end;

(*************************************************)
{
 Procedure name : FIND_PATH

 Purpose : this procedure allows the player to
   find a path in the current scenario path file.

 Parameters : CURRENT_PATH - record containing the
   data for the path being searched for.
   CURRENT_PATH_NAME - string containing the name
   of the path being searched for.

 Called by : UPDATE_PATH_DATA
                                                 }
(*************************************************)
procedure find_path (var current_path : path_record;
```

```
                        var current_path_name : string5);

var
   path_data : path_record;
   search_string : string5;

(************************************************)
{
 Procedure name : GET_SEARCH_STRING

 Purpose : this procedure prompts the player for
   the name of the path that he wants to find.

 Parameters : none.

 Called by : FIND_PATH
                                                 }
(************************************************)
function  get_search_string : string5;

var
   number_of_fields : integer;
   field_list : field_array;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Path to search for :';
   label_x := 26;  label_y := 12;
   str_val := null_string;
   x1 := 48;  y1 := 12;  x2 := 52;  y2 := 12;
   field_type := strg;
   end;
save_screen;
draw_window (21,10,60,14, blue, lightgray, null_string);
shade_window (21,10,60,14, black);
edit_screen (number_of_fields, field_list, not
abort_allowed);
get_search_string := upper_case (field_list[1].str_val);
key := null;
restore_screen
end;

begin
search_string := get_search_string;
if search_string <> null_string then
   begin
   taread (paths, path_data, search_string, partialmatch);
   if ok then
      begin
```

```
      current_path := path_data;
      current_path_name := current_path.path_name
      end
   end
end;

begin
initialize_path_fields;
save_screen;
clear_area (2,4,79,22);
display_edit_list_help_line;
center_text (24, 'Page Up - next path     Page Down -
previous path', red);
center_text (5, 'Path data', cyan);
get_first_path (current_path, current_path_name);
repeat
   save_screen;
   clear_area (2,6,79,22);
   if current_path_name <> null_string then
      display_path_data (current_path)
   else
      center_text (14, 'There are currently no paths in the
database', white);
   key := get_key;
   case key of
      home_key  : get_first_path (current_path,
current_path_name);
      page_up   : get_next_path  (current_path,
current_path_name);
      page_down : get_prev_path  (current_path,
current_path_name);
      f4        : print_screen;
      f5        : add_path       (current_path,
current_path_name);
      f6        : delete_path    (current_path,
current_path_name);
      f7        : edit_path      (current_path,
current_path_name);
      f8        : find_path      (current_path,
current_path_name)
   end;
   restore_screen
until key = escape;
restore_screen
end;

begin
save_screen;
clear_area (1,2,80,25);
draw_window (1,3,80,23, white, blue, 'Scenario / Network');
menu_x1 := 25;
```

```
menu_y1 := 7;
menu_x2 := 55;
menu_y2 := 19;
repeat
   choice := menu_selection (null_string, 'Scenario
information|'+
                                                  'Node data|'+
                                                  'Path data|'+
                                                  'Network
utility|'+
                                                  'Return\');
    case choice of
       1 : update_scenario_information;
       2 : update_node_data;
       3 : update_path_data;
       4 : network_utility
    end
until choice = 5;
menu_x1 := menu_x1_default;
menu_y1 := menu_y1_default;
menu_x2 := menu_x2_default;
menu_y2 := menu_y2_default;
restore_screen
end;


begin
end.
^Z
```

```
(************************************************)
{
 Unit name : TIMESTEP

 Purpose : this unit contains the procedures that
   conduct all of the actions that occur each
   time step.  This includes incrementing the
   game time, processing the events list, firing
   rounds, and attriting forces.
                                                 }
(************************************************)
unit timestep;

interface
{$I-}

uses crt, utility, gameutil, global, event, commands,
taccess, tahigh;

procedure check_for_start_of_hostilities;
procedure display_sitrep;
procedure process_events_list;
procedure stock_atp;
procedure process_field_trains;
procedure process_ammo_trucks;
procedure process_fire_units;
procedure generate_messages;

implementation

const
   fire_unit   = 0;
   ammo_truck  = 1;
   in_position = 0;
   under_fire  = 1;
   moving      = 2;

(************************************************)
{
 Procedure name : ADD_MESSAGE

 Purpose : this procedure adds a message to the
 message buffer file for display at a later time.

 Parameters : NEW_MESSAGE - message record to be
   added to the buffer.

 Called by : several procedures that generate
   messages throughout this unit.
                                                 }
(************************************************)
```

```
procedure add_message (new_message : message_record);

var
   record_number : longint;

begin
addrec (messages, record_number, new_message);
addkey (message_type_index, record_number,
new_message.message_type);
flushfile (messages);
flushindex (message_type_index)
end;

(************************************************)
{
 Procedure name : DELETE_MESSAGE

 Purpose : this procedure deletes a message from
 the message buffer after it has been displayed.

 Parameters : RECORD_NUMBER - number of the record
   that corresponds to the message to be deleted.

 Called by : DISPLAY_MESSAGES
                                                 }
(************************************************)
procedure delete_message (record_number : longint);

var
   message : message_record;

begin
getrec (messages, record_number, message);
deletekey (message_type_index, record_number,
message.message_type);
deleterec (messages, record_number);
flushfile (messages);
flushindex (message_type_index)
end;

(************************************************)
{
 Procedure name : CHECK_FOR_START_OF_HOSTILITIES

 Purpose : this procedure prompts the user for
 determining whether hostilities have commenced.

 Parameters : none.

 Called by : PLAY_WARGAME
                                                 }
```

```
(************************************************)
procedure check_for_start_of_hostilities;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Commence hostilities (Yes/No) ?';
   label_x := 24;  label_y := 13;
   str_val := 'N';
   x1 := 57;  y1 := 13;  x2 := 57;  y2 := 13;
   field_type := ch;  valid_char_set := ['Y','N']
   end;
save_screen;
draw_window (18,11,63,15, white, blue, null_string);
display_edit_screen_help_line;
edit_screen (number_of_fields, field_list, not
abort_allowed);
hostilities_started := field_list[1].str_val[1] = 'Y';
restore_screen
end;

(************************************************)
{
 Procedure name : DISPLAY_SITREP

 Purpose : this procedure displays a sitrep for
 the player with information on the firing units,
 the battalion trains, and the ammo trucks.

 Parameters : none.

 Called by : GENERATE_MESSAGES
             ISSUE_COMMAND
                                                 }
(************************************************)
procedure display_sitrep;

var
   number_of_screens : integer;
   screen : integer;

(************************************************)
{
 Procedure name : DISPLAY_FIRING_UNIT_SITREP

 Purpose : this procedure displays a sitrep for
 the fire unit passed as a parameter.

 Parameters : FIRING_UNIT_NUMBER - number of the
   firing unit for which the sitrep will be
```

```
   displayed.

 Called by : DISPLAY_SITREP
                                                    }
(************************************************)
procedure display_firing_unit_sitrep (firing_unit_number :
integer);

var
   number_string : string80;

begin
save_screen;
with firing_units[firing_unit_number] do
   begin
   center_text (4, 'Firing Unit', blue);
   put_string (20, 6,  'Firing unit name :  ');
   put_string (58, 6,  firing_unit_name);
   if sections_in_operating_condition > 0.0 then
      begin
      put_string (20, 7,  'Location :  ');
      put_string (58, 7,  location);
      put_string (20, 8,  'Rounds on hand :  ');
      str (rounds_on_hand, number_string);
      put_string (58, 8,  number_string);
      put_string (20, 9,  'Ammo status :  ');
      put_string (58, 9,  ammo_status);
      put_string (20, 10, 'Rounds fired from current
position :  ');
      str (rounds_fired_from_position, number_string);
      put_string (58, 10, number_string);
      put_string (20, 11, 'Time in position (hours) :  ');
      str ((time_in_position / 60):4:2, number_string);
      put_string (58, 11, number_string);
      put_string (20, 12, 'Firing sections still operating :
');
      str (sections_in_operating_condition:4:2,
number_string);
      put_string (58, 12, number_string);
      put_string (20, 13, 'Status :  ');
      put_string (58, 13, firing_status);
      put_string (20, 14, 'Vulnerability :  ');
      put_string (58, 14, vulnerability_status)
      end
   else
      center_text (8, 'K I L L E D', blue)
   end;
key := get_key;
restore_screen
end;
```

```
(************************************************)
{
 Procedure name : DISPLAY_AMMO_TRUCK_SITREP

 Purpose : this procedure displays a sitrep for
 the range of ammo trucks passed as parameters.

 Parameters : FIRST_TRUCK - number of the first
   truck to be displayed.
   LAST_TRUCK - number of the last truck to be
   displayed.

 Called by : DISPLAY_SITREP
                                                 }
(************************************************)
procedure display_ammo_truck_sitrep (first_truck, last_truck
: integer);

var
   truck : integer;
   percent : string6;

begin
save_screen;
center_text (4, 'Ammo Trucks', blue);
center_text (6,
  'Bumper    Convoy    Mission    Unit  to    Full/
Location    Effective', blue);
center_text (7,
  '  #                              resupply    empty
        %    ', blue);
center_text (8,
  '------    ------    -------    --------    -----
--------    ---------', blue);
for truck := first_truck to last_truck do
   with ammo_trucks[truck] do
      begin
      put_string (5, 9 + truck - first_truck,
bumper_number);
      if not killed then
         begin
         put_string (15, 9 + truck - first_truck,
convoy_name);
         put_string (27, 9 + truck - first_truck,
mission_assigned);
         put_string (37, 9 + truck - first_truck,
firing_unit_to_resupply);
         put_string (49, 9 + truck - first_truck,
load_status);
         put_string (58, 9 + truck - first_truck, location);
```

```
            str (effective_percent:4:2, percent);
            put_string (70, 9 + truck - first_truck, percent)
            end
         else
            put_string (22, 9 + truck - first_truck, 'K I L L E
D')
         end;
key := get_key;
restore_screen
end;

(************************************************)
{
 Procedure name : DISPLAY_TRAINS_SITREP

 Purpose : this procedure displays a sitrep for
 the battalion trains.

 Parameters : none.

 Called by : DISPLAY_SITREP
                                                 }
(************************************************)
procedure display_trains_sitrep;

begin
save_screen;
center_text (4, 'Field Trains', blue);
center_text (6, 'Location :  '+ battalion_trains.location,
blue);
key := get_key;
restore_screen
end;

begin
save_screen;
clear_area (1,2,80,25);
draw_window (1,2,80,25, blue, lightgray, 'SITREP');
center_text (23, 'page up - next screen     page down -
previous screen', red);
center_text (24, 'hit ESC when done viewing sitrep', red);
if number_of_ammo_trucks > 12 then
   number_of_screens := number_of_firing_units + 3
else
   number_of_screens := number_of_firing_units + 2;
screen := 1;
repeat
   if screen in [1..number_of_firing_units] then
      display_firing_unit_sitrep (screen)
   else if screen = number_of_firing_units + 1 then
      display_trains_sitrep
```

```
   else if screen = number_of_firing_units + 2 then
      begin
      if number_of_ammo_trucks < 12 then
         display_ammo_truck_sitrep (1,
number_of_ammo_trucks)
      else
         display_ammo_truck_sitrep (1, 12)
      end
   else
      display_ammo_truck_sitrep (13, number_of_ammo_trucks);

   if key = page_up then
      begin
      if screen = number_of_screens then
         screen := 1
      else
         incr (screen)
      end
   else if key = page_down then
      begin
      if screen = 1 then
         screen := number_of_screens
      else
         decr (screen)
      end
   else if key = 113 shl 8 then
      check_event_list
until key = escape;
restore_screen
end;

(************************************************)
{
 Procedure name : AMOUNT_OF_ATTRITION

 Purpose : this procedure determines the rate of
 attrition of all units during the game based on
 the parameters passed.

 Parameters : UNIT_TYPE - indicates whether the
   unit to be attrited is a truck or fire unit.
   UNIT_NUMBER - indicates which fire unit or
   truck is to be attrited.
   UNIT_POSTURE - indicates the current posture
   of the unit to be attrited.

 Called by : PROCESS_FIELD_TRAINS
             PROCESS_AMMO_TRUCKS
             PROCESS_FIRE_UNITS
                                                }
(************************************************)
```

```
function amount_of_attrition (unit_type : integer;
                              unit_number : integer;
                              unit_posture : integer) :
real;

type
   alpha_array = array [fire_unit..ammo_truck] of
                    array [in_position..moving] of real;

const
   alpha_air  : alpha_array = ( (0.000005, 0.000005,
0.0000025),
                                (0.00001,  0.00001,
0.000005) );
   alpha_arty : alpha_array = ( (0.000004, 0.000004,
0.000002),
                                (0.000008, 0.000008,
0.000004) );
var
   temp_node : node_record;
   temp_path : path_record;
   casualties_per_minute : real;

function y_air : integer;

var
   y : integer;

begin
if day_time then
   y := 2
else
   y := 1;
case unit_posture of
   in_position, under_fire :
      begin
      with temp_node do
         begin
         case position_type of
            'R' : y := y + 2;
            'U' : y := y + 1
         end;
         case cover_concealment of
            'L' : y := y + 3;
            'M' : y := y + 2;
            'H' : y := y + 1
         end
         end;
      if unit_type = fire_unit then
         with firing_units[unit_number] do
            begin
```

```
            case vulnerability_status of
               'H' : y := y + 3;
               'M' : y := y + 2;
               'L' : y := y + 1
            end;
            if unit_posture = under_fire then
               case firing_status of
                  'M' : y := y + 2;
               else
                        y := y + 1
               end
            end
      else
         with ammo_trucks[unit_number] do
            begin
            case vulnerability_status of
               'H' : y := y + 3;
               'M' : y := y + 2;
               'L' : y := y + 1
             end;
             if unit_posture = under_fire then
                case moving of
                   false : y := y + 2;
                   true  : y := y + 1
                end
             end
      end;
   moving :
      with temp_path do
         case vulnerability of
            'H' : y := y + 3;
            'M' : y := y + 2;
            'L' : y := y + 1
         end
end;
y_air := y
end;

function y_arty : integer;

var
   y : integer;

begin
y := y_air;
y_arty := y
end;

begin
case unit_type of
   fire_unit  :
```

```
      begin
      if unit_posture in [in_position, under_fire] then
         taread (nodes, temp_node,
                 firing_units[unit_number].location,
exactmatch)
      else
         taread (paths, temp_path,
                 firing_units[unit_number].location,
exactmatch);
      with firing_units[unit_number] do
         casualties_per_minute :=
sections_in_operating_condition *
           ( (alpha_air  [unit_type][unit_posture] * y_air)
+
             (alpha_arty [unit_type][unit_posture] * y_arty)
)
      end;
   ammo_truck :
      begin
      if unit_posture in [in_position, under_fire] then
         taread (nodes, temp_node,
                 ammo_trucks[unit_number].location,
exactmatch)
      else
         taread (paths, temp_path,
                 ammo_trucks[unit_number].location,
exactmatch);
      with ammo_trucks[unit_number] do
         casualties_per_minute := effective_percent *
           ( (alpha_air  [unit_type][unit_posture] * y_air)
+
             (alpha_arty [unit_type][unit_posture] * y_arty)
)
       end
end;
amount_of_attrition := casualties_per_minute *
game_parameters.time_step_size
end;

(************************************************)
{
 Procedure name : PROCESS_EVENTS_LIST

 Purpose : this procedure reads events from the
 events list and determines if it is time to
 execute that particular event and does so if
 the time is current.

 Parameters : none.

 Called by : EXECUTE_NEXT_TIME_STEP
```

```
                                                    }
(************************************************)
procedure process_events_list;

var
   current_dtg : string15;
   last_time_to_process : string10;
   next_time_key : string10;
   record_number : longint;
   event : event_record;

(************************************************)
{
 Procedure name : UNIT_ALIVE

 Purpose : this procedure determines whether or
 not a unit is still alive before it processes
 an event record for that unit.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
                                                    }
(************************************************)
function unit_alive (event : event_record): boolean;

begin
case event.unit_type of
   'T' : unit_alive := true;
   'A' : with ammo_trucks[truck_number (event.unit_name)] do

            unit_alive := not killed;
   'F' : with firing_units[unit_number (event.unit_name)] do

            unit_alive := (sections_in_operating_condition >
0.0)
end
end;

(************************************************)
{
 Procedure name : OCCUPY_NODE

 Purpose : this procedure handles an occupy node
 event.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
                                                    }
(************************************************)
```

```
procedure occupy_node (event : event_record);

begin
case event.unit_type of
   'T' : begin
         battalion_trains.location := event.node;
         battalion_trains.moving := false
         end;
   'A' : with ammo_trucks[truck_number (event.unit_name)] do

            begin
            location := event.node;
            moving := false
            end;
   'F' : with firing_units[unit_number (event.unit_name)] do

            begin
            location := event.node;
            if rounds_on_hand > 0 then
               firing_status := 'H'
            else
               firing_status := 'C'
            end
end
end;

(************************************************)
{
 Procedure name : TRANSIT_NODE

 Purpose : this procedure handles a transit node
 event.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
                                                 }
(************************************************)
procedure transit_node (event : event_record);

begin
case event.unit_type of
   'T' : battalion_trains.location := event.path;
   'A' : with ammo_trucks[truck_number (event.unit_name)] do

            location := event.path;
   'F' : with firing_units[unit_number (event.unit_name)] do

            location := event.path
end
end;
```

```
(************************************************)
{
 Procedure name : WAIT_AT_NODE

 Purpose : this procedure handles a wait at node
 event.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
                                                 }
(************************************************)
procedure wait_at_node (event : event_record);

begin
case event.unit_type of
   'T' : battalion_trains.location := event.node;
   'A' : with ammo_trucks[truck_number (event.unit_name)] do

            location := event.node;
   'F' : with firing_units[unit_number (event.unit_name)] do

            location := event.node
end
end;

(************************************************)
{
 Procedure name : DEPART_NODE

 Purpose : this procedure handles a depart node
 event.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
                                                 }
(************************************************)
procedure depart_node (event : event_record);

var
   capacity : integer;
   temp_node : node_record;
   temp_path : path_record;

begin
case event.unit_type of
   'T' : with battalion_trains do
            begin
            location := event.path;
```

```
                time_in_position := 0;
                vulnerability_status := 'A';
                moving := true
                end;
      'A' : with ammo_trucks[truck_number (event.unit_name)] do

                begin
                location := event.path;
                taread (paths, temp_path, event.path,
exactmatch);
                vulnerability_status := temp_path.vulnerability;

                moving := true
                end;
      'F' : with firing_units[unit_number (event.unit_name)] do

                begin
                capacity := round
(sections_in_operating_condition *
                                    section_max_rounds_capacity);

                if rounds_on_hand > capacity then
                   begin
                   taread (nodes, temp_node, location,
exactmatch);
                   temp_node.ammo_count :=
                      temp_node.ammo_count + (rounds_on_hand -
capacity);
                   taupdate (nodes, temp_node, location);
                   taflush (nodes);
                   rounds_on_hand := capacity
                   end;
                location := event.path;
                time_in_position := 0;
                rounds_fired_from_position := 0;
                vulnerability_status := 'A';
                firing_status := 'M'
                end
end
end;

(************************************************)
{
 Procedure name : SHOOT

 Purpose : this procedure handles a shooting
 event.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
```

```
                                                        }
(*************************************************)
procedure shoot (event : event_record);

(*************************************************)
{
 Procedure name : DETERMINE_AMMO_STATUS

 Purpose : this procedure determines the units
 ammo status after the firing has occured.

 Parameters : UNIT_NUMBER - unit that is firing

 Called by : SHOOT
                                                        }
(*************************************************)
procedure determine_ammo_status (unit_number : integer);

begin
with firing_units[unit_number] do
   begin
   if rounds_on_hand <= 0 then
      ammo_status := 'O'
   else if rounds_on_hand > (0.35 *
sections_in_operating_condition *
                              section_max_rounds_capacity)
then
      ammo_status := 'S'
   else if rounds_on_hand < (0.1 *
sections_in_operating_condition *
                              section_max_rounds_capacity)
then
      ammo_status := 'C'
   else
      ammo_status := 'L'
   end
end;

begin
if hostilities_started then
   with firing_units[unit_number (event.unit_name)] do
      if firing_status = 'H' then
         begin
         rounds_on_hand := rounds_on_hand -
            round (event.volleys *
sections_in_operating_condition);
         rounds_fired_from_position :=
rounds_fired_from_position +
            round (event.volleys *
sections_in_operating_condition);
```

```
            determine_ammo_status (unit_number
(event.unit_name))
            end
end;

(************************************************)
{
 Procedure name : RETURN_FROM_ATP

 Purpose : this procedure handles a return from
 the atp event.

