SPARTAN: An Instructional High Resolution Land Combat Model

THESIS

David Keith Cox
Captain, USA

AFIT/GOR/ENS/92M-7

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohi

SPARTAN: An Instructional High Resolution Land Combat Model

THESIS

David Keith Cox
Captain, USA

AFIT/GOR/ENS/92M-7

Approval for public release; distribution unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	Master's Thesis

4. TITLE AND SUBTITLE
SPARTAN: AN INSTRUCTIONAL HIGH RESOLUTION LAND COMBAT MODEL

5. FUNDING NUMBERS

6. AUTHOR(S)
David K. Cox, Captain, USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/92M-7

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This project developed an instructional high resolution land combat simulation model. The purpose of this model is to demonstrate common techniques of modeling used in the present generation of UL Army land combat models. This model is stochastic and uses an event scheduling type of discrete event time algorithm. The model represents a maximum of two six-man infantry squads in direct fire combat. The individual soldiers move, search for targets, engage targets, and react to hostile encounters. The data values provided with the model are all generic and hypothetical. The model is only intended as a demonstration tool and has no validity for analytic use. A preprocessor is provided with the model and Appendix D provides a user's manual.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Combat Model, High Resolution, Stochastic Model Educational, Simulation, Land Warfare	201
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1992	Master's Thesis

4. TITLE AND SUBTITLE
SPARTAN: AN INSTRUCTIONAL HIGH RESOLUTION LAND COMBAT MODEL

5. FUNDING NUMBERS

6. AUTHOR(S)
David K. Cox, Captain, USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GOR/ENS/92M-7

9. SPONSORING, MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This project developed an instructional high resolution land combat simulation model. The purpose of this model is to demonstrate common techniques of modeling used in the present generation of UL Army land combat models. This model is stochastic and uses an event scheduling type of discrete event time algorithm. The model represents a maximum of two six-man infantry squads in direct fire combat. The individual soldiers move, search for targets, engage targets, and react to hostile encounters. The data values provided with the model are all generic and hypothetical. The model is only intended as a demonstration tool and has no validity for analytic use. A preprocessor is provided with the model and Appendix D provides a user's manual.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Combat Model, High Resolution, Stochastic Model Educational, Simulation, Land Warfare	201
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements.***

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

AFIT/GOR/ENS/92M-7

SPARTAN: An Instructional High Resolution Land Combat Model

THESIS

Presented to the Faculty of the
Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

David K. Cox, B.S.
Captain, USA

March 1992

THESIS APPROVAL

STUDENT: Captain David K. Cox CLASS: GOR 92-M

THESIS TITLE: SPARTAN: An Instructional High Resolution Land Combat Model

DEFENSE DATE: 4 MAR 92

COMMITTEE:	NAME/DEPARTMENT	SIGNATURE
Advisor	Major Michael W. Garrambone/ENS	
Co-Advisor	Major Bruce W. Morlan/ENC	

Preface

The goal of this thesis was to develop a high resolution land combat model for use as an instructional aid in land combat modeling courses. The model development began at the conceptual level and progressed through code development to initial implementation. This model displays modeling concepts that are representative of the present generation of US Army high resolution analytic combat models and provides facilities for students to observe the model components and understand their operation.

This thesis provides essential background material, the development methodology, details of the model's design and implementation, and classroom materials to support its operational use. SPARTAN is a simple representation of an analytic model that will serve as a useful tool in the presentation of high resolution land combat modeling techniques.

I wish to thank MAJ Michael Garrambone and MAJ Bruce Morlan for their guidance and invaluable assistance in the development of this thesis. I also wish to thank the other officers in the land combat modeling course who provided timely feedback on necessary model improvements. Lastly, I must thank my wife Phyllis and my kids Joshua and Courtney for their constant support and understanding of my efforts.

David Keith Cox

Table of Contents

List of Figures

List of Tables

Abstract

This project developed a high resolution land combat model for the purpose of demonstrating current modeling techniques. The complex and dynamic nature of combat simulation models makes it very difficult to teach the concepts of combat modeling without some type of hands-on experience for the student. A literature search of current military models showed no model existed that was well suited for this educational purpose.

This model [SPARTAN] was developed to represent current modeling techniques in use with the present generation of US Army models. The model is primarily a small scale direct fire attrition model under the definitions of the Army Model Improvement Program [AMIP] (12). The model represents the soldiers performing the actions of movement, searching for targets, selecting targets, engaging targets, and reacting to enemy encounters. The model development process focused on using a sound methodology for the code development, and using modeling techniques similar to those in the Army's two premier high resolution models [JANUS and the Combined Arms and Support Task Force Evaluation Model [CASTFOREM]]. SPARTAN contains numerous features that allow the student to observe in great detail how the model represents the various activities of the soldiers. An educational assessment of the model was performed by students and faculty at the Air Force Institute of Technology.

SPARTAN: AN INSTRUCTIONAL HIGH RESOLUTION LAND COMBAT MODEL

I. Introduction

1.1 Background

Within the defense community, computer-aided wargames and combat simulation models have become important training and analysis tools. One of the primary users of computer modeling techniques, the US Army Training and Doctrine Command, defines a model as "a representation of the real world by a series of mathematical and judgemental relationships (16:85)," and a simulation model as "a model of combat in which play is automated and governed by decision rules input into the simulation before play begins. An automated wargame (16:86)." In most cases, combat models are descriptive in that they provide a history of a battle with the initial conditions of the conflict provided by the analyst. A distinguishing characteristic of a simulation model from other types of analytic models is that it solves the problem by portraying the process as a chronological sequence of events acted out in a step by step manner till some form of termination condition is encountered (7:3).

A General Accounting Office [GAO] report on models in use by the federal government lists modeling analysis as a key factor influencing policy makers (11:116). Leaders use analytic results from combat simulation models to make decisions in virtually all areas of procurement, force structure, and national defense policy. During the concept phase of system development, simulation models are often used to study alternative designs, and define system requirements. In the operational testing phase, simulations may be used to investigate system performance, determine support requirements or develop tactics for a particular system (22:712).

Combat simulations are used extensively also as training tools at all echelons in the military. In the US Army, all battalion and brigade commander designees receive two weeks of tactics training using the JANUS combat simulation model as a tool to improve their command and control skills.

While there are many model advocates, there are also many people within Congress and the defense community who feel that using models for analysis is not always accurate. A GAO report cites numerous misuses of model analyses and poor analytic techniques to support these contentions (11:i--iv). This same GAO report to Congress emphasized the need for a greater understanding of the inherent processes of combat and the methods used in models to simulate these

processes (11:104). A key point is that the output from a simulation can only be as good as the input that the analyst provides for the model.

Since modern warfare is such a complex and dynamic activity, the simulations built to model these conflicts are normally very complex tools. For example, the Army's high resolution combat model JANUS(T) uses more than 85,000 lines of computer code. A typical JANUS battle simulates the activities of over 600 combatants in an air and land warfare scenario where any or all of these combatants can be performing numerous different tasks (14:5). Adding to the complexity of the combat systems is the need to represent terrain and the battlefield environment in which the combat occurs. According to an experienced JANUS operator, this model requires a minimum of three months for an analyst to learn the very basics of the operating processes (40).

Simulation modeling is not an exact science and can never be expected to completely represent reality. In any given situation, the analyst can only replicate a limited number of the combat processes and the most important environmental factors. To analyze model output and to make intelligent decisions from the analysis, it is imperative that the combat modeler fully understand the various methods used to model combat and the numerous simplifying assumptions inherent in any simulation model.

1.2 Problem Statement Sorely lacking in the modeling community has been a small model which serves as an instructional tool to demonstrate clearly the processes of combat simulation. With such a tool, the student could observe the often opaque processes operating inside the model and understand the relationships of the combat processes. Of the present analytical models in use, such as JANUS(T), none are suitable instructional aids applicable towards supporting a university course of instruction in combat modeling. Indeed, a review of the Joint Services military model catalogues indicates there are no models that specifically meet this educational need (26:App M).

The purpose of this thesis has been to develop and implement a high resolution combat model that would be a useful instructional tool in a land combat modeling course. This model demonstrates current Army methods of modeling direct fire combat processes in a simple structured environment. The model can support and enhance the curriculum by improving the conceptual understanding of the modeling processes.

1.3 Model Requirements

Wargaming literature discusses many approaches to modeling combat, yet most experts seem to agree that modern analysts still do not fully understand the very dynamic combat processes (11:100--102). For any given activity

there are usually a number of methods of modeling the process. The algorithms and techniques used in the SPARTAN model in most cases are those found in common use with the present generation of Army models. A goal of the Army Model Improvement Program [AMIP] is to standardize algorithms among models for consistency throughout the Army (12:12). SPARTAN uses the simplest techniques available that provide a sufficient level of detail. With these goals in mind, it was possible to build a nearly transparent model that displays the characteristics of a typical land combat model as presently used within the defense community.

The model developed is a high resolution, stochastic simulation model which represents a two-sided combat scenario between two homogeneous forces with several entities. Each of these entities has numerous attributes that describe its characteristics and capabilities. The processes that the model simulates are the following:

-terrain representation;
-time representation;
-entity movement;
-searching;
-target acquisition/line of sight;
-target selection;
-target engagement/destruction;
-ammunition expenditure;
-reaction to fire;
-command and control.

A key element of SPARTAN is the ability to interrupt the simulation at any point within the execution. This model interrupt feature allows the student to check the

status of the entities and observe the processes occurring throughout model execution.

1.4 Modeling Definitions

This section defines a number of terms that will provide a more accurate description of the modeling task.

1.4.1 High Resolution. Combat models fall into two major categories, high resolution and low or aggregated resolution. A high resolution model is one that describes the activities and interactions of individual entities. These may be individual soldiers or a weapon system such as an individual tank, or aircraft. Each entity has its own particular characteristic attributes and the model uses these attributes to simulate performance of the entity. The model maintains historical performance output on individual entities. In contrast, low resolution models aggregate forces at various levels into subordinate units. These units then have parameters that describe the overall capabilities of these units. As an example, a high resolution model monitors the position of each soldier while a low resolution model typically keeps a center of mass location for a unit or group of soldiers.

1.4.2 Stochastic Processes. As with most real world activities, in any given combat situation there are usually numerous possible outcomes, and seldom is there any great certainty about which outcome will occur. A stochastic

model describes the uncertainty of events on a real battlefield by using probability theory. The possibility that an event will produce a certain outcome is given a percentile chance of occurrence and a random number draw decides whether the event did occur. This accomodation of chance tends to make the model more believable, and provides a closer representation of reality (5:88).

1.4.3 Scenario. The combat scenario is the motivating force that drives the model. It includes the terrain, the environment, the forces of both sides, and the specific circumstances in which the events are to occur. In this model, the scenario provided is a generic situation based on a commercially developed board game. Unlike most combat models, this model design does not have the specific purpose of providing analytic results. Instead, the purpose of this model is to demonstrate model operations; therefore, the data used is only representative of typical data, and makes no claim of being statistically or historically derived.

1.4.4 Terrain Representation. The process of representing terrain involves partitioning the maneuver space into small regularly shaped polygons that represent sections of terrain. Each of these boxes have attributes associated with it, such as an elevation, a vegetation index, and a trafficability index.

1.4.5 Entity Movement. A model simulates movement of entities across the terrain based on the entities mission and encounters with the enemy. The movement rate is a function of an entity's inherent movement characteristics, combat posture, and terrain characteristics. Operators must specify movement routes as input data for each particular scenario or the entity may react to a situation and modify its movement routes according to some programmed decision logic.

1.4.6 Target Engagement. Target engagement involves the stochastic processes of detecting, selecting and firing at an enemy. Each entity searchs for targets within the limits of its perception, and selects a particular target from those in view. It then fires a weapon at that target. For each shot fired, the model determines a probability of hit and kill. The model uses these probabilities and a random number draw to determine the damage inflicted on the target. Some models only determine deaths, while others include many categories of wounds and damage.

1.4.7 Model Documentation. Documentation is the supporting material provided with the simulation software. This material should provide sufficient information to the potential user to understand the uses and limitations of the model, the functional processes of the model, and the re-

quired input to the model. The documentation that accompanies this model includes the following:

-purpose;
-model overview;
-scenario;
-method of development;
-discussion of modeling techniques and algorithms;
-schematics of model processes;
-user's manual.

1.5 Approach and Methodology

This section briefly describes the process of developing the model.

1.5.1 A major requirement at the beginning was to review literature and interview knowledgeable individuals on the most appropriate techniques to use for development of this model.

1.5.2 The next requirement was to develop a specific scenario that includes the terrain map, force characteristics and the circumstances of the battle. This scenario provides all the necessary input data required for the model.

1.5.3 The lecture notes on high resolution modeling by James K. Hartman were used as a basis to develop a general structure for the model (23). These notes helped define the activities of the various subroutines and illustrated some of the subroutine and database relationships.

1.5.4 Various programming languages were evaluated before selecting the Microsoft QuickBASIC programming lan-

guage. This language was chosen for its relative simplicity, graphics capabilities, and modular programming features.

1.5.5 The model building process started with a simulation of entities moving on the terrain. When this prototype operated satisfactorily, additional capabilities were added and tested. This iterative procedure progressively upgraded the model until the model included all the processes of being able to move, shoot, and react to hostile fire.

1.5.6 Verification ensured that the computer code was operating as intended. This was followed up with a number of individuals performing various laboratory tests with the model. The tests provided feedback from novice operators and experienced modelers on various improvements (19:237). This required several cycles of testing and updating to certify the model for its intended purpose as an instructional demonstration tool.

1.6 Equipment

The equipment needed in this thesis effort was that required to perform the computer programming tasks:

IBM AT Compatible microcomputer with minimum EGA color graphics;
Microsoft DOS 3.3;
Microsoft QuickBASIC 4.5 programming language;
Microsoft QuickBASIC 4.5 User's Manual;
MATHCAD 2.5 Mathematics Software.

The only equipment required to run the model is the standard IBM personal computer with an EGA or better monitor.

1.7 Thesis Organization

Chapter II is a literature review that summarizes pertinent information about the combat modeling processes, discusses methodologies for simulation development, and a rationale for the choice of programming languages. Chapter III discusses the methodology of formulating, building, and coding the simulation model. The process of going from problem definition to a functioning simulation is discussed in detail. Chapter IV provides detailed discussion of the algorithms used to model the combat processes. Lastly, Chapter V concludes the thesis and presents recommendations for further study.

II. Literature Review

2.1 Introduction

The purpose of this chapter is to review the literature pertaining to high resolution combat modeling. The focus of Section 2 is a look at current high resolution models in the US Army. This section discusses some of the more common methods used to simulate combat processes in US Army models. Sections 3 and 4 discuss several approaches to discrete event modeling and several methodologies for model development. Section 5 discusses the rationale behind the choice of a computer programming language.

2.2 High Resolution Combat Models Used by the US Army

Since the early 1960's, the United States Army has developed and used several generations of high resolution combat models. The Army Model Improvement Program [AMIP] has selected two analytic high resolution models for standardization within the US Army (9). These two models are JANUS [A] 2.0, and the Combined Arms and Support Task Force Evaluation Model [CASTFOREM]. Both of these models simulate battalion task force size elements on the battlefield [9]. To give some perspective of this simulation, a typical mechanized battalion task force might include over 600 soldiers with over 100 armored vehicles. AMIP establishes

six functional areas that land combat models can be expected to simulate (12:12--13):

1) maneuver;
2) fire support;
3) air defense;
4) combat service support;
5) intelligence and electronic warfare;
6) force control/command and control.

Both JANUS and CASTFOREM represent most of these systems to some degree. A model can represent any number of these areas or a single one. AMIP would categorize SPARTAN as a "functional area model" for maneuver forces. This category represents "activities directly related to the application of direct combat power" (12:35). The basic elements necessary to model direct fire combat according to Hartman (23:1--10) are

1) battlefield environment;
2) movement representation;
3) target detection algorithms;
4) target selection criteria;
5) target engagement;
6) time advance mechanism.

The next few sections review these necessary elements as modeled in JANUS [A] 2.0 and CASTFOREM.

2.2.1 Battlefield Environment. JANUS and CASTFOREM both use similar methods of representing terrain and the environment.

JANUS uses a terrain file to represent a three dimensional battlefield. The battlefield is divided into square grid cells by evenly spaced vertical and horizontal lines.

These grid cells can be of variable width [from 25 to 200 meters] (14:33). The terrain file maintains an average elevation, trafficability factor, and a ground clutter factor for each cell (14:34). CASTFOREM uses the same techniques for terrain representation although the grid size is smaller, at 25 to 100 meters (13:3-1). Both models also consider a number of atmospheric conditions that are not a factor in the model developed in this research.

2.2.2 Movement Representation. JANUS and CASTFOREM use similar methods for movement. Each entity in a model has certain movement capabilities specified in its table of attributes. Prior to model execution, the operator designates movement control points. The entity then moves along its designated route at speeds dictated by its attributes and the mobility conditions of the route. JANUS attempts to move each entity 50 meters during each movement phase, but may modify this distance for adverse terrain conditions. After a movement, the time to make this move is computed and used to determine when the next movement will occur (14:-411). One major difference between the two models is that a CASTFOREM entity has a shortest path algorithm that allows the entity to deviate from the movement route if the terrain warrants while the JANUS entity can only leave the original movement route because of operator input after the model queries the operator. One specific instance of this is when

the model stops at a minefield and waits until the operator provides a course of action for the entity (14:277;13:3-169). An example of the differences between the models is that a JANUS entity would drive into a mountain and stop while a CASTFOREM entity would find a path around it.

2.2.3 Target Detection Algorithms. Target detection is generally modeled as three factors. The first is the visual signature of the target being observed. The next factor is the transmission of the signature through the intevening atmosphere, and the last factor is the ability of the observer's sensor to see the target signature. The target detection algorithms used in JANUS and CASTFOREM come from the Night Vision Electro-Optical Laboratory [NVEOL] detection model (25:28;13:3-59). A Rand study included the comment that "the best experimental data on the probability of target acquisition by a human observer, through direct vision . . . are probably those obtained by the Army's Night Vision Laboratory (NVL)" (1:3). These algorithms account for detailed modeling of target definition, range, obscurance, and visual capabilities of the observer (14:354). In the NVEOL algorithms, detection probability is the product of two terms. These are "P_1, the probability of detection with unlimited observation time, . . . and P_2, a time-dependent term that takes account of search sectors, field of view, and coverage during a scan time" (1:3).

P_1 is often referred to as the probability of acquisition. This value represents the target's signature for the specific conditions at the time of the search and the capabilities of the observer's sighting system to sense the signature. A P_1 value greater than the observing sensor's minimum threshold indicates that the target can potentially be detected. When a target has a P_1 value that exceeds an observer's threshold value then it goes on a target detection list.

$$P_1 = \frac{(C/M)^{2.7+0.7(C/M)}}{1+(C/M)^{2.7+(C/M)}} \qquad (1)$$

C is the number of resolution cells present in the area of the target dimensions. This value is a function of sensor quality, target contrast, and propagation effects. In JANUS, C is always at 3.5 as a simplification of the model (1:8). M is a scale factor for background clutter of the target (1:4).

$$P_2 = 1-EXP[-(C/M)(t/6.8)] \qquad (2)$$

$$t = 2(FV/SS) \qquad (3)$$

The value of 2/6.8 sec is an empirically derived value for a glimpse time. Search sector (SS) is the total search area observed while field of view (FV) is the portion of the search sector that the observer can view at any instant (1:7).

The target detection routine computes a P_2 value for each potential acquisition. The P_2 equation computes a probability based on the amount of time that the observer was looking in the area of the target. The probability of

detection is compared to a uniform [0,1] random variate. A drawn number less than the P_2 value determines actual observation. This discussion is a brief summary of an extensive discussion of the target detection theory in Rand Note N-3087-DR&E/A/AF (1:3--10). Additional discussion is available in (13;14;23;25).

An important element of terrain representation and target detection is the determination of line of sight [LOS] between the observer and the target. The requirement here is for an algorithm that takes three dimensional terrain coordinates and determines whether there are any intervening terrain features or clutter that would block or reduce the chance of observation between two points (38:823). Both models use similar methods to check the LOS. JANUS samples the elevation of each grid while CASTFOREM checks the elevation of all the grid boundaries along the path between the two points(14:348;13:3-2).

The method of computation is similar in both cases. The height of the observer is computed as its elevation plus the height of the observation system which might be human eye level or a weapon system sight elevation. The height of the target is the elevation at the target's location plus the height of the target (14:348). In the JANUS algorithm, the intervening terrain is sampled at regular intervals. At each point, LOS height is computed and compared against the

terrain height to determine whether LOS is blocked. If LOS still exists then ground clutter and obscuration are checked and a LOS degradation factor is computed that represents partial attenuation (14:348--349). Ground clutter in JANUS can be both vegetation and manmade structures.

2.2.4 Target Selection. Both Army models use ranking schemes for target selection. JANUS determines single shot kill probability [SSPK] for each observed target, and sums the SSPK for all observed targets. Each target has a probability of being selected for engagement proportional to its SSPK. The model then draws a random number that specifies which target to select (14:370--372).