 Parameters : EVENT - event to be processed.

 Called by : PROCESS_EVENTS_LIST
                                                 }
(************************************************)
procedure return_from_atp (event : event_record);

var
   new_message : message_record;

(************************************************)
{
 Procedure name : CREATE_MESSAGE

 Purpose : this procedure creates a message and
 puts it in the buffer if a truck has returned
 from the atp and the trains have moved.

 Parameters : TRUCK_NUMBER - truck that has
   returned to trains location.

 Called by : RETURN_FROM_ATP
                                                 }
(************************************************)
procedure create_message (truck_number : integer);

var
   new_message : message_record;
   message : message_record;
   record_number : longint;
   message_exists : boolean;
   key : char;

begin
with ammo_trucks[truck_number] do
   begin
   new_message.message_type := '8';
   if convoy_name = null_string then
      begin
```

```
      new_message.unit_type := 'A';
      new_message.unit_name := bumper_number
      end
   else
      begin
      new_message.unit_type := 'C';
      new_message.unit_name := convoy_name
      end;
   new_message.location := location
   end;
message_exists := false;
clearkey (message_type_index);
key := '8';
nextkey (message_type_index, record_number, key);
while ok and not message_exists do
   begin
   getrec (messages, record_number, message);
   if message.unit_name = new_message.unit_name then
      message_exists := true;
    nextkey (message_type_index, record_number, key)
    end;
if not message_exists then
   add_message (new_message)
end;

begin
with ammo_trucks[truck_number (event.unit_name)] do
   begin
   location := event.node;
   moving := false;
   if atp_rounds_on_hand >= round (ammo_capacity *
effective_percent) then
      begin
      atp_rounds_on_hand :=
         atp_rounds_on_hand - round (ammo_capacity *
effective_percent);
      load_status := 'F'
      end;
   if location <> battalion_trains.location then
      create_message (truck_number (event.unit_name))
   end
end;

begin
current_dtg := game_dtg;
last_time_to_process :=
   inc_dtg_to_timekey
      (current_dtg, round (game_parameters.time_step_size /
2.0));
clearkey (time_index);
nextkey (time_index, record_number, next_time_key);
```

```
while ok and (next_time_key <= last_time_to_process) do
   begin
   getrec (event_list, record_number, event);
   if unit_alive (event) then
      case event.event_type of
         'O' : occupy_node (event);
         'T' : transit_node (event);
         'W' : wait_at_node (event);
         'D' : depart_node (event);
         'F' : shoot (event);
         'R' : return_from_atp (event)
      end;
   delete_event (record_number);
   nextkey (time_index, record_number, next_time_key)
   end
end;

(************************************************)
{
 Procedure name : STOCK_ATP

 Purpose : this procedure determines whether or
 not to restock the atp each time based on the
 start of a new day.

 Parameters : none.

 Called by : EXECUTE_NEXT_TIME_STEP
                                                 }
(************************************************)
procedure stock_atp;

var
   total_firing_units : real;
   i : integer;

begin
if new_day then
   begin
   total_firing_units := 0;
   for i := 1 to number_of_firing_units do
      total_firing_units := total_firing_units +
         firing_units[i].sections_in_operating_condition;
   atp_rounds_on_hand :=
      trunc (commanders_guidance.bn_csr *
total_firing_units)
   end
end;

(************************************************)
{
```

```
 Procedure name : PROCESS_FIELD_TRAINS

 Purpose : this procedure checks the trains each
 time step to determine the vulnerability, if it
 is receiving incomming, and attrition of any
 trucks at the trains location.

 Parameters : none.

 Called by : EXECUTE_NEXT_TIME_STEP
                                                    }
(*************************************************)
procedure process_field_trains;

(*************************************************)
{
 Procedure name : DETERMINE_VULNERABILTY_LEVEL

 Purpose : this procedure checks the current
 level of vulnerability of the field trains based
 on time in position.

 Parameters : none.

 Called by : PROCESS_FIELD_TRAINS
                                                    }
(*************************************************)
procedure determine_vulnerability_level;

begin
with battalion_trains do
   begin
   if time_in_position <
      round
(commanders_guidance.vulnerability_threshold_time * 60) then

         vulnerability_status := 'A'
   else if time_in_position >
      round
(commanders_guidance.vulnerability_threshold_time * 60) then

         vulnerability_status := 'H'
   else if time_in_position >
      round
(commanders_guidance.vulnerability_threshold_time * 60 * 2)
then
         vulnerability_status := 'C'
   end
end;

(*************************************************)
```

```
{
 Procedure name : CHECK_FOR_INCOMING

 Purpose : this procedure checks the trains level
 of vulnerability to determine if they should
 receive enemy artillery fire.

 Parameters : none.

 Called by : PROCESS_FIELD_TRAINS
                                                  }
(************************************************)
procedure check_for_incoming;

(************************************************)
{
 Procedure name : DISPLACE_FOR_INCOMING

 Purpose : this procedure prompts the player to
 determine whether the trains and any trucks at
 that location should displace after receiving
 incoming.

 Parameters : none.

 Called by : CHECK_FOR_INCOMING
                  }
(************************************************)
function displace_for_incoming : boolean;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Displace (Yes/No) ?';
   label_x := 30;  label_y := 20;
   str_val := 'N';
   x1 := 51;  y1 := 20;  x2 := 51;  y2 := 20;
   field_type := ch;  valid_char_set := ['Y','N']
   end;
save_screen;
draw_window (10,15,71,23, blue, lightgray, null_string);
center_text (17, 'Battalion trains are receiving incoming
artillery fire.', blue);
center_text (18, 'Request permission for emergency
displacement.', blue);
edit_screen (number_of_fields, field_list, not
abort_allowed);
displace_for_incoming := field_list[1].str_val[1] = 'Y';
restore_screen
end;
```

```
(************************************************)
{
 Procedure name : DISPLACE_TRAINS

 Purpose : this procedure creates the event
 records necessary to displace the trains and any
 trucks at the trains if desired by the player.

 Parameters : none.

 Called by : CHECK_FOR_INCOMING
                    }
(************************************************)
procedure displace_trains;

var
   i : integer;
   displace_event : event_record;
   current_time : string15;

begin
with displace_event do
   begin
   event_type := 'O';
   current_time := game_dtg;
   time_key := inc_dtg_to_timekey (current_time, 30);
   node := battalion_trains.location
   end;
with battalion_trains do
   begin
   displace_event.unit_type := 'T';
   displace_event.unit_name := 'TRAIN';
   moving := true;
   time_in_position := 0;
   vulnerability_status := 'A';
   add_event (displace_event)
   end;
for i := 1 to number_of_ammo_trucks do
   with ammo_trucks[i] do
      if (not killed) and (location =
battalion_trains.location) then
         begin
         displace_event.unit_type := 'A';
         displace_event.unit_name := bumper_number;
         moving := true;
         add_event (displace_event)
         end
end;

(************************************************)
```

```
{
 Procedure name : ASSESS_CASUALTIES

 Purpose : this procedure eill assess casualties
 for any trucks at the trains location when they
 received the incoming artillery fire.

 Parameters : none.

 Called by : CHECK_FOR_INCOMING
                    }
(************************************************)
procedure assess_casualties;

var
   i : integer;

begin
for i := 1 to number_of_ammo_trucks do
   with ammo_trucks[i] do
      if (not killed) and (location =
battalion_trains.location) then
         begin
         vulnerability_status :=
battalion_trains.vulnerability_status;
         effective_percent := effective_percent -
            amount_of_attrition (ammo_truck, i, under_fire);

         if effective_percent < 0.0 then
            effective_percent := 0.0
         end
end;

begin
with battalion_trains do
   if ((vulnerability_status = 'C') and (random < 0.05)) or
      ((vulnerability_status = 'H') and (random < 0.01)) or
      ((vulnerability_status = 'A') and (random < 0.005))
then
         begin
         if displace_for_incoming then
            displace_trains;
         assess_casualties
         end
end;

(************************************************)
{
 Procedure name : PUT_VULNERABILITY_MESSAGE

 Purpose : this procedure will put a message in
```

```
 the buffer if the vulnerability of the trains has
 changed this time step.

 Parameters : none.

 Called by : PROCESS_FIELD_TRAINS
                    }
(************************************************)
procedure put_vulnerability_message;

procedure create_message;

var
   new_message : message_record;

begin
with battalion_trains do
   begin
   new_message.message_type := '7';
   new_message.unit_type := 'T';
   new_message.unit_name := 'TRAIN';
   if vulnerability_high then
      new_message.vulnerability := 'H'
   else
      new_message.vulnerability := 'C'
   end;
add_message (new_message)
end;

begin
with battalion_trains do
   case vulnerability_status of
      'A' : if vulnerability_high or vulnerability_critical
then
               begin
               vulnerability_high := false;
               vulnerability_critical := false
               end;
      'H' : if not vulnerability_high then
               begin
               vulnerability_critical := false;
               vulnerability_high := true;
               create_message
               end;
      'C' : if not vulnerability_critical then
               begin
               vulnerability_high := false;
               vulnerability_critical := true;
               create_message
               end
   end
```

```
end;

begin
if not battalion_trains.moving then
   with battalion_trains do
      begin
      time_in_position := time_in_position +
game_parameters.time_step_size;
      determine_vulnerability_level;
      if hostilities_started then
         check_for_incoming;
      put_vulnerability_message
      end
end;

(************************************************)
{
 Procedure name : PROCESS_AMMO_TRUCKS

 Purpose : this procedure checks the trucks each
 time step to determine if they have completed a
 resupply mission, if they have sufficient rest,
 attrit them if moving, and update their stats.

 Parameters : none.

 Called by : EXECUTE_NEXT_TIME_STEP
                                                 }
(************************************************)
procedure process_ammo_trucks;

(************************************************)
{
 Procedure name : ASSESS_CASUALTIES

 Purpose : this procedure checks the trucks each
 time step to determine if they are moving and if
 so what attrition will be assessed to them.

 Parameters : none.

 Called by : PROCESS_AMMO_TRUCKS
                                                 }
(************************************************)
procedure assess_casualties;

var
   i : integer;

begin
for i := 1 to number_of_ammo_trucks do
```

```
   with ammo_trucks[i] do
      if (not killed) and moving then
         begin
         effective_percent := effective_percent -
            amount_of_attrition (ammo_truck, i,
timestep.moving);
         if effective_percent < 0.0 then
            effective_percent := 0.0;
         killed := (effective_percent = 0.0)
         end
end;

(************************************************)
{
 Procedure name : CHECK_FOR_RESUPPLY

 Purpose : this procedure checks the trucks each
 time step to determine if they have completed
 a resupply mission, either to a unit or to a
 node.  If so, it creates a message to that
 effect.

 Parameters : none.

 Called by : PROCESS_AMMO_TRUCKS
                                                }
(************************************************)
procedure check_for_resupply;

var
   i : integer;
   temp_node : node_record;

procedure create_message (message_type : char);

var
   new_message : message_record;
   message : message_record;
   message_exists : boolean;
   key : char;
   record_number : longint;

begin
with ammo_trucks[i] do
   begin
   new_message.message_type := message_type;
   if convoy_name = null_string then
      begin
      new_message.unit_type := 'A';
      new_message.unit_name := bumper_number
      end
```

```
      else
         begin
         new_message.unit_type := 'C';
         new_message.unit_name := convoy_name
         end;
      if message_type = '1' then
         new_message.location := node_to_resupply
      else
         new_message.location := firing_unit_to_resupply
      end;
   message_exists := false;
   clearkey (message_type_index);
   key := message_type;
   nextkey (message_type_index, record_number, key);
   while ok and not message_exists do
      begin
      getrec (messages, record_number, message);
      if message.unit_name = new_message.unit_name then
         message_exists := true;
       nextkey (message_type_index, record_number, key)
       end;
   if not message_exists then
      add_message (new_message)
   end;

   begin
   for i := 1 to number_of_ammo_trucks do
      with ammo_trucks[i] do
         if (not killed) and (mission_assigned = 'Y') then
            begin
            if (firing_unit_to_resupply = 'PREPO') and (not
   moving) and
               (node_to_resupply = location) and (load_status =
   'F') then
                  begin
                  load_status := 'E';
                  taread (nodes, temp_node, node_to_resupply,
   exactmatch);
                  temp_node.ammo_count := temp_node.ammo_count
   +
                     round (ammo_capacity * effective_percent);

                  taupdate (nodes, temp_node, location);
                  taflush (nodes);
                  create_message ('1');
                  mission_assigned := 'N';
                  firing_unit_to_resupply := null_string;
                  node_to_resupply := null_string
                  end
            else if (not moving) and (load_status = 'F') and
               (firing_unit_to_resupply <> null_string) and
```

```
            (firing_units[unit_number
(firing_unit_to_resupply)].location =
            location) then
               begin
               load_status := 'E';
               with firing_units[unit_number
(firing_unit_to_resupply)] do
                  begin
                  rounds_on_hand := rounds_on_hand +
                     round (ammo_capacity *
effective_percent);
                  if firing_status = 'C' then
                     firing_status := 'H'
                  end;
               create_message ('2');
               mission_assigned := 'N';
               firing_unit_to_resupply := null_string;
               node_to_resupply := null_string
               end
         end
end;

(************************************************)
{
 Procedure name : CHECK_FOR_REST

 Purpose : this procedure checks the trucks each
 time step to determine if they have accrued
 sufficient rest to accomplish any assigned
 mission.  It also updates the amount of rest
 and the time since they began resting.

 Parameters : none.

 Called by : PROCESS_AMMO_TRUCKS
                                                }
(************************************************)
procedure check_for_rest;

var
   i : integer;

begin
for i := 1 to number_of_ammo_trucks do
   with ammo_trucks[i] do
      if not killed then
         begin
         time_since_rest_began :=
            time_since_rest_began +
game_parameters.time_step_size;
         if (not moving) and (mission_assigned = 'N') then
```

```
            amount_of_rest := amount_of_rest +
game_parameters.time_step_size;
         if amount_of_rest >=
            round (commanders_guidance.crew_rest_per_day *
60) then
               begin
               time_since_rest_began := 0;
               amount_of_rest := 0
               end
         end
end;

(*************************************************)
{
 Procedure name : CALCULATE_TRUCK_STATS

 Purpose : this procedure updates the amount of
 time that a truck is not available due to
 casualties each time step.

 Parameters : none.

 Called by : PROCESS_AMMO_TRUCKS
                                                  }
(*************************************************)
procedure calculate_truck_stats;

var
   i : integer;

begin
for i := 1 to number_of_ammo_trucks do
   with ammo_trucks[i] do
      casualty_time := casualty_time +
         round ((1.0 - effective_percent) *
game_parameters.time_step_size)
end;

begin
if hostilities_started then
   assess_casualties;
check_for_resupply;
check_for_rest;
calculate_truck_stats
end;

(*************************************************)
{
 Procedure name : PROCESS_FIRE_UNITS

 Purpose : this procedure checks the fire units
```

```
 each time step for firing, attrition, ammo
 resupply, and updates it vulnerability, ammo
 status, and statistics.

 Parameters : none.

 Called by : EXECUTE_NEXT_TIME_STEP
                                                    }
(*************************************************)
procedure process_fire_units;

var
   i : integer;

(*************************************************)
{
 Procedure name : CHECK_FOR_AMMO_RESUPPLY

 Purpose : this procedure checks the fire units
 each time step to determine if they have
 arrived at a node at which they are supposed to
 pick up ammunition.

 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(*************************************************)
procedure check_for_ammo_resupply;

var
   temp_node : node_record;

begin
with firing_units[i] do
   if (ammo_pickup_mission) and (location =
ammo_pickup_location) then
      begin
      taread (nodes, temp_node, location, exactmatch);
      rounds_on_hand := rounds_on_hand +
temp_node.ammo_count;
      temp_node.ammo_count := 0;
      taupdate (nodes, temp_node, location);
      taflush (nodes)
      end
end;

(*************************************************)
{
 Procedure name : SHOOT_ROUNDS
```

```
 Purpose : this procedure checks the fire units
 each time step to have them fire at a rate
 consistent with their ammo status and the
 battalion csr.

 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(*************************************************)
procedure shoot_rounds;

var
   rounds_to_fire : integer;

begin
with firing_units[i] do
   begin
   rounds_to_fire := trunc (commanders_guidance.bn_csr *
rate_percent_csr *
                            sections_in_operating_condition
*
                            game_parameters.time_step_size /
(60 * 24));
   rounds_to_fire := abs (normal_rv (rounds_to_fire, 5));
   if rounds_to_fire >
      (sustained_rate_of_fire *
sections_in_operating_condition *
       game_parameters.time_step_size) then
          rounds_to_fire := trunc (sustained_rate_of_fire *

sections_in_operating_condition *

game_parameters.time_step_size);
   if rounds_to_fire > rounds_on_hand then
      rounds_to_fire := rounds_on_hand;
   rounds_on_hand := rounds_on_hand - rounds_to_fire;
   rounds_fired_from_position := rounds_fired_from_position
+ rounds_to_fire;
   if rounds_on_hand <= 0 then
      firing_status := 'C'
   end
end;

(*************************************************)
{
 Procedure name : DETERMINE_VULNERABILITY_LEVEL

 Purpose : this procedure checks the fire units
 each time step to update their vulnerability
 status based on rounds fired from position and
```

```
 time in position.

 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(*************************************************)
procedure determine_vulnerability_level;

begin
with firing_units[i], commanders_guidance do
   if (time_in_position < round
(vulnerability_threshold_time * 60)) and
      (rounds_fired_from_position <
vulnerability_threshold_rounds) then
         vulnerability_status := 'A'
   else if (time_in_position < round
(vulnerability_threshold_time * 60)) and
      (rounds_fired_from_position >
vulnerability_threshold_rounds) then
         vulnerability_status := 'H'
   else if (time_in_position > round
(vulnerability_threshold_time * 60)) and
      (rounds_fired_from_position <
vulnerability_threshold_rounds) then
         vulnerability_status := 'H'
   else
         vulnerability_status := 'C'
end;

(*************************************************)
{
 Procedure name : ASSESS_CASUALTIES

 Purpose : this procedure checks the fire units
 each time step to assess casualties based on
 vulnerabilty and whether they are receiving
 incoming this time step.

 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(*************************************************)
procedure assess_casualties;

var
   j : integer;

(*************************************************)
{
```

```
 Procedure name : CHECK_FOR_INCOMING

 Purpose : this procedure checks the fire units
 each time step to determine whether or not they
 will receive enemy artillery fire and assess
 casualties accordingly.

 Parameters : none.

 Called by : ASSESS_CASUALTIES
                                                    }
(************************************************)
procedure check_for_incoming;

function displace_for_incoming : boolean;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Displace (Yes/No) ?';
   label_x := 30;  label_y := 20;
   str_val := 'N';
   x1 := 51;  y1 := 20;  x2 := 51;  y2 := 20;
   field_type := ch;  valid_char_set := ['Y','N']
   end;
save_screen;
draw_window (10,15,71,23, blue, lightgray, null_string);
center_text (17, 'Firing unit ' +
firing_units[i].firing_unit_name +
                  ' is receiving incoming artillery fire.',
blue);
center_text (18, 'Request permission for emergency
displacement.', blue);
edit_screen (number_of_fields, field_list, not
abort_allowed);
displace_for_incoming := field_list[1].str_val[1] = 'Y';
restore_screen
end;

procedure displace_firing_unit;

var
   j : integer;
   displace_event : event_record;
   current_time : string15;

begin
with displace_event do
   begin
   event_type := 'O';
```

```
   current_time := game_dtg;
   time_key := inc_dtg_to_timekey (current_time, 30);
   node := firing_units[i].location
   end;
with firing_units[i] do
   begin
   displace_event.unit_type := 'F';
   displace_event.unit_name := firing_unit_name;
   firing_status := 'M';
   time_in_position := 0;
   rounds_fired_from_position := 0;
   vulnerability_status := 'A';
   add_event (displace_event)
   end;
for j := 1 to number_of_ammo_trucks do
   with ammo_trucks[j] do
      if (not killed) and (location =
firing_units[i].location) then
         begin
         displace_event.unit_type := 'A';
         displace_event.unit_name := bumper_number;
         moving := true;
         add_event (displace_event)
         end
end;

procedure assess_incoming_casualties;

var
   j : integer;

begin
with firing_units[i] do
   begin
   sections_in_operating_condition :=
sections_in_operating_condition -
      amount_of_attrition (fire_unit, i, under_fire);
   if sections_in_operating_condition < 0.0 then
      sections_in_operating_condition := 0.0
   end;
for j := 1 to number_of_ammo_trucks do
   with ammo_trucks[j] do
      if (not killed) and (location =
firing_units[i].location) then
         begin
         vulnerability_status :=
firing_units[i].vulnerability_status;
         effective_percent := effective_percent -
            amount_of_attrition (ammo_truck, j, under_fire);

         if effective_percent < 0.0 then
```

```
            effective_percent := 0.0
         end
end;

begin
with firing_units[i] do
   if ((vulnerability_status = 'C') and (random < 0.05)) or
      ((vulnerability_status = 'H') and (random < 0.01)) or
      ((vulnerability_status = 'A') and (random < 0.005))
then
         begin
         if displace_for_incoming then
            displace_firing_unit;
         assess_incoming_casualties
         end
end;

begin
with firing_units[i] do
   begin
   sections_in_operating_condition :=
sections_in_operating_condition -
      amount_of_attrition (fire_unit, i, in_position);
   if sections_in_operating_condition < 0.0 then
      sections_in_operating_condition := 0.0
   end;
for j := 1 to number_of_ammo_trucks do
   with ammo_trucks[j] do
      if (not killed) and
         (ammo_trucks[j].location =
firing_units[i].location) then
            begin
            vulnerability_status :=
firing_units[i].vulnerability_status;
            effective_percent := effective_percent -
               amount_of_attrition (ammo_truck, j,
in_position);
            if effective_percent < 0.0 then
               effective_percent := 0.0
            end;
check_for_incoming
end;

(************************************************)
(
 Procedure name : PUT_VULNERABILITY_MESSAGE

 Purpose : this procedure will create a message to
 be placed in the buffer if the firing units
 vulnerability level has changed this time step.
```

```
 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(************************************************)
procedure put_vulnerability_message;

procedure create_message;

var
   new_message : message_record;

begin
with firing_units[i] do
   begin
   new_message.message_type := '7';
   new_message.unit_type := 'F';
   new_message.unit_name := firing_unit_name;
   if vulnerability_high then
      new_message.vulnerability := 'H'
   else
      new_message.vulnerability := 'C'
   end;
add_message (new_message)
end;

begin
with firing_units[i] do
   case vulnerability_status of
      'A' : if vulnerability_high or vulnerability_critical
then
               begin
               vulnerability_high := false;
               vulnerability_critical := false
               end;
      'H' : if not vulnerability_high then
               begin
               vulnerability_critical := false;
               vulnerability_high := true;
               create_message
               end;
      'C' : if not vulnerability_critical then
               begin
               vulnerability_high := false;
               vulnerability_critical := true;
               create_message
               end
   end
end;

(************************************************)
```

```
{
 Procedure name : DETERMINE_AMMO_STATUS

 Purpose : this procedure will check the firing
 unit each time step in order to update its ammo
 status.

 Parameters : none.

 Called by : PROCESS_FIRE_UNIT
                                                    }
(************************************************)
procedure determine_ammo_status;

begin
with firing_units[i] do
   if rounds_on_hand > (0.35 * section_max_rounds_capacity *

                        sections_in_operating_condition) then

      ammo_status := 'S'
   else if rounds_on_hand <= 0 then
      ammo_status := 'O'
   else if rounds_on_hand < (0.1 *
section_max_rounds_capacity *
                        sections_in_operating_condition) then

      ammo_status := 'C'
   else
      ammo_status := 'L'
end;

(************************************************)
{
 Procedure name : PUT_AMMO_STATUS_MESSAGE

 Purpose : this procedure will create a message to
 be placed in the buffer if the firing units
 ammo status has changed this time step.

 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(************************************************)
procedure put_ammo_status_message;

procedure create_message;

var
   new_message : message_record;
```

```
begin
with firing_units[i] do
   begin
   new_message.message_type := '6';
   new_message.unit_type := 'F';
   new_message.unit_name := firing_unit_name;
   if ammo_low then
      new_message.ammo_status := 'L'
   else if ammo_critical then
      new_message.ammo_status := 'C'
   else if ammo_out then
      new_message.ammo_status := 'O'
   end;
add_message (new_message)
end;

begin
with firing_units[i] do
   case ammo_status of
      'S' : if ammo_out or ammo_critical or ammo_low then
               begin
               ammo_out := false;
               ammo_critical := false;
               ammo_low := false
               end;
      'L' : if not ammo_low then
               begin
               ammo_low := true;
               ammo_critical := false;
               ammo_out := false;
               create_message
               end;
      'C' : if not ammo_critical then
               begin
               ammo_critical := true;
               ammo_low := false;
               ammo_out := false;
               create_message
               end;
      'O' : if not ammo_out then
               begin
               ammo_out := true;
               ammo_low := false;
               ammo_critical := false;
               create_message
               end
   end
end;

(************************************************)
```

```
{
 Procedure name : CALCULATE_FIRING_UNIT_STATS

 Purpose : this procedure will update the stats
 for a firing unit each time step based on the
 time that it was critically short ammo,
 critically vulnerable, and time lost due to
 casualties.

 Parameters : none.

 Called by : PROCESS_FIRE_UNITS
                                                    }
(*************************************************)
procedure calculate_firing_unit_stats;

begin
with firing_units[i] do
   begin
   if firing_status = 'H' then
      total_availability_time :=
         total_availability_time + round
(game_parameters.time_step_size *
         sections_in_operating_condition);
   if vulnerability_status = 'C' then
      critically_vulnerable_time :=
         critically_vulnerable_time + round
(game_parameters.time_step_size *
         sections_in_operating_condition);
   if ammo_status = 'C' then
      critically_short_time :=
         critically_short_time + round
(game_parameters.time_step_size *
         sections_in_operating_condition)
   end
end;

begin
for i := 1 to number_of_firing_units do
   with firing_units[i] do
      if sections_in_operating_condition > 0.0 then
         begin
         check_for_ammo_resupply;
         if (hostilities_started) and (firing_status = 'H')
then
            shoot_rounds;
         if firing_status = 'M' then
            begin
            sections_in_operating_condition :=
               sections_in_operating_condition -
               amount_of_attrition (fire_unit, i, moving);
```

```
            if sections_in_operating_condition < 0.0 then
               sections_in_operating_condition := 0.0
            end
         else
            begin
            time_in_position :=
               time_in_position +
game_parameters.time_step_size;
            determine_vulnerability_level;
            if hostilities_started then
               assess_casualties;
            if sections_in_operating_condition > 0.0 then
               begin
               put_vulnerability_message;
               determine_ammo_status;
               put_ammo_status_message
               end
            end;
         calculate_firing_unit_stats
         end
end;

(************************************************)
{
 Procedure name : GENERATE_MESSAGES

 Purpose : this procedure will display the sitrep
 and message buffer each time step if it is
 required based on desired frequency of sitreps
 and whether or not the message buffer has any-
 thing in it.

 Parameters : none.

 Called by : EXECUTE_NEXT_TIME_STEP
                                                 }
(************************************************)
procedure generate_messages;

procedure display_messages;

var
   message : message_record;
   message_type : char;
   record_number : longint;
   buffer : array [1..8] of string;
   i : integer;

function message_text (message : message_record): string;

var
```

```
   buffer : string;

begin
with message do
   begin
   case message_type of
      '1' : begin
            buffer := 'Ammo prepositioned at node ' +
location;
            if unit_type = 'C' then
               buffer := buffer + ', convoy ' + unit_name +
' requests orders'
            else
               buffer := buffer + ', truck ' + unit_name + '
requests_orders'
            end;
      '2' : begin
            buffer := 'Ammo delivered to unit ' + location;
            if unit_type = 'C' then
               buffer := buffer + ', convoy ' + unit_name +
' requests orders'
            else
               buffer := buffer + ', truck ' + unit_name + '
requests_orders'
            end;
      '3' : begin
            end;
      '4' : begin
            end;
      '5' : begin
            end;
      '6' : begin
            case ammo_status of
              'O' : buffer := 'Unit ' + unit_name + ' is out
of ammo';
              'C' : buffer := 'Unit ' + unit_name + ' ammo
status is critical';
              'L' : buffer := 'Unit ' + unit_name + ' ammo
status is low'
            end
            end;
      '7' : begin
            case unit_type of
               'T' : if vulnerability = 'H' then
                        buffer := 'Trains have reached high
vulnerability'
                     else
                        buffer := 'Trains have reached
critical vulnerability';
               'F' : if vulnerability = 'H' then
                        buffer := 'Unit '+ unit_name +
```

```
                                    ' has reached high
vulnerability'
                        else
                           buffer := 'Unit '+ unit_name +
                                    ' has reached critical
vulnerability'
                end
                end;
         '8' : begin
                if unit_type = 'C' then
                   buffer := 'Convoy ' + unit_name + ' unable to
locate trains'
                else
                   buffer := 'Truck ' + unit_name + ' unable to
locate trains'
                end
      end
      end;
message_text := buffer
end;

procedure display_buffer;

var
   i : integer;

begin
save_screen;
draw_window (6,14,75,25, blue, lightgray, 'MESSAGES');
for i := 1 to 8 do
   put_string (10, 15 + i, buffer[i]);
center_text (24, 'press any key', red);
key := get_key;
key := null;
restore_screen
end;

begin
clearkey (message_type_index);
nextkey (message_type_index, record_number, message_type);
repeat
   for i := 1 to 8 do
      buffer[i] := null_string;
   i := 1;
   while ok and (i < 8) do
      begin
      getrec (messages, record_number, message);
      buffer[i] := message_text (message);
      delete_message (record_number);
      incr (i);
```

```
      nextkey (message_type_index, record_number,
message_type)
      end;
   if buffer[1] <> null_string then
      display_buffer
until not ok
end;

begin
display_messages;
time_since_last_sitrep :=
   time_since_last_sitrep + game_parameters.time_step_size;
if time_since_last_sitrep >=
commanders_guidance.unit_sitrep_frequency then
   begin
   time_since_last_sitrep := 0;
   display_sitrep
   end
end;


begin
end.^Z
```

```
(************************************************)
{
 Unit name : COMMANDS

 Purpose : this unit contains the procedures that
   allow the player to input commands and have
   them executed as is appropriate for each
   command.
                                                 }
(************************************************)
unit commands;

interface
{$I-}

uses dos, crt, utility, gameutil, global, taccess, tahigh;

procedure create_truck_convoy;
procedure remove_truck_convoy;
procedure ammo_resupply_mission;
procedure cancel_resupply_mission;
procedure ammo_truck_ammo_pickup;
procedure fire_unit_ammo_pickup;
procedure cancel_fire_unit_pickup;
procedure move_unit;
procedure issue_fire_order;
procedure change_firing_rate;
procedure cancel_command;

implementation

type
   name_array = array [1..24] of string5;
   route_array = array [1..2] of array [1..11] of string5;
var
   route : route_array;
   current_path : integer;

(************************************************)
{
 Procedure name : UNIQUE_CONVOY_NAME

 Purpose : this procedure checks a name entered
 for a convoy to insure that it has not been used
 to name a fire unit, truck, or other convoy.

 Parameters : STRING_VALUE - string to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by CREATE_TRUCK_CONVOY
                                                 }
```

```
(************************************************)
{$F+}
function unique_convoy_name (string_value : string80):
boolean;

var
   convoy_exists : boolean;

begin
suppress_messages := true;
string_value := upper_case (string_value);
convoy_exists := (string_value = null_string) or
                 (string_value = 'TRAIN') or
                 (truck_number (string_value) <> 0) or
                 (unit_number  (string_value) <> 0) or
                 (valid_convoy (string_value));
if convoy_exists then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'convoy must have a unique name',
null_string);
unique_convoy_name := not convoy_exists;
suppress_messages := false
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_AMMO_TRUCK_FOR_CONVOY

 Purpose : this procedure checks a name entered
 to determine whether or not it can be added to
 a new convoy.

 Parameters : STRING_VALUE - string to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by CREATE_TRUCK_CONVOY
                                                 }
(************************************************)
{$F+}
function valid_ammo_truck_for_convoy (string_value :
string80): boolean;

var
   truck_exists : boolean;

begin
suppress_messages := true;
string_value := upper_case (string_value);
truck_exists := (string_value = null_string) or
```

```
                (valid_ammo_truck (string_value));
if not truck_exists then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'ammo truck does not exist',
null_string);
valid_ammo_truck_for_convoy := truck_exists;
suppress_messages := false
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_TRUCK_OR_CONVOY

 Purpose : this procedure checks a name entered
 to determine whether or not it corresponds to
 an existing truck or convoy.

 Parameters : STRING_VALUE - string to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by AMMO_RESUPPLY_MISSION
                                                 }
(************************************************)
{$F+}
function valid_truck_or_convoy (string_value : string80):
boolean;

var
   truck_or_convoy_exists : boolean;

begin
suppress_messages := true;
string_value := upper_case (string_value);
truck_or_convoy_exists := valid_ammo_truck (string_value) or

                          valid_convoy (string_value);
if not truck_or_convoy_exists then
   display_error_message ('INPUT ERROR', null_string,
                          'truck/convoy entered does not
exist',
                          'or truck is part of a convoy',
null_string);
valid_truck_or_convoy := truck_or_convoy_exists;
suppress_messages := false
end;
{$F-}

(************************************************)
{
```

```
 Procedure name : VALID_UNIT_TO_RESUPPLY

 Purpose : this procedure checks a name entered
 to determine whether or not it corresponds to
 an existing node or unit to be resupplied.

 Parameters : STRING_VALUE - string to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by AMMO_RESUPPLY_MISSION
                                                  }
(************************************************)
{$F+}
function valid_unit_to_resupply (string_value : string80):
boolean;

var
   unit_exists : boolean;

begin
suppress_messages := true;
string_value := upper_case (string_value);
unit_exists := (valid_unit (string_value)) or (string_value
= 'PREPO');
if not unit_exists then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'unit entered does not exist',
null_string)
else if string_value = 'PREPO' then
   begin
   number_of_fields := 3;
   with field_list[3] do
      begin
      put_string (label_x, label_y, label_string);
      if length (str_val) > (x2 - x1 + 1) then
         str_val [0] := chr (x2 - x1 + 1);
      put_string (x1, y1, str_val)
      end
   end
else
   begin
   number_of_fields := 2;
   field_list[3].str_val := null_string;
   with field_list[3] do
      put_string (label_x, label_y, '
      ')
   end;
valid_unit_to_resupply := unit_exists;
suppress_messages := false
end;
```

```
{$F-}

(************************************************)
{
 Procedure name : TRUCKS_AT_ATP

 Purpose : this procedure checks a name entered
 to determine whether or not it corresponds to
 an existing truck or convoy that is at the
 trains and ready to pickup ammo.

 Parameters : STRING_VALUE - string to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by AMMO_TRUCK_AMMO_PICKUP
                                                 }
(************************************************)
{$F+}
function trucks_at_atp (string_value : string80): boolean;

var
   truck_or_convoy_valid : boolean;
   i : integer;

begin
suppress_messages := true;
string_value := upper_case (string_value);
truck_or_convoy_valid := valid_ammo_truck (string_value) or
                         valid_convoy (string_value);
if truck_or_convoy_valid then
   begin
   if valid_ammo_truck (string_value) then
      truck_or_convoy_valid :=
          (ammo_trucks[truck_number (string_value)].location
=
           battalion_trains.location) and
          (ammo_trucks[truck_number
(string_value)].load_status = 'E') and
          (ammo_trucks[truck_number
(string_value)].mission_assigned = 'N')
   else
      begin
      truck_or_convoy_valid := false;
      for i := 1 to number_of_ammo_trucks do
         if (ammo_trucks[i].convoy_name = string_value) and
            (ammo_trucks[i].location =
battalion_trains.location) and
            (ammo_trucks[i].load_status = 'E') and
            (ammo_trucks[i].mission_assigned = 'N') then
               truck_or_convoy_valid := true
      end
```

```
   end;
if not truck_or_convoy_valid then
   display_error_message ('INPUT ERROR', null_string,
                          'truck/convoy entered does not
exist,',
                          'is not at trains, has a mission,
or',
                          'is not out of ammunition');
trucks_at_atp := truck_or_convoy_valid;
suppress_messages := false
end;
{$F-}

(************************************************)
{
 Procedure name : TIME_NOT_PAST

 Purpose : this procedure checks a dtg that has
 been entered for a command to insure that it
 has not past.

 Parameters : STRING_VALUE - dtg to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by any command that requires a dtg be entered.
                                                 }
(************************************************)
{$F+}
function time_not_past (string_value : string80): boolean;

var
   time_equivalent : datetime;

begin
if valid_dtg (string_value) then
   begin
   dtg_to_datetime (string_value, time_equivalent);
   if time_relative (time_equivalent, game_time) = before
then
      begin
      time_not_past := false;
      display_error_message ('INPUT ERROR', null_string,
null_string,
                             'time entered has already
past', null_string);
      end
   else
      time_not_past := true
   end
else
   begin
```

```
   time_not_past := false;
   display_error_message ('INPUT ERROR', null_string,
                          'dtg format : ''05 0530Z JAN
89''',
                          'spaces may be omitted',
null_string)
   end
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_NODE_FOR_ROUTE

 Purpose : this procedure checks a node to insure
 that it lies on the route being entered for a
 movement command.

 Parameters : STRING_VALUE - node to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by MOVE_UNIT.
                                                 }
(************************************************)
{$F+}
function valid_node_for_route (string_value : string80):
boolean;

begin
string_value := upper_case (string_value);
if valid_node (string_value) then
   route[2][1] := string_value
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_PATH_IN_ROUTE

 Purpose : this procedure checks a path to insure
 that it lies on the route being entered for a
 movement command.

 Parameters : STRING_VALUE - path to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by MOVE_UNIT.
                                                 }
(************************************************)
{$F+}
```

```
function valid_path_in_route (string_value : string80):
boolean;

var
   node_has_path : boolean;
   path_has_node : boolean;
   path_exists : boolean;
   temp_path : path_record;
   temp_node : node_record;
   i : integer;

procedure update_fields_displayed;

begin
if (key = up_arrow) and (current_path > 2) then
   begin
   decr (number_of_fields);
   with field_list[current_path] do
      begin
      str_val := null_string;
      put_string (label_x, label_y,
                  '

')
      end;
   decr (current_path)
   end
else if (key = enter) or (key = down_arrow) or
        ((key = f2) and (string_value <> null_string)) then
   begin
   route[1][current_path] := temp_path.path_name;
   if temp_path.start_node = temp_node.node_name then
      route[2][current_path] := temp_path.end_node
   else
      route[2][current_path] := temp_path.start_node;
   with field_list[current_path] do
      put_string (label_x + 21, label_y,
                  'to node :  '+ route[2][current_path]);
   if (current_path < 11) then
      begin
      incr (number_of_fields);
      with field_list[current_path + 1] do
         begin
         put_string (label_x, label_y, label_string);
         if length (str_val) > (x2 - x1 + 1) then
            str_val [0] := chr (x2 - x1 + 1);
         put_string (x1, y1, str_val)
         end;
      incr (current_path)
      end
   else if key <> f2 then
      key := null
```

```
   end
end;

begin
suppress_messages := true;
string_value := upper_case (string_value);
path_exists := (key = up_arrow) or
   ((key = f2) and (string_value = null_string) and
(current_path > 2));
if not path_exists then
   begin
   taread (paths, temp_path, string_value, exactmatch);
   path_exists := ok;
   if not path_exists then
      display_error_message ('INPUT ERROR', null_string,
null_string,
                             'path entered does not exist',
null_string)
   else
      begin
      taread (nodes, temp_node, route[2][current_path - 1],
exactmatch);
      node_has_path := false;
      for i := 1 to 6 do
         if temp_node.paths[i] = temp_path.path_name then
            node_has_path := true;
      path_has_node := (temp_path.start_node =
temp_node.node_name) or
                       (temp_path.end_node =
temp_node.node_name);
      path_exists := node_has_path and path_has_node;
      if not path_exists then
         display_error_message ('INPUT ERROR', null_string,
                                'path does not connect', 'to
previous node',
                                 null_string)
      end
   end;
if path_exists then
   update_fields_displayed;
valid_path_in_route := path_exists;
suppress_messages := false
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_UNIT_TO_MOVE

 Purpose : this procedure checks a unit entered to
 insure that it is a valid unit in this game and
```

```
 that it is not already pending a move.