CASTFOREM has a much more elaborate method of determining target selection. Initial input to the model determines which targets have the highest priorities for each type of entity or even whether the target would be engaged at all by this entity. As an example, the modeler might designate infantry fighting vehicles as the highest priority target for M2 infantry fighting vehicles. Among competing targets, CASTFOREM also uses a ranking algorithm:

$$rank = \frac{(tgt\ dimension)(tgt\ contrast)}{(observer/tgt\ effect)} * (flash) \qquad (4)$$

The flash factor is a variable that gives the target a higher priority if it is firing a weapon. This represents the idea that an observer can most easily see a firing

target and will consider it a more dangerous opponent. The observer/target effect includes factors of range, target motion, and intervisibility (13:3-59).

2.2.5 Engagement Assessment. JANUS and CASTFOREM have different methods of determining kill probabilities. JANUS uses a look-up table to determine a SSPK against a target. This table accounts for the type of weapon, type and exposure aspect of the target, motion of target and observer, as well as the range to the target (14:372). A random draw then determines whether the shot hit the target entity. In JANUS, an entity is either dead or alive, that is, there are no levels of damage (14:374).

For hit probability, CASTFOREM uses "a normal bivariate distribution with a bias off the aimpoint and dispersion" (13:3-178). It computes a probability of kill, and level of damage based on the location of the impact. CASTFOREM accounts for accumulation of damage and multiple levels of "kills" (13:3-179).

There are many different definitions of probability of kill and just as many methods of computing them. Many times probabilities of kill are conditional on the occurrence of other events. Some of these might be hitting the target, location of hit, range to the target, or angle of impact. With JANUS, the data is empirically derived from testing by the Ballistics Research Laboratories [BRL], so many of these

factors are accounted for, but for the analyst it is always important to know the test conditions. The JANUS Single Shot/Burst Kill Probabilities [SSPK] account for the following factors:

- range to target unit;
- motion category of firing unit;
- motion category of target unit;
- protection category (exposed or defilade);
- aspect angle of target (head-on or flank).

The CASTFOREM algorithm was also empirically derived with BRL data (13:2-77).

2.3 Conceptual Framework

In the previous sections, the discussion focused on the fundamental concepts of simulating direct fire combat with emphasis on modeling the combat processes. In the remainder of the chapter, the aim is a review of different concepts of structuring simulations, methods of discrete event simulation, and programming languages.

William T. Morris describes model development as an "enrichment" process. Any model should start with a well developed logical structure. From this simple beginning the model goes through an iterative process of modification and testing until the model meets the original objective requirements. He stresses to simplify until it works, then elaborate and enrich (31:B-709).

2.3.1 Model Development. Just as an outline is essential to writing any lengthy document, a methodology must be

established for capturing the complexity of a combat model. Richard Nance has observed during his modeling career that

> the development of a small program or a small model requires little discipline and almost no supportive techniques. Both can be developed fairly rapidly and with little control on the conceptual representation and the eventual implementation. In part, this explains the criticism often leveled at new graduates in computer science or engineering: they do not know how to solve real world problems. (32:220)

Nance defines two roles for a methodology. These are "1) conceptual guidance in the modeling task, and 2) definition of needs for environment designers (32:220)." A Winter Simulation Conference article by Joseph Derrick and Osman Balci considers three types of guidance to be important in the development of discrete event models. These are static design, dynamic design, and implementation methods (17:716). This article reviewed a number of different discrete event methodologies, several of which are applicable to combat modeling.

The first consideration is the choice of an implementation methodology of which Derrick reviews five types. A key discriminator between methods of implementation is the technique used by each for time advance. There are generally two methods of looking at time advance (28:9). These being:

> Next Event Time Advance - this method initializes the time clock at zero and updates the clock to the time of the next most imminent event and continues until no other events exist.

> Fixed Increment Time Advance - a specific increment of time is added to the clock and all processes of the system are checked to see if any status changes or events should have occurred in that increment, if so the system is updated to reflect that the event occurred at the end of the increment. A subset of this approach is to use variable length time steps.

In this case, there is little choice in an implementation methodology since the intent of this model is to emulate JANUS and CASTFOREM, both of which use an event scheduling framework. The article's discussion of the methodologies supports event scheduling as an appropriate choice. Derrick describes the event scheduling routine as an efficient method of execution when the simulation involves numerous independent and less interactive entities (17:714). This statement is an apt description of a combat model with its many combatants having a wide range of activities in most cases. Derrick's other implementation methodologies for discrete event simulation were activity scanning, process interaction, and transaction flow which have characteristics that make them unattractive for combat modeling. Activity scanning requires the model to check each activity at each time advance, thus making it relatively inefficient in a discrete event model (17:714). Process interaction and transaction flow are both process oriented, and are most efficient in models representing series of queues and servers (17:712). John Evans states in his discussion on alternative strategies,

> the event-scheduling approach . . . is probably the most natural way to proceed and is frequently used when starting from scratch using a general-purpose programming language. (21:79)

In his Introduction to Simulation, William Biles' discussion states that the event scheduling approach concentrates on the events and the resulting changes in the system state. Future events go chronologically into an event calendar. Time advances as each event is pulled off the event calendar. The simulation must have a method to pull the next event off the list, advance time, transfer control of the program to the next event, generate new events and sort the event calendar as each new event arrives to it (6:8). The mechanisms to perform these tasks are discussed in greater detail in Section 2.3.3.

In terms of static and dynamic development frameworks, Evans does not see any distinct advantages among conceptual frameworks except that some are better in particular applications (21:78). The two methodologies suited to event scheduling are structured modeling [SM] and conical methodology [CM] (16:716). Of these, SM orients on building modules around queueing events, (8:253) a situation not encountered in combat models.

Conical methodology as described by Nance "prescribes a top-down model definition followed by a bottom-up model specification" (32:221). CM stresses a lifecycle approach to simulation development. The process begins with a decom

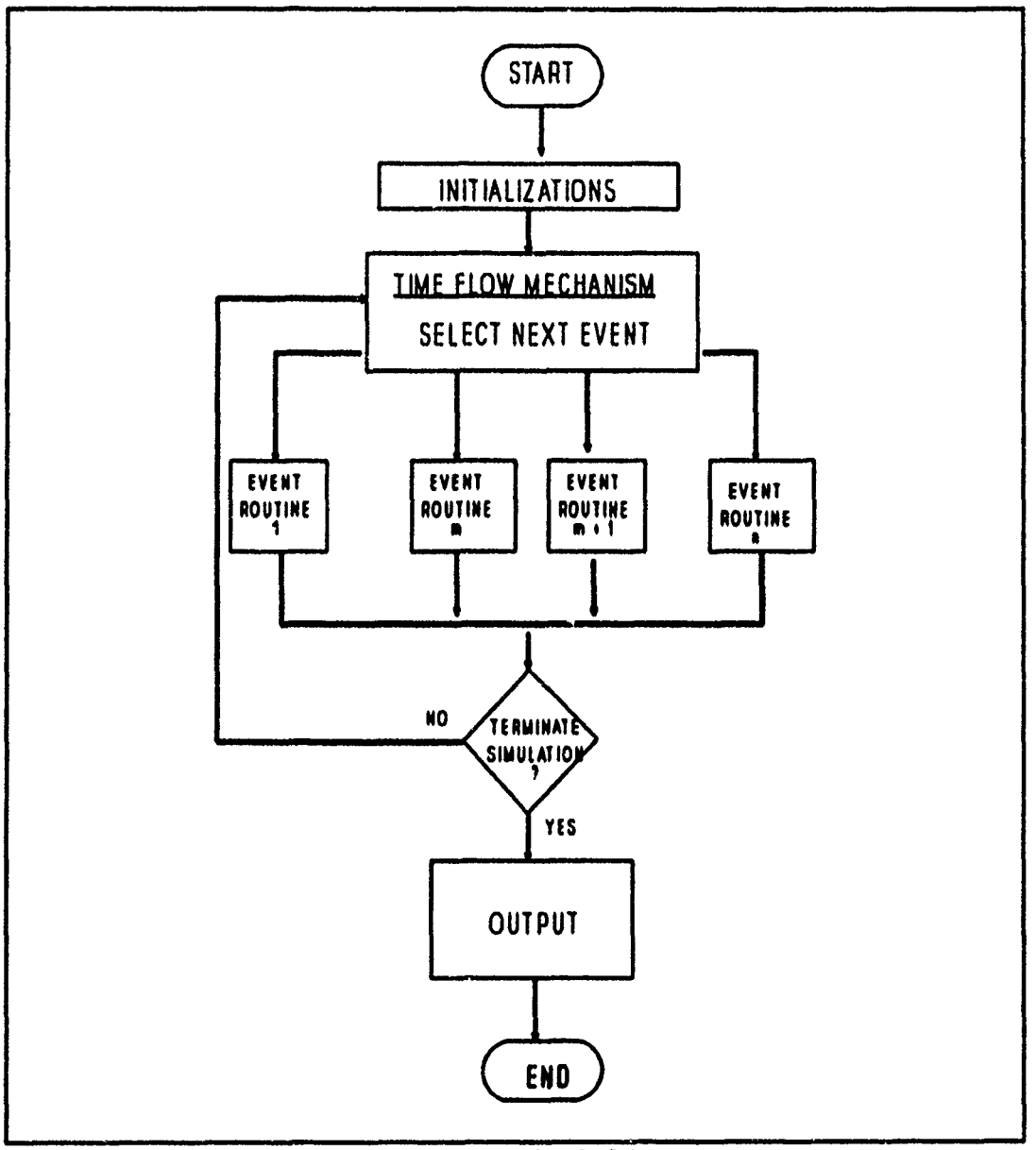

Figure 1 Event Scheduling Process

position of the model elements and their relationships as entities, attributes, and activities. From here, the model builder develops the dynamic relationships of the elements. A hierarchical and functional decomposition allow the development of a structured program plan. The next phase of the

process is an iterative process of building and refining. This usually results in some redefinitions of the problem and further iterations. The final phase is verification and validation of the model. An ongoing task throughout the process is documentation of the model and the development process (32:221-223).

While the conical methodology gives general guidance on the process of creating and maintaining a viable model, Osman Balci provides some specific guidance on the steps necessary to implement an event scheduling program. These are (2:290):

1) Identify the objects and attributes.
2) Identify the attributes of the system.
3) Define what causes a change in the value of an attribute as an event.
4) Write a subroutine to execute each event.
5) Follow the logic of an event scheduling routine with an event list to develop the simulation program.

2.3.2 Model Validation and Verification.

The conical methodology emphasizes that validation and verification should be performed throughout the development of a simulation model. In simple terms, "model verification examines whether the computerized model runs as intended," and "validation examines the correspondence of the model and its outputs to perceived reality (4:14)". Validation and verification is a very controversial subject with numerous different approaches to the problem. A 1979 GAO report on model assessment reviewed many of the more popular methods

and attempted to find the common thoughts within the various methods (10). The result is "a minimal set of criteria deemed necessary for model evaluation (4:15)." The following paragraphs list the areas deemed important in model assessment.

Documentation - should provide the level of detail necessary to allow model users to understand the processes, the underlying assumptions, the limitations and the results of any model validation effort. "At a minimum, the model documentation should describe the data structure, the key elements of the model, the general flow logic, and all the variables...(36:78)."

Validity

Theoretical validity - ensuring that the model's mathematical modeling techniques are appropriate for the level of detail required by the user.

Data validity - involves establishing the accuracy, completeness, and appropriateness of the original data and verifying the methods in which the data is transformed within the model (3:21).

Operational validity - is the issue of divergence of model results from "real world" values. It also involves testing the underlying assumptions and theories to determine how well they represent the phenomenon. A key part of operational validity is some basic agreement between the

model's sponsors and developers as to the types of validation required and the amount of "realism" expected (4:16).

Computer model verification - "is ensuring that the coding of the conceptual model is correct (4:16)." The mathematical and logical relationships should be correctly formulated. Throughout the development process, each component of the model should be checked separately and as a functioning part of the whole model for proper operation. The following list provides six methods for verifying code (4:16):

1) structured programming methods;
2) program testing;
3) tracing the simulation;
4) logical relationship checks;
5) comparison to analytic models;
6) graphics.

Maintainability - this is the issue of determining whether the model will be able to correctly represent the system under analysis throughout the life cycle of the system. Maintainability requires that a model be designed so that it can be modified as required. Two important aspects of maintainability are review and update. Review requirements are that the model proponent schedules periodic looks at the model to ensure that the model still uses the best data readily available and to determine whether modifications are necessary on the model. The update requirement is to ensure that the model proponent has developed some procedure or guideline to indicate when the model requires im-

provements in order to continue to meet its design goals (10:22).

Usability - this category includes a number of factors involved with the ease of use of the model. An important factor is to determine availability of the required data. Another key point is whether the model output is understandable, and whether the output can be modified to suit particular studies. A short list of additional topics under usability would include portability, run time, and set up time (10:23).

The GAO authors have left the terms in the list, defined in general terms, so they can be applied to a wide variety of models. Some subjects are emphasized in specific models more than others depending on the degree of detail required from the model and user defined requirements (4:-15). Performing an evaluation of a simulation using these categories should provide a comprehensive assessment of the model's worth and applicability.

2.3.3 Scheduling Future Events. The heart of any discrete event scheduling simulation is the event file synchronization structure. This mechanism provides order to the operation of the model. Luis Rodriguez states in his comparison of these structures that "the most important factor involved in the total execution time of a simulation is the time required to file an event" (35:189). He looks

at linked list structures which most general simulation languages use and compares them to several newer techniques. This 1982 report states that there are faster scheduling techniques, but linked lists are still the predominant method in use (35:189). More complicated methods with multiple lists are much faster, but have a requirement for dynamic parameters which make them more difficult to use (35:189).

Both JANUS and CASTFOREM use similar discrete event scheduling approaches to model continuous time (2). The master time flow mechanism in JANUS is an event scheduling routine that uses multiple linked lists (27). CASTFOREM is written in SIMSCRIPT II.5 (13:v) which uses a discrete event time advance mechanism of multiple linked linear lists similar to the JANUS model (30:73).

As previously discussed, these routines simulate continuous time by advancing time to the occurrence time of the next event. Richard Nance provided a simple description of an event as any state change in the model (17:711). The linked linear list is a sorted data record that keeps the future events in a sequential order of occurrence (28:71). As the model pulls the next event from the list, time is updated, the event is processed, new events are added to the future event list, and the list is relinked. This process continues until a specified termination point is reached.

This point might be a designated time limit, a specified attrition level or any number of switches specified by the user that would be triggered by some event occurrence (2:-290).

In whatever form it takes, event set management basically involves database manipulation where the data consists of event records with fields containing event types, and occurrence times as the minimum information required to maintain the event calendar (41:153). There are two general forms of data storage, these being sequential, commonly called computed address method, and random access or link addressing method (42:54). The sequential form is a simpler method of storage since the only requirement is to keep the data [events] stored in chronological order. The only tools required are routines to add events, sort events and pull events off the list. The drawback to this method is that it is inefficient because it must re-sort and manipulate all the events on the event calendar for each iteration. An iteration being the cycle of selecting and executing the next event. It is considered a brute force method (28:138).

If a sequential data storage technique is to be used then an efficient sorting routine is a critical element. A 1991 comparison of sorting routines shows that:

> For nearly sorted or midsized files (a few thousand elements), Shellsort [named for the developer Shell] performs as well as or better than any other known algorithm, including quicksort. Furthermore it is an

> in-place sorting algorithm requiring little extra space and is easy to code. (43:88)

A 1985 comparison by Dudewicz also showed that Shellsort is faster than other methods when inserting into a sorted list (18:293). The heapsort and quicksort were 2-4 times faster at sorting a random list (18:293); however, this is not a big concern since the key property of an event list is that new events are inserted into already sorted lists.

The more common method of random access requires much more programming overhead to reduce the processing time required to manipulate the lists. If a random access approach is taken for event management then there are a plethora of possible techniques. Since the basic structure used by the JANUS and CASTFOREM models is the doubly linked linear list, it is discussed here for completeness. The doubly linked linear list structure consists of two lists, pointers and event records. A pointer is a variable that references a location in a data structure (42:54). In a double linked list, each list may have several pointers that identify the head, tail, and also intermediate locations in the list. Event records each have predecessor links and successor links that identify the storage locations of the event that precedes and the event that follows an event. In this manner all events have a reference to the other events in the list and can be accessed without being physically sequenced in order. One list stores the active event

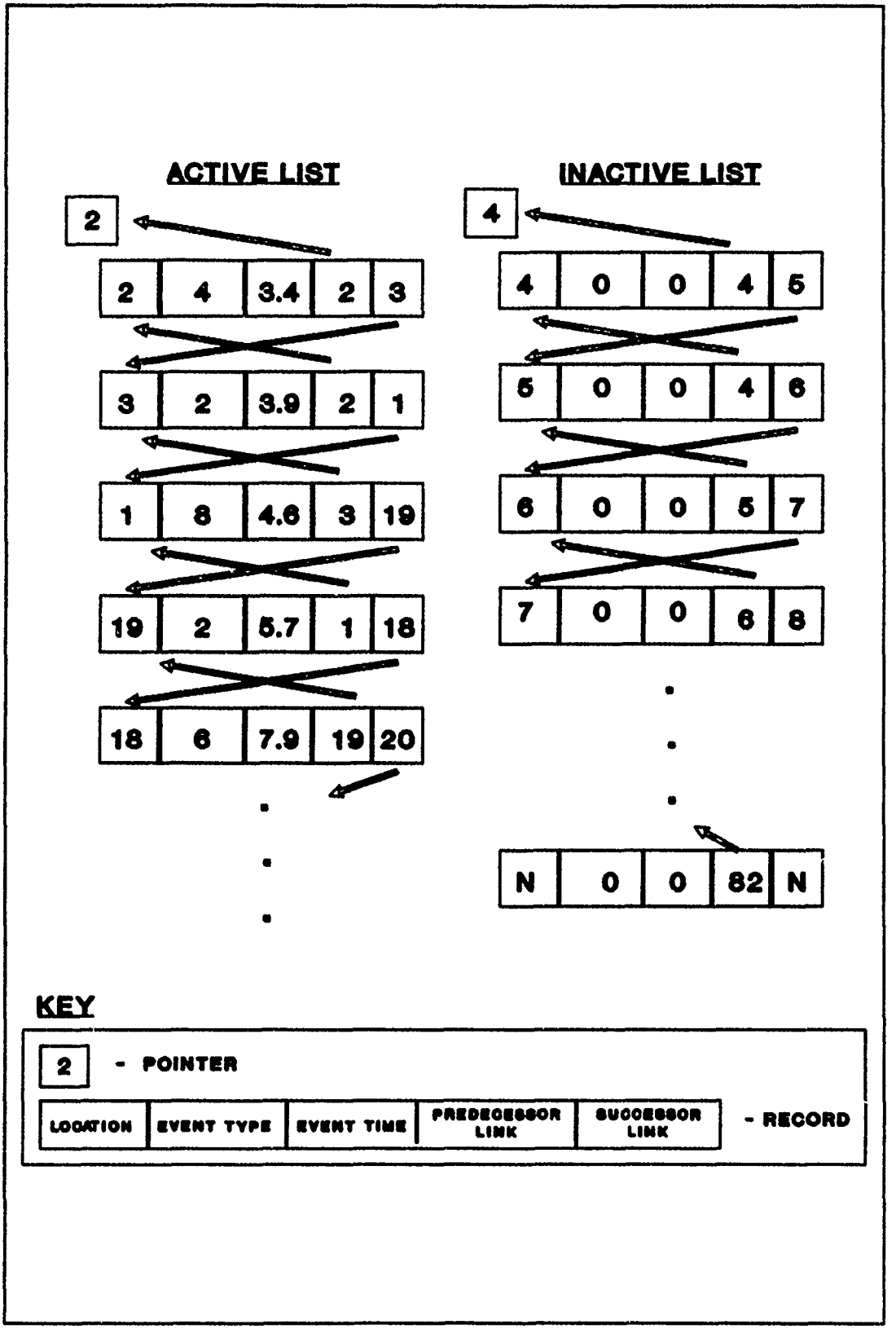

Figure 2 Example of Doubly Linked List Structure

records while the second is a list of available record spaces (28:135-136). Figure 2 shows how the records are indexed and linked in each of the two lists.

A complicating requirement in the development of simulation languages is the need to preempt events already in the list. One method is to pull all the active events from the event calendar for any entity that is eliminated from the simulation. In a Winter Simulation Conference article, Henriksen provides a description of a simple alternative. If a soldier kills another soldier who is already on the schedule to perform an event, the dead soldier's activity cannot be allowed to occur. Rather than pulling the event out of the list, as a simulation language would do, the kill triggers an attribute switch on the soldier that the event modules check before execution (24:352). In general, switches are a useful method of modifying the model's program logic as the simulation proceeds and are used throughout JANUS and CASTFOREM.