 Parameters : STRING_VALUE - unit to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by MOVE_UNIT.
                                                    }
(************************************************)
{$F+}
function valid_unit_to_move (string_value : string80):
boolean;

var
   unit_exists : boolean;
   i : integer;

begin
suppress_messages := true;
string_value := upper_case (string_value);
unit_exists := valid_unit (string_value) or
               valid_ammo_truck (string_value) or
               valid_convoy (string_value) or
               (string_value = 'TRAIN');
if not unit_exists then
   display_error_message ('INPUT ERROR', null_string,
                          'unit entered does not exist',
                          'or cannot move separately',
null_string)
else
   begin
   if string_value = 'TRAIN' then
      unit_exists := not battalion_trains.pending_movement
   else if valid_unit (string_value) then
      unit_exists := not firing_units[unit_number
(string_value)].
                          pending_movement
   else if valid_ammo_truck (string_value) then
      unit_exists := not ammo_trucks[truck_number
(string_value)].
                          pending_movement
   else
      begin
      unit_exists := true;
      for i := 1 to number_of_ammo_trucks do
         with ammo_trucks[i] do
            if (convoy_name = string_value) and
(pending_movement) then
               unit_exists := false
      end;
   if not unit_exists then
      display_error_message ('INPUT ERROR', null_string,
```

```
                                'unit is already moving or is',
                                'pending execution of a
movement', null_string)
   end;
if unit_exists then
   begin
   if valid_unit (string_value) then
      begin
      number_of_fields := 4;
      with field_list[4] do
         begin
         put_string (label_x, label_y, label_string);
         if length (str_val) > (x2 - x1 + 1) then
            str_val [0] := chr (x2 - x1 + 1);
         put_string (x1, y1, str_val)
         end
      end
   else
      begin
      number_of_fields := 3;
      field_list[4].str_val := null_string;
      with field_list[4] do
      put_string (label_x, label_y, '
            ')
      end
   end;
valid_unit_to_move := unit_exists;
suppress_messages := false
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_NODE_FOR_RESUPPLY

 Purpose : this procedure checks a node entered
 to insure that it lies along the entered route
 so that resupply can occur there.

 Parameters : STRING_VALUE - node to be checked.

 Called by : used as a parameter to EDIT_SCREEN
   by MOVE_UNIT.
                                                }
(************************************************)
{$F+}
function valid_node_for_resupply (string_value : string80):
boolean;

var
```

```
   i : integer;
   node_in_route : boolean;

begin
string_value := upper_case (string_value);
node_in_route := string_value = null_string;
if not node_in_route then
   for i := 1 to 12 do
      if string_value = route[2][i] then
         node_in_route := true;
if not node_in_route then
   display_error_message ('INPUT ERROR', null_string,
                          'node entered is not along',
                          'the entered route', null_string);

valid_node_for_resupply := node_in_route
end;
{$F-}

(*************************************************)
{
 Procedure name : CREATE_TRUCK_CONVOY

 Purpose : this procedure allows the player to
 create a convoy of trucks to be used as a group
 for other commands to be entered.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                 }
(*************************************************)
procedure create_truck_convoy;

var
   names : name_array;
   i : integer;

procedure set_up_fields;

var
   i, j : integer;

begin
number_of_fields := 25;
with field_list[1] do
   begin
   label_string := 'New convoy name :';
   label_x := 11;  label_y := 9;
   str_val := null_string;
   x1 := 30;  y1 := 9;  x2 := 34;  y2 := 9;
```

```
   field_type := eval; eval_function := unique_convoy_name
   end;
for i := 0 to 2 do
   for j := 1 to 8 do
      with field_list[1 + j + (i * 8)] do
         begin
         label_string := null_string;
         label_x := 11;  label_y := 11;
         str_val := null_string;
         x1 := 3 + (j * 8);  y1 := 13 + i;  x2 := 7 + (j *
8);  y2 := 13 + i;
         field_type := eval; eval_function :=
valid_ammo_truck_for_convoy
         end;
field_list[2].label_string := 'Enter bumper numbers for
trucks in convoy :'
end;

function valid_input (var names : name_array): boolean;

var
   input_is_valid : boolean;
   location : string10;
   ammo_status : char;
   i : integer;

begin
input_is_valid := false;
for i := 1 to 24 do
   if names[i] <> null_string then
      input_is_valid := true;
if input_is_valid then
   begin
   i := 0;
   repeat
      incr (i)
   until names[i] <> null_string;
   location := ammo_trucks[truck_number
(names[i])].location;
   ammo_status := ammo_trucks[truck_number
(names[i])].load_status;
   for i := 1 to 24 do
      if (names[i] <> null_string) and
         ((location <> ammo_trucks[truck_number
(names[i])].location) or
          (ammo_status <> ammo_trucks[truck_number
(names[i])].load_status) or
          (ammo_trucks[truck_number
(names[i])].mission_assigned = 'Y')) then
             input_is_valid := false
   end;
```

```
if not input_is_valid then
   display_error_message ('COMMAND ERROR',
                          'all trucks must be in same
location,',
                          'have no mission assigned, and',
                          'have the same ammo status to be',

                          'part of the same convoy');
valid_input := input_is_valid
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Create Truck Convoy', cyan);
repeat
   edit_screen (number_of_fields, field_list,
abort_allowed);
   field_list[1].str_val := upper_case
(field_list[1].str_val);
   for  i := 1 to 24 do
      names[i] := upper_case (field_list[i + 1].str_val)
until (key = escape) or valid_input (names);
if key <> escape then
   for i := 1 to 24 do
      if names[i] <> null_string then
         ammo_trucks[truck_number(names[i])].convoy_name :=
            field_list[1].str_val;
restore_screen
end;

(************************************************)
{
 Procedure name : REMOVE_TRUCK_CONVOY

 Purpose : this procedure allows the player to
 remove trucks from a convoy so that they will
 now be treated individually by the player.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                 }
(************************************************)
procedure remove_truck_convoy;

var
   i : integer;
```

```
procedure set_up_fields;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Convoy to be separated :';
   label_x := 25;  label_y := 12;
   str_val := null_string;
   x1 := 51;  y1 := 12;  x2 := 55;  y2 := 12;
   field_type := eval; eval_function := valid_convoy
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Remove Truck Convoy', cyan);
edit_screen (number_of_fields, field_list, abort_allowed);
field_list[1].str_val := upper_case (field_list[1].str_val);

if (key = f2) or (key = enter) then
   for i := 1 to number_of_ammo_trucks do
      if ammo_trucks[i].convoy_name = field_list[1].str_val
then
         ammo_trucks[i].convoy_name := null_string;
restore_screen
end;

(************************************************)
{
 Procedure name : AMMO_RESUPPLY_MISSION

 Purpose : this procedure allows the player to
 assign a truck or convoy the mission of ammo
 delivery to either a fire unit or node.  This
 command must be issued with a corresponding
 movement order to insure that applicable units
 are at the proper locations for delivery.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure ammo_resupply_mission;

var
   i : integer;
```

```
procedure set_up_fields;

begin
number_of_fields := 2;
with field_list[1] do
   begin
   label_string := 'Ammo truck bumper # or convoy name :';
   label_x := 19;  label_y := 10;
   str_val := null_string;
   x1 := 57;  y1 := 10;  x2 := 61;  y2 := 10;
   field_type := eval; eval_function :=
valid_truck_or_convoy
   end;
with field_list[2] do
   begin
   label_string := 'Unit to resupply :';
   label_x := 19;  label_y := 12;
   str_val := null_string;
   x1 := 39;  y1 := 12;  x2 := 43;  y2 := 12;
   field_type := eval; eval_function :=
valid_unit_to_resupply
   end;
with field_list[3] do
   begin
   label_string := 'Node to resupply :';
   label_x := 19;  label_y := 14;
   str_val := null_string;
   x1 := 39;  y1 := 14;  x2 := 43;  y2 := 14;
   field_type := eval; eval_function := valid_node
   end
end;

function valid_input : boolean;

var
   input_is_valid : boolean;
   i : integer;

begin
input_is_valid := not ((field_list[3].str_val = null_string)
and
   ((field_list[2].str_val = null_string) or
    (field_list[2].str_val = 'PREPO')));
if input_is_valid then
   begin
   suppress_messages := true;
   if valid_ammo_truck (field_list[1].str_val) then
      begin
      input_is_valid := ammo_trucks[truck_number
(field_list[1].str_val)].
```

```
         mission_assigned = 'N';
      if not input_is_valid then
         display_error_message ('COMMAND ERROR',
null_string,
                                'ammo truck already has a',
                                'mission assigned',
null_string)
      end
   else
      begin
      for i := 1 to number_of_ammo_trucks do
         if (ammo_trucks[i].convoy_name =
field_list[1].str_val) and
            (ammo_trucks[i].mission_assigned = 'Y') then
               input_is_valid := false;
      if not input_is_valid then
         display_error_message ('COMMAND ERROR',
null_string,
                                'convoy already has a',
                                'mission assigned',
null_string)
      end;
   suppress_messages := false
   end;
valid_input := input_is_valid
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Ammunition Resupply', cyan);
repeat
   edit_screen (number_of_fields, field_list,
abort_allowed);
   field_list[1].str_val := upper_case
(field_list[1].str_val);
   field_list[2].str_val := upper_case
(field_list[2].str_val);
   field_list[3].str_val := upper_case
(field_list[3].str_val);
until (key = escape) or valid_input;
if key <> escape then
   begin
   if truck_number (field_list[1].str_val) <> 0 then
      begin
      ammo_trucks[truck_number (field_list[1].str_val)].
         mission_assigned := 'Y';
      ammo_trucks[truck_number (field_list[1].str_val)].
         firing_unit_to_resupply := field_list[2].str_val;
```

```
      ammo_trucks[truck_number (field_list[1].str_val)].
         node_to_resupply := field_list[3].str_val
      end
   else
      for i := 1 to number_of_ammo_trucks do
         if ammo_trucks[i].convoy_name =
field_list[1].str_val then
            begin
            ammo_trucks[i].mission_assigned := 'Y';
            ammo_trucks[i].firing_unit_to_resupply :=
field_list[2].str_val;
            ammo_trucks[i].node_to_resupply :=
field_list[3].str_val
            end
   end;
restore_screen
end;

(************************************************)
{
 Procedure name : CANCEL_RESUPPLY_MISSION

 Purpose : this procedure allows the player to
 cancel the resupply mission issued to a truck
 or convoy by the AMMO_RESUPPLY_COMMAND.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure cancel_resupply_mission;

var
   i : integer;

procedure set_up_fields;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Ammo truck bumper # or convoy name :';
   label_x := 19;  label_y := 12;
   str_val := null_string;
   x1 := 57;  y1 := 12;  x2 := 61;  y2 := 12;
   field_type := eval; eval_function :=
valid_truck_or_convoy
   end
end;
```

```
begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Cancel Resupply Mission', cyan);
edit_screen (number_of_fields, field_list, abort_allowed);
field_list[1].str_val := upper_case (field_list[1].str_val);

if (key = f2) or (key = enter) then
   for i := 1 to number_of_ammo_trucks do
      if ((ammo_trucks[i].bumper_number =
field_list[1].str_val) or
          (ammo_trucks[i].convoy_name =
field_list[1].str_val)) and
         (ammo_trucks[i].mission_assigned = 'Y') and
         ((ammo_trucks[i].node_to_resupply <> null_string)
or
          (ammo_trucks[i].firing_unit_to_resupply <>
null_string)) then
               begin
               ammo_trucks[i].mission_assigned := 'N';
               ammo_trucks[i].node_to_resupply :=
null_string;
               ammo_trucks[i].firing_unit_to_resupply :=
null_string
               end;
restore_screen
end;

(************************************************)
{
 Procedure name : FIRE_UNIT_AMMO_PICKUP

 Purpose : this procedure allows the player to
 issue an order to a fire unit to pick up ammo
 at a specified node.  This command must be
 issued with a corresponding movement order to
 get the fire unit to the specified node.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure fire_unit_ammo_pickup;

procedure set_up_fields;

begin
number_of_fields := 2;
```

```
with field_list[1] do
   begin
   label_string := 'Fire unit to pick up ammo :';
   label_x := 24;  label_y := 11;
   str_val := null_string;
   x1 := 53;  y1 := 11;  x2 := 57;  y2 := 11;
   field_type := eval; eval_function := valid_unit
   end;
with field_list[2] do
   begin
   label_string := 'Location of ammo :';
   label_x := 24;  label_y := 13;
   str_val := null_string;
   x1 := 44;  y1 := 13;  x2 := 48;  y2 := 13;
   field_type := eval; eval_function := valid_node
   end
end;

function valid_input : boolean;

var
   input_is_valid : boolean;

begin
input_is_valid := field_list[2].str_val <> null_string;
if input_is_valid then
   begin
   with firing_units[unit_number (field_list[1].str_val)] do

      input_is_valid := ammo_pickup_mission;
   if not input_is_valid then
      display_error_message ('COMMAND ERROR', null_string,
                             'fire unit already has an',
                             'ammo pickup mission assigned',
null_string)
   end;
valid_input := input_is_valid
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Fire Unit Ammunition Pickup', cyan);
repeat
   edit_screen (number_of_fields, field_list,
abort_allowed);
   field_list[1].str_val := upper_case
(field_list[1].str_val);
```

```
   field_list[2].str_val := upper_case
(field_list[2].str_val)
until (key = escape) or valid_input;
if key <> escape then
   with firing_units[unit_number (field_list[1].str_val)] do

      if not ammo_pickup_mission then
         begin
         ammo_pickup_mission := true;
         ammo_pickup_location := field_list[2].str_val
         end;
restore_screen
end;

(************************************************)
{
 Procedure name : CANCEL_FIRE_UNIT_PICKUP

 Purpose : this procedure allows the player to
 cancel an order to a fire unit to pick up ammo
 at a specified node as directed to by a
 FIRE_UNIT_AMMO_PICKUP order.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure cancel_fire_unit_pickup;

procedure set_up_fields;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Fire unit :';
   label_x := 33;  label_y := 12;
   str_val := null_string;
   x1 := 46;  y1 := 12;  x2 := 50;  y2 := 12;
   field_type := eval; eval_function := valid_unit
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Cancel Ammo Pickup Mission', cyan);
edit_screen (number_of_fields, field_list, abort_allowed);
```

```
field_list[1].str_val := upper_case (field_list[1].str_val);

if (key = f2) or (key = enter) then
   with firing_units[unit_number (field_list[1].str_val)] do

      if ammo_pickup_mission then
         begin
         ammo_pickup_mission := false;
         ammo_pickup_location := null_string
         end;
restore_screen
end;

(************************************************)
{
 Procedure name : AMMO_TRUCK_AMMO_PICKUP

 Purpose : this procedure allows the player to
 issue an order to a truck or convoy that is
 at the trains location and empty to proceed to
 atp for ammo resupply.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure ammo_truck_ammo_pickup;

var
   command_number : string6;
   new_event_1 : event_record;
   new_event_2 : event_record;
   i : integer;

procedure set_up_fields;

begin
number_of_fields := 2;
with field_list[1] do
   begin
   label_string := 'Ammo truck or convoy to pickup ammo :';
   label_x := 19;  label_y := 11;
   str_val := null_string;
   x1 := 58;  y1 := 11;  x2 := 62;  y2 := 11;
   field_type := eval; eval_function := trucks_at_atp
   end;
with field_list[2] do
   begin
   label_string := 'Departure time :';
   label_x := 19;  label_y := 13;
```

```
   str_val := game_dtg;
   x1 := 37;  y1 := 13;  x2 := 51;  y2 := 13;
   field_type := eval; eval_function := time_not_past
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Ammo Truck Ammunition Pickup', cyan);
put_string (55, 6, 'Command number :');
str (command_serial_number, command_number);
put_string (72, 6, command_number);
color_foreground (55,6,77,6, yellow);
edit_screen (number_of_fields, field_list, abort_allowed);
field_list[1].str_val := upper_case (field_list[1].str_val);

if key = f2 then
   begin
   with new_event_1 do
      begin
      event_type := 'D';
      serial_number := command_serial_number;
      dtg_to_timekey (field_list[2].str_val, time_key);
      unit_type := 'A';
      unit_name := null_string;
      node := null_string;
      path := 'ATP'
      end;
   new_event_2 := new_event_1;
   with new_event_2 do
      begin
      event_type := 'R';
      time_key := inc_dtg_to_timekey (field_list[2].str_val,

         abs (normal_rv
(game_parameters.avg_time_trains_to_atp, 10)));
      path := null_string
      end;
   if truck_number (field_list[1].str_val) <> 0 then
      begin
      new_event_1.unit_name := field_list[1].str_val;
      new_event_2.unit_name := field_list[1].str_val;
      new_event_2.node :=
         ammo_trucks[truck_number
(field_list[1].str_val)].location;
      ammo_trucks[truck_number (field_list[1].str_val)].
         mission_assigned := 'Y';
      add_event (new_event_1);
```

```
      add_event (new_event_2)
      end
   else
      begin
      for i := 1 to number_of_ammo_trucks do
         if ammo_trucks[i].convoy_name =
field_list[1].str_val then
            begin
            new_event_1.unit_name :=
ammo_trucks[i].bumper_number;
            new_event_2.unit_name :=
ammo_trucks[i].bumper_number;
            new_event_2.node := ammo_trucks[i].location;
            ammo_trucks[i].mission_assigned := 'Y';
            add_event (new_event_1);
            add_event (new_event_2)
            end
      end;
   incr (command_serial_number)
   end;
restore_screen
end;


(************************************************)
{
 Procedure name : ISSUE_FIRE_ORDER

 Purpose : this procedure allows the player to
 issue an order to a fire order to execute a
 fire order with the specified number of volleys
 and at the specified time.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                 }
(************************************************)
procedure issue_fire_order;

var
   command_number : string6;
   new_event : event_record;

procedure set_up_fields;

begin
number_of_fields := 3;
with field_list[1] do
   begin
   label_string := 'Unit to fire :';
```

```
   label_x := 25;  label_y := 10;
   str_val := null_string;
   x1 := 41;  y1 := 10;  x2 := 45;  y2 := 10;
   field_type := eval; eval_function := valid_unit
   end;
with field_list[2] do
   begin
   label_string := 'Time to fire :';
   label_x := 25;  label_y := 12;
   str_val := game_dtg;
   x1 := 41;  y1 := 12;  x2 := 55;  y2 := 12;
   field_type := eval; eval_function := time_not_past
   end;
with field_list[3] do
   begin
   label_string := 'Number of volleys :';
   label_x := 25;  label_y := 14;
   x1 := 46;  y1 := 14;  x2 := 48;  y2 := 14;
   field_type := int; int_min_value := 0; int_max_value :=
maxint;
   int_value := 1;
   str (int_value, str_val)
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Fire Order', cyan);
put_string (55, 6, 'Command number :');
str (command_serial_number, command_number);
put_string (72, 6, command_number);
color_foreground (55,6,77,6, yellow);
edit_screen (number_of_fields, field_list, abort_allowed);
field_list[1].str_val := upper_case (field_list[1].str_val);

if key = f2 then
   begin
   with new_event do
      begin
      event_type := 'F';
      serial_number := command_serial_number;
      dtg_to_timekey (field_list[2].str_val, time_key);
      unit_type := 'F';
      unit_name := field_list[1].str_val;
      volleys := field_list[3].int_value
      end;
   add_event (new_event);
   incr (command_serial_number)
```

```
   end;
restore_screen
end;

(************************************************)
{
 Procedure name : CHANGE_FIRING_RATE

 Purpose : this procedure allows the player to
 change the rate at which a unit fires during the
 execution of each time step.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure change_firing_rate;

procedure set_up_fields;

begin
number_of_fields := 2;
with field_list[1] do
   begin
   label_string := 'Unit to change :';
   label_x := 27;  label_y := 11;
   str_val := null_string;
   x1 := 45;  y1 := 11;  x2 := 49;  y2 := 11;
   field_type := eval; eval_function := valid_unit
   end;
with field_list[2] do
   begin
   label_string := 'Firing rate (% CSR) :';
   label_x := 27;  label_y := 13;
   str (1.0, str_val);
   x1 := 50;  y1 := 13;  x2 := 53;  y2 := 13;
   field_type := float; float_min_value := 0.01;
float_max_value := 1.0
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Change Unit Firing Rate', cyan);
edit_screen (number_of_fields, field_list, abort_allowed);
field_list[1].str_val := upper_case (field_list[1].str_val);
```

```
if key = f2 then
   firing_units[unit_number
(field_list[1].str_val)].rate_percent_csr :=
      field_list[1].float_value;
restore_screen
end;

(************************************************)
{
 Procedure name : CANCEL_COMMAND

 Purpose : this procedure allows the player to
 cancel a command that was issued with a time for
 execution if it has not passed.  It requires that
 the serial number of the command to be cancelled
 be entered.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                 }
(************************************************)
procedure cancel_command;

var
   record_number : longint;

procedure set_up_fields;

begin
number_of_fields := 1;
with field_list[1] do
   begin
   label_string := 'Number of command to cancel :';
   label_x := 24;  label_y := 12;
   str_val := null_string;
   x1 := 55;  y1 := 12;  x2 := 57;  y2 := 12;
   field_type := int;
   int_min_value := 0; int_max_value :=
command_serial_number - 1
   end
end;

begin
set_up_fields;
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Cancel Command', cyan);
edit_screen (number_of_fields, field_list, abort_allowed);
if (key = f2) or (key = enter) then
```

```
   begin
   findkey (serial_number_index, record_number,
field_list[1].str_val);
   while ok do
      begin
      delete_event (record_number);
      findkey (serial_number_index, record_number,
field_list[1].str_val)
      end
   end;
restore_screen
end;

(************************************************)
{
 Procedure name : MOVE_UNIT

 Purpose : this procedure allows the player to
 issue a movement order to any of the units in
 the game.  This is done by first entering the
 route for the movement and then the information
 for the unit and the time to movement.  It also
 allows the player to instruct a fire unit to
 resupply in route at a specified node.

 Parameters : none.

 Called by : ISSUE_COMMAND
                                                }
(************************************************)
procedure move_unit;

var
   command_number : string6;
   valid_input : boolean;

(************************************************)
{
 Procedure name : GET_ROUTE

 Purpose : this procedure allows the player to
 enter the route to taken by a unit that is to
 be moved with this movement order.  It allows
 a route with up to 10 nodes in it.

 Parameters : none.

 Called by : MOVE_UNIT
                                                }
(************************************************)
procedure get_route;
```

```
var
   i : integer;
   j : integer;

procedure set_up_fields;

var
   i : integer;

begin
number_of_fields := 2;
with field_list[1] do
   begin
   label_string := 'Start node :';
   label_x := 24;  label_y := 10;
   str_val := null_string;
   x1 := 38;  y1 := 10;  x2 := 42;  y2 := 10;
   field_type := eval; eval_function := valid_node_for_route

   end;
for i := 2 to 11 do
   with field_list[i] do
      begin
      label_string := 'along path :';
      label_x := 24;  label_y := 10 + i;
      str_val := null_string;
      x1 := 38;  y1 := 10 + i;  x2 := 42;  y2 := 10 + i;
      field_type := eval; eval_function :=
valid_path_in_route
      end
end;

begin
set_up_fields;
save_screen;
put_string (26,25, 'Accept route   ');
center_text (8, 'Enter route for movement', cyan);
for i := 1 to 2 do
   for j := 1 to 11 do
      route [i][j] := null_string;
current_path := 2;
valid_input := false;
repeat
   edit_screen (number_of_fields, field_list,
abort_allowed);
   for i := 1 to number_of_fields do
      field_list[i].str_val := upper_case
(field_list[i].str_val);
   if key <> escape then
      valid_input := field_list[2].str_val <> null_string
```

```
until (key = escape) or valid_input;
restore_screen
end;

(************************************************)
{
 Procedure name : GET_MOVEMENT_INFO

 Purpose : this procedure allows the player to
 enter the unit to be moved, the departure or
 arrival time, and, if it is a fire unit, the
 location for a resupply in route.

 Parameters : none.

 Called by : MOVE_UNIT
                                                }
(************************************************)
procedure get_movement_info;

procedure set_up_fields;

begin
number_of_fields := 3;
with field_list[1] do
   begin
   label_string := 'Unit to move :';
   label_x := 18;  label_y := 10;
   str_val := null_string;
   x1 := 34;  y1 := 10;  x2 := 38;  y2 := 10;
   field_type := eval; eval_function := valid_unit_to_move
   end;
with field_list[2] do
   begin
   label_string := 'Time for movement :';
   label_x := 18;  label_y := 12;
   str_val := game_dtg;
   x1 := 39;  y1 := 12;  x2 := 53;  y2 := 12;
   field_type := eval; eval_function := time_not_past
   end;
with field_list[3] do
   begin
   label_string := 'Is this a departure or arrival time
(D/A) :';
   label_x := 18;  label_y := 14;
   str_val := 'D';
   x1 := 63;  y1 := 14;  x2 := 63;  y2 := 14;
   field_type := ch;  valid_char_set := ['D','A']
   end;
with field_list[4] do
   begin
```

```
   label_string := 'Node to resupply at :';
   label_x := 18;  label_y := 16;
   str_val := null_string;
   x1 := 41;  y1 := 16;  x2 := 45;  y2 := 16;
   field_type := eval; eval_function :=
valid_node_for_resupply
   end
end;

begin
set_up_fields;
save_screen;
center_text (8, 'Enter movement information', cyan);
edit_screen (number_of_fields, field_list, abort_allowed);
field_list[1].str_val := upper_case (field_list[1].str_val);

field_list[4].str_val := upper_case (field_list[4].str_val);

valid_input := key = f2;
restore_screen
end;

(************************************************)
{
 Procedure name : CREATE_EVENT_RECORDS

 Purpose : this procedure creates the event
 records that correspond to the information
 entered in this movement order.

 Parameters : none.

 Called by : MOVE_UNIT
                                                 }
(************************************************)
procedure create_event_records;

type
   route_array = array [2..11] of integer;
var
   route_info : route_array;
   total_estimated_time  : integer;
   new_event : event_record;
   wait_event : event_record;
   wait_exists : boolean;
   command_start_time : datetime;
   command_time : string15;
   path_number : integer;
   i : integer;
```

```
procedure calculate_movement_times (var route_info :
route_array;
                                    var total_estimated_time
: integer);

var
   i : integer;
   path_number : integer;
   speed : real;
   calculated_time : integer;
   estimated_time : integer;
   total_calculated_time : integer;
   temp_path : path_record;
   tracked_vehicle : boolean;

begin
path_number := 2;
total_estimated_time := 0;
total_calculated_time := 0;
tracked_vehicle := unit_number (field_list[1].str_val) <> 0;

repeat
   taread (paths, temp_path, route[1][path_number],
exactmatch);
   if tracked_vehicle then
      begin
      estimated_time :=
         round (((1.0 /
game_parameters.avg_track_convoy_speed) * 60.0)
               * temp_path.length);
      case temp_path.road_condition of
         'P' : speed := 0.95 *
game_parameters.avg_track_convoy_speed;
         'M' : speed := 1.00 *
game_parameters.avg_track_convoy_speed;
         'G' : speed := 1.05 *
game_parameters.avg_track_convoy_speed
      end;
      calculated_time :=
         abs (normal_rv (round (((1.0 / speed) * 60.0) *
temp_path.length), 5))
      end
   else
      begin
      estimated_time :=
         round (((1.0 /
game_parameters.avg_wheel_convoy_speed) * 60.0)
               * temp_path.length);
      case temp_path.road_condition of
         'P' : speed := 0.95 *
game_parameters.avg_wheel_convoy_speed;
```

```
         'M' : speed := 1.00 *
game_parameters.avg_wheel_convoy_speed;
         'G' : speed := 1.05 *
game_parameters.avg_wheel_convoy_speed
      end;
      calculated_time :=
         abs (normal_rv (round (((1.0 / speed) * 60.0) *
temp_path.length), 5))
      end;
   route_info[path_number] := calculated_time;
   total_estimated_time := total_estimated_time +
estimated_time;
   total_calculated_time := total_calculated_time +
calculated_time;
   incr (path_number)
until (path_number > 11) or (route[1][path_number] =
null_string);
for i := path_number to 11 do
   route_info[i] := 0
end;

begin
calculate_movement_times (route_info, total_estimated_time);

dtg_to_datetime (field_list[2].str_val, command_start_time);

if field_list[3].str_val = 'A' then
   dec_time (command_start_time, total_estimated_time);
if time_relative (command_start_time, game_time) = before
then
   command_start_time := game_time;
datetime_to_dtg (command_start_time, command_time);
path_number := 1;
repeat
   wait_exists := false;
   if path_number = 1 then
      begin
      with new_event do
         begin
         event_type := 'D';
         serial_number := command_serial_number;
         dtg_to_timekey (command_time, time_key);
         path := route[1][path_number + 1]
         end
      end
   else if (path_number = 11) or (route[1][path_number + 1]
= null_string) then
      begin
      with new_event do
         begin
         event_type := 'O';
```

```
            serial_number := command_serial_number;
            time_key :=
               inc_dtg_to_timekey (command_time,
route_info[path_number]);
            node := route[2][path_number]
            end
         end
      else
         begin
         if route[2][path_number] = field_list[4].str_val then
            begin
            wait_exists := true;
            with wait_event do
               begin
               event_type := 'W';
               serial_number := command_serial_number;
               time_key :=
                  inc_dtg_to_timekey (command_time,
route_info[path_number]);
               node := route[2][path_number]
               end
            end;
         with new_event do
            begin
            event_type := 'T';
            serial_number := command_serial_number;
            if wait_exists then
               time_key :=
                 inc_dtg_to_timekey (command_time,
game_parameters.time_step_size)
            else
               time_key :=
                  inc_dtg_to_timekey (command_time,
route_info[path_number]);
            path := route[1][path_number + 1]
            end
         end;
      if unit_number (field_list[1].str_val) <> 0 then
         begin
         new_event.unit_type := 'F';
         new_event.unit_name := field_list[1].str_val;
         add_event (new_event);
         if wait_exists then
            begin
            wait_event.unit_type := 'F';
            wait_event.unit_name := field_list[1].str_val;
            add_event (wait_event)
            end
         end
      else if truck_number (field_list[1].str_val) <> 0 then
         begin
```

```
      new_event.unit_type := 'A';
      new_event.unit_name := field_list[1].str_val;
      add_event (new_event)
      end
   else if field_list[1].str_val = 'TRAIN' then
      begin
      new_event.unit_type := 'T';
      new_event.unit_name := field_list[1].str_val;
      add_event (new_event)
      end
   else
      begin
      new_event.unit_type := 'A';
      for i := 1 to number_of_ammo_trucks do
         if ammo_trucks[i].convoy_name =
field_list[1].str_val then
            begin
            new_event.unit_name :=
ammo_trucks[i].bumper_number;
            add_event (new_event)
            end
      end;
   incr (path_number)
until (path_number = 12) or (route[1][path_number] =
null_string)
end;

begin
save_screen;
clear_area (2,3,79,22);
display_command_help_line;
center_text (4, 'Move Unit', cyan);
put_string (55, 6, 'Command number :');
str (command_serial_number, command_number);
put_string (72, 6, command_number);
color_foreground (55,6,77,6, yellow);
get_route;
if valid_input then
   begin
   get_movement_info;
   if valid_input then
      begin
      create_event_records;
      incr (command_serial_number)
      end
   end;
restore_screen
end;


begin
```

```
suppress_messages := false
end.^Z
```

```
(************************************************)
{
 Unit name : GAMEUTIL

 Purpose : this unit contains a number of general
   purpose procedures that are specific to the
   game and that are used by procedures or
   functions that are in more than one unit.
                                                 }
(************************************************)
unit gameutil;

interface

uses dos, utility, global, taccess, tahigh;

type
   time_relation_type = (before, after, same);

var
   suppress_messages : boolean;

function  always_true (string_value : string80): boolean;
procedure inc_time (var time : datetime; increment :
integer);
procedure dec_time (var time : datetime; decrement :
integer);
procedure dtg_to_datetime (dtg : string15; var datetime_rec
: datetime);
procedure datetime_to_dtg (datetime_rec : datetime; var dtg
: string15);
procedure dtg_to_timekey (dtg : string15; var timekey :
string10);
function  time_relative (time1, time2 : datetime):
time_relation_type;
function  time_in_range (time1, time2, checktime :
datetime): boolean;
function  inc_dtg_to_timekey (var start_dtg : string15;
                              increment : integer  ):
string;
procedure determine_day_or_night;
function  normal_rv (mean : integer; sd : integer): integer;

function  unit_number (name : string5): integer;
function  truck_number (name : string5): integer;
procedure add_event (new_event : event_record);
procedure delete_event (record_number : longint);
function  valid_unit (string_value : string80): boolean;
function  valid_ammo_truck (string_value : string80):
boolean;
function  valid_convoy (string_value : string80): boolean;
```

```
function  valid_node (string_value : string80): boolean;
procedure close_all_files;

implementation

function  always_true (string_value : string80): boolean;

begin
always_true := true
end;

(************************************************)
{
 Procedure name : INC_TIME

 Purpose : this procedure increments a time that
   is represented by the Turbo provided datetime
   type and returns it.  The increment is to be
   specified in minutes.

 Parameters : TIME - datetime record to be
   incremented.
   INCREMENT - number of minutes to increment the
   datetime record.

 Called by :
                                                 }
(************************************************)
procedure inc_time (var time : datetime; increment :
integer);

begin
with time do
   begin
   min := min + increment;
   if min > 59 then
      begin
      hour := hour + (min div 60);
      min := min - ((min div 60) * 60)
      end;
   if hour > 23 then
      begin
      day := day + (hour div 24);
      hour := hour - ((hour div 24) * 24)
      end;
   if day > month_data [month].days then
      begin
      day := day - month_data [month].days;
      month := month + 1
      end;
   if month > 12 then
```

```
      begin
      month := 1;
      year := year + 1
      end
   end;
end;

(************************************************)
{
 Procedure name : DEC_TIME

 Purpose : this procedure decrements a time that
   is represented by the Turbo provided datetime
   type and returns it.  The decrement is to be
   specified in minutes.

 Parameters : TIME - datetime record to be
   decremented.
   DECREMENT - number of minutes to decrement the
   datetime record.

 Called by :
                                                  }
(************************************************)
procedure dec_time (var time : datetime; decrement :
integer);

var
   dec_mins  : integer;
   dec_hours : integer;
   dec_days  : integer;

begin
dec_days  := decrement div 1440;
decrement := decrement mod 1440;
dec_hours := decrement div 60;
decrement := decrement mod 60;
dec_mins  := decrement;
with time do
   begin
   if min >= dec_mins then
      min := min - dec_mins
   else
      begin
      min := 60 - (dec_mins - min);
      incr (dec_hours)
      end;
   if hour >= dec_hours then
      hour := hour - dec_hours
   else
      begin
```

```
            hour := 24 - (dec_hours - hour);
            incr (dec_days)
            end;
         while dec_days >= day do
            begin
            dec_days := dec_days - day;
            if month > 1 then
               month := month - 1
            else
               begin
               month := 12;
               year := year - 1
               end;
            day := month_data[month].days
            end;
         day := day - dec_days
         end
      end;

      (************************************************)
      {
       Procedure name : DTG_TO_DATETIME

       Purpose : this procedure converts a string that
         represents a valid dtg and converts it to the
         date and time in the form of the datetime
         type record.

       Parameters : DTG - string that contains the dtg
         to be converted.
         DATETIME_REC - record to contain the result of
         the conversion.

       Called by :
                                                      }
      (************************************************)
      procedure dtg_to_datetime (dtg : string15; var datetime_rec
      : datetime);

      var
         error_code : integer;

      begin
      dtg := remove_blanks (dtg);
      val (copy (dtg, 1, 2), datetime_rec.day , error_code);
      if copy (dtg, 3, 2) = '24' then
         datetime_rec.hour := 0
      else
         val (copy (dtg, 3, 2), datetime_rec.hour, error_code);
      val (copy (dtg, 5, 2), datetime_rec.min , error_code);
```

```
datetime_rec.month := ord (str_to_month (copy (dtg, 8, 3)))
+ 1;
val (copy (dtg, 11, 2), datetime_rec.year, error_code);
datetime_rec.year := datetime_rec.year + 1900
end;

(************************************************)
{
 Procedure name : DATETIME_TO_DTG

 Purpose : this procedure converts a datetime
   record into a string that represents a valid
   dtg.

 Parameters : DATETIME_REC - record to be con-
   verted into a dtg string.
   DTG - string to contain the converted datetime
   record.

 Called by :
                                                 }
(************************************************)
procedure datetime_to_dtg (datetime_rec : datetime; var dtg
: string15);

var
   string_val : string15;

begin
with datetime_rec do
   begin
   str (day, string_val);
   if day < 10 then
      begin
      dtg [1] := '0';
      dtg [2] := string_val [1]
      end
   else
      begin
      dtg [1] := string_val [1];
      dtg [2] := string_val [2]
      end;
   dtg [3] := blank;
   str (hour, string_val);
   if (hour = 0) and (min = 0) then
      begin
      dtg [4] := '2';
      dtg [5] := '4'
      end
   else if hour < 10 then
      begin
```

```
            dtg [4] := '0';
            dtg [5] := string_val [1]
            end
         else
            begin
            dtg [4] := string_val [1];
            dtg [5] := string_val [2]
            end;
         str (min, string_val);
         if min < 10 then
            begin
            dtg [6] := '0';
            dtg [7] := string_val [1]
            end
         else
            begin
            dtg [6] := string_val [1];
            dtg [7] := string_val [2]
            end;
         dtg [8] := 'Z';
         dtg [9] := blank;
         insert (month_data [month].name, dtg, 10);
         dtg [13] := blank;
         str (year - 1900, string_val);
         insert (string_val, dtg, 14)
         end
      end;

      (************************************************)
      {
       Procedure name : DTG_TO_TIMEKEY

       Purpose : this procedure takes a string that
         is a valid dtg and converts it to a string that
         contains the same information but can be used
         as an alphabetic key for event records.

       Parameters : DTG - string that contains the valid
         dtg to converted for use as a key.
         TIMEKEY - string that will contain the key
         after conversion from dtg format.

       Called by :
                                                       }
      (************************************************)
      procedure dtg_to_timekey (dtg : string15; var timekey :
      string10);

      var
         month : month_type;
         month_val : string;
```

```
begin
dtg := remove_blanks (dtg);
insert (copy (dtg, 11, 2), timekey, 1);
month := str_to_month (copy (dtg, 8, 3));
str (ord (month) + 1, month_val);
if ord (month) < 10 then
   insert ('0' + month_val, timekey, 3)
else
   insert (month_val, timekey, 3);
insert (copy (dtg, 1, 6), timekey, 5)
end;

(************************************************)
{
 Procedure name : TIME_RELATIVE

 Purpose : this procedure compares two records of
   type datetime and determines whether the first
   one represents a time that occurs before, after
   or at the same time as the second record.

 Parameters : TIME1 - the first record of type
   datetime to be compared against.
   TIME2 - the second record of type datetime to
   be compared against the first one.

 Called by :
                                                 }
(************************************************)
function  time_relative (time1, time2 : datetime):
time_relation_type;

begin
if time1.year < time2.year then time_relative := before
else if time1.year > time2.year then time_relative := after
else
   begin
   if time1.month < time2.month then time_relative := before

   else if time1.month > time2.month then time_relative :=
after
   else
      begin
      if time1.day < time2.day then time_relative := before
      else if time1.day > time2.day then time_relative :=
after
      else
         begin
         if time1.hour < time2.hour then time_relative :=
before
```

```
            else if time1.hour > time2.hour then time_relative
        := after
            else
               begin
               if time1.min < time2.min then time_relative :=
        before
               else if time1.min > time2.min then time_relative
        := after
               else
                  time_relative := same
               end
            end
         end
      end
    end;

    (*************************************************)
    {
     Procedure name : TIME_IN_RANGE

     Purpose : this procedure will check the third
       parameter of type datetime and indicate whether
       or not it falls between the times represented
       by the first two datetime records.

     Parameters : TIME1 - the first record of type
       datetime to be compared against.
       TIME2 - the second record of type datetime to
       be compared against.
       CHECKTIME - the record of type datetime to be
       compared to the interval represented by TIME1
       and TIME2.