2.4 Random Number Generation

Any simulation model of a system with inherent random processes must have a means of obtaining random variates. Two methods to accomplish are to draw random variates from a table or to use an arithmetic method to generate a numerical value in accordance with the appropriate distribution (28:-421). Both methods are widely used in simulation. Random

numbers derived in this manner are referred to as pseudo-random numbers. This is because both methods are reproducible deterministic methods of generating a stream of random variates. Drawing numbers from a table is computationally time consuming, but can be faster than arithmetically computing random variates of complex distributions (28:421). Tables are also useful when empirically derived distribution values are available. There are a wide variety of arithmetic methods available to provide random variates. An article, by Arne Thesen, compares numerous methods to compute uniform, exponential, normal, and gamma distributions. His comparisons are based on the qualities of computational efficiency, ease of programming, non-degeneracy, randomness of sequence, and independence from seeds and other variates (41:157). The uniform distribution with an interval of [0,1] is a key element in the numerical computation of random variates. Most other distributions can be obtained by transformations of uniform variates (28:420).

JANUS uses the U[0,1] generator and tables to obtain random variates. To model weapons effects, empirically derived probabilities of kill are sampled from a data table, and compared against a U[0,1] value to determine the results (14:A2). The target selection algorithm uses a U[0,1] value to pick targets. The NVEOL target detection model uses the U[0,1] distribution for comparison to a table of exponen-

tially distributed values in the continuous search algorithm(1:8). CASTFOREM also uses the uniform distribution for comparison to exponential distributions as well as a bivariate normal distribution to compute probabilities of kill (13:2-77). Designers of both models have chosen to use look up tables rather than using computationally demanding transform algorithms (14;13:2-83). Thesen's article discusses why the transform operations that are simple to program become very computationally demanding. Primarily, it is the result of using the logarithmic function " which uses Taylor series expansions with a large number of terms (41:159)."

2.5 Programming Language

There are a wide variety of programming languages that are used for simulation programming. Most of these fall into two categories. The first is general purpose simulation languages, the best known of which are SLAM II, SIMSCRIPT, and GPSS. The advantages of these are that they provide a "core of facilities" such as random variate generation, entity management, and event list management (28:263). The disadvantage is that any potential user must have access to these languages (39:93). The second category includes the higher level programming languages such as FORTRAN, C, PASCAL, and BASIC. One advantage of these languages is portability, which means the program can be easy adapted to run on different computer systems. With high level languag-

es, even the compiler may not be necessary if the simulation can be used in an executable format. Another advantage is that higher level programming languages permit the application of good programming techniques and a person learning simulation can see the hierachical structure of event subroutines and data files (39:94). Andrew Seila, a professor at the University of Georgia, has found "persons learning discrete event simulation benefit by seeing the data structures used and operations that are performed on them in the simulation program (39:94)."

Since the model under development is instructional in nature, the features of portability, ease of use, and graphics capability are of prime importance. FORTRAN is in wide use in military simulations (28:253) and would be the author's choice except that it does not facilitate use of graphics except with add-on software packages. An article in the Journal of Pascal, Ada, and Modula-2 comparing the programming languages Pascal and C provided a good analysis of each language (44). This article was then used as a base of comparison with a similar article on QuickBASIC version 4.5. This analysis provided some measure of the strengths and features of the different languages. C is a very capable language, but is not user friendly. It would require an extensive training period before a student could work on the model code. Pascal has all the necessary capabilities for

simulation programs, but is not as well suited to modular programming as QuickBASIC version 4.5 (44:10). QUICKBASIC version 4.5 also meets all the requirements above while also being in a format similar to FORTRAN. It is well suited as an educational language. It allows modular construction, it automatically checks syntax and it has debugging tools that were invaluable to an inexperienced programmer (37:295). The article describing QuickBASIC states, "The new ANSI BASIC standard includes many new features, some of which have outdone Pascal, Modula-2, and even C" (37:295). Quick-BASIC 4.5 has all the features of a modern structured programming language.

III. Model Development Process

3.1 Introduction

This chapter describes the general model building process using to create the SPARTAN combat model. The flowchart in Figure 3 shows the overall process required to develop the model.

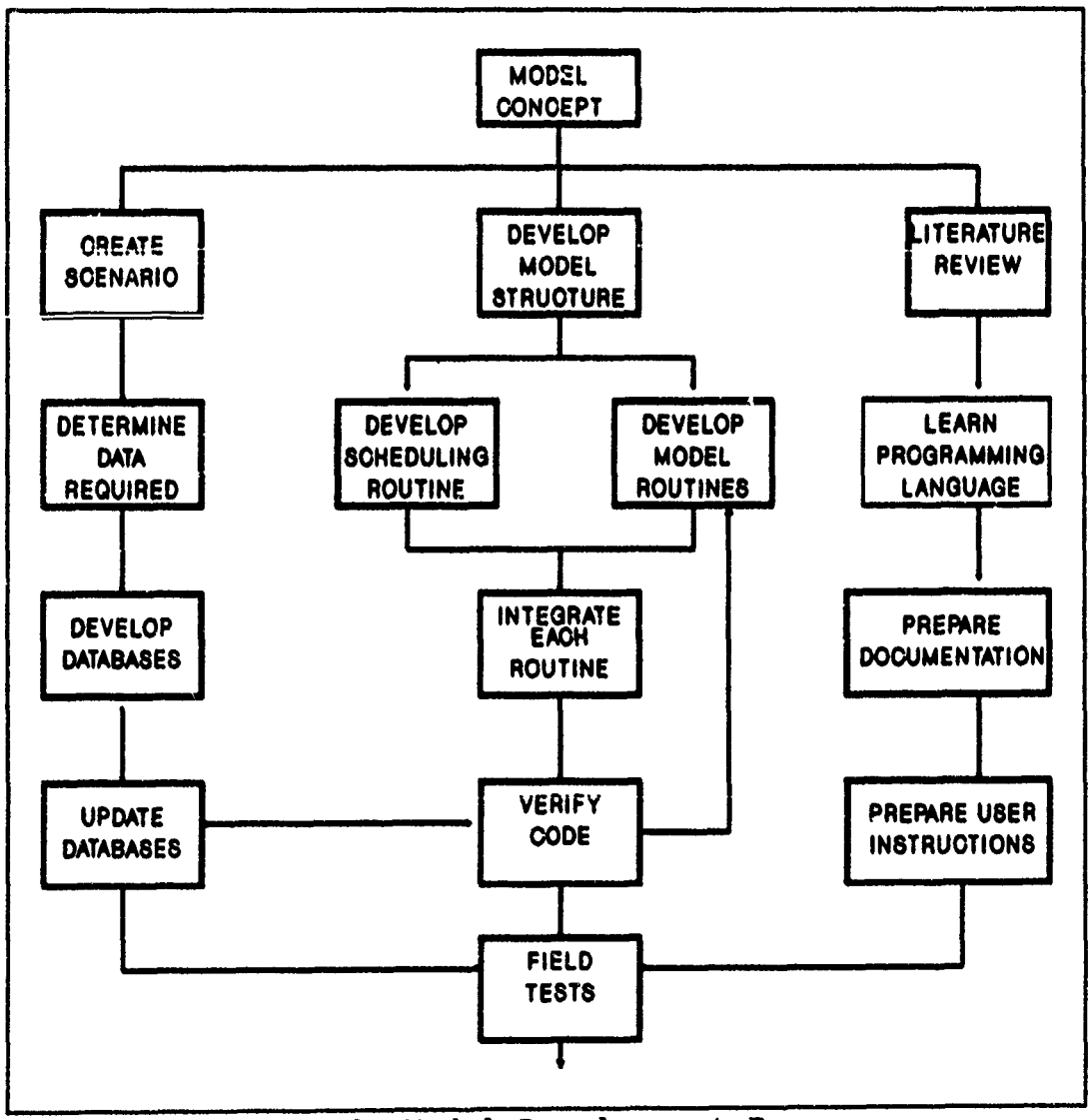

Figure 3 Model Development Process

The process appears much cleaner on the chart than it was in actual development. At each stage of the development, new information was included and often algorithms were revised. In each case, there was a cascade effect that required changing elements such as the documentation, the help files or the preprocessor.

As the flowchart depicts, time management was required to concurrently create the model code, construct the model support tools, and prepare the documentation. The chart portrays three parallel processes, but to a certain extent each of the processes was dependent on the others, and were done in a somewhat synchronous fashion.

3.2 Development Methodology

The general principles of the conical methodology were applied as a framework for simulation development. An article by Richard Nance is the primary source for this approach. Figure 4 is an extracted outline of this methodology (31:38--43). Further information can be found in (2,3,4,17,33).

The following sections progress from problem formulation through implementation of the functioning simulation model. An important procedure in the systematic development approach to this complex problem was to interject several steps in the process between the conceptual problem statement and the coding process that define the specifications of the static and dynamic aspects of the model. The goal

I. Statement of the study objectives

A. Definitions
B. Assumptions regarding objectives

II. Modeling environment

A. Modeling effort available
B. Modeling assumptions

1. Boundaries
2. Interactions with environment

a) Input description
b) Assumptions on model/environment feedback or cross effects
c) Output and format decisions

III. Model Definition

A. Identifying the objects and their attributes
B. Submodels with possible sublevels

a) Value attributes
b) Relational attributes

IV. Model Validation and Verification Procedures

A. Validation tests
B. Verification criteria and tests

V. Model experimentation

Figure 4 Conical Methodology Outline

was to determine what objects and activities were being modeled and the relationships that were to exist between them. This provided sufficient groundwork to start creating functions and subprograms that would interact in a logical progression. Knowing the data requirements before the programming began meant that it was much simpler to determine when to pass data and when to call values from an array. It also facilitated a modular structure that limited

data access by subprograms to their required portions. The following sections provide a more detailed discussion of the process used to develop SPARTAN.

3.3 Problem Definition

The first requirement of the conical methodology is to determine the real nature of the problem and what techniques should be used to solve this problem. At this stage in the development, it is important that the analyst and user look at the various techniques available to solve the problem. Simulation is only one tool that should only be applied when it is the best tool available. In this instance the problem to be solved as discussed in chapter I is:

> build a high resolution land combat model for wargaming analysis of small scale direct fire conflict between two homogeneous forces. This model should include common solution techniques to the basic processes of a maneuver warfare model as outlined in AMIP (12:12). The model should be simple to operate with a minimum amount of instruction, so that it is a useful compliment to a course of instruction in high resolution combat modeling.

In this particular case, the problem identifies the simulation model solution as the best tool to use.

The problem statement provides a starting point for looking at the project in terms of requirements. Some of which are:

- The SPARTAN program must be capable of running on all IBM XT compatible computers [512K] with color graphics capability.
  - -- This will enable the widest possible range of users to be able to operate the model.

-- This requirement goes back to the idea of building the model with the user in mind. In this case, the users are students.

- The user will be able to operate the model and understand its internal processes with little effort beyond reading the documentation.

  -- Students operating this model may have little or no previous experience with computers or modeling.

- The model will illustrate some of the common techniques used by the present generation of Army models to represent combat.

  -- This is in keeping with the combat modeling course objectives.

- The model will provide animated color graphics depicting the battle in progress.

  -- A graphical representation of a complex simulation is the best means of ensuring that the analyst can comprehend and interpret all the interactions in even a simple combat simulation (37:822).

- The model will require no human participation once it is started, but will provide in-progress status to the user.

  -- This is a normal feature of analytic models which ensures the model will have the same result for each run with the same initial conditions.

- The simulation should run faster than real time if it is to be useful.

  -- The intent of the model is for the student to work with it. If the student has to wait for extended periods on each run, he will be less inclined to make good use of the model.

  -- Even a model such as CASTFOREM that may run 24 hours for a 2 hour battle incorporates many time saving features at the expense of some realism.

- Preprocessors should be part of the model to simplify the operation of the model, so the user can focus on the processes in the model.

    -- The intent of this model is to illustrate a typical combat model and not to teach programming techniques, so the coding in the model should be as transparent as possible to the user.

- An online help facility and instructional material should be part of the model.

    -- This will simplify the operation of the model and enable the student to quickly refer to a discussion of the various processes modeled in SPARTAN.

    -- This ensures that some form of documentation is available even if a paper copy is not available.

3.3.1 Definitions.

1) "An object - is anything that can be characterized by one or more attributes to which values are assigned (33:193)." Usually an entity or event.

2) "Attributes - record information about the object that is useful for modeling task; they assume values as needed to record changes in the object's state (33:193)."

3) "System state - a collection of variables the values of which define the state of the system at a given point in time (6:8)."

4) "Set - a collection of associated entities (6:8)."

3.3.2 Assumptions Regarding Objectives.

1) This model demonstrates modeling concepts and will never be used for analytic work.

2) The scope and level of detail of the model were limited by the requirement to complete the project in

five months for the start of the next combat modeling course sequence.

3) The slow operating speed of personal computers required some further simplifications to keep the speed at a reasonable rate.

4) The event step approach provides a reasonable representation of the real process.

3.4 Modeling Environment

3.4.1 Modeling Effort. A major consideration in the development of any model must be an appraisal of the time and effort that can be allocated to the project. In this case, the project required approximately one half of a man-year of effort.

3.4.2 Modeling Assumptions. The following list is by no means comprehensive, but does discuss some of the major assumptions used in developing the model and the initial scenario. Chapter IV contains many additional assumptions within the discussions of the individual combat processes.

1) Boundaries. This model has a strict boundary that limits any external influence of the outcome. The interactions of the soldier entities are modeled in a somewhat sterile environment where only the terrain and the enemy have an impact on the soldiers. There is no logistics support or higher level command and control aside from what the soldier files have available upon initiation of the simulation.

2) Missions. At the beginning of the simulation, each soldier has a mission as defined by his direction of movement, speed, and posture. He will orient on that mission until such time as his squad leader causes him to adjust or the soldier reacts to contact with the enemy.

3) A soldier's performance attributes will not change over time or as a result of changes in the battlefield situation. As an example, a soldier entity will never become tired.

3.5 Model Definition

3.5.1 Objects and Attributes. Identifying the objects and their attributes may also be referred to as the static specification of the model. An initial requirement in laying out the model is to determine what objects can either change the state of the model or cause some action to occur. The objects used in this model are:

Grids - The data records for terrain representation
Events - The records controlling actions of the model
Soldiers - The operational entities

Each of the objects in the model have a certain number of attributes that provide a more descriptive representation of the object. It was important to determine early what attributes of each object should be represented in the model. The determination of attribute requirements goes back to the initial model objectives. At this point, it is important to remember that a model is just a representation

of reality and the level of detail should only be that necessary to address the problem at hand. In this instance, the model is expected to demonstrate the modeling processes, rather than act as an analytic tool; therefore, the attributes required will only be those necessary for the combat processes noted in chapter I.

Table 1 Terrain Attributes

Variable	Type	Description
horz	int	the horizontal grid index from left to right (20 meters apart) 1 - 50
vert	int	the vertical grid index from bottom to top (20 meters apart) 1-50
elev	real	the average grid height range of 0.0 - 69.0 meters
mobfac	real	a trafficability index that affects speed of entity movement range 0.0 - 1.0

Table 2 Event Attributes

Variable	Type	Description
time	real	scheduled occurrence time of the event range is 0000.0 - 9999.0
type	int	identifies type of event range from 1 - 7
actor	int	identifies the soldier performing the event range from 1 - 12
predecessor link	int	pointer to event that preceeds this event range 1 - 99
successor link	int	pointer to event that follows this event range 1 - 99

Table 3 Soldier Attributes

Variable	Type	Description
x	real	soldier's current horizontal coordinate range of 0.0 - 1000.0m
y	real	soldier's current vertical coordinate range of 0.0 - 1000.0m
z	real	soldier's current elevation range of 0.0 - 69.0m
xlast	real	soldier's last horizontal coordinate range of 0.0 - 1000.0m
ylast	real	soldier's last vertical coordinate range of 0.0 - 1000.0m
size	real	soldier's height range of 1.6 - 2.0m
speed	real	movement factor range of 0.0 - 40.0 units per move
dir	real	direction of travel/orientation range of 0.0 - 2 * pi radians
moving	int	a flag that gives movement intent (0) stopped: (1) moving
wpnrng	real	effective range of soldier's weapon range of 300 - 600m
ammo	int	ammunition available to soldier range 0 - 200 rounds
status	int	flag indicates whether soldier is (0) dead: (1) alive: (2) wounded
posture	real	value indicates whether figure is (.25) prone: (.5) crouch: (1.0) upright
incmd	int	indicate squad leader (1) leader (0) subordinate
atkdir	real	original movement direction range of 0.0 - 2 * pi radians
tgteng	int	which target soldier has selected to engage range of 1 - 12
side	int	flag indicates allegiance of soldier (-1) red: (1) blue

The only system attributes in this model are the index attribute time represented by the real variable [time], the

atmospheric attenuation coefficient, and the counter on the number of events processed. The system always starts at a clock time of 000.0 seconds.

3.5.2 Model Activities.

The activities performed by the simulation fall into three basic categories. The first category includes those activities that are considered to occur instantaneously with no significant duration. In most cases, this is a simplifying assumption since most of them would require some time duration. Within this category, some are represented as events and some are performed as "bookkeeping", to update the attributes as required. The second category activities have some duration and are represented by a start event and a stop event. And the last category are those tasks required to perform the system maintenance activities.

Instantaneous Activities

line of sight [LOS]: This activity checks for intervening terrain between the observer and target. Provides a result of 1 [LOS exists] or 0 [LOS is blocked].

direct fire engagement: This activity is the action of firing a weapon and determining the bullet's time of flight. It then schedules the bullet's impact.

determine target size

determine observer to target ranges

develop target lists

decrement ammunition when expended

determine probability of hit

determine probability of kill

plot graphic representation of entities

plot graphic representation of engagements

change posture of soldiers

change direction of travel on orders from the squad leader

impact of the projectile

Time Duration Events

start move: This event determines a new location that the soldier will move to and the time it takes to get there. Soldier's attributes reflect new position.

stop move: This event updates the soldier's graphic location and schedules his next startmove.

search for targets: This is a constant process, but rather than start and stop, a new search cycle is scheduled by the time computed for the previous cycle.

reaction to fire: This event is called by a firing sequence and the target decides what action to take after being fired upon.

target selection: picking a target off the target list using a random number draw and some simple decision logic This requires some decision time.

System Maintenance Activities

initialize data sets and event list

maintain event list [future event calendar]

select next event

update time clock

add new events to list [schedule events]

delete events from list when they become obsolete

generate pseudorandom variates

transfer program control among event subprograms as required

terminating the program based on user specifications

store historical data for battle updates and final output

compute summarized data for the final output

3.6 Model Development

This stage of the process consisted of creating the data-files, developing an event scheduling routine, and adding routines that performed the various combat processes.

3.6.1 Creating a Database. Creation of the database had to be the first step in the model formulation because all other portions of the model rely on calling values from data files. The four primary data files are the soldier attributes, the terrain file, the probability tables and the initial events for each entity that start the simulation. The data files took several forms before they reached their present state. Preprocessors were developed for each file that allow the student to create a generic file, view the contents of the file, and edit the file attributes to create unique scenarios. Initially, this was accomplished using record arrays. Records can store many attributes that can be called with a single variable name. This makes passing variables between routines much less complicated. The drawback to this method was that the files were not in a readable ASCII format. It was felt that the student users of the model should be able to view the contents of the file

without a special viewing program. As a result, flat files in ASCII format were used instead. The use of flat files required greater storage space for the data arrays and more lines of code to access each individual data element, but in hindsight the model is much easier to understand especially for students not familiar with record data structures.

The data elements in each file are the same attributes listed in section 3.5.

3.6.2 Event Set Management. The most difficult task was creating a simple event scheduling routine. This routine took several forms before the present doubly linked list approach was adopted.

Since simplicity was a goal, a sequential event list sorted between events was initially created. This program used the Shellsort as discussed in chapter II and was relatively easy to implement. The event records contained four fields for event type, time, and two entity identifiers. Computational efficiency was an immediate problem that killed this approach. The time required to manipulate the event list slowed even a simple movement routine to the point where it was impractical to continue.

The next approach was to use a doubly linked list implementation designed by MAJ Morlan. This scheduling routine is a simple version of the doubly linked list that runs efficiently and meets all the requirements of the model. This routine uses two linked lists. These are the

active list which maintains the location of the future events and the inactive list which maintains the locations of the unused storage locations. One pointer is used for each list to point to the head of the list. Each event record consists of five elements. These are the event type, event time, event actor, predecessor link, and successor link. The original goal was to have enough data in each event record to describe who would do it, what would be done, when it would happen and to whom. A grammatical analogy would have been a complete sentence structure with subject, verb and direct object. In the final approach, the second entity identifier was dropped because it is simpler to access the name from the soldier's attribute array when needed rather than manipulate that field with each event list update. In its simplest fashion, this scheduling routine was checked in all conceivable requirements to ensure it functions properly. These cases include

1) Adding an event to the top of the list;
2) Adding an event to the middle of the list;
3) Adding an event to the bottom of the list;
4) Removing an event from the top of the list;
5) Removing an event from the middle of the list;
6) Removing an event from the bottom of the list.

Some other cases were eliminated by requiring that the initial events are loaded in chronological order and the program automatically goes to output if no active events are on the list.

The subprograms required to perform event set management are the following:

1) addevent - adds events to active event list [calendar];
2) move - moves the event from one list to the other;
3) remove - calls move to pull an event from top middle or bottom of list;
4) clock - pulls the next event off active event calendar;
5) event - updates system clock and calls next event;
6) linkempties - initializes the file by giving all records initial link values;
7) initialize - loads all the data files required to start the model.