     Called by :
                                                      }
    (*************************************************)
    function time_in_range (time1, time2, checktime : datetime):
    boolean;

    begin
    if time_relative (time1, time2) = before then
       time_in_range := ( (time_relative (checktime, time1) in
    [after, same]) and
                          (time_relative (checktime, time2) in
    [before, same]) )
    else
       time_in_range := ( (time_relative (checktime, time2) in
    [after, same]) and
                          (time_relative (checktime, time1) in
    [before, same]) )
    end;
```

```
(************************************************)
{
 Procedure name : INC_DTG_TO_TIMEKEY

 Purpose : this procedure will take a string that
   represents a valid dtg and increment the time
   and date that it represents by the specified
   number of minutes and return the result in the
   form of a string to be used as a key to an
   event record.

 Parameters : START_DTG - the string that
   represents a valid dtg to be incremented.
   INCREMENT - the number of minutes to
   increment the time represented by the dtg.

 Called by :
                                                 }
(************************************************)
function inc_dtg_to_timekey (var start_dtg : string15;
                             increment : integer  ): string;


var
   final_datetime : datetime;
   final_dtg : string15;
   final_timekey : string10;

begin
dtg_to_datetime (start_dtg, final_datetime);
inc_time (final_datetime, increment);
datetime_to_dtg (final_datetime, final_dtg);
dtg_to_timekey (final_dtg, final_timekey);
start_dtg := final_dtg;
inc_dtg_to_timekey := final_timekey
end;

(************************************************)
{
 Procedure name : DETERMINE_DAY_OR_NIGHT

 Purpose : this procedure uses the BMNT end EENT
   entered by the player to determine whether the
   game time is at day or night.

 Parameters : none.

 Called by :
                                                 }
(************************************************)
```

```
procedure determine_day_or_night;

var
   sun_up : integer;
   sun_down : integer;
   error : integer;

begin
val (copy (commanders_guidance.bmnt, 1, 4), sun_up, error);
val (copy (commanders_guidance.eent, 1, 4), sun_down,
error);
day_time := (((game_time.hour * 100) + game_time.min) >=
sun_up) and
            (((game_time.hour * 100) + game_time.min) <
sun_down)
end;

(************************************************)
{
 Procedure name : NORMAL_RV

 Purpose : this procedure will return a normally
   distributed random variable with the specified
   mean and standard deviation.

 Parameters : MEAN - the mean of the distribution.
   SD - the standard deviation of the
   distribution.

 Called by :
                                                 }
(************************************************)
function  normal_rv (mean : integer; sd : integer): integer;


var
   x_random : real;
   y_random : real;
   y_calculated : real;

begin
repeat
   x_random := (random * 6.0) - 3.0;
   y_random := random * (1 / sqrt (2.0 * pi));
   y_calculated := exp (-1.0 * sqr (x_random) / 2.0) / sqrt
(2.0 * pi)
until y_random <= y_calculated;
normal_rv := round (x_random * sd + mean)
end;

(************************************************)
```

```
{
 Procedure name : UNIT_NUMBER

 Purpose : this procedure will take the name of
   a unit and return the number of that unit in
   the array of firing units.  It will return a
   0 if the name doews not correspond to a
   firing unit.

 Parameters : NAME - the name of the firing unit
   to be checked.

 Called by :
                                                  }
(************************************************)
function unit_number (name : string5): integer;

var
   firing_unit_number : integer;

begin
firing_unit_number := 0;
repeat
   incr (firing_unit_number)
until (firing_unit_number = number_of_firing_units) or
      (name =
firing_units[firing_unit_number].firing_unit_name);
if name <> firing_units[firing_unit_number].firing_unit_name
then
   unit_number := 0
else
   unit_number := firing_unit_number
end;

(************************************************)
{
 Procedure name : TRUCK_NUMBER

 Purpose : this procedure will take the bumper
   number of a truck and return the number of that
   truck in the array of ammo trucks.  It will
   return a value of 0 if the bumper number does
   not correspond to a truck.

 Parameters : NAME - the bumper number of the
   truck to be checked.

 Called by :
                                                  }
(************************************************)
function truck_number (name : string5): integer;
```

```
var
   ammo_truck_number : integer;

begin
ammo_truck_number := 0;
repeat
   incr (ammo_truck_number)
until (ammo_truck_number = number_of_ammo_trucks) or
      (name = ammo_trucks[ammo_truck_number].bumper_number);

if name <> ammo_trucks[ammo_truck_number].bumper_number then

   truck_number := 0
else
   truck_number := ammo_truck_number
end;

(************************************************)
{
 Procedure name : ADD_EVENT

 Purpose : this procedure will take the event
   passed to it and add it to the file that
   contains the events list of events not yet
   executed.

 Parameters : NEW_EVENT - the event record to be
   added to the events list.

 Called by :
                                                 }
(************************************************)
procedure add_event (new_event : event_record);

var
   record_number : longint;
   command_number : string6;

begin
str (new_event.serial_number, command_number);
addrec (event_list, record_number, new_event);
addkey (time_index, record_number, new_event.time_key);
addkey (serial_number_index, record_number, command_number);

flushfile (event_list);
flushindex (time_index);
flushindex (serial_number_index);
if new_event.event_type = 'D' then
   begin
   suppress_messages := true;
```

```
   case new_event.unit_type of
      'T' : battalion_trains.pending_movement := true;
      'A' : ammo_trucks[truck_number
(new_event.unit_name)].pending_movement :=
                true;
      'F' : firing_units[unit_number
(new_event.unit_name)].pending_movement :=
                true
   end;
   suppress_messages := false
   end
end;

(************************************************)
{
 Procedure name : DELETE_EVENT

 Purpose : this procedure will take the event
   corresponding to the record number passed to
   the procedure and delete it from the events
   list.  If the event was related to a cancelled
   movement order it also takes the unit involved
   and moves it to the nearest node if it was
   on a path.

 Parameters : RECORD_NUMBER - the number of the
   record to be deleted from the events list.

 Called by :
                                                 }
(************************************************)
procedure delete_event (record_number : longint);

var
   event : event_record;
   command_number : string6;
   temp_path : path_record;

begin
getrec (event_list, record_number, event);
str (event.serial_number, command_number);
if event.event_type in ['O','T','W','D'] then
   begin
   suppress_messages := true;
   case event.unit_type of
      'T' : with battalion_trains do
               begin
               if not valid_node (battalion_trains.location)
then
                  begin
```

```
                    taread (paths, temp_path, location,
exactmatch);
                    location := temp_path.start_node
                    end;
                 pending_movement := false
                 end;
      'A' : with ammo_trucks[truck_number (event.unit_name)]
do
                 begin
                 if not valid_node
                    (ammo_trucks[truck_number
(event.unit_name)].location) then
                       begin
                       taread (paths, temp_path, location,
exactmatch);
                       location := temp_path.start_node
                       end;
                 pending_movement := false
                 end;
      'F' : with firing_units[unit_number (event.unit_name)]
do
                 begin
                 if not valid_node
                    (firing_units[unit_number
(event.unit_name)].location) then
                       begin
                       taread (paths, temp_path, location,
exactmatch);
                       location := temp_path.start_node
                       end;
                 pending_movement := false
                 end
   end;
   suppress_messages := false
   end;
deletekey (serial_number_index, record_number,
command_number);
deletekey (time_index, record_number, event.time_key);
deleterec (event_list, record_number);
flushfile (event_list);
flushindex (time_index);
flushindex (serial_number_index)
end;

(************************************************)
{
 Procedure name : VALID_UNIT

 Purpose : this procedure will determine whether
   or not the name passed corresponds to a valid
   unit that it still alive.
```

```
 Parameters : STRING_VALUE - name of unit to be
   checked.

 Called by :
                                                    }
(*************************************************)
{$F+}
function valid_unit (string_value : string80): boolean;

var
   unit_exists : boolean;

begin
string_value := upper_case (string_value);
unit_exists := (unit_number (string_value) <> 0) and
   (firing_units[unit_number (string_value)].
      sections_in_operating_condition > 0.0);
if (not suppress_messages) and (not unit_exists) then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'unit entered does not exist',
null_string);
valid_unit := unit_exists
end;
{$F-}

(*************************************************)
{
 Procedure name : VALID_AMMO_TRUCK

 Purpose : this procedure will determine whether
   or not the name passed corresponds to a valid
   truck that is still alive.

 Parameters : STRING_VALUE - name of truck to be
   checked.

 Called by :
                                                    }
(*************************************************)
{$F+}
function valid_ammo_truck (string_value : string80):
boolean;

var
   ammo_truck_good : boolean;

begin
string_value := upper_case (string_value);
ammo_truck_good := (truck_number (string_value) <> 0) and
```

```
   (ammo_trucks [truck_number
(string_value)].effective_percent > 0.0) and
   (ammo_trucks[truck_number (string_value)].convoy_name =
null_string);
if (not suppress_messages) and (not ammo_truck_good) then
   display_error_message ('INPUT ERROR', null_string,
                          'truck entered does not exist',
                          'or is part of another convoy',
null_string);
valid_ammo_truck := ammo_truck_good
end;
{$F-}

(************************************************)
{
 Procedure name : VALID_CONVOY

 Purpose : this procedure will determine whether
   or not the name passed corresponds to a valid
   convoy that is still in operation.

 Parameters : STRING_VALUE - name of convoy to be
   checked.

 Called by :
                                                  }
(************************************************)
{$F+}
function valid_convoy (string_value : string80): boolean;

var
   ammo_truck_number : integer;
   convoy_exists : boolean;

begin
string_value := upper_case (string_value);
convoy_exists := false;
if string_value <> null_string then
   for ammo_truck_number := 1 to number_of_ammo_trucks do
       if (ammo_trucks[ammo_truck_number].effective_percent
> 0.0) and
          (string_value =
ammo_trucks[ammo_truck_number].convoy_name) then
             convoy_exists := true;
if (not suppress_messages) and (not convoy_exists) then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'convoy does not exists',
null_string);
valid_convoy := convoy_exists
end;
```

```
{$F-}

(************************************************)
{
 Procedure name : VALID_NODE

 Purpose : this procedure will determine whether
   or not the name passed corresponds to a valid
   node created for this scenario.

 Parameters : STRING_VALUE - name of node to be
   checked.

 Called by :
                                                 }
(************************************************)
{$F+}
function valid_node (string_value : string80): boolean;

var
   temp_node : node_record;
   node_exists : boolean;

begin
string_value := upper_case (string_value);
taread (nodes, temp_node, string_value, exactmatch);
node_exists := ok;
if (not suppress_messages) and (not node_exists) then
   display_error_message ('INPUT ERROR', null_string,
null_string,
                          'node does not exists',
null_string);
valid_node := node_exists
end;
{$F-}

(************************************************)
{
 Procedure name : CLOSE_ALL_FILES

 Purpose : this procedure will close all files
   that are used throughout the game.

 Parameters : none.

 Called by :
                                                 }
(************************************************)
procedure close_all_files;

begin
```

```
        taclose (nodes);
        taclose (paths);
        closefile (event_list);
        closeindex (time_index);
        closeindex (serial_number_index);
        closefile (messages);
        closeindex (message_type_index)
        end;


        begin
        end.^Z
```

```
(************************************************)
{
 Unit name : UTILITY

 Purpose : this unit contains a number of general
   purpose procedures that perform functions
   related to input and output of information
   using menus and full screen editing.
                                                 }
(************************************************)
unit utility;

interface
{$I-}

uses dos, crt;

const
   null_string = '';
   blank       = #32;
   null        = 00;
   backspace   = 08;
   enter       = 13;
   escape      = 27;
   space       = 32;
   f1          = 59  shl 8;
   f2          = 60  shl 8;
   f3          = 61  shl 8;
   f4          = 62  shl 8;
   f5          = 63  shl 8;
   f6          = 64  shl 8;
   f7          = 65  shl 8;
   f8          = 66  shl 8;
   f9          = 67  shl 8;
   f10         = 68  shl 8;
   f11         = 133 shl 8;
   f12         = 134 shl 8;
   home_key    = 71  shl 8;
   up_arrow    = 72  shl 8;
   page_up     = 73  shl 8;
   left_arrow  = 75  shl 8;
   right_arrow = 77  shl 8;
   end_key     = 79  shl 8;
   down_arrow  = 80  shl 8;
   page_down   = 81  shl 8;
   insert_key  = 82  shl 8;
   delete_key  = 83  shl 8;
   menu_x1_default = 1;
   menu_x2_default = 80;
   menu_y1_default = 3;
   menu_y2_default = 23;
```

```
      menu_forecolor_default   = white;
      menu_backcolor_default   = blue;
      option_forecolor_default = blue;
      option_backcolor_default = lightgray;
      field_forecolor_default  = yellow;
      field_backcolor_default  = magenta;
      max_fields_per_screen = 25;
      abort_allowed = true;
   type
      string1  = string [1];
      string2  = string [2];
      string3  = string [3];
      string4  = string [4];
      string5  = string [5];
      string6  = string [6];
      string7  = string [7];
      string8  = string [8];
      string9  = string [9];
      string10 = string [10];
      string15 = string [15];
      string20 = string [20];
      string25 = string [25];
      string30 = string [30];
      string35 = string [35];
      string40 = string [40];
      string45 = string [45];
      string50 = string [50];
      string55 = string [55];
      string60 = string [60];
      string65 = string [65];
      string70 = string [70];
      string75 = string [75];
      string80 = string [80];
      option_string = string;
      eval_function_type = function (string_value : string80):
   boolean;
      month_type = (JAN, FEB, MAR, APR, MAY, JUN,
                    JUL, AUG, SEP, OCT, NOV, DEC,
   INVALID_MONTH);
      data_type = (int, float, ch, strg, eval, enum, dtg,
   time);
      month_record = record
         days : integer;
         name : string3
      end;
      field_record = record
         label_string : string80;
         label_x, label_y : integer;
         str_val : string80;
         x1, y1, x2, y2 : integer;
         case field_type : data_type of
```

```
         int   : ( int_value : longint;
                   int_min_value,
                   int_max_value : integer );
         float : ( float_value : real;
                   float_min_value,
                   float_max_value : real );
         ch    : ( valid_char_set : set of char );
         enum  : ( number_of_enum_values : integer;
                   valid_enum_values : array [1..10] of
string10 );
         eval  : ( eval_function : eval_function_type );

      end;
   field_array = array [1..max_fields_per_screen] of
field_record;
   font = array [1..7] of array [1..7] of char;

var
   key : integer;
   menu_x1, menu_x2, menu_y1, menu_y2 : integer;
   menu_forecolor,   menu_backcolor   : integer;
   option_forecolor, option_backcolor : integer;
   field_forecolor , field_backcolor  : integer;
   default_mask : dirstr;
   letters : array ['A'..'Z'] of font;
   digits  : array ['0'..'9'] of font;
   fontfile : file of font;
   number_of_fields : integer;
   field_list : field_array;
   month_data : array [1..12] of month_record;

procedure incr (var number : integer);
procedure decr (var number : integer);
function  remove_blanks (string_value : string80): string80;

function  upper_case (string_value : string80): string80;
function  str_to_month (string_value : string3): month_type;

function  valid_dtg (string_value : string15): boolean;
function  valid_time (string_value : string5): boolean;
function  insert_on : boolean;
procedure toggle_insert;
function  get_key : integer;
procedure initialize_screen;
procedure save_screen;
procedure restore_screen;
procedure print_screen;
procedure remove_cursor;
procedure restore_cursor;
procedure beep;
function  printer_ready : boolean;
```

```
procedure color_background (x1, y1, x2, y2, color :
integer);
procedure color_foreground (x1, y1, x2, y2, color :
integer);
procedure clear_area (x1, y1, x2, y2 : integer);
procedure put_char (column, line : integer; character :
char);
function  get_char (column, row : integer): char;
procedure put_string (column, line : integer; text :
string80);
procedure put_font_string (column, line : integer; text :
string10; color : integer);
procedure center_text (line : integer; text : string80;
color : integer);
procedure draw_window (x1, y1, x2, y2 : integer;
                       forecolor, backcolor : integer;
                       title : string80);
procedure shade_window (x1, y1, x2, y2, color : integer);
procedure display_error_message (title, line1, line2, line3,
line4 : string55);
function  edit_field (field : field_record): string80;
procedure display_edit_screen_help_line;
procedure display_edit_list_help_line;
procedure display_command_help_line;
procedure display_add_record_help_line;
procedure edit_screen (var number_of_fields : integer;
                       var field_data : field_array;
                       abort_possible : boolean);
function  menu_selection (menu_title : string80;
                          menu_options : option_string):
integer;
function  get_file (prompt, message : string80; search_mask
: dirstr): pathstr;


implementation

type
   screen = array [1..25] of array [1..80] of integer;
var
   keyboard_status : byte absolute $0040:$0017;
   register_values : registers;
   screen_image : screen;
   cursor_x : integer;
   cursor_y : integer;
   stack_pointer : integer;
   screen_image_stack : array [1..7] of screen;
   cursor_x_stack     : array [1..7] of integer;
   cursor_y_stack     : array [1..7] of integer;
   loop : char;
const
```

```
   screen_location : ^screen = ptr ($B800,$0000);

procedure incr (var number : integer);

begin
number := number + 1
end;

procedure decr (var number : integer);

begin
number := number - 1
end;

function remove_blanks (string_value : string80): string80;

var
   position : byte;

begin
position := pos (blank, string_value);
while position <> 0 do
   begin
   delete (string_value, position, 1);
   position := pos (blank, string_value)
   end;
remove_blanks := string_value
end;

function  upper_case (string_value : string80): string80;

var
   position : byte;

begin
for position := 1 to length (string_value) do
   string_value [position] := upcase (string_value
[position]);
upper_case := string_value
end;

function  str_to_month (string_value : string3): month_type;


begin
if string_value = 'JAN' then
   str_to_month := JAN
else if string_value = 'FEB' then
   str_to_month := FEB
else if string_value = 'MAR' then
   str_to_month := MAR
```

```
else if string_value = 'APR' then
   str_to_month := APR
else if string_value = 'MAY' then
   str_to_month := MAY
else if string_value = 'JUN' then
   str_to_month := JUN
else if string_value = 'JUL' then
   str_to_month := JUL
else if string_value = 'AUG' then
   str_to_month := AUG
else if string_value = 'SEP' then
   str_to_month := SEP
else if string_value = 'OCT' then
   str_to_month := OCT
else if string_value = 'NOV' then
   str_to_month := NOV
else if string_value = 'DEC' then
   str_to_month := DEC
else
   str_to_month := INVALID_MONTH
end;

function  valid_dtg (string_value : string15): boolean;

var
   date  : longint;
   time  : longint;
   year  : longint;
   zone  : char;
   month : month_type;
   error_code : integer;

begin
string_value := upper_case (string_value);
string_value := remove_blanks (string_value);
if length (string_value) = 12 then
   begin
   val (copy (string_value, 1, 2),  date, error_code);
   val (copy (string_value, 3, 4),  time, error_code);
   val (copy (string_value, 11, 2), year, error_code);
   zone :=  char (string_value [7]);
   month := str_to_month (copy (string_value, 8, 3));
   valid_dtg :=  (error_code = 0) and
                 (month <> INVALID_MONTH) and
                 (date > 0) and (date <= month_data [ord
(month)].days) and
                 (time >= 0) and (time <= 2400) and
                 (zone in ['A'..'Z']) and
                 (year >= 70) and (year <= 99)
   end
else
```