Several brief examples will illustrate the process performed by the scheduling routine of adding and removing activities to the end or in the middle of a list [refer to the illustration in Figure 2]. In the first case, an event is added to the head of a list. This requires both list pointer values to change, the predecessor and successor links of the new event to be set, and the predecessor link of the next event to be altered. The next example is when an event is added to the middle of a list. In this case, the links of the new event have to be set as well as the successor link of the event occurring just before the new event and the predecessor link of the event immediately after. The pointers at the head of the list are unaffected. In this next example, an event is removed from the top of the list. This task sets the predecessor value of the next event on the list to the pointer value then moves the event over to the inactive list. The last example illustrates removing an event from the middle of a list. Here the successor link of the event ahead is set to the value of the

event following the removed event and the predecessor link of the following event is set to the value of the event ahead of the removed event. Once again the removed event is then placed in the inactive list.

3.6.3 Model Enrichment. The process of model building was cyclic and involved making gradual improvements to the basic model and testing the proper functioning of the code before adding to the model again. The goal of this approach was to minimize the debugging required as the model grew in complexity.

The first step in this process was to develop an initialization subprogram that accessed all necessary datafiles, and created the storage arrays for each file. These arrays store the terrain attributes, soldier attributes, initial events, and probability tables. In each case, a debug routine was used to ensure that the data was being correctly stored in the arrays and in the correct format.

Integrity of the data was an important issue. This involved ensuring the correctness and accuracy of the numerical values. The model required a mixture of real, and integer values, but to simplify storage and access all the data arrays were stored as single precision real values. This minimized the number of arrays, but meant that certain fields had to be converted to integer values prior to use. The use of incorrect data types and incorrect variable

ranges proved to be common problems when debugging the subprograms.

The combat processes were added to the model in the order required to test their performance. The movement subprograms were the first to be implemented. These were followed by the search routine, engagement routine and finally the decision logic processes. In each case, the model was run under various conditions at each stage to ensure it was functioning as required. The worst debug problems were those that appeared long after the component was initially integrated into the model. On these occasions it was much more difficult to isolate the problem.

A typical improvement cycle involved creating a simple subprogram and having it print out all the data values it required. This would ensure that the correct data was being provided to the routine. Next the subprogram would be expanded to include the necessary logic and some simple algorithms. This would also be printed to the screen and worked with until correct. It usually took several upgrades to get each subprogram to function properly. An important element was always to fix any problem before adding to the model again. On the few occasions when this method was not strictly adhered to, many hours were spent searching through the code for errors.

An additional task with each subprogram was to document the code as it was written. This internal documentation

made the code much more understandable in the later stages of development and ensured that others could understand and work with the code. A second set of eyes was often invaluable to solving code errors.

3.6.4 Probability Distributions.

A variety of probability distributions were used within the model for various processes. An effort was made to maintain simplicity rather than accuracy by limiting the distributions used in SPARTAN. The QUICKBasic 4.5 uniform [0,1] pseudorandom number generator is the basis for all stochastic processes. A triangular distribution function was included that uses the QB4.5 U[0,1] random variates in a transform operation. This distribution was chosen because the transform operation is efficient and the output can be used to represent both symmetric and skewed distributions. The function is given a low, high and mode values and returns a value within this range. The algorithm for the transform was adapted from Pritsker (34:713). This transform provides a rough approximation to a normal distribution when the mode is centered and the extreme values are assumed to be within two standard deviations from the mean. Probability tables were created using the equations in chapter II, for those distributions that use exponential or normal distributions. This is similar to the method used by both JANUS and CASTFOREM for weapons data. In the case of these two models, the data is the product of extensive

weapon testing. The values in the SPARTAN tables were created by using some known values from the modeling literature and approximating the unknown parameters so that the values in the table were reasonable. Even though the tables do not have accurate values, they still have distributions that are representative of the empirical data [e.g. probabilities of hit still vary exponentially with range even if the range of values is not accurate]. These tables were created using MATHCAD templates which generated datafiles containing the required values [See Appendix A for templates].

3.6.5 Instructional Components.

Since SPARTAN had an instructional purpose, a number of features had to be added to the model that might not be standard for a purely analytic model. The first and foremost was that it had to be simple to operate. The intended user might have very limited knowledge of computer operations, and a short amount of time to work with the model. Next, a help function was added to the model that provides basic information on operating the model and a brief overview of the various processes portrayed by the simulation model. The help function was set up as a hierachical menu that enables easy access to brief general topic discussions and subsequent access to more detailed subtopics.

A preprocessor [STARTUP] was developed that enables the user to create, modify and review the terrain, soldier

attribute, and initial event datafiles. STARTUP is a single menu driven program that provides sufficient information to modify the example scenario or create files to support new scenarios. This program also includes a help function that explains the requirements for each datafile and includes instructions on how to set up and operate the SPARTAN model. The program code for the STARTUP preprocessor is included in APPENDIX B.

Within each phase of the model's operation, instructional help screens are available to give the student a better understanding of the overall process. When the model begins operation, it queries the user whether he desires to observe the datafiles being loaded into memory. The intent is to show the format and composition of the required data files. The set up screen follows and gives the user a variety of possible termination conditions to choose among. The student has the option of terminating after a certain number of events, after a specified time, or at some level of attrition for either side.

During the operation of SPARTAN, several windows can be created that suspend execution and allow the student to evaluate the status of the run. The soldier attribute window provides a view of a limited number of the attributes on each soldier, so the current status of each soldier can be assessed. Another window displays an overall battle status to include values such as the number of soldiers on

each side that are dead, alive or wounded as well as the number of ammunition rounds remaining. A target list window displays the detected targets on each soldier's target list and the associated probabilities of detection. This gives the student a better basis for understanding the target selection and engagement process. The last window provides a list of the next twelve events on the future event calendar. This window is set up to display the linked list logic of the scheduling routine along with a short description of each listed event. The last feature of note is the graphic display of the detection process. When a soldier has successfully detected an enemy soldier then a blue line is momentarily drawn between the positions to indicate detection. This was initially just a debug feature, but was left in to give the student a better feel for what the soldiers can see.

The output option at the end of the simulation run provides the student with a variety of output types and different formats. It displays some of the basic functions of a postprocessor by providing summarized values, final attribute values, and a history file. Several formats for providing output are represented by screen displays, output files or even sending the output to a printer [by using printscreen]. One screen display shows several summary data elements that are used as typical measures of effectiveness in analytic models. Another screen provides the final

attribute values for all the soldiers. Additionally, the model records in a file all the events as they occur in chronological order.

3.6.6 Human Factors Considerations.

Several elements of the simulation model were evaluated and modified to make it easier to use and more functional as an educational tool. These were primarily improving the screen displays, minimizing key strokes and eliminating potential mishaps or student errors.

In a text on human factors, McCormick specifies two objectives to be met when creating visual displays: "the display must be able to be seen clearly and the design should help the viewer to correctly perceive the meaning of the display (30:85). In SPARTAN, the screen displays were designed to provide a sharp contrast between the text and the background, so the text is very legible and readible. Additionally, all the data displays were structured to provide a simple, understandable format, and kept as uncluttered as possible.

The graphical display of the battle was also modified. Colors were chosen that are representative of the object or can quickly be associated with the object. Simple examples include blue for blue soldiers, and a red icon for red soldiers. To enhance the student's perception of the events, certain events have special effects that prompt the user to notice them. A rifle shot has a distinct noise and

a red flash. The wounding of a soldier is indicated by a momentary red burst and a distinct noise while the death of a soldier has a more pronounced yellow burst and the icon changes to a dull gray color.

To simplify the operation of the model, key strokes were eliminated wherever possible and specific choices were provided to the user. Where the user is required to provide input to the model, specific instructions are provided, so the user knows exactly what keys to hit or values to input. Specific choices are delineated and examples or default values are provided. The menus are set up to only accept the correct values and to continue to prompt the user when incorrect values are given. The simulation has default value provided, so it will run correctly, even if nothing is provided by the user.

3.7 Model Assessment

Assessment of the SPARTAN combat model began with a critical look at the original project objectives and continued throughout model development. The evaluation criteria established by the 1979 GAO report were used as guidelines for assessing the modeling effort (10). SPARTAN presented some unique assessment issues since it was a demonstrator rather than a true "analytic" simulation model. As a result, some of the GAO criteria received more emphasis than others depending on their degree of applicability.

3.7.1 Assessment Process.

Model assessment occurred at three distinct levels. Initially, each aspect of the model was evaluated by the author, following this faculty advisors evaluated the model and provided feedback on necessary improvements and further guidance on meeting the project objectives. The final assessment was a series of laboratory tests performed by personnel with backgrounds similar to those of the projected student audience. This last phase used "blind testing" as discussed in James Dunnigan's book, The Complete Wargames Handbook for the primary model assessment technique (19:2-37). Blind testing involved issuing prototype software and user's manuals to a test audience with no additional instructions. The test audience was told to read the user's manual and attempt to operate SPARTAN. They were asked to identify any features of the model that were difficult to understand or distracted from the learning objectives. Additionally, they were asked to provide a subjective evaluation of whether SPARTAN could provide a significant improvement of a student's understanding of the modeling process over that presently received in the course without any available modeling demonstrators. The blind tests were performed in two phases with two students in the first group and three students in the second group. Improvement recommendations were incorporated into the model after each

phase. The following sections discuss the GAO criteria as it applied to SPARTAN.

3.7.2 Documentation.

The documentation provided with the model had two particular audiences. The first was the student who is expected to operate the model and hopefully become familiar with the fundamentals of land combat modeling. The second audience was the person who intends to learn the details of the model's operation and who may desire to modify the model program code.

The user's manual and the online help screens were provided for the student. These two sources of information were intended to provide the user with sufficient information to operate the SPARTAN system and to understand the general modeling concepts used in the simulation. The comments from the test audience generally focused on ways to improve descriptions of the model, and discrepancies between the model and the user's manual. Most felt that the user's manual was about the right length, but some wanted greater detail in the users manual. The trade off here was between keeping the user's manual concise and providing sufficient detail. In response, greater detail was provided in the online help screen. This made the information available to the interested student, but kept the user's manual as concise as possible.

The thesis document was intended to provide a comprehensive discussion of the modeling process, underlying assumptions, limitations, and the specific techniques used to model the combat processes. Chapters III and IV provide detailed discussions of the logic and techniques used in SPARTAN. The appendices provide internally documented copies of all the programming code used in SPARTAN. The internal documentation was written as the code was developed. Comments were included in the code to explain the structuring of the code, how variables are used and specific implementation issues of the QuickBASIC code. This thesis should provide sufficient information to enable someone to modify the programming code after a couple weeks of learning the programming language and the details of the simulation model.

3.7.3 Validation.

The GAO report discussed assessing three areas of model validity. These were data validity, theoretical validity, and operational validity (10:5). From the project's beginnings, there was never any intention of creating a "valid" representation of a realistic battlefield, rather the objective was to demonstrate present day techniques used in analytic models for modeling the combat processes. The data for SPARTAN was fabricated, so there would be no misconceptions about the level of realism represented by SPARTAN. There is a measure of theoretical validity in SPARTAN be-

cause most of the algorithms are adapted from those that have been previously tested and validated for use in US Army models. However even with proven algorithms, there is no real validity when the data and parameters are not empirically produced. Operational validity also does not apply to SPARTAN because it is contingent on theoretical and data validity. SPARTAN does not model or attempt to model a "real world" situation. The real question of validity is whether SPARTAN meets its original objectives of providing a useful tool for demonstrating land combat modeling techniques. The results of the blind testing were that SPARTAN would be a beneficial addition to the reading and lecture material presently used to teach high resolution land combat modeling. The personnel tested felt, in particular, that it provided good insight into the search process and the various component parts of a combat model.

3.7.4 Verification.

Verification of the code required ensuring that each of the algorithms performed as intended, and that each algorithm functioned properly in concert with the rest of the model under all possible conditions. Numerous simulation runs were performed with a wide range of parameter values and variations to the scenario data. The goal was to find any problems and correct them before adding further complexity to the model. This was the point in the process where many of the limits on parameter values were established.

Most of the code problems were identified and corrected as each subprogram was added to the base model. There were a few occasions when model inconsistencies were revealed in the blind testing that required extensive searching. Several of these were due to inadequate definition of variable types, but were subsequently corrected. The QuickBASIC interpreter made the task of correcting programming errors much less onerous than it would have been with FORTRAN or many other languages. The interpreter runs the code in uncompiled form, so after each correction it was a simple matter to rerun the program. Even after an extensive number of runs, errors probably still exist in the program code, but all errors identified by the testing program have been corrected.

3.7.5 Maintainability.

One of the original goals of SPARTAN was to provide a structured program code that would be easy to understand and enhance as desired. The modular design of the program and the extensive internal documentation should allow a programmer to understand the operation of each routine, and how it relates to the other processes occurring within SPARTAN. The variable names were chosen to be descriptive of their use, so they would aid in understanding the code.

The issues of reviewing and updating the model are left to the eventual user, but it is hoped that a policy is

established that ensures problems encountered by students are addressed and corrected when possible.

3.7.6 Portability.

SPARTAN was designed to run on most IBM compatible personal computers. It runs on any IBM compatible personal computers at the Air Force Institute of Technology and should operate on most student's personal computers. The minimum requirement of 512k random access memory (RAM) is a requirement of compiled QuickBASIC version 4.5. The EGA color monitor requirement was necessary to achieve sufficient clarity in the graphics. The code has been tested successfully at various speeds between 12MHZ to 25MHZ. It operates much more rapidly from a hard disk drive, but will operate from any disk drive with 360k or more capacity.

3.7.7 Useability.

After refinements from the first cycle of blind testing, the new users encountered no great difficulty in understanding the instructions and were able to operate the model successfully. In its simplest form, the model with the example scenario can be operated by a novice modeler in less than an hour of time. No significant computer skills are required other than to access the correct directory, and type in SPARTAN at the prompt. This satisfies one of the major objectives of the project. SPARTAN is a simple instructional aid that can be a benefit to all students.

3.8 Conclusion

The intent of Chapter III has been to provide an overview of the development process used to create the SPARTAN model. Using the conical methodology, the following components of the model were developed using specifications derived from the original problem statement guidance of the sponsor.

1) input data files

2) STARTUP preprocessor program

3) SPARTAN combat model with subprograms for:

- initialization
- terrain representation
- individual soldier movement
- line of sight determination
- target acquisition and detection
- target selection
- target engagement
- damage assessment
- reaction to fire
- limited command and control
- online help facilities

4) model documentation

5) user's manual

6) student study guides

Chapter IV will provide a more detailed discussion of the various combat processes modeled in SPARTAN.

IV. Combat Processes

4.1 Introduction

This chapter discusses the techniques used in SPARTAN to simulate the different combat processes. In most cases, these ten techniques are representative of the common methods used by Army models. In each case, the origins of the modeling process, the author's rationale, and the implementation methods will be discussed. All the processes addressed can logically be grouped into categories of movement activities, engagement activities, and the thought processes of decision logic. This chapter organizes the discussion of the processes into these three categories to facilitate a coherent discussion of the process flow.

4.2 Movement Processes

The movement process in SPARTAN uses soldier attributes and terrain attributes to position soldier entities on the represented battlefield and move soldier's to new locations according to the scenario guidelines and the soldier's status.

4.2.1 Modeling Movement in Army Models.

Army models commonly use a movement routine that represents the continuous movement of entities with a series of fixed size movement steps. The models actually represent the entities presence only at these discrete step locations. The variability in movement rates is reflected in the time required to travel the fixed distance. This type of movement representation minimizes the processing time required for movement while providing a sufficient level of detail. A fixed time

step method might require several iterations to move an entity over the same distance. The movement algorithms in SPARTAN are similar to, but greatly simplified versions of those found in JANUS(14:411). The actual movement algorithms were extracted from DARCOM-P 706-101 (20:40-15).

$$X = X+20*COS(DIR) \quad (5)$$

$$Y = Y+20*SIN(DIR) \quad (6)$$

These two equations use the trigonometric functions to add an incremental value to each coordinate based on a fixed movement distance and a direction of movement for each soldier. These simple equations are applicable in all possible directions [0.0 - 6.28 radians].

SPARTAN moves an entity 20 meters for each move, regardless of the terrain. This is a departure from the JANUS model, which tries to move entities in 50 meter increments, and can modify the distance when obstacles are encountered. SPARTAN operates on a much smaller scale and currently, does not model obstacle circumvention. Unlike JANUS, each soldier in SPARTAN has an original direction of movement rather than a route of movement with varying directions. Changes in direction occur only as a result of a soldier's tactical decisions. In this aspect, SPARTAN is employing a method similar to CASTFOREM which uses movement routes, but allows tactical decisions to affect the choice of these routes.

As stated earlier, movement time is the factor that varies with conditions [See Equation 7]. Each soldier has attributes for movement speed and posture while his location has an associated trafficability index. All of these factors are included in the movement time algorithm. As an example, a soldier with a high movement speed will have

more moves of shorter time duration than another soldier with a lower movement speed. As a result, the quicker soldier will move a further distance in the same allotted time.

4.2.2 The SPARTAN Movement Process.

Movement is performed by two subprograms named startmove and endmove. These routines move each entity along a single direction of movement at a rate of 20 meters per move whenever the soldier's movement attribute switch is on and his speed is greater than zero.

When startmove is called, it begins by performing several status checks to determine whether the soldier should move. The routine checks his posture, his present movement rate, the terrain trafficability, and his proximity to enemy forces. Several artificial rules are imposed on the movement process. These are that a soldier stop moving when within 100m of a living enemy soldier and when he crosses the terrain maneuver boundaries [edges of the view screen]. These rules eliminate the issue of close quarters combat and allow the terrain datafile to be limited in size to the viewing screen area. When a soldier does move, equations [5] and [6] are used to compute the soldier's new coordinates. His present location is stored as the previous location and both sets are updated in the soldier's attribute array.

Startmove calls the procedure for updating graphics. This subprogram erases the soldier's symbol at the old location and redraws his symbol at the new location. The rapid speed of the drawing provides an animated effect of the entity moving across the screen.

The time duration of the move is a uniform random factor modified for the soldier's speed, present grid trafficability factor, and

posture. This movement time is added to the present event time and an endmove event is scheduled at that later time. The random variable represents subtle variations in the soldier's movement rate and terrain conditions or perhaps to model minor direction changes of the soldier moving along a tactical route that improves his level of cover and concealment.

$$MOVETIME = TIME + \frac{RND * SCALEFACTOR}{SPEED * MOBILITYFACTOR * POSTURE} \qquad (7)$$

The primary function of the endmove event is to schedule the next startmove event. The time duration between moves is a random value from a triangular distribution with a mode of ten seconds.

The tactical situation affects a soldier's movement by altering his attribute array. A soldier reacting to being fired upon may alter his speed, change posture, change direction or just stop. Similarly, if a designated squad leader engages an enemy soldier then he may cause the other squad members to change their direction and move towards the enemy he selected to fire upon. In the event that the squad leader or his target are killed, the squad leader's soldiers return to their original direction of movement.

The low level of detail in this movement simulation results in the following list of limitations and or assumptions inherent in the process.

1) The entities in effect are in an iterative process of moving and stopping rather than continuous movement.

2) There might be situations where the soldier might not want to move the entire twenty meters in one bound, but the algorithms have no options.

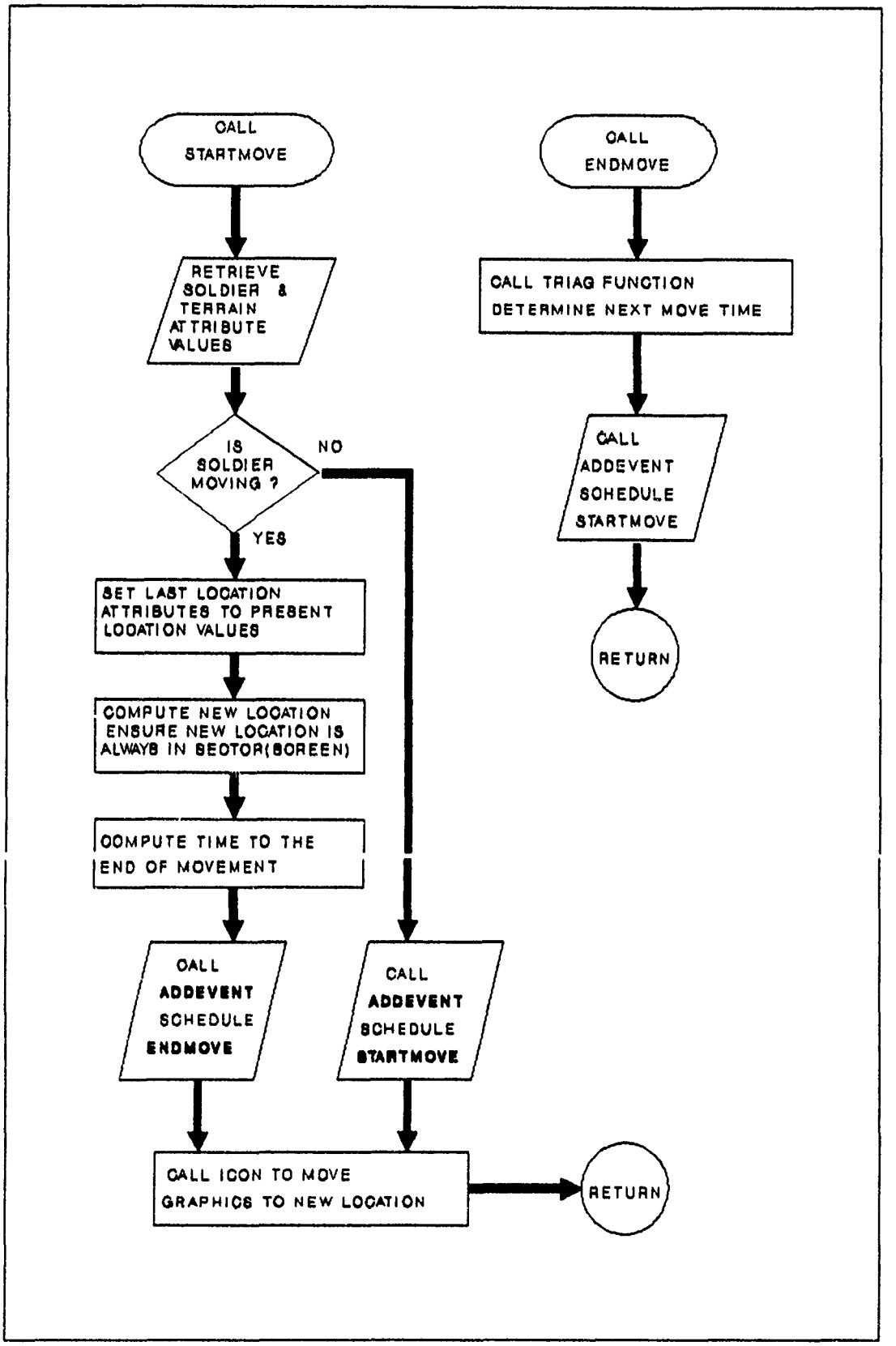

Figure 5 Movement Process

3) The slope of the terrain does not affect the rate of movement.

4) SPARTAN assumes the trafficability of the grid where the soldier starts is not significantly different anywhere along his 20m path.

5) After an endmove and before the next startmove, the soldier's moving status does not change to indicate a halt.

6) Fatigue is not a factor in this movement.

7) Movement computations are scaled for the screen graphics rather than being a good model of reality.

8) Posture changes are instantaneous and do not affect a move in progress.

4.3 Engagement Processes Modeled in SPARTAN

The target engagement process includes a sequential list of activities from target acquisition through target destruction. This section will discuss the various processes of searching for targets, selecting targets, and engaging the targets.

Before getting into the details of each process, it is useful to see how the processes relate in the activities of a typical SPARTAN soldier. For all soldiers, the first step is to search for targets. Every 10-20 seconds, the computer scans the entire area around each soldier. To perform this search, the computer must determine if line of sight exists between the scanning soldier and all enemy soldiers. If line of sight exists then the routine computes whether each enemy with line of sight is providing enough of a signature for the observer to acquire. The computer then checks for a possible detection for each of those enemy targets that could be acquired. Any target that can be detected is placed in the observer's target list. If the observer has successfully detected one or more targets, then a selection algorithm

models the process of deciding whether to shoot at any of the enemy and then which particular one to engage. If a target was selected, the soldier then takes the time to aim and fire at the enemy. If the enemy soldier is struck and killed, the firer returns to his searching activities, otherwise, he reengages the enemy as long as he can see the enemy. Throughout this process, any failure to advance to the next stage of the process returns the soldier to his initial search activities. With this in mind, it will be easier to understand each of the subordinate processes and the soldier's decision process.

4.3.1 SPARTAN Search Process.

SPARTAN uses a simplified version of the continuous search algorithm discussed in Hartman (23:4--32). This is the same basic search process used by CASTFOREM and JANUS. This type of search model requires that three factors must be addressed before a target is successfully detected. These are: the target gives off a signature that can be acquired by the observer's detection device, it is physically possible to see from the observer to the target, and the observer must look in the direction of the target long enough to pick up the target's signature.

At this point, it is necessary to point out a significant difference between the JANUS, CASTFOREM and SPARTAN models. CASTFOREM determines a search sector for each sensor as a function of time (13: 3-79), JANUS and SPARTAN do not. JANUS has a set of rules governing its search sector. On the move, each entity searches 360 degrees, but when stationary JANUS uses a fixed 180 degree arc centered on the direction of travel or it uses the sensor field of view when the observer is in

defilade (14:365). SPARTAN does not determine a specific sector of observation for each soldier. It is assumed that in each search cycle, the soldier performs a scan in all directions. The primary reason for eliminating sectors was to avoid the associated computational overhead. In effect, the model assumes that the soldiers are prudent and performing a comprehensive search each search cycle. What this eliminates is the possibility of focusing on likely enemy locations, defining sectors of responsibility among groups of soldiers or modeling the tendency of many soldiers to look only in their direction of travel.

4.3.1.1 Line of Sight [LOS].

The first task performed by the search module is to check for line of sight between the observer soldier and all the enemy. Performing this check first eliminates some unnecessary calculations that would be performed if the target's acquisition potential were checked initially.

LOS is a separate function within the combat module. The observer's identity is passed to the function and for each living enemy soldier, it returns either a 1 to indicate that LOS exists or a 0 to indicate that terrain obstructs the view between the observer and one of the enemy.

The method used to check LOS is depicted in Figure 6. The height of the observer's visual sensor [his eyes] are computed as his elevation plus his height attribute. The enemy's height is the enemy's elevation plus his height. The function then computes the distance between the two soldiers and determines the number of checks required to sample the intervening elevations at 10m increments. 10m was chosen to ensure each 20m grid cell is sampled. The function then uses a simple difference

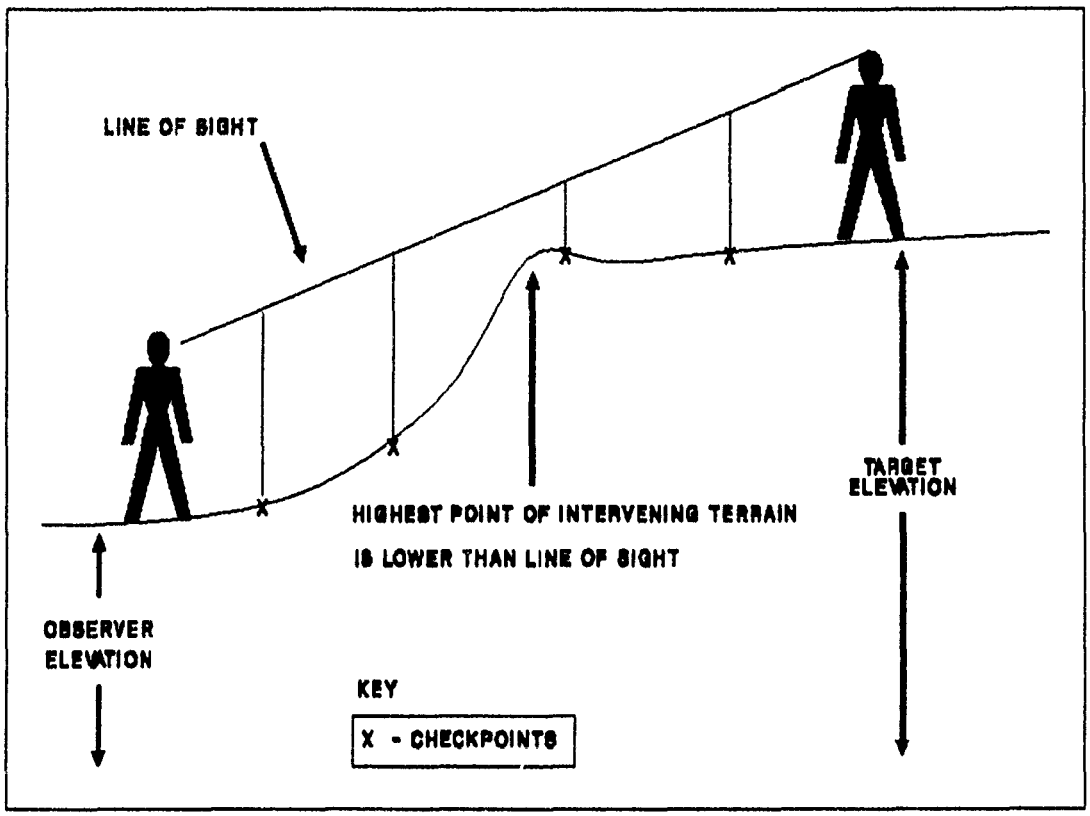

Figure 6 Line of Sight Check

equation to determine the LOS elevation at each check point. If the LOS elevation is less than or equal to the grid elevation then the LOS function returns an indication that LOS does not exist. This function does not account for any partial obscuration that might occur due to vegetation or buildings.

Atmospheric attenuation is accounted for by using a simple linear modifier. This modifier assumes a uniform background weather effect within the model (23:3-20). The modifier increases the minimum threshold level required for target acquisition [See Section 4.3.1.2]. This attenuation effect might be the result of fog, rain, dust or darkness that hinders the transmission of the target signature.

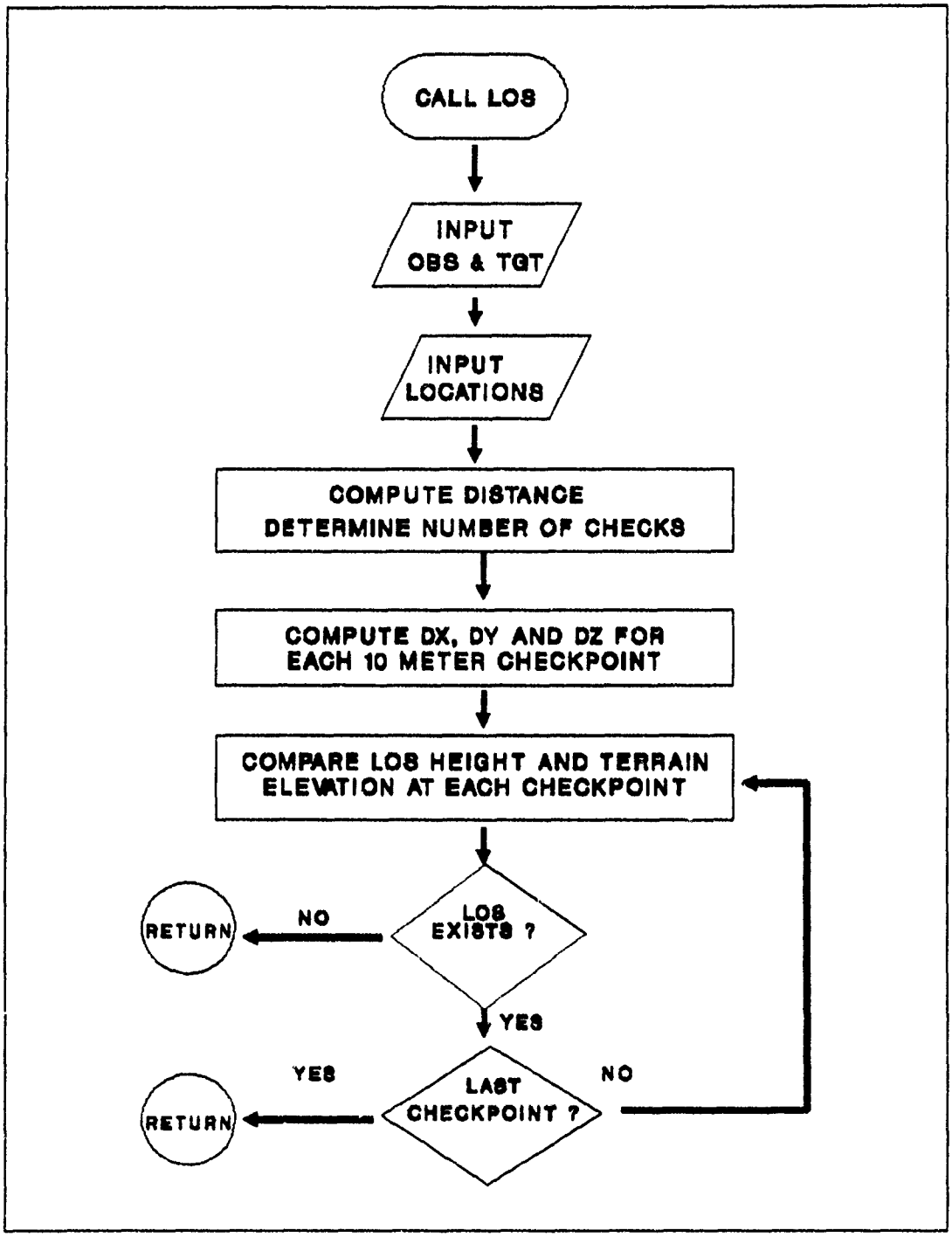

Figure 7 LOS Process

4.3.1.2 Target Acquisition.

The process of target acquisition determines whether a target gives off a sufficient signature for the observer to detect. The

equation used to model acquisition is a simplified version of equation [1]. This equation is discussed in Bailey's analysis of the JANUS detection algorithms (1:5). See Appendix A for the MATHCAD template used to create the acquisition tables.

$$p1 = 1 - \exp\left(-.84 * \left(\frac{C}{M}\right)^{2.4}\right) \qquad (8)$$

M is a constant value of 3.5. This scaling factor accounts for the probability required to yield a target identification.

C is the number of resolvable cycles for the target which is a factor of target height divided by distance of the target.

.84 is a scaling factor added to Bailey's algorithm to get appropriate values since no empirical data was used on the sensory capabilities of human eyesight.

Using equation [8], a table of acquisition values was created. This table is indexed by target posture and target range. After computing target range and looking up the target posture, the search subprogram selects an acquisition probability and compares it to a threshold value. A value exceeding the threshold indicates that the target can be acquired and possibly detected. The minimum acquisition threshold for SPARTAN was set arbitrarily using the author's best judgement at .3 for conditions of no attenuation. This is the point where the atmospheric attenuation coefficient affects the search process. If the user sets the attenuation coefficient lower than 1.0, it raises the threshold value for acquisition by a proportional amount.

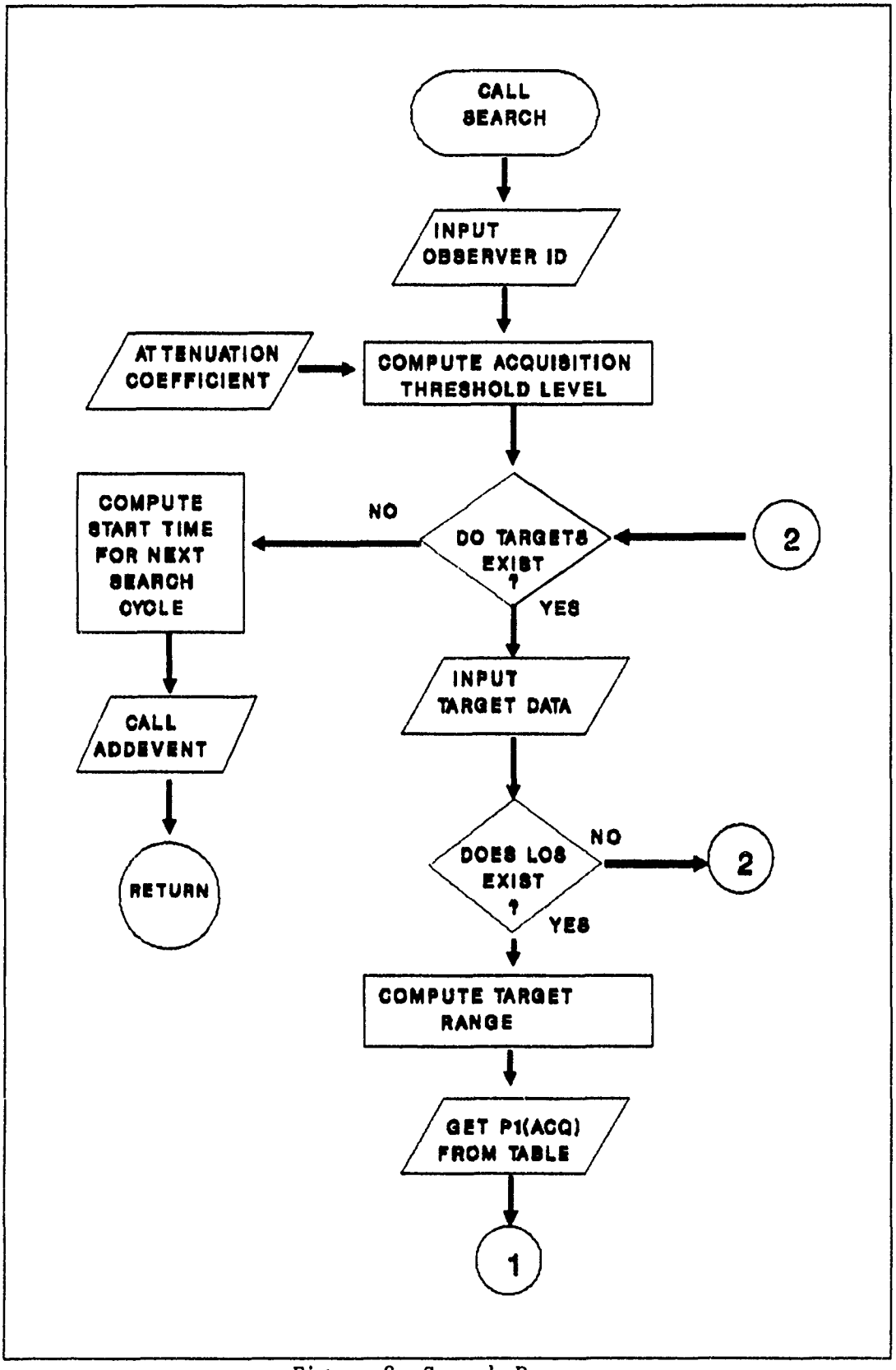

Figure 8 Search Process

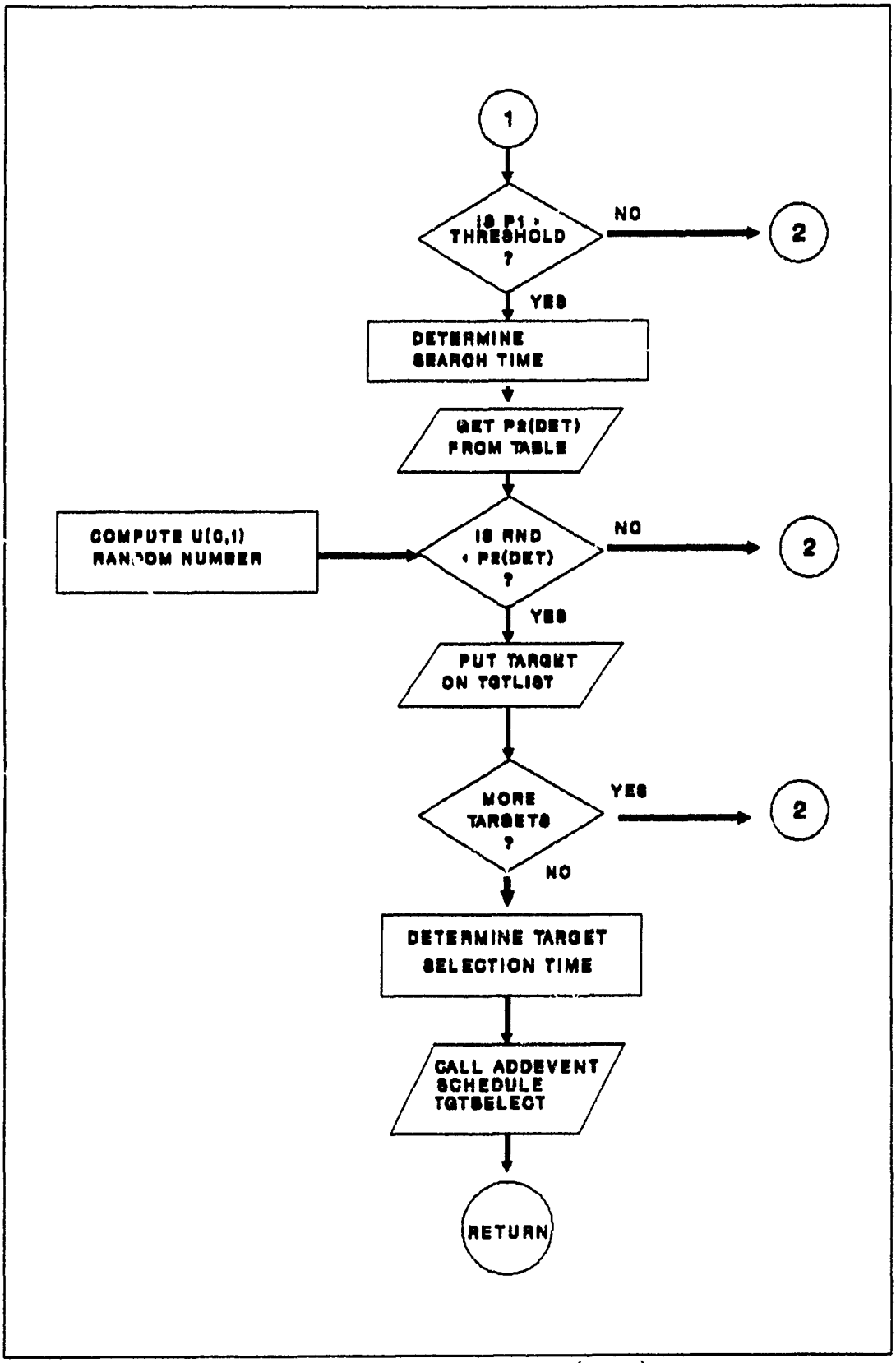

Figure 9 Search Process (cont)

4.3.1.3 Target Detection.

If a target can be acquired then the possibility of detection is checked by the search subprogram. For each acquireable target, a random search time is computed using a triangular distribution with a mode of 2.0 and a range of [.4 - 4.0]. This value is assumed to be the time [in seconds] that the observer scans the area containing the potential target. With this time and the target range, the search routine draws a U[0,1] random variate and compares it to a P_i value drawn from the detection probability tables. If the random variate is less than or equal to the P_i value then a detection occurs, the targets identification and P_i values are stored in the observer's target list and the observer is scheduled to perform a target selection.

The detection probability tables were created using the NVEOL equation for P_i as discussed in Chapter II. These tables are indexed by target range and search time.

$$P_2 = 1 - \exp\left(-\left(\frac{C}{M}\right) * \left(\frac{t}{6.8}\right)\right) \qquad (9)$$

C and M are the same variables discussed with equation [8]. This table does not differentiate between any target postures.

t is the random search time in seconds with a range of [0.4 - 4.0].

6.8 is an empirical value from the original NVEOL equation.

The JANUS and CASTFOREM versions of this algorithm are much more comprehensive. In addition to range and search time, they include: atmospheric attenuation, contrast between the target and its background,

sky brightness, movement of the observer, target movement and whether the target is producing a firing signature.

4.3.2 Target Selection.

The target selection subprogram is scheduled at the completion of a soldier's cycle if one or more targets was detected during the search. The two purposes of the target selection routine are to determine whether the conditions are suitable for the soldier to fire at a detected enemy soldier and to determine which of multiple targets should be engaged. The process used by SPARTAN is similar to the decision logic modeled in JANUS.

The selection routine begins by comparing the target distance to the effective range of the observer's weapon. At this point, the model also accounts for any observer's range estimation error by allowing up to 100m of variation in the computed range estimate. This estimate is compared to the observer's weapon range. If the target is in range, then the probabilities for all targets on the observer's target list are summed and compared to a threshold of .20. If the summed value is less than .20, the soldier is assumed to have decided that the target cannot be effectively engaged and the observer goes back to searching. If the soldier's decision is to engage, the detection probability values in the target list are normalized so they sum to 1. A U[0,1] random variate is drawn and compared to this range of detection probabilities. The targets with the larger probabilities are most likely to be picked. The result of this method is that target with a high probability of being hit will usually be chosen. JANUS uses a similar method, except it determines single shot probabilities of kill [SSPK] for each target on

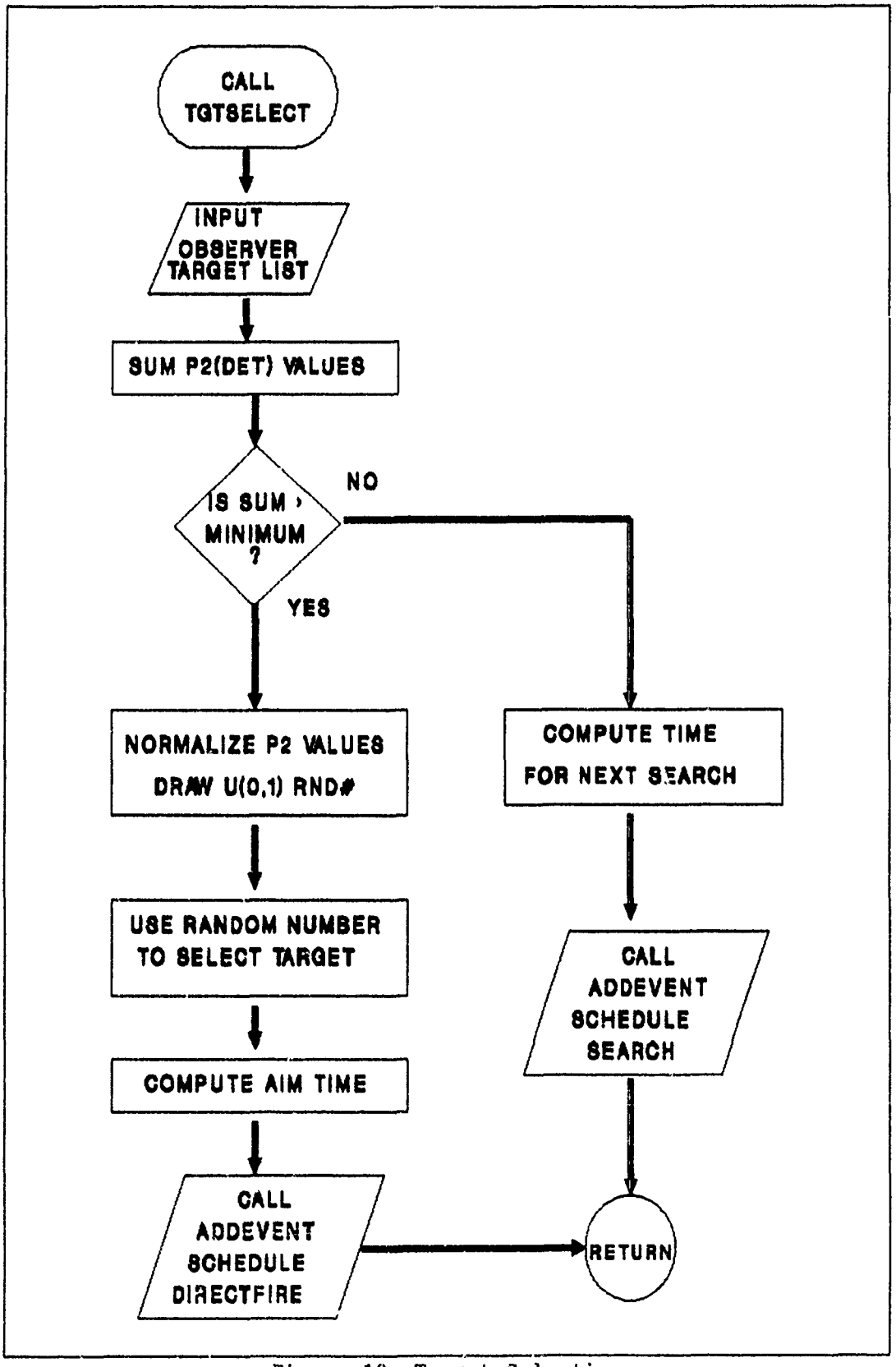

Figure 10 Target Selection

the list and uses this parameter to select a target. A comparison of SSPKs and detection probabilities shows a strong correlation since both are highly correlated with target range. With this in mind, the author decided to use detection probabilities as a surrogate for the SSPK, so that additional computations would not be required.

If a target is selected, its identifier is placed in the observer's attribute as the target to be engaged, and a direct fire event is scheduled. The target selection occurs as an instantaneous event, but a five second delay between target detection and selection is intended to account for the decision time.

4.3.3 Target Engagement.

Soldier's in SPARTAN are limited to semi-automatic direct fire weapons. The model incorporates probability of hit tables for single shot weapons with effective ranges of 300, 400, 500, and 600 meters. The weapon range is one of the soldier's attributes. The time of flight for each weapon is the same, but the probabilities of hit vary significantly.

Before the target selection routine calls a direct fire engagement, it computes a delay time that accounts for loading and aiming the weapon. The direct fire subprogram starts by checking LOS to ensure the target has not moved out of view during the time required to select and aim at the target. The process then ensures that ammunition is available, and if so decrements one round from the soldier's supply. The subprogram, subsequently, computes a time of flight for the bullet and calls the impact subprogram at that time in the future. The subprogram also creates an auditory and visual effect to represent the bullet's

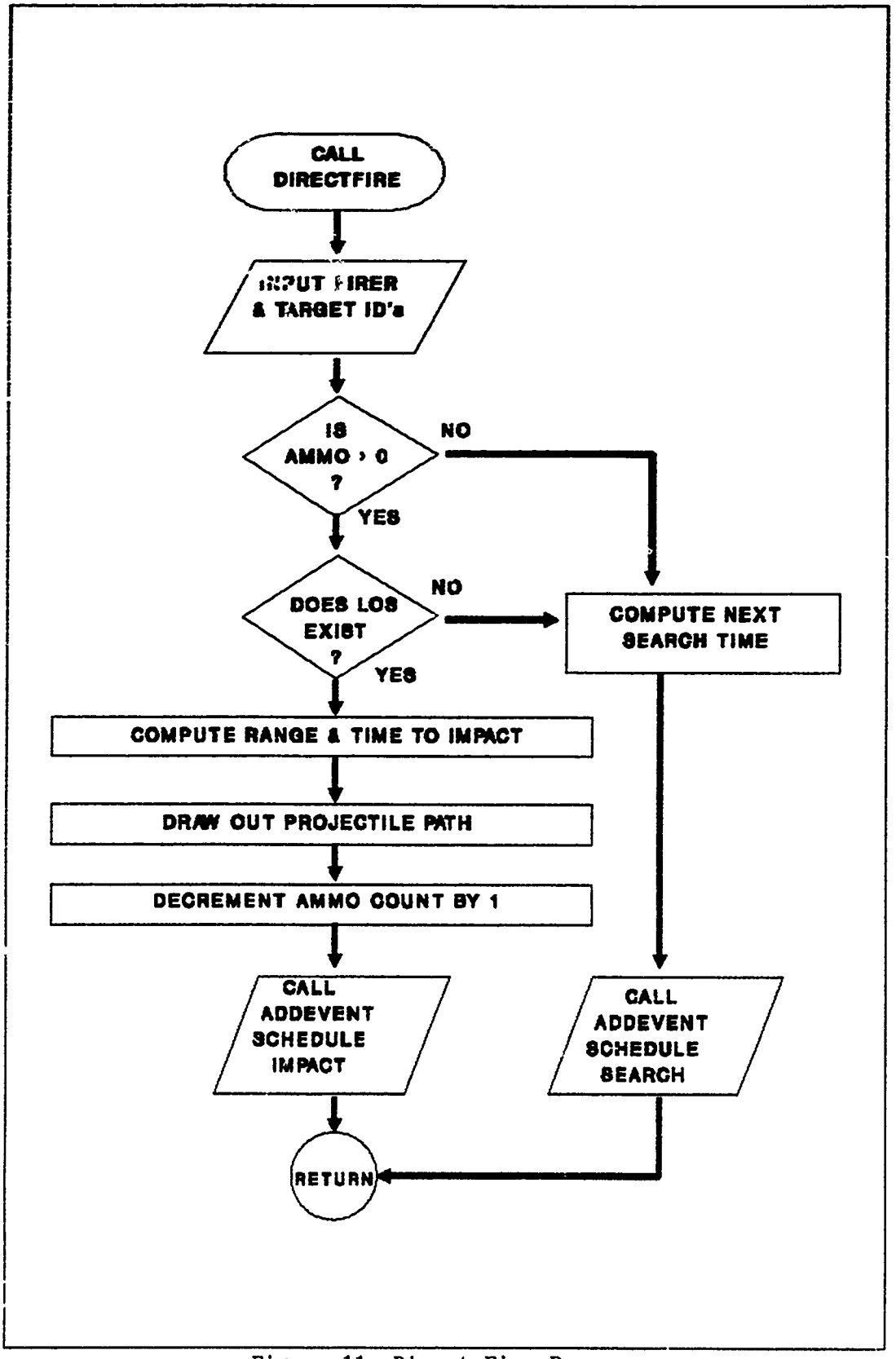

Figure 11 Direct Fire Process

flight. Initially, the impact subprogram determines whether the bullet hits the target soldier. The probability of hit is taken from a table stored in main memory and indexed by weapon range, engagement range and target posture. The probabilities were originally computed using a bivariate normal approximation equation provided by Hartman (23:7-17). This equation assumes no bias and a circular error distribution. The basic equation is:

$$P_{hit} = 1 - \exp\left(-\frac{R^2}{2 * \sigma^2}\right) \tag{10}$$

$$R^2 = X^2 + Y^2 \tag{11}$$

The numerical values required to apply these equations were arbitrarily selected to create hit tables with the desired range of values. Additionally, this algorithm assumes a circular target, so a fraction of the probability was taken to represent the portion of the circle occupied by the target.

The impact algorithm draws the appropriate probability from the datafiles and compares it with a U[0,1] random variate. A random value greater than the probability indicates a missed shot with the result that the firer will attempt to reengage while a "react to fire" subprogram is called for the target soldier. If the round strikes the target then there is a 30 percent chance of killing the soldier, and a 70 percent chance of only wounding him. When a soldier is killed, several things occur. An extended noise and color burst indicate his death and his symbol changes to a red color. His status attribute goes to zero, his movement goes to zero, and his posture goes to prone. Lastly, the "killsoldier" subprogram is called which removes all active

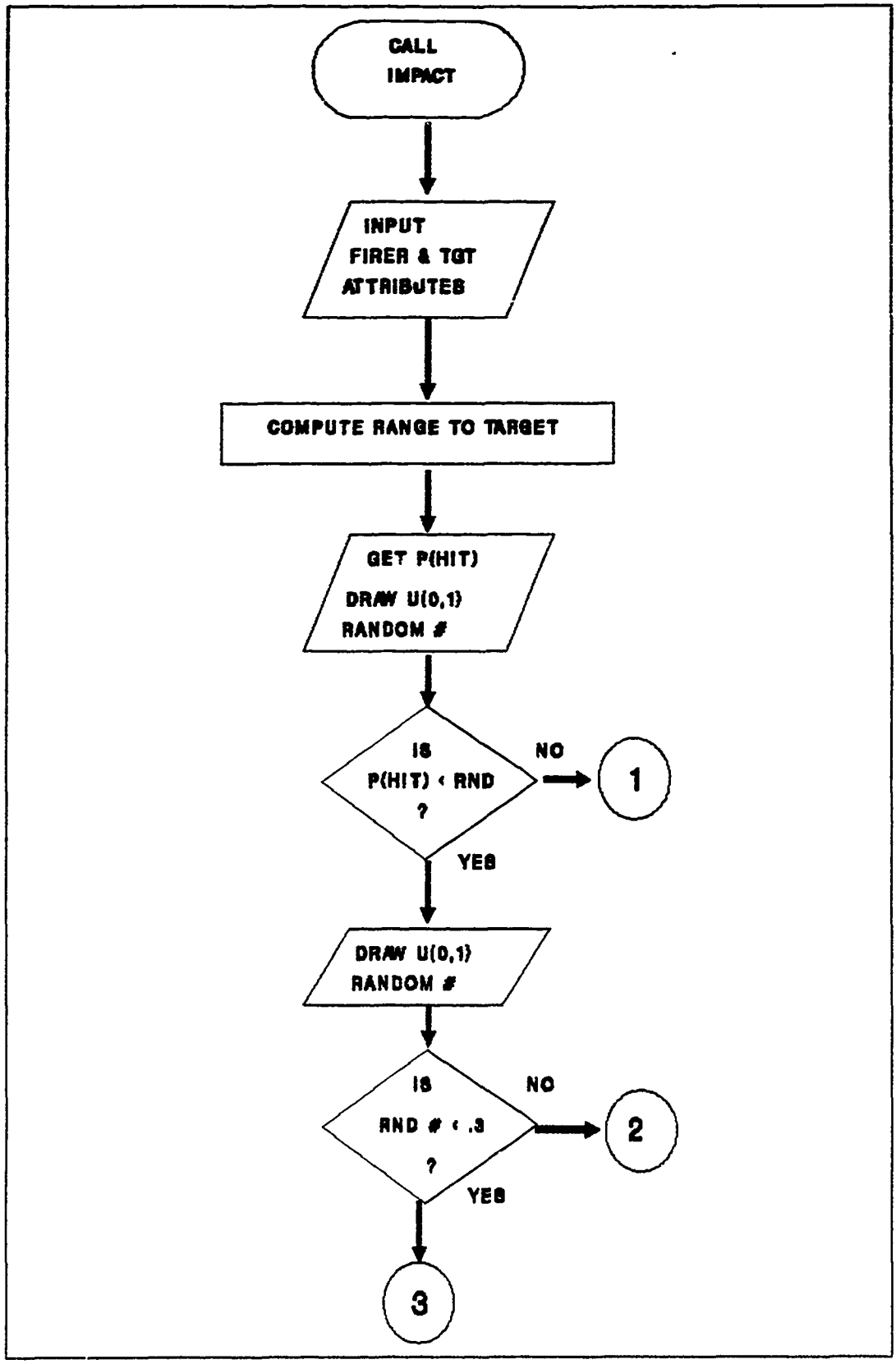

Figure 12 Bullet Impact Process

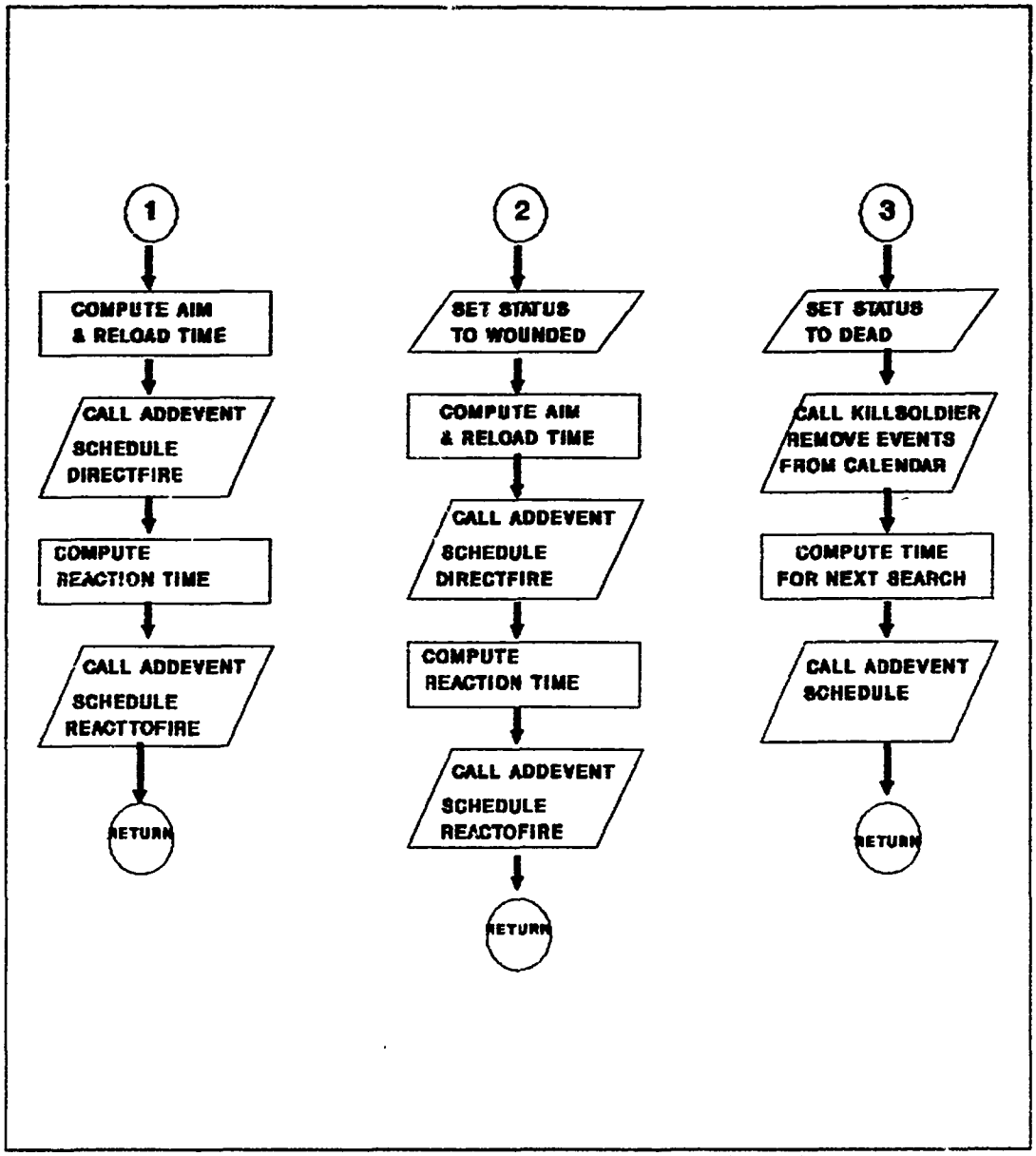

Figure 13 Bullet Impact Process (cont)

events of the dead soldier from the future event calendar. This prevents the possibility of a soldier performing an event or affecting the battle after his death.

A wounded soldier is indicated by a short color and sound burst. When a wounding occurs, the firer is scheduled to reengage, and the target entity is scheduled for a reaction to fire event. The only effect a wound has on a soldier is to reduce his movement speed. There

is no cumulative effect from multiple wounds and suppression is not modeled by SPARTAN.

4.4 Modeling Decision Logic

The point of modeling decision logic is to determine which of the available courses of action would be taken by an individual in a given situation. This requires making a judgement based on parameters that significantly effect the situation. Some decisions may be specified in certain circumstances and some may be probabilistic to reflect uncertainty. The principal tool for performing decision logic in SPARTAN is the IF - THEN - ELSE block which mimics the decision process by monitoring a specified set of conditions. These blocks are used extensively throughout the program. Virtually, every step of each process has alternatives that require some choice. The previous sections have made numerous references to these decisions. In the following paragraphs, the discussion will focus on a two processes that emphasize decision logic.

4.4.1 Reacting to Fire.

This subprogram requires a soldier to take some action when fired upon by the enemy. The method chosen to model this decision was a conditional block with three possible branches. Branches are chosen stochastically with a U[0,1] random variate. The first choice has a 40 percent chance of occurrence. This branch causes the soldier to assume a prone posture and slow down to a crawl speed. This is intended to replicate a soldier seeking cover. The next choice has a 20 percent probability and causes the soldier to reverse directions as if to back away. The last option has the soldier remain in an standing posture and

advancing toward the enemy. These alternatives were chosen, so distinct choices would occur that were discernable to the student observing the simulation run. They were selected with no concern for realism and could be changed to reflect other courses of action.

4.4.2 Command and Control.

One portion of the model was intended to represent limited command and control processes. On each side, one soldier can be designated as a squad leader. The student has the option of having squad members follow certain instructions from the squad leader.

When the squad leader is in control, all other squad members will reorient towards any target engaged by the squad leader. No one else in the squad has the ability to communicate or cause others to react. The squad leader is in effect the only "voice" in the unit. The inspiration for this logic was a squad communications system used by the US Army in the early 1980's. This system gave all the subordinates a helmet mounted receiver while the squad leader had the only transmitter. The result was very limited one way communications. In this model, messages are instantaneous and always received. The simulation logic requires that all the subordinates will resume their original orientation if the squad leader or his current target is killed. Note the simplifying assumption here that a soldier knows the status of any enemy that he is observing.

There are numerous other instances where command and control logic is inherent in the process. The initial attribute values of the soldiers have the effect of specifying a simplistic mission order for each soldier. A soldier's direction, rate of advance, and initial

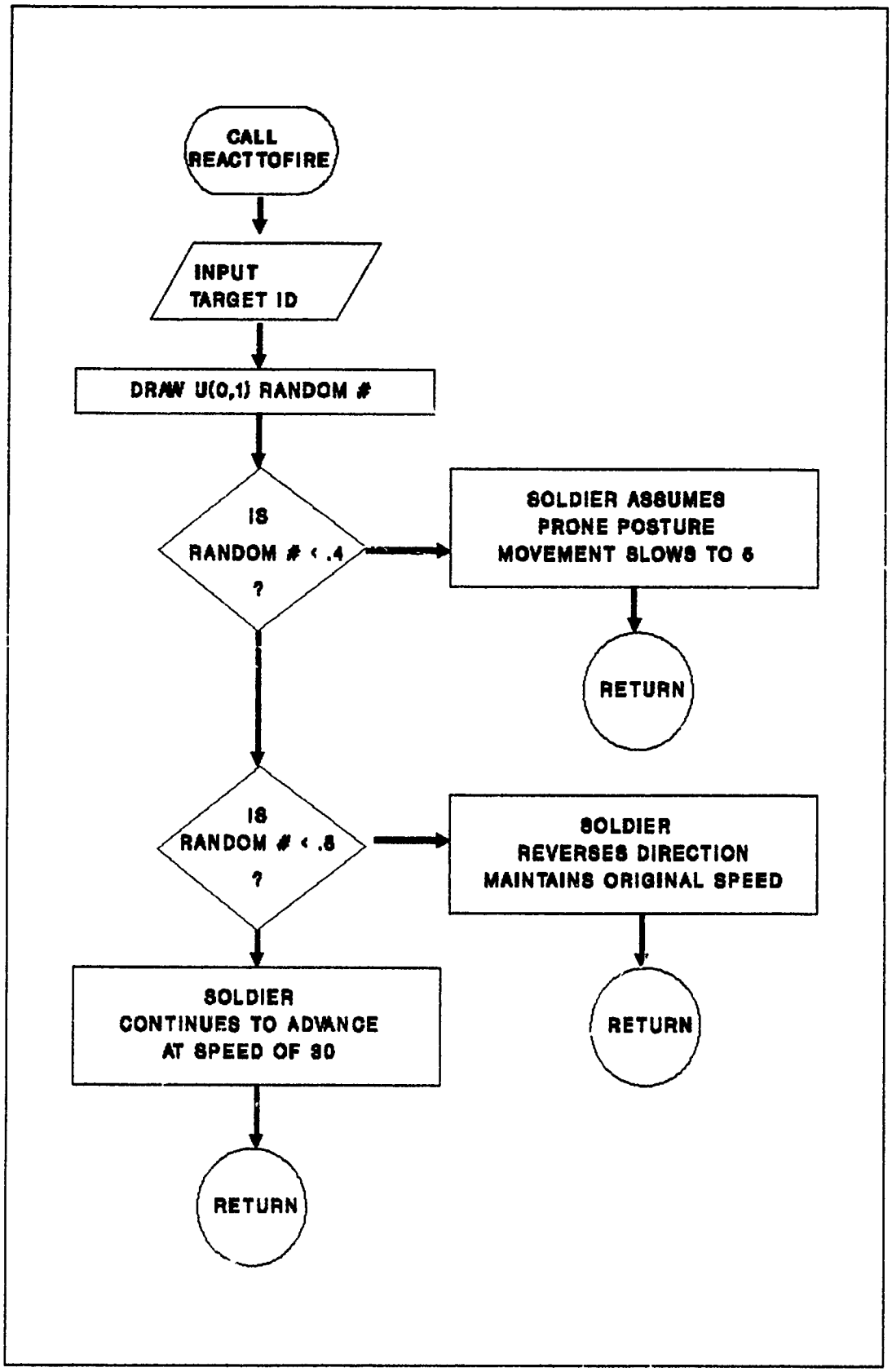

Figure 14 React to Fire Process

location can represent crude forms of tactics. The decision logic that determines when a soldier will fire at a potential target might represent a unit's standard operating procedures for fire control. Lastly, the absence of communications in the model can be representative of situations where poor communications actually exist.

4.5 Conclusion

Chapter IV has provided a discussion of the major combat processes modeled in SPARTAN. The intent was to provide a sufficient level of detail for the reader to understand how the processes operate, how they interact within the simulation, and why specific methods were chosen for use in the model.

Chapter V provides an assessment of whether the model development effort met the original study objects, and provides some recommendations for future enhancements of SPARTAN.

V. Conclusion

5.1 Summary

This thesis developed a new high resolution land combat model as an educational tool to supplement courses of instruction for future combat modelers. The goal was to develop a simulation model that illustrates some of the more important topics of combat modeling as discussed by James Hartman and other authors (1,5,7,18). Some of the important topics demonstrated are:

1) Time keeping and an implementation technique for event set management and synchronization.

2) Algorithms used to model movement, terrain, target detection, target selection, weapon accuracy, and attrition.

3) Techniques to model decision logic of the soldier as well as simple command and control issues.

4) Stochastic techniques for representing the occurrence of randomness on the battlefield.

5) Data requirements and storage techniques for various model components.

6) The overall process of developing a combat simulation model from concept to implementation.

7) An example of the components for a typical combat model such as the scenario input, a preprocessor, the simulation model, various types of output, and accompanying documentation.

Unlike many other thesis efforts that have focused on one aspect of a combat model and have gone into great detail with that particular aspect; this thesis effort used the six months of available time to perform the entire process of developing a model, with a very limited time scope.

At this point, it is important to remember that the objective was to create a model that displays modeling techniques rather than a model to perform combat analysis. The modeling techniques in SPARTAN are similar, in most cases, to those found in the current family of high resolution combat models used by the US Army. Since this model did not have an analytic objective, the issue of data validity was totally avoided. Any attempt to implement accurate data would have required an effort equal to the task of developing just the computer model.

Throughout the process, it was important to stay focused on the original objectives. At each stage, the ever present temptation was to add some extra level of detail or refinement to the model. One drawback of not having accurate data was the time required to scale model parameters, so the model output would appear reasonable. This is particularly evident in the representation of time. Time goes by much more quickly in the simulation than it could in reality. This is not to say that the specific event times are unrealistic. The apparent shortcoming is that the model only represents a limited number of the many activities that a soldier would be constantly performing as he traversed a battlefield. Thus the model does not account adequately for the time taken by these other activities.

One aspect of this modeling project that differs from most was the level of transparency required. Normally, a modeler strives to minimize the mechanical nature of the model. SPARTAN, on the other hand, was intended to demonstrate these mechanisms, so there was always a decision on which was more important. One example of this was leaving in the debug feature that displays a target detection. This should help the

student understand the sequential search process that preceeds a target engagement.

The single biggest hurdle to overcome in developing SPARTAN was to implement a viable event set management algorithm. After several attempts with methods that proved slow and laborious such as sorting sequential lists, MAJ Morlan's design for a doubly linked list became the control logic for SPARTAN. Another significant task was maintaining the documentation that goes with the model. The documentation for SPARTAN includes detailed comment lines in the code, the users manual, the online instruction screens, and the thesis document. Every modification required updating all of these documents.

Some of the other design objectives that guided development were ease of use, simplicity, and portability. The model meets these requirements. Any student can read the users manual and operate SPARTAN with the example scenario in less than an hour. The set up prompts provide the student with simple options that require little knowledge of modeling and leave little room for student error. Simple help screens should enable the student to learn as much about the model as one desires. The software and data files for SPARTAN can be maintained on a single 360K floppy disk that fits on most IBM compatible personal computers. This means that SPARTAN is readily accessible to virtually any student.

The QuickBASIC version 4.5 programming language proved to be a good choice as a development tool. It facilitates the use of structured programming and the interpreter greatly simplified the tasks of coding and debugging the program. The language syntax is quite understandable

and was easy to learn. Other features that proved useful were the color graphics functions and the ability to simultaneously maintain several screens of graphic images that can be called up on demand.

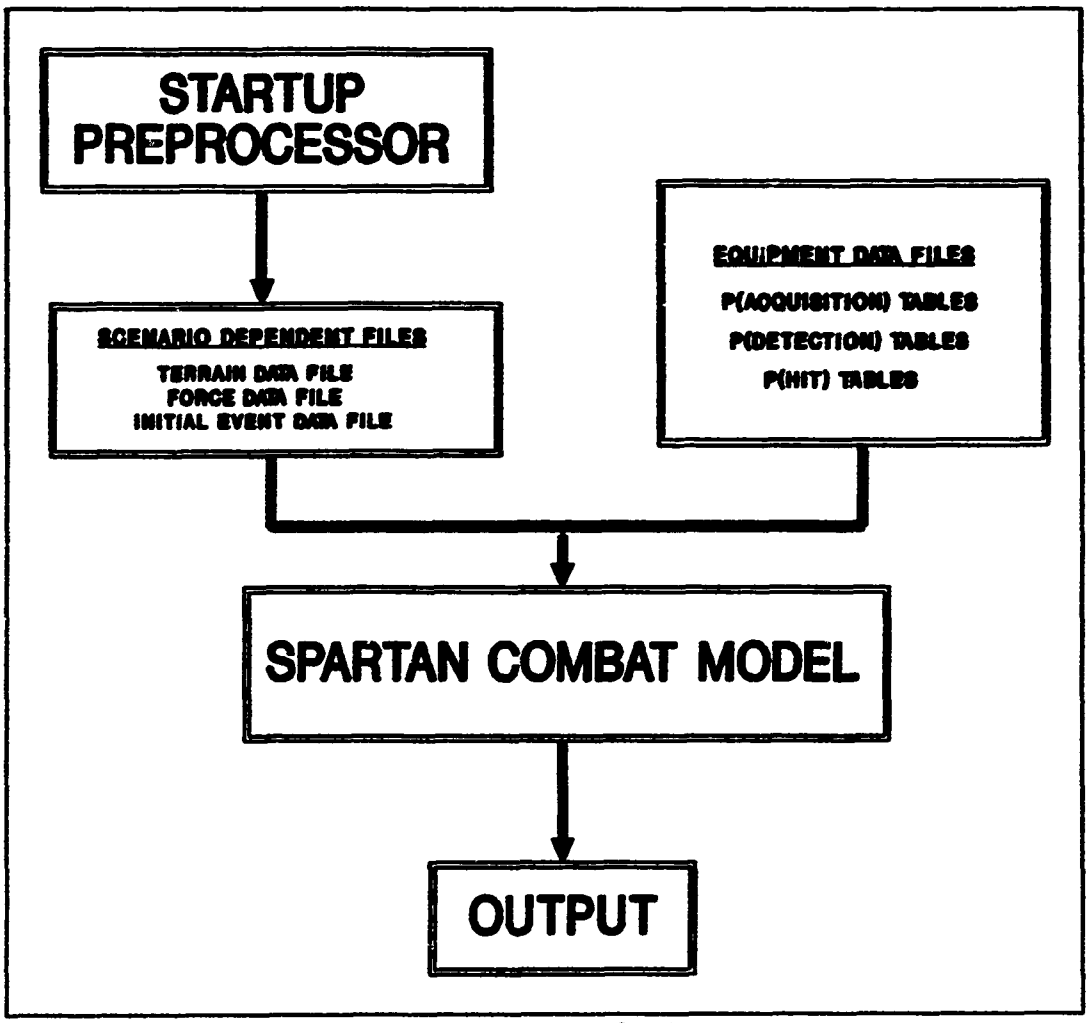

Figure 15 SPARTAN Simulation Process

5.2 Recommendations

The simplicity of SPARTAN lends itself to a myriad of possible enhancements. The possibilities include multiple types of entities, multiple weapon types, greater detail in the various processes, more refined graphics displays and a better array of available output. As William T. Morris stated in his article "On the Art of Modeling",

> one begins with very simple models, quite distinct from reality and attempts to move in an evolutionary fashion toward more elaborate models which more nearly reflect the complexity of the actual management situation. (31:B-709)

SPARTAN was intended to represent a finished product, but as just stated, a simple model can always be enriched. The structured programming approach used to develop the code facilitates a future process of "elaboration and enrichment". This section will discuss a logical progression of possible improvements that have already been identified.

1) While SPARTAN was developed with the goal of making it easy to improve, no great effort was made to determine the storage and memory limitations of QuickBASIC 4.5. It would be prudent to establish these size limitations prior to any major improvements to the program.

2) Improve the movement process to allow input of detailed movement paths for each soldier. Additionally, increase the level of detail in the attributes, so that at any point in time it can be determined whether the soldier is moving or stationary. This would allow greater detail in the acquisition and engagement processes.

3) Incorporate obstacles, and greater terrain detail such as vegetation, background clutter, and speed adjustments due to slope.

4) Establish a better time line for the occurrence of events, so time representation is more realistic.

5) Obtain realistic sensor and weapons data to increase the level of realism.

6) Enhance target definition modeling by incorporating target to background contrast and a more realistic representation of atmospheric attenuation.

7) Incorporate multiple weapon types to include indirect fire systems such as grenade launchers or mortars, and consider ways to portray minefields and booby traps.

8) The graphics can be greatly enhanced. The terrain representation needs to be much clearer, and perhaps a pattern can be incorporated to display the trafficability index of each grid. Additionally, the icons would be much more informative if they displayed the soldier's status and posture.

9) The decision processes are very rudimentary and could be vastly improved. Communications also could be improved to incorporate transmission time delays and misinterpretation.

10) Improve the user interface with the model by making instructions clearer and simplifying the overall process by steps such as minimizing the number of query prompts and user key strokes.

This list could be much more extensive, but the ideas provided seem to be reasonable steps that stay within the original context of the modeling project.

A final note, giving this model to students as an instructional tool requires that some means should be established to get feedback from the students after each use of the model. Their suggestions then could be incorporated into a new model. This would ensure that SPARTAN remains a viable tool for a long time to come.

5.3 Conclusion

This thesis effort provides the military modeling community with a viable instructional tool for illustration of high resolution land

combat modeling. The scope of the project encompassed all the primary aspects of simulation modeling from the initial problem statement through coding and implementation of a functional model. The end result of this project is an "analytic type" combat simulation that includes a preprocessor, and online instructional displays. These features provide the student with sufficient information to use the model and to learn how the model simulates the various combat processes.

The end result of this thesis effort is a high resolution land combat model with many of the key features of actual analytic models of much greater size and complexity.

Appendix A: Computation Templates for Probability Tables

This appendix discusses the computational methods used to create the various probability tables used in SPARTAN. The equations are adapted from those discussed by Hartman (23) for probability of hit and the Night Vision Electro-Optical Laboratory algorithms for acquisition and detection as discussed by Bailey (1). Since the intent of SPARTAN was to model only the most basic aspects of the combat processes, the algorithms were simplified by replacing some variables with constants [these are noted in the templates]. The data for these tables was created arbitrarily. The goal was to provide the model with "reasonable" values that would produce reasonable results. There was no intent to validate any of the data or the results of the model.

MATHCAD [version 2.5] mathematics software was used to create the various probability tables. The software was quite useful since it allows the user to create a reuseable template that displays all the computations along with text comments for internal documentation.

Probability of Acquisition

The template on the following page was used to create the P_1 acquisition tables. The equations were adapted from those discussed by Bailey in a Rand report on the NVEOL algorithms used in JANUS(14:3--10). Each value in the P_1 table represents the probability of detecting a target given unlimited time.

The following MATHCAD 2.5 template computes probabilities of acquisition using simplified versions of the Night Vision Electro-Optical Laboratories (NVEOL) optical sensor algorithms as used in JANUS and CASTFOREM.

All linear measurements are in meters and all angular measurements are in radians or milliradians.

ORIGIN ≡ 1 i := 1 ..10 j := 1 ..3

This is the average target height in each posture. $L := \begin{bmatrix} .5 \\ 1 \\ 2 \end{bmatrix}$

This is the range between sensor and target. $R_i := 100 \cdot i$

r is the resolution of the sensor expressed in resolvable cycles per milliradian. This is arbitrarily set to 1.0. The NVEOL model provides an equation to determine r in the visual range, but it would require determining several factors such as sky brightness and target to background contrast.

r := 1.

C is an array of values for resolvable cycles of the target in each posture.

$$C_{i,j} := \frac{L_j}{R_i} \cdot 1000 \cdot r$$

$$C = \begin{vmatrix} 5 & 10 & 20 \\ 2.5 & 5 & 10 \\ 1.667 & 3.333 & 6.667 \\ 1.25 & 2.5 & 5 \\ 1 & 2 & 4 \\ 0.833 & 1.667 & 3.333 \\ 0.714 & 1.429 & 2.857 \\ 0.625 & 1.25 & 2.5 \\ 0.556 & 1.111 & 2.222 \\ 0.5 & 1 & 2 \end{vmatrix}$$

M is the fixed scale value to account for the percent probability required to yield an identification decision.

M := 3.5 for all uses in SPARTAN

This section computes the probability of detection P1 given unlimited time (Bailey:5).

$$pl_{i,j} := 1 - e^{-.84 \cdot \left[\frac{C_{i,j}}{M}\right]^{2.4}}$$

$$pl = \begin{vmatrix} 0.862 & 1 & 1 \\ 0.312 & 0.862 & 1 \\ 0.132 & 0.526 & 0.981 \\ 0.069 & 0.312 & 0.862 \\ 0.041 & 0.197 & 0.686 \\ 0.026 & 0.132 & 0.526 \\ 0.018 & 0.093 & 0.403 \\ 0.013 & 0.069 & 0.312 \\ 0.01 & 0.052 & 0.246 \\ 0.008 & 0.041 & 0.197 \end{vmatrix}$$

Probability of Detection

The algorithm for probability of detection [P_2] also was adapted from Bailey's report (1:7). For SPARTAN, the template on the following page computes P_2 as a value that varies according to range and the time spent looking in the sector containing the potential target.

Target posture is not considered in the computation of P_2. As a means of saving main memory only one P_2 table is used instead of the three that would be required if each posture was considered. Target posture is accounted for in the P_1 algorithm. The target size for crouched posture is used as an average value throughout the SPARTAN P_2 algorithm.

Search time has a range of .4 - 4.0 seconds with a mean of 2.0 seconds. This implies that the observer scanned the sector containing the target for 2.0 seconds on the average. There are ten possible search times.

$$j := 1 \ldots 10 \qquad t_j := .4 \cdot j$$

As a simplification, the P2 table was computed to vary only with respect to range and time. All resolution values use the crouch posture as inaverage value. This was done to minimize the number of look up tables in storage since there is a limit to the amount of available memory in many personal computers.

Values for C and M are those created in the probability of acquisition template.

This is the NVL equation for probability of detection.

$$P2_{i,j} := 1 - e^{-\left\lceil \frac{C_{i,2}}{M} \right\rceil \cdot \left\lceil \frac{t_j}{6.8} \right\rceil}$$

Probability of detection table with time represented by columns and range by rows.

$$P2 = \begin{vmatrix} 0.155 & 0.285 & 0.396 & 0.489 & 0.568 & 0.635 & 0.692 & 0.739 & 0.78 & 0.814 \\ 0.081 & 0.155 & 0.223 & 0.285 & 0.343 & 0.396 & 0.445 & 0.489 & 0.531 & 0.568 \\ 0.054 & 0.106 & 0.155 & 0.201 & 0.244 & 0.285 & 0.324 & 0.361 & 0.396 & 0.429 \\ 0.041 & 0.081 & 0.118 & 0.155 & 0.189 & 0.223 & 0.255 & 0.285 & 0.315 & 0.343 \\ 0.033 & 0.065 & 0.096 & 0.126 & 0.155 & 0.183 & 0.21 & 0.236 & 0.261 & 0.285 \\ 0.028 & 0.054 & 0.081 & 0.106 & 0.131 & 0.155 & 0.178 & 0.201 & 0.223 & 0.244 \\ 0.024 & 0.047 & 0.069 & 0.092 & 0.113 & 0.134 & 0.155 & 0.175 & 0.194 & 0.213 \\ 0.021 & 0.041 & 0.061 & 0.081 & 0.1 & 0.118 & 0.137 & 0.155 & 0.172 & 0.189 \\ 0.019 & 0.037 & 0.054 & 0.072 & 0.089 & 0.106 & 0.123 & 0.139 & 0.155 & 0.17 \\ 0.017 & 0.033 & 0.049 & 0.065 & 0.081 & 0.096 & 0.111 & 0.126 & 0.14 & 0.155 \end{vmatrix}$$

Probabilities of Hit

The purpose of the MATHCAD template on the following page was to create tables of hit probabilities using an algorithm discussed by Hartman (23:7-17). This algorithm computes a bivariate normal distribution of hits. This algorithm assumes Cov (x,y) = 0, and the center of the impact point distribution is the original aimpoint. Additionally, the variance is the same in both directions. The equation is intended to model hits on a circular target rather than a human silhouette, so as a gross approximation, a percentage of the circular area is used to represent a soldier's target surface.

The purpose for using this equation was to provide the model with data that varied over a range of possible values much the same way empirical data might, if it were available. The parameters are totally arbitrary. The table shown at the end of the template is for a 600m effective weapon. Tables were created for four weapons with different effective ranges. The only difference between weapons is the standard error of each weapon.

The values in the table were judged to be reasonable by the author based on his limited experience with military small arms. For an actual study, the probabilities would be determined from weapons testing or possibly obtained from the US Army Ballistic Research Laboratory.

This template computes the values for the phit600 table.

This block defines the indices for the range and posture variables in the bivariate normal equation.

ORIGIN ≡ 1 i := 1 ..10 j := 1 ..3

As stated in the assumptions, bias = 0

These are the three target posture modifiers.

prone posture := .25 (posture 1) crouched posture := .5 (posture 2) standing posture := 1.0 (posture 3)

A simple formula was used to create standard error values for four types of direct fire weapons. Each weapon has a standard error value for each 100m increment out to 1000m. These σ are totally hypothetical since there was never any intent to get accurate data for particular weapon types.

$$\sigma_i := .6 + \sqrt{i \cdot .1}$$

NOTE: Starting σ for each weapon type
- phit300 uses σ = 1.0
- phit400 uses σ = 0.9
- phit500 uses σ = 0.8
- phit600 uses σ = 0.6

This formula computes a relative value for target surface area based on a 2 meter x .5 meter figure modified for the posture.

$$R_{i,j} := \sqrt{2 \cdot posture_j \cdot .5}$$

Below is the Hartman equation used to approximate a bivariate normal hit distribution.

$$phit_{i,j} := 1 - e^{-\left[\frac{\left[R_{i,j}\right]^2}{2 \cdot \sigma_i^2}\right]}$$

phit =

Target Posture: prone	crouch	standing	Range
0.138	0.258	0.449	100m
0.108	0.204	0.366	200m
0.091	0.173	0.316	300m
0.079	0.152	0.28	400m
0.071	0.136	0.254	500m
0.064	0.124	0.233	600m
0.059	0.114	0.215	700m
0.054	0.106	0.201	800m
0.051	0.099	0.188	900m
0.048	0.093	0.177	1000m

Appendix B: STARTUP Preprocessor Program Listing

This appendix provides a listing of the QuickBASIC version 4.5 program code for the STARTUP preprocessor. STARTUP is a menu-driven program that enables the user to create, edit, and review the three scenario dependent files used in SPARTAN. These three files are the initial event datafile, the terrain attribute file, and the soldier attribute file. Specific information on the composition of these files is available in Section 3 of the User's Manual in Appendix D.

The code is organized into five modules with subprograms within each module. The startup.bas module creates the user menus and calls the specific functions requested by the user. The initevnt.bas module contains the subprograms that support the initial event datafile. The terrain.bas module contains subprograms that support the terrain data file functions. The soldier.bas module supports the soldier attribute file, and lastly the util.bas module subprograms support functions in all the other modules.

There are two special notes about this code. QuickBASIC does not use a line continuation feature, so ampersands [&] have been used in this text version of the code to indicate a line extension. In the help subprogram, all the screen text was deleted. Only an example of the help menu structure remains.

```
'*************************************************************************
'*                                                                       *
'*                            STARTUP.BAS                                *
'*                                                                       *
'*************************************************************************
'
' PURPOSE  This program is a preprocessor for the SPARTAN combat
' model.  It allows the user to modify three key input files for the
' combat model (terrain, initial events, and soldier attributes).
'
' This program contains five modules. This main module has a function
' to link the others together and query the user as to what task is to be
' performed.  There is one module for each file to be modified. Each
' module allows the user to create, edit and display the specific file.
' The last module contains utility subprograms that can be used
' throughout the program.

' All subprograms and functions that could be called or defined in this
' module are explicitly declared at the beginning of each module.

DECLARE SUB ground ()           ' user interface to terrain editor
DECLARE SUB soldaten ()         ' user interface to soldier file editor
DECLARE SUB Createarray ()      ' creates generic soldier attribute file
DECLARE SUB Displayarray ()     ' displays soldier attribute file
DECLARE SUB Editarray ()        ' editor for soldier attribute file
DECLARE SUB Create ()           ' creates a generic terrain data file
DECLARE SUB DisplayFile ()      ' displays the terrain data file
DECLARE SUB EditFile ()         ' editor for the terrain map
DECLARE SUB TerrainMap ()       ' displays a relief map of terrain file
DECLARE SUB initialevents ()    ' user interface to event editor
DECLARE SUB Displayevents ()    ' displays the initial event file
DECLARE SUB Createevents ()     ' creates a generic event file
DECLARE SUB Editevents ()       ' editor for the initial event file
DECLARE SUB opening ()          ' creates an opening presentation screen
DECLARE SUB help ()             ' online help facility for users
DECLARE SUB frame (left%, right%, top%, bottom%, fore%, back%)

DIM SHARED movers(12, 20) AS SINGLE  'dimensions soldier's attribute
                                     'array

CLS             'clears the screen at the start of the program
COLOR 1, 7     'sets a white foreground and blue background
CALL opening    'puts an opening presentation screen up

DO     'queries the user for the next task until he is done
CLS
left% = 10: right% = 70: top% = 5: bottom% = 15: fore% = 1: back% = 7
CALL frame(left%, right%, top%, bottom%, fore%, back%)
```

```
LOCATE 7, 25
PRINT "MASTER MENU": PRINT
LOCATE 8, 25
PRINT "1)  Work on event file"
LOCATE 9, 25
PRINT "2)  Work on terrain file"
LOCATE 10, 25
PRINT "3)  Work on soldier attribute file"
LOCATE 11, 25
PRINT "4)  Read Help file"
LOCATE 12, 25
PRINT "5)  Exit the program"
LOCATE 14, 25
PRINT "Type your selection  (1 to 5)"
'Wait for the user to select a key.
ch$ = INPUT$(1)

' use SELECT to process a response

SELECT CASE ch$
 CASE "1"
  CALL initialevents
 CASE "2"
  CALL ground
 CASE "3"
  CALL soldaten
 CASE "4"
  CALL help
 CASE "5"
   EXIT DO   'terminates the program
 CASE ELSE
   BEEP      'error trap
   ch$ = INPUT$(1)
END SELECT
LOOP
CLS
END


SUB ground
'*************************************************************************
'PURPOSE this routine presents the terrain editor menu and calls the
'appropriate subprograms from the terrain module
'*************************************************************************

 COLOR 1, 7
 DO
  CLS
  left% = 10: right% = 70: top% = 4: bottom% = 16: fore% = 1: back% = 7
  CALL frame(left%, right%, top%, bottom%, fore%, back%)
```

```
  LOCATE 6, 20
  PRINT "EDITOR MENU FOR TERRAIN DATA FILE"

  LOCATE 8, 20
  PRINT "1)  Create a new file."
  LOCATE 9, 20
  PRINT "2)  Edit an existing file."
  LOCATE 10, 20
  PRINT "3)  View the terrain map."
  LOCATE 11, 20
  PRINT "4)  Review the file (after editing only)."
  LOCATE 12, 20
  PRINT "5)  Exit the program."
  LOCATE 14, 20
  PRINT "Type your selection  (1 to 5)"
  'Wait for the user to select a key.
  ch$ = INPUT$(1)

  ' use select to process a response

SELECT CASE ch$   'calls the requested subprogram
  CASE "1"
   Create
  CASE "2"
   EditFile
  CASE "3"
   TerrainMap
  CASE "4"
   DisplayFile
  CASE "5"
   EXIT DO
  CASE ELSE
   BEEP
   ch$ = INPUT$(1)
END SELECT
LOOP

END SUB


SUB initialevents
'************************************************************************
'PURPOSE  this routine presents the event editor menu and calls the
'requested subprograms from the initevnt module
'************************************************************************

 DO
  COLOR 1, 7
  CLS
  left% = 10: right% = 70: top% = 4: bottom% = 15: fore% = 1: back% = 7
  CALL frame(left%, right%, top%, bottom%, fore%, back%)
```

```
  LOCATE 6, 20
  PRINT "EDITOR MENU FOR INITIAL EVENT FILE"



  LOCATE 8, 20
  PRINT "1)  Create a new event file."
  LOCATE 9, 20
  PRINT "2)  Edit event file."
  LOCATE 10, 20
  PRINT "3)  Review the file."
  LOCATE 11, 20
  PRINT "4)  Exit the program."
  LOCATE 13, 20
  PRINT "Type your selection  (1 to 4)"
  'Wait for the user to select a key.
  ch$ = INPUT$(1)

  ' use SELECT to process a response

SELECT CASE ch$
  CASE "1"
   Createevents
  CASE "2"
   Editevents
  CASE "3"
   Displayevents
   CASE "4"
   EXIT DO
  CASE ELSE
   BEEP
   ch$ = INPUT$(1)
 END SELECT
LOOP

END SUB


SUB soldaten
'************************************************************************
'PURPOSE this routine presents the soldier attribute editor menu and
'calls the requested subprograms from the soldier module
'************************************************************************

 COLOR 1, 7
 DO
  CLS
  left% = 10: right% = 70: top% = 4: bottom% = 15: fore% = 1: back% = 7
  CALL frame(left%, right%, top%, bottom%, fore%, back%)

  LOCATE 6, 20
  PRINT "EDITOR MENU FOR SOLDIER ATTRIBUTE FILE"
```

```
  LOCATE 8, 20
  PRINT "1)  Create a new soldier file."
  LOCATE 9, 20
  PRINT "2)  Modify soldier file."
  LOCATE 10, 20
  PRINT "3)  Review the file."
  LOCATE 11, 20
  PRINT "4)  Exit the program."
  LOCATE 13, 20
  PRINT "Type your selection  (1 to 4)"
  'Wait for the user to select a key.
  ch$ = INPUT$(1)

  ' use SELECT to process a response

SELECT CASE ch$
  CASE "1"
   PRINT "BUILD A FILE"
   Createarray
  CASE "2"
   Editarray
  CASE "3"
   Displayarray
   CASE "4"
   EXIT DO
  CASE ELSE
   BEEP
   ch$ = INPUT$(1)
 END SELECT
 LOOP

END SUB
```

```
'*************************************************************************
' The purpose of this module is to create and edit initial events
' for each soldier in the SPARTAN Combat Model.
'*************************************************************************

DECLARE SUB Createevents ()
DECLARE SUB Displayevents ()
DECLARE SUB Editevents ()
OPTION BASE 1  'sets initial array index to 1

DIM SHARED events(24, 3) AS SINGLE   'this establishes dimensions
                                     'for the event array
END


SUB Createevents
'*************************************************************************
'PURPOSE this routine creates a generic initial event list for SPARTAN
'*************************************************************************
CLS
OPEN "event.exp" FOR OUTPUT AS #1    'this block loads the file
FOR N = 1 TO 12                 'this loop creates a search event for
   time = 1! + .15 * N          'each soldier
   WRITE #1, 2, time, N
NEXT N

FOR L = 1 TO 12                 'this loop creates a startmove event for
   time = 2! + .17 * L          'each soldier
   WRITE #1, 1, time, L
NEXT L
CLOSE #1
CLS

' This block provides a formatted display of the file in ASCII text
OPEN "event.exp" FOR INPUT AS #1
LOCATE 1, 1
PRINT "     Event type   time  actor          Event type   time   actor"

I = 0
DO UNTIL EOF(1)
   I = I + 1
   INPUT #1, etype%, time!, actor%
   IF (I < 15) THEN
     LOCATE I + 1, 6
   ELSE
     LOCATE I - 13, 40
   END IF
   PRINT USING " ##) ###   ###.##  ###  "; I; etype%; time!; actor%
LOOP
CLOSE #1
LOCATE 23, 2
```

```
PRINT "For your review ... Hit any key to continue ...": k$ = INPUT$(1)
END SUB

SUB Displayevents
'*************************************************************************
'PURPOSE  to allow the student to review the initial event list
'*************************************************************************
CLS
' This routine provides a formatted file display in ASCII text
OPEN "event.exp" FOR INPUT AS #1
LOCATE 1, 1
PRINT "     Event type   time  actor          Event type   time   actor"

I = 0
DO UNTIL EOF(1)
 I = I + 1
 INPUT #1, etype%, time!, actor%
 IF (I < 15) THEN
   LOCATE I + 1, 6
 ELSE
   LOCATE I - 13, 40
 END IF
 PRINT USING " ##) ###   ###.##  ###  "; I; etype%; time!; actor%
LOOP
 CLOSE #1
 LOCATE 23, 1
 PRINT " Hit any key to return to main menu": k$ = INPUT$(1)
END SUB


SUB Editevents STATIC
'*************************************************************************
'PURPOSE  this routine allows the user to edit the file event.exp
'*************************************************************************

OPEN "event.exp" FOR INPUT AS #1  'this block loads events into an array
I = 0
DO UNTIL EOF(1)
  I = I + 1
  INPUT #1, etype%, etime!, eactor%
  events(I, 1) = etype%: events(I, 2) = etime!: events(I, 3) = eactor%
LOOP
CLOSE #1

DO    'this loop modifies the events in the array
 CLS
 LOCATE 1, 1
 INPUT "Which event is to be modified ?", N
 PRINT events(N, 1), events(N, 2), events(N, 3)
 INPUT "New event type is ?  ", events(N, 1)
 INPUT "New event time is ?  ", events(N, 2)
 INPUT "New event actor is?  ", events(N, 3)
```

```
 INPUT "Edit another event ? Yes - <enter>  No - <n> to quit.", answer$
LOOP UNTIL answer$ = "n" OR answer$ = "N"



OPEN "event.exp" FOR OUTPUT AS #1  'this puts event array back in file
 FOR J = 1 TO I
 WRITE #1, events(J, 1), events(J, 2), events(J, 3)
 NEXT J
CLOSE #1

END SUB
```

```
'*********************************************************************
'PURPOSE  this module is to create or edit a terrain file for the
'SPARTAN Combat Model.  It writes the data to filename board.dat. If
'you want multiple terrain files you need to copy them over with DOS
'commands prior to rerunning the create option of this program.

'The resulting data files are in ASCII format and are readable with a
'text editor or this program
'*********************************************************************

OPTION BASE 1

DECLARE SUB TerrainMap ()
DECLARE SUB Create ()
DECLARE SUB DisplayFile ()
DECLARE SUB EditFile ()

DIM SHARED terrain(2500, 4) AS SINGLE

SCREEN 9
COLOR 1, 7
WINDOW (0, 0)-(1010, 1010)

DIM SHARED clr(10) AS INTEGER
'defining colors for the map
FOR I = 1 TO 7 + 9
  clr(I) = I
NEXT I

END


SUB Create
'*********************************************************************
'PURPOSE  this routine creates a generic terrain data file that can be
'modified to meet specific scenario requirements
'*********************************************************************
CLS
SCREEN 9, , 1, 1
COLOR 1, 7
LOCATE 12, 25
PRINT " TERRAIN FILE IS BEING LOADED "
OPEN "board.exp" FOR OUTPUT AS #1
N = 0
FOR a = 1 TO 50
  FOR B = 1 TO 50
   N = N + 1
   vert = a          'indexes the north-south grids
   horz = B          'indexes the east-west grids
   elev = 20!        'inputs a 20meter elevation for all grids
   mobfac = 1!       'inputs a real value of 1.0 for mobfac
```

```
   WRITE #1, vert, horz, elev, mobfac
  NEXT B
NEXT a
CLOSE #1
LOCATE 24, 2
PRINT "Do you wish to view the file yes <y> no <enter> ?":ans$ = INPUT$(1)
IF (ans$ = "Y" OR ans$ = "y") THEN
CLS
LOCATE 1, 2: PRINT " USE THE <PAUSE> KEY TO STOP SCROLLING  THEN <ENTER>
& TO CONTINUE"
LOCATE 2, 7
PRINT " X COORD  Y COORD  ELEVATION  MOBILITY FACTOR "
PRINT "**************************************************************************
& **********"

VIEW PRINT 4 TO 25

OPEN "board.exp" FOR INPUT AS #1
FOR I = 1 TO 50
   FOR J = 1 TO 50
     INPUT #1, horz, vert, elev, mobfac
     PRINT USING "        ####     ####     ###.###     #.##"; horz; vert;
& elev; mobfac
   NEXT J
NEXT I
CLOSE #1
ELSE
END IF
CLS
SCREEN 9  'eliminates view print option
END SUB


SUB DisplayFile
'**************************************************************************
'PURPOSE  prints out a listing of the block of grid cells chosen from
'the terrain file
'**************************************************************************
CLS
PRINT "Which section of terrain do you wish to view. The file is in a"
PRINT "linear list format, so give a start and endnumber"
PRINT "To compute record number  (x - 1) * 50 + y"
INPUT " Starting number... ", start
INPUT " Ending number ...  ", ending
CLS
FOR I = start TO ending
 PRINT terrain(I, 1), terrain(I, 2), terrain(I, 3), terrain(I, 4)
NEXT I
PRINT " Hit any key to return to main menu":  k$ = INPUT$(1)
END SUB
```

```
SUB EditFile
'*************************************************************************
' PURPOSE  This routine queries the user for a specific grid location.
' Those coordinates are converted into a record number and the user is
' prompted to update the elements of that record.
'*************************************************************************

CLS
LOCATE 12, 12
PRINT " Wait just a moment while file is loaded into a buffer."

OPEN "board.exp" FOR INPUT AS #1
  I = 0
  DO UNTIL EOF(1)
    I = I + 1
    INPUT #1, X, Y, elev, mobfac
    terrain(I, 1) = X: terrain(I, 2) = Y: terrain(I, 3) = elev:
& terrain(I, 4) = mobfac
  LOOP
CLOSE #1

DO
CLS
INPUT "What are the coordinates of the grid to be modified ?", X, Y
xy = 0
FOR I = 1 TO X
  FOR J = 1 TO Y
    xy = xy + 1
  NEXT J
NEXT I

terrain(xy, 1) = X
terrain(xy, 2) = Y
INPUT "elev (0 - 69meters)...", terrain(xy, 3)
INPUT "mobfac (0 - 1.0)...", terrain(xy, 4)

INPUT "Edit another grid ? Yes - <enter>  No - <n> to quit.", answer$
LOOP UNTIL answer$ = "n" OR answer$ = "N"

LOCATE 12, 16
PRINT "One moment while the terrain record is updated."
OPEN "board.exp" FOR OUTPUT AS #1
FOR I = 1 TO 2500
  WRITE #1, terrain(I, 1), terrain(I, 2), terrain(I, 3), terrain(I, 4)
NEXT I
CLOSE #1
END SUB
```

```
SUB TerrainMap
'*************************************************************************
'PURPOSE this routine draws a relief map that represents relief features
'*************************************************************************

CLS
SCREEN 9
COLOR 7, 0
WINDOW (0, 0)-(1010, 1010)
'defining colors for the map
FOR I = 1 TO 7
  clr(I) = I + 9
NEXT I

OPEN "board.exp" FOR INPUT AS #1
N = 0
FOR I = 1 TO 50
  FOR J = 1 TO 50
   INPUT