```
   valid_dtg := false
end;

function  valid_time (string_value : string5): boolean;

var
   time : longint;
   zone : char;
   error_code : integer;

begin
string_value := upper_case (string_value);
val (copy (string_value, 1, 4),  time, error_code);
zone := char (string_value [5]);
valid_time :=  (error_code = 0) and
               (time >= 0) and (time <= 2400) and
               (zone in ['A'..'Z'])
end;

function insert_on : boolean;

begin
insert_on := ((keyboard_status and $80) = $80)
end;

procedure toggle_insert;

begin
if insert_on then
   keyboard_status := keyboard_status and $7F
else
   keyboard_status := keyboard_status or $80
end;

function get_key : integer;

var
   key_value : integer;

begin
key_value := ord (readkey);
if key_value = null then
   key_value := ord (readkey) shl 8;
get_key := key_value
end;

procedure initialize_screen;

begin
window (1,1,80,25);
textbackground (black);
```

```
textcolor (yellow);
clrscr
end;

procedure save_screen;

begin
incr (stack_pointer);
screen_image_stack [stack_pointer] := screen_location^;
cursor_x_stack [stack_pointer] := wherex;
cursor_y_stack [stack_pointer] := wherey
end;

procedure restore_screen;

begin
screen_location^ := screen_image_stack [stack_pointer];
gotoxy (cursor_x_stack [stack_pointer], cursor_y_stack
[stack_pointer]);
decr (stack_pointer)
end;

procedure print_screen;

begin
intr ($5, register_values)
end;

procedure remove_cursor;

begin
gotoxy (1,1);
register_values.ax := $0100;
register_values.cx := $2607;
intr ($10, register_values)
end;

procedure restore_cursor;

begin
register_values.ax := $0100;
register_values.cx := $0607;
intr ($10, register_values)
end;

procedure beep;

begin
sound (1000);
delay (100);
nosound
```

```
end;

function printer_ready : boolean;

begin
register_values.ah := $02;
register_values.dx := $00;
intr ($17, register_values);
printer_ready := (register_values.ah and $08) = $00
end;

procedure color_background (x1, y1, x2, y2, color :
integer);

var
   line,
   column : integer;
   value  : integer;

begin
for line := y1 to y2 do
   for column := x1 to x2 do
      begin
      value := screen_location^ [line,column];
      value := (value and $0FFF) + (color shl 12);
      screen_location^ [line,column] := value
      end
end;

procedure color_foreground (x1, y1, x2, y2, color :
integer);

var
   line,
   column : integer;
   value  : integer;

begin
for line := y1 to y2 do
   for column := x1 to x2 do
      begin
      value := screen_location^ [line,column];
      value := (value and $F0FF) + (color shl 8);
      screen_location^ [line,column] := value
      end
end;

procedure clear_area (x1, y1, x2, y2 : integer);

var
   line,
```

```
   column : integer;
   value  : integer;

begin
for line := y1 to y2 do
   for column := x1 to x2 do
      begin
      value := screen_location^ [line,column];
      value := (value and $FF00) + ord (space);
      screen_location^ [line,column] := value
      end
end;

procedure put_char (column, line : integer; character :
char);

var
   value : integer;

begin
value := screen_location^ [line,column];
value := (value and $FF00) + ord (character);
screen_location^ [line,column] := value
end;

function get_char (column, row : integer): char;

begin
get_char := chr ( lo (screen_location^ [row, column]))
end;

procedure put_string (column, line : integer; text :
string80);

var
   i : integer;

begin
for i := 1 to length (text) do
   put_char (column + i - 1, line, text [i])
end;

procedure put_font_string (column, line : integer; text :
string10; color : integer);

var
   k : integer;

procedure put_font_char (column, line : integer; character :
char; color : integer);
```

```
var
   i, j : integer;

begin
if (column < 74) and (line < 19) then
   for i := 1 to 7 do
      for j := 1 to 7 do
         begin
         if (character in ['a'..'z']) or (character in
['A'..'Z']) then
            put_char (i + column - 1, j + line - 1, letters
[character] [j, i])
         else if character in ['0'..'9'] then
            put_char (i + column - 1, j + line - 1, digits
[character] [j, i])
         else
            put_char (i + column - 1, j + line - 1, blank);
         color_foreground (i + column - 1, j + line - 1,
                           i + column - 1, j + line - 1,
color)
         end
end;

begin
for k := 0 to length (text) - 1 do
   put_font_char (column + (k * 8), line, text [k+1], color)

end;

procedure center_text (line : integer; text : string80;
color : integer);

var
   indent : integer;

begin
indent := (((lo(windmax) - lo(windmin) + 1) - length (text))
div 2) + 1;
put_string ((lo(windmin) + indent), line, text);
color_foreground ((lo(windmin) + indent), line,
                  (lo(windmin) + indent + length (text) -
1), line, color)
end;

procedure draw_window (x1, y1, x2, y2 : integer;
                       forecolor, backcolor : integer;
                       title : string80);

var
   line,
   column,
```

```
   indent : integer;

begin
clear_area (x1,y1,x2,y2);
color_background (x1,y1,x2,y2, backcolor);
color_foreground (x1,y1,x2,y2, forecolor);
for column := x1 to x2 do
   begin
   put_char (column, y1, #205);
   put_char (column, y2, #205);
   end;
for line := y1 to y2 do
   begin
   put_char (x1, line, #186);
   put_char (x2, line, #186);
   end;
put_char (x1, y1, #201);
put_char (x1, y2, #200);
put_char (x2, y1, #187);
put_char (x2, y2, #188);
if length (title) <> 0 then
   begin
   indent := ((x2 - x1 + 1) - (length (title) + 2)) div 2;
   put_string (x1 + indent,  y1, ' '+ title +' ')
   end
end;

procedure shade_window (x1, y1, x2, y2, color : integer);

var
   position : integer;

begin
position := x1 + 1;
repeat
   put_char (position, y2 + 1, #219);
   color_foreground (position, y2 + 1, position, y2 + 1,
color);
   incr (position)
until position > x2 + 2;
position := y1 + 1;
repeat
   put_char (x2 + 1, position, #219);
   put_char (x2 + 2, position, #219);
   color_foreground (x2 + 1, position, x2 + 2, position,
color);
   incr (position)
until position > y2
end;
```

```
procedure display_error_message (title, line1, line2, line3,
line4 : string55);

begin
save_screen;
draw_window (21,9,60,17, yellow, red, title);
shade_window (21,9,60,17, black);
center_text (11, line1, yellow);
center_text (12, line2, yellow);
center_text (13, line3, yellow);
center_text (14, line4, yellow);
center_text (16, 'press any key', white);
key := get_key;
key := null;
restore_screen
end;

function edit_field (field : field_record): string80;

var
   screen_string : string80;
   exit_edit_field : boolean;
   i : integer;

procedure highlight_field (field : field_record);

begin
with field do
   begin
   color_background (x1-1, y1, x2+1, y2, field_backcolor);
   color_foreground (x1-1, y1, x2+1, y2, field_forecolor);
   if field_type in [strg, dtg] then
      begin
      i := x2 + 1;
      repeat
         decr (i);
         gotoxy (i + 1, y1)
      until (i = x1 - 1) or (get_char (i, y1) <> blank)
      end
   else
      gotoxy (x1, y1)
   end
end;

procedure restore_field (field : field_record);

begin
with field do
   begin
   color_background (x1-1, y1, x2+1, y2, menu_backcolor);
   color_foreground (x1-1, y1, x2+1, y2, menu_forecolor);
```

```
      if field_type in [strg, dtg] then
         begin
         i := x2 + 1;
         repeat
            decr (i);
            gotoxy (i + 1, y1)
         until (i = x1 - 1) or (get_char (i, y1) <> blank)
         end
      else
         gotoxy (x1, y1)
      end
   end;

   procedure display_edit_field_help_screen;

   begin
   save_screen;
   remove_cursor;
   draw_window  (16,5,65,19, blue, lightgray, 'help');
   shade_window (16,5,65,19, black);
   center_text (12, 'NO HELP AVAILABLE AT THIS TIME', blue);
   center_text (17, 'press any key to return', blue);
   key := get_key;
   key := null;
   restore_cursor;
   restore_screen
   end;

   begin
   restore_cursor;
   if insert_on then
      begin
      put_string (77,25, 'INS');
      color_foreground (77,25,79,25, blue)
      end
   else
      put_string (77,25, '   ');
   highlight_field (field);
   exit_edit_field := false;
   repeat
      key := get_key;
      case key of
         up_arrow, down_arrow, escape, enter,
         f2, f3, f4, f5, f6, f7, f8, f9, f10,
         page_up, page_down :
              exit_edit_field := true;
         f1 : display_edit_field_help_screen;
         insert_key : begin
                      if insert_on then
                         begin
                         put_string (77,25, 'INS');
```

```
                        color_foreground (77,25,79,25, blue)
                        end
                     else
                        put_string (77,25, '   ')
                     end;
      backspace : if wherex = field.x1 then
                        beep
                     else if wherex = field.x2 + 1 then
                        begin
                        gotoxy (wherex - 1, field.y1);
                        put_char (wherex, wherey, blank)
                        end
                     else
                        begin
                        gotoxy (wherex - 1, field.y1);
                        for i := wherex + 1 to field.x2 do
                           put_char (i - 1, field.y1, get_char
(i, field.y1));
                        put_char (field.x2, field.y1, blank)
                        end;
      delete_key : if wherex = field.x2 + 1 then
                        beep
                     else
                        begin
                        for i := wherex + 1 to field.x2 do
                           put_char (i - 1, field.y1, get_char
(i, field.y1));
                        put_char (field.x2, field.y1, blank)
                        end;
      left_arrow : if wherex = field.x1 then
                        beep
                     else
                       gotoxy (wherex - 1, field.y1);
      right_arrow : if wherex = field.x2 + 1 then
                        beep
                     else
                        gotoxy (wherex + 1, field.y1);
      home_key : gotoxy (field.x1, field.y1);
      end_key  : begin
                 i := field.x2 + 1;
                 repeat
                    decr (i);
                    gotoxy (i + 1, field.y1)
                 until (i = field.x1 - 1) or
                       (get_char (i, field.y1) <> blank)
                 end;
   else
      begin
      if insert_on then
         begin
         if wherex = field.x2 + 1 then
```

```
                beep
             else
                begin
                for i := field.x2 downto wherex + 1 do
                   put_char (i, field.y1, get_char (i - 1,
field.y1));
                put_char (wherex, field.y1, chr (key));
                gotoxy (wherex + 1, field.y1)
                end
             end
          else
             begin
             if wherex = field.x1 then
                begin
                for i := field.x1 to field.x2 do
                   put_char (i, field.y1, blank);
                put_char (field.x1, field.y1, chr (key));
                gotoxy (field.x1 + 1, field.y1)
                end
             else if wherex = field.x2 + 1 then
                beep
             else
                begin
                put_char (wherex, field.y1, chr (key));
                gotoxy (wherex + 1, field.y1)
                end
             end
          end
       until exit_edit_field;
       for i := field.x1 to field.x2 do
          screen_string [i - field.x1 + 1] := get_char (i,
       field.y1);
       i := field.x2 + 1;
       repeat
          decr (i)
       until (i = field.x1 - 1) or (get_char (i, field.y1) <>
       blank);
       screen_string [0] := chr (i - field.x1 + 1);
       edit_field := screen_string;
       restore_field (field);
       remove_cursor
       end;

       procedure display_edit_screen_help_line;

       begin
       clear_area (1,25,80,25);
       put_char (1, 25, #24);
       put_char (2, 25, #25);
       put_char (3, 25, #60);
```

```
put_char (4, 25, #217);
put_string (5, 25, '-Choose  F1-Help  F4-Print screen
ESC-Done');
color_foreground (1,25,4,25, red);
color_foreground (5,25,11,25, blue);
color_foreground (14,25,15,25, red);
color_foreground (16,25,20,25, blue);
color_foreground (23,25,24,25, red);
color_foreground (25,25,37,25, blue);
color_foreground (40,25,42,25, red);
color_foreground (43,25,47,25, blue);
color_background (1,25,80,25, lightgray)
end;

procedure display_edit_list_help_line;

begin
clear_area (1,25,80,25);
put_char (1, 25, #24);
put_char (2, 25, #25);
put_char (3, 25, #60);
put_char (4, 25, #217);
put_string (5, 25, '-Choose');
put_string (13, 25, 'F1-Help F4-Print F5-Add F6-Delete '+
                    'F7-Edit F8-Search ESC-Done');
color_foreground (1,25,4,25, red);
color_foreground (5,25,11,25, blue);
color_foreground (13,25,14,25, red);
color_foreground (15,25,20,25, blue);
color_foreground (21,25,22,25, red);
color_foreground (23,25,29,25, blue);
color_foreground (30,25,31,25, red);
color_foreground (32,25,36,25, blue);
color_foreground (37,25,38,25, red);
color_foreground (39,25,46,25, blue);
color_foreground (47,25,48,25, red);
color_foreground (49,25,54,25, blue);
color_foreground (55,25,56,25, red);
color_foreground (57,25,64,25, blue);
color_foreground (65,25,67,25, red);
color_foreground (68,25,72,25, blue);
color_background (1,25,80,25, lightgray)
end;

procedure display_command_help_line;

begin
clear_area (1,25,80,25);
put_char (1, 25, #24);
put_char (2, 25, #25);
put_char (3, 25, #60);
```

```
put_char (4, 25, #217);
put_string (5, 25, '-Choose  F1-Help  F2-Execute command  '+
                   'F4-Print  ESC-Abort');
color_foreground (1,25,4,25, red);
color_foreground (5,25,11,25, blue);
color_foreground (14,25,15,25, red);
color_foreground (16,25,20,25, blue);
color_foreground (23,25,24,25, red);
color_foreground (25,25,40,25, blue);
color_foreground (43,25,44,25, red);
color_foreground (45,25,50,25, blue);
color_foreground (53,25,55,25, red);
color_foreground (56,25,61,25, blue);
color_background (1,25,80,25, lightgray)
end;

procedure display_add_record_help_line;

begin
clear_area (1,25,80,25);
put_char (1, 25, #24);
put_char (2, 25, #25);
put_char (3, 25, #60);
put_char (4, 25, #217);
put_string (5, 25, '-Choose  F1-Help  F2-Add record  '+
                   'F4-Print  ESC-Abort');
color_foreground (1,25,4,25, red);
color_foreground (5,25,11,25, blue);
color_foreground (14,25,15,25, red);
color_foreground (16,25,20,25, blue);
color_foreground (23,25,24,25, red);
color_foreground (25,25,35,25, blue);
color_foreground (38,25,39,25, red);
color_foreground (40,25,45,25, blue);
color_foreground (48,25,50,25, red);
color_foreground (51,25,56,25, blue);
color_background (1,25,80,25, lightgray)
end;

procedure edit_screen (var number_of_fields : integer;
                       var field_data : field_array;
                       abort_possible : boolean);

var
   field_number : integer;
   error_value : integer;
   good_value_entered : boolean;
   i : integer;

procedure display_edit_screen_help_screen;
```

```
begin
save_screen;
remove_cursor;
draw_window  (16,5,65,19, blue, lightgray, 'help');
shade_window (16,5,65,19, black);
center_text (12, 'NO HELP AVAILABLE AT THIS TIME', blue);
center_text (17, 'press any key to return', blue);
key := get_key;
key := null;
restore_cursor;
restore_screen
end;

procedure print_bad_int_message;

begin
display_error_message ('INPUT ERROR',
                       'integer input error',
                        null_string,
                       'input is either not a valid number
or',
                       'is out of allowable range of
values')
end;

procedure print_bad_float_message;

begin
display_error_message ('INPUT ERROR',
                       'floating point input error',
                        null_string,
                       'input is either not a valid number
or',
                       'is out of allowable range of
values')
end;

procedure print_bad_dtg_message;

begin
display_error_message ('INPUT ERROR',
                       'dtg input format error',
                        null_string,
                       'proper format : ''05 0530Z JAN
89''',
                       'spaces may be omitted')
end;

procedure print_bad_time_message;
```

```
begin
display_error_message ('INPUT ERROR',
                       'time input error',
                        null_string,
                       'proper format : ''0530Z''',
                        null_string)
end;

procedure print_bad_chr_message;

begin
display_error_message ('INPUT ERROR',
                       'character input error',
                        null_string,
                       'character entered is not allowed',
                       'in this field')
end;

procedure print_bad_enum_message;

begin
display_error_message ('INPUT ERROR',
                       'character input error',
                        null_string,
                       'characters entered are not allowed',

                       'in this field')
end;

begin
save_screen;
for field_number := 1 to number_of_fields do
   with field_data [field_number] do
      begin
      put_string (label_x, label_y, label_string);
      if length (str_val) > (x2 - x1 + 1) then
         str_val [0] := chr (x2 - x1 + 1);
      if field_type in [int, float] then str_val :=
         remove_blanks (str_val);
      put_string (x1, y1, str_val)
      end;
field_number := 1;
repeat
   with field_data [field_number] do
      begin
      str_val := edit_field (field_data [field_number]);
      if (key = enter) or (key = up_arrow) or (key =
down_arrow) or
         (key = page_up) or (key = page_down) or (key = f4)
or
         ((key = escape) and not (abort_possible)) or
```

```
          ((key = f2) and (abort_possible)) then
          begin
          case field_type of
             int    : begin
                      val (str_val, int_value, error_value);
                      good_value_entered := (error_value = 0)
and
                                            (int_value >=
int_min_value) and
                                            (int_value <=
int_max_value);
                      if not good_value_entered then
                         print_bad_int_message
                      end;
             float  : begin
                      val (str_val, float_value, error_value);

                      good_value_entered := (error_value = 0)
and
                                            (float_value >=
float_min_value) and
                                            (float_value <=
float_max_value);
                      if not good_value_entered then
                         print_bad_float_message
                      end;
             ch     : begin
                      str_val := upper_case (str_val);
                      good_value_entered :=
                         str_val [1] in valid_char_set;
                      if not good_value_entered then
                         print_bad_chr_message
                      end;
             strg   : begin
                      good_value_entered := true
                      end;
             enum   : begin
                      str_val := upper_case (str_val);
                      good_value_entered := false;
                      for i := 1 to number_of_enum_values do
                         good_value_entered :=
                            str_val = valid_enum_values [i];
                      if not good_value_entered then
                         print_bad_enum_message
                      end;
             dtg    : begin
                      str_val := upper_case (str_val);
                      good_value_entered := valid_dtg
(str_val);
                      if not good_value_entered then
                         print_bad_dtg_message
```

```
                        end;
                time    : begin
                          str_val := upper_case (str_val);
                          good_value_entered := valid_time
(str_val);
                          if not good_value_entered then
                             print_bad_time_message
                          end;
                eval    : good_value_entered := eval_function
(str_val)
            end;
            if good_value_entered then
               case key of
                  f4          : print_screen;
                  up_arrow    : if field_number = 1 then
                                   begin
                                   if not abort_possible then
                                      field_number :=
number_of_fields
                                   end
                                else
                                   decr (field_number);
                  down_arrow,
                  enter       : if field_number =
number_of_fields then
                                   field_number := 1
                                else
                                   incr (field_number)
               end
            end
         end
until (key = escape) or
      ((key = page_down) and not abort_possible) or
      ((key = page_up) and not abort_possible) or
      ((key = enter) and (number_of_fields = 1)) or
      ((key = f2) and (abort_possible));
restore_screen
end;

function menu_selection (menu_title : string80;
                         menu_options : option_string):
integer;

type
   menu_option = record
      option : string80;
      option_number : integer;
      x1, y1, x2, y2 : integer
      end;
   option_array = array [1..20] of menu_option;
```

```
var
   option_list : option_array;
   number_menu_options : integer;

procedure create_option_list (menu_options : option_string;
                              var option_list :
option_array;
                              var number_menu_options :
integer);

var
   position : integer;
   option_string_length : integer;
   opt_num : integer;
   skip_lines : integer;

begin
position := 1;
number_menu_options := 0;
while menu_options [position] <> '\' do
      begin
      incr (number_menu_options);
      with option_list [number_menu_options] do
         begin
         option_string_length := 0;
         while not (menu_options [position] in ['\','|']) do

            begin
            incr (option_string_length);
            option [option_string_length] := menu_options
[position];
            incr (position)
            end;
         option [0] := chr (option_string_length);
         option_number := number_menu_options;
         x1 := ((menu_x2 - menu_x1 + 1 - length (option))
div 2) + menu_x1;
         x2 := x1 + length (option) - 1;
         if menu_options [position] = '|' then incr
(position)
         end
      end;
if number_menu_options < 10 then
   skip_lines :=
      (((menu_y2 - menu_y1) - (2 * number_menu_options - 1))
div 2) + 1
else
   skip_lines :=
      (((menu_y2 - menu_y1) - (2 * ((number_menu_options +
1) div 2) - 1)) div 2) + 1;
for opt_num := 1 to number_menu_options do
```

```
   with option_list [opt_num] do
      begin
      if number_menu_options < 10 then
         begin
         x1 := ((menu_x2 - menu_x1 + 1 - length (option))
div 2) + menu_x1;
         y1 := (menu_y1 + skip_lines - 1) + (2 * opt_num -
1)
         end
      else
         begin
         if opt_num <= (number_menu_options + 1) div 2 then
            begin
            y1 := (menu_y1 + skip_lines - 1) + (2 * opt_num
- 1);
            x1 := menu_x1 + 4
            end
         else
            begin
            y1 := (menu_y1 + skip_lines - 1) +
                  (2 * (opt_num - ((number_menu_options + 1)
div 2)) - 1);
            x1 := menu_x1 + ((menu_x2 - menu_x1 + 1) div 2)
+ 2
            end
         end;
      x2 := x1 + length (option) - 1;
      y2 := y1
      end
end;

procedure display_menu_screen (menu_title : string80;
                               option_list : option_array;
                               number_menu_options :
integer);

procedure display_options (option_list : option_array;
                           number_menu_options : integer);

var
   opt_num : integer;

begin
for opt_num := 1 to number_menu_options do
   with option_list [opt_num] do
      put_string (x1, y1, option)
end;

procedure display_menu_help_line;

begin
```

```
clear_area (1,25,80,25);
put_char (1, 25, #24);
put_char (2, 25, #25);
put_string (3, 25, '-Choose');
put_char (12, 25, #60);
put_char (13, 25, #217);
put_string (14, 25, '-Select');
put_string (23, 25, 'F1-Help');
color_foreground (1,25,2,25, red);
color_foreground (3,25,9,25, blue);
color_foreground (12,25,13,25, red);
color_foreground (14,25,20,25, blue);
color_foreground (23,25,24,25, red);
color_foreground (25,25,30,25, blue);
color_background (1,25,80,25, lightgray)
end;

begin
draw_window (menu_x1, menu_y1, menu_x2, menu_y2,
             menu_forecolor, menu_backcolor, menu_title);
display_options (option_list, number_menu_options);
display_menu_help_line
end;

function get_option (option_list : option_array;
                     number_menu_options : integer):
integer;

var
   opt_num : integer;

procedure highlight_option (field : menu_option);

begin
with field do
   begin
   color_background (x1-1, y1, x2+1, y2, option_backcolor);
   color_foreground (x1-1, y1, x2+1, y2, option_forecolor)
   end
end;

procedure restore_option (field : menu_option);

begin
with field do
   begin
   color_background (x1-1, y1, x2+1, y2, menu_backcolor);
   color_foreground (x1-1, y1, x2+1, y2, menu_forecolor)
   end
end;
```

```
procedure display_menu_help_screen;

begin
save_screen;
draw_window  (16,6,65,20, blue, lightgray, 'help');
shade_window (16,6,65,20, black);
center_text (13, 'NO HELP AVAILABLE AT THIS TIME', blue);
center_text (19, 'press any key to return', blue);
key := get_key;
key := null;
restore_screen
end;

begin
opt_num := 1;
highlight_option (option_list [opt_num]);
repeat
   key := get_key;
   case key of
        enter    :
           get_option := opt_num;
        up_arrow :
           begin
           restore_option (option_list [opt_num]);
           if opt_num = 1 then
              opt_num := number_menu_options
           else decr (opt_num);
           highlight_option (option_list [opt_num])
           end;
        down_arrow :
           begin
           restore_option (option_list [opt_num]);
           if opt_num = number_menu_options then
              opt_num := 1
           else incr (opt_num);
           highlight_option (option_list [opt_num])
           end;
        left_arrow :
           if (number_menu_options >= 10) and
              (opt_num > (number_menu_options + 1) div 2)
then
              begin
              restore_option (option_list [opt_num]);
              opt_num := opt_num - ((number_menu_options +
1) div 2);
              highlight_option (option_list [opt_num])
              end;
        right_arrow :
           if (number_menu_options >= 10) and
              (opt_num + ((number_menu_options + 1) div 2)
               <= number_menu_options) then
```

```
                begin
                restore_option (option_list [opt_num]);
                opt_num := opt_num + ((number_menu_options +
1) div 2);
                highlight_option (option_list [opt_num])
                end;
          f1 :
             display_menu_help_screen
   end
until key = enter
end;

begin
save_screen;
create_option_list (menu_options, option_list,
number_menu_options);
display_menu_screen (menu_title, option_list,
number_menu_options);
menu_selection := get_option (option_list,
number_menu_options);
restore_screen
end;

function get_file (prompt, message : string80; search_mask :
dirstr): pathstr;

const
   number_fields = 1;
   fields : array [1..number_fields] of field_record =
            ( (label_string : 'Directory :  '; label_x : 18;
label_y : 7;
                str_val : null_string;
                x1 : 31; y1 : 7; x2 : 60; y2 : 7;
                field_type : strg) );
   file_window_x1 = 10;
   file_window_y1 = 10;
   file_window_x2 = 71;
   file_window_y2 = 21;
   file_window_backcolor = cyan;
   file_window_forecolor = blue;

type
   file_list_ptr = ^file_list_record;
   file_list_record = record
      line     : string60;
      previous : file_list_ptr;
      next     : file_list_ptr
   end;

var
   file_list   : file_list_ptr;
```

```
   path_valid  : boolean;
   list_length : integer;
   heap_state  : pointer;
   file_error  : integer;

procedure display_getfile_help_screen;

begin
save_screen;
draw_window  (16,5,65,19, blue, lightgray, 'help');
shade_window (16,5,65,19, black);
center_text (12, 'NO HELP AVAILABLE AT THIS TIME', blue);
center_text (17, 'press any key to return', blue);
key := get_key;
key := null;
restore_screen
end;

procedure display_getfile_screen (prompt, message :
string80; search_mask : dirstr);

procedure display_getfile_help_line;

begin
put_char (1, 25, #24);
put_char (2, 25, #25);
put_char (3, 25, #26);
put_char (4, 25, #27);
put_string (5, 25, '-Choose');
put_char (14, 25, #60);
put_char (15, 25, #217);
put_string (16, 25, '-Select');
put_string (25, 25, 'F1-Help');
put_string (34, 25, 'F4-New mask');
put_string (47, 25, 'ESC-Quit');
color_foreground (1,25,4,25, red);
color_foreground (5,25,11,25, blue);
color_foreground (14,25,15,25, red);
color_foreground (16,25,22,25, blue);
color_foreground (25,25,26,25, red);
color_foreground (27,25,31,25, blue);
color_foreground (34,25,35,25, red);
color_foreground (36,25,44,25, blue);
color_foreground (47,25,49,25, red);
color_foreground (50,25,54,25, blue);
color_background (1,25,80,25, lightgray)
end;

begin
draw_window (menu_x1, menu_y1, menu_x2, menu_y2,
             menu_forecolor, menu_backcolor, prompt);
```

```
draw_window (file_window_x1, file_window_y1, file_window_x2,
file_window_y2,
             file_window_forecolor, file_window_backcolor,
search_mask);
put_string (fields [1].label_x, fields [1].label_y,
            fields [1].label_string + search_mask);
center_text (fields [1].label_y - 2, message,
file_window_backcolor);
display_getfile_help_line
end;

function check_path (search_mask : dirstr): boolean;

var
   file_info  : searchrec;
   valid_path : boolean;

begin
findfirst (search_mask, anyfile, file_info);
file_error := doserror;
if file_error <> 0 then
   begin
   save_screen;
   draw_window  (26,13,55,16, yellow, red, null_string);
   shade_window (26,13,55,16, black);
   case doserror of
         3 : begin
             center_text (14, 'PATH NOT FOUND', yellow);
             center_text (15, 'press any key', white);
             key := get_key;
             key := null
             end;
        18 : begin
             center_text (14, 'NO FILES FOUND', yellow);
             center_text (15, 'press any key', white);
             key := get_key;
             key := null
             end;
       152 : begin
             center_text (14, 'DISK  ERROR', yellow);
             center_text (15, 'Retry or Abort', yellow);
             color_foreground (34,15,34,15, white);
             color_foreground (43,15,43,15, white);
             repeat
                key := get_key;
                if chr (key) in ['a','A'] then
                   begin
                   restore_screen;
                   check_path := (file_error = 0);
                   exit
                   end
```

```
                  else if chr (key) in ['r', 'R'] then
                     valid_path := check_path (search_mask)
               until (file_error <> 152) or (chr (key) in
['a','A']);
               end
   else
      begin
      center_text (14, 'PATH ERROR', yellow);
      center_text (15, 'press any key', white);
      key := get_key;
      key := null
      end
   end;
   restore_screen
   end;
check_path := (file_error = 0);
end;

function get_file_list (search_mask : dirstr;
                        var list_length : integer):
file_list_ptr;

const
   blank_60 = '                                    '+
              '                                    ';


var
   number_of_files : integer;
   file_info : searchrec;
   column    : integer;
   list_head : file_list_ptr;
   list_tail : file_list_ptr;

begin
number_of_files := 1;
list_length := 1;
column := 2;
new (list_head);
list_head^.line := blank_60;
list_head^.next := nil;
list_head^.previous := nil;
list_tail := list_head;
findfirst (search_mask, anyfile, file_info);
while (doserror = 0) do
   begin
   delete (list_tail^.line, column, length
(file_info.name));
   insert (file_info.name, list_tail^.line, column);
   if file_info.attr = directory then
      list_tail^.line [column + length (file_info.name)] :=
'\';
```

```
   findnext (file_info);
   if doserror = 0 then
      begin
      incr (number_of_files);
      if (number_of_files mod 4) = 1 then
         begin
         new (list_tail^.next);
         list_tail^.next^.next := nil;
         list_tail^.next^.previous := list_tail;
         list_tail^.next^.line := blank_60;
         list_tail := list_tail^.next;
         incr (list_length);
         column := 2
         end
      else
         column := column + 15
      end
   end;
get_file_list := list_head
end;

function get_file_name (search_mask : dirstr;
                        file_list   : file_list_ptr;
                        list_length : integer     ):
pathstr;

var
   list_pointer : file_list_ptr;
   done : boolean;
   file_name : pathstr;
   dir  : dirstr;
   name : namestr;
   ext  : extstr;

procedure highlight_file;

begin
color_foreground (file_window_x1 + wherex,
file_window_y1 + wherey,
                  file_window_x1 + wherex + 13,
file_window_y1 + wherey,
                  yellow);
color_background (file_window_x1 + wherex,
file_window_y1 + wherey,
                  file_window_x1 + wherex + 13,
file_window_y1 + wherey,
                  magenta)
end;

procedure restore_file;
```

```
begin
color_foreground (file_window_x1 + wherex,
file_window_y1 + wherey,
                  file_window_x1 + wherex + 13,
file_window_y1 + wherey,
                  file_window_forecolor);
color_background (file_window_x1 + wherex,
file_window_y1 + wherey,
                  file_window_x1 + wherex + 13,
file_window_y1 + wherey,
                  file_window_backcolor)
end;

begin
draw_window (file_window_x1, file_window_y1,
             file_window_x2, file_window_y2,
             file_window_forecolor, file_window_backcolor,
search_mask);
window (file_window_x1 + 1, file_window_y1 + 1,
        file_window_x2 - 1, file_window_y2 - 1);
textbackground (file_window_backcolor);
textcolor (file_window_forecolor);
clrscr;
list_pointer := file_list;
done := false;
repeat
   put_string (file_window_x1 + wherex, file_window_y1 +
wherey,
               list_pointer^.line);
   if (list_pointer^.next <> nil) and (wherey <> 10) then
      begin
      gotoxy (wherex, wherey + 1);
      list_pointer := list_pointer^.next
      end
   else
      begin
      gotoxy (1,1);
      list_pointer := file_list;
      done := true
      end
until done;
highlight_file;
repeat
   key := get_key;
   case key of
      enter :
         begin
         file_name := copy (list_pointer^.line, wherex + 1,
12);
         if pos ('\', file_name) = 0 then
            begin
```

```
            fsplit (search_mask, dir, name, ext);
            get_file_name := dir + file_name
            end
         else
            begin
            save_screen;
            draw_window  (26,13,55,16, yellow, red,
null_string);
            shade_window (26,13,55,16, black);
            center_text (14, 'CANNOT SELECT SUBDIRECTORY',
yellow);
            center_text (15, 'press any key', white);
            key := get_key;
            key := null;
            restore_screen
            end
         end;
      f1 :
         display_getfile_help_screen;
      up_arrow,
      down_arrow,
      left_arrow,
      right_arrow :
         begin
         restore_file;
         case key of
            up_arrow :
               if wherey <> 1 then
                  begin
                  gotoxy (wherex, wherey - 1);
                  list_pointer := list_pointer^.previous
                  end
               else if (wherey = 1) and
(list_pointer^.previous <> nil) then
                  begin
                  list_pointer := list_pointer^.previous;
                  insline;
                  put_string (file_window_x1 + 1,
file_window_y1 + wherey,
                              list_pointer^.line)
                  end;
            down_arrow :
               if (wherey <> 10) and (list_pointer^.next <>
nil) and
                  (list_pointer^.next^.line [wherex + 1] <>
blank) then
                  begin
                  gotoxy (wherex, wherey + 1);
                  list_pointer := list_pointer^.next
                  end
```

```
                else if (wherey = 10) and (list_pointer^.next
<> nil) and
                        (list_pointer^.next^.line [wherex +
1] <> blank) then
                   begin
                   list_pointer := list_pointer^.next;
                   gotoxy (wherex, 1);
                   delline;
                   gotoxy (wherex, 10);
                   put_string (file_window_x1 + 1,
file_window_y1 + wherey,
                               list_pointer^.line)
                   end;
             left_arrow :
                if wherex <> 1 then
                   gotoxy (wherex - 15, wherey)
                else if wherex = 1 then
                   begin
                   if wherey <> 1 then
                      begin
                      list_pointer := list_pointer^.previous;

                      gotoxy (wherex + 45, wherey - 1)
                      end
                   else if (wherex = 1) and
                           (list_pointer^.previous <> nil)
then
                      begin
                      list_pointer := list_pointer^.previous;

                      insline;
                      put_string (file_window_x1 + 1,
file_window_y1 + wherey,
                                  list_pointer^.line);
                      gotoxy (wherex + 45, wherey)
                      end
                   end;
             right_arrow :
                if (wherex <> 46) and
                   (get_char (file_window_x1 + wherex + 16,
                              file_window_y1 + wherey      )
<> blank) then
                   gotoxy (wherex + 15, wherey)
                else if wherex = 46 then
                   begin
                   if wherey <> 10 then
                      begin
                      list_pointer := list_pointer^.next;
                      gotoxy (1, wherey + 1)
                      end
```

```
                    else if (wherey = 10) and
(list_pointer^.next <> nil) then
                        begin
                        list_pointer := list_pointer^.next;
                        gotoxy (1,1);
                        delline;
                        gotoxy (1,10);
                        put_string (file_window_x1 + 1,
file_window_y1 + wherey,
                                    list_pointer^.line)
                        end
                    end
               end;
            highlight_file
            end
         end
until (key = enter) or (key = escape) or (key = f4);
restore_file;
textbackground (black);
textcolor (yellow);
window (1,1,80,25)
end;

begin
save_screen;
mark (heap_state);
repeat
   display_getfile_screen (prompt, message, search_mask);
   path_valid := false;
   repeat
      search_mask := edit_field (fields [1]);
      if key = enter then
         path_valid := check_path (search_mask)
   until (key = escape) or (path_valid);
   if path_valid then
      begin
      file_list := get_file_list (search_mask, list_length);

      get_file := get_file_name (search_mask, file_list,
list_length)
      end
until (key = enter) or (key = escape);
if key = escape then
   get_file := null_string;
release (heap_state);
restore_screen
end;

begin

menu_x1 := menu_x1_default;
```

```
menu_x2 := menu_x2_default;
menu_y1 := menu_y1_default;
menu_y2 := menu_y2_default;
menu_forecolor := menu_forecolor_default;
menu_backcolor := menu_backcolor_default;
option_forecolor := option_forecolor_default;
option_backcolor := option_backcolor_default;
field_forecolor  := field_forecolor_default;
field_backcolor  := field_backcolor_default;
default_mask := 'a:\*.*';
stack_pointer := 0;
month_data [1].days := 31;
month_data [2].days := 28;
month_data [3].days := 31;
month_data [4].days := 30;
month_data [5].days := 31;
month_data [6].days := 30;
month_data [7].days := 31;
month_data [8].days := 31;
month_data [9].days := 30;
month_data [10].days := 31;
month_data [11].days := 30;
month_data [12].days := 31;
month_data [1].name := 'JAN';
month_data [2].name := 'FEB';
month_data [3].name := 'MAR';
month_data [4].name := 'APR';
month_data [5].name := 'MAY';
month_data [6].name := 'JUN';
month_data [7].name := 'JUL';
month_data [8].name := 'AUG';
month_data [9].name := 'SEP';
month_data [10].name := 'OCT';
month_data [11].name := 'NOV';
month_data [12].name := 'DEC';

assign (fontfile, 'fontfile');
reset  (fontfile);
if ioresult = 0 then
   begin
   for loop := 'A' to 'Z' do
      read (fontfile, letters [loop]);
   for loop := '0' to '9' do
      read (fontfile, digits [loop]);
   close (fontfile)
   end
else
   begin
   initialize_screen;
   remove_cursor;
   draw_window  (19,12,61,16, yellow, red, null_string);
```

```
    center_text (13, 'FONTFILE NOT FOUND', yellow);
    center_text (14, 'CHECK DISK,  THEN TRY AGAIN', yellow);
    center_text (15, 'press any key', white);
    key := get_key;
    key := null;
    restore_cursor;
    initialize_screen;
    halt
    end

end.
^Z
```

LIST OF REFERENCES

1. U.S. Department of the Army, Operations, Field Manual 100-5, Washington, D.C.: U.S. Government Printing Office, 1982.

2. Wohl, Joseph, "Force Management Decision Requirements for Air Force Tactical Command and Control," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 9, Sept 1981.

3. Taylor, James G., Professor, Naval Postgraduate School, Monterey, CA, OS3636 Class Notes, "C3 Architecture", Summer Quarter, 1988.

4. Giles, G. and Sandene, J., "An Introduction to C3 Architecture," M.S. Thesis, Naval Postgraduate School, Monterey, CA, March 1988.

5. U.S. Department of the Army, Field Artillery Cannon Battalion, Field Manual 6-20-1, Washington, D.C.: U.S. Government Printing Office, 1979.

6. Clausewitz, Carl von, On W_, Edited and Translated by Michael Howard and Peter Paret, Princeton University Press, 1976.

7. IBM Corp. Federal Systems Division, "1982 Final Report Integration of C3 Architecture," (Naval Postgraduate School Microfiche AD-B072-182L).

8. Orr, George, Combat Operations C3I: Fundamentals and Interactions, Air University Press, 1983.

BIBLIOGRAPHY

Barr, D.R., Poock, G.K., and Richards, F.R., "Experimental Manual, Part I: Experimentation Methodology," Naval Postgraduate School NPS55-78-032, Nov, 78, (NPS Microfiche AD-A067-539).

Martin, Terry L., Command, Control, and Communications Mission and Organization: A Primer, M.S. Thesis, Naval Postgraduate School, Monterey, CA, March 1984.

Schoderbeck, Peter P., Schoderbeck, Charles G., and Kefalas, Asterios, Management Systems: Conceptual Considerations, Third Edition, Business Publications, INC., Plano, TX, 1985.

Tiede, Roland V., and Leake, Lewis A., "A Method for Evaluating the Combat Effectiveness of a Tactical Information System in a Field Army.," Operations Research, Opns. Res. 19, pages 585-604, 1971.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Commandant US Army Field Artillery School ATTN: ATSF-AF Ft Sill, Oklahoma 73503-5600	1
4.	Commander US Army Signal Center and Ft Gordon ATTN: ATZH-POO Ft Gordon, Georgia 30905-5300	1
5.	Commander US Army Combined Arms Center ATTN: ATZL-SWP-P Ft Leavenworth, Kansas 66027-5300	1
6.	Curricular Office, Code 39 Naval Postgraduate School Monterey, California 93943-5000	1
7.	C^3 Academic Group ATTN: Professor Carl Jones, Code 74 Naval Postgraduate School Monterey, California 93943-5000	1
8.	Commander US Army Combined Arms Center ATTN: ATZL-CAC-I Ft Leavenworth, Kansas 66027-5300	1
9.	Department of Operations Research ATTN: Professor Samuel Parry, Code 55Py Naval Postgraduate School Monterey, California 93943-5000	2

10. Department of Operations Research — 2
ATTN: LCDR William Walsh, Code 55Wa
Naval Postgraduate School
Monterey, California 93943-5000

11. Test and Experimentation Command — 2
ATTN: CPT Michael W. Schneider
Ft Hood, Texas 76544

12. Commanding General — 1
Training and Doctrine Command
Ft Monroe, Virginia 23351

13. Commanding General — 1
5th United States Army
Fort Sam Houston, Texas 78234

14. Defense Communications Agency — 2
Center for C^3 Systems
ATTN: CPT Anthony R. Ferrara
Arlington Hall Station
Arlington, Virginia 22212-5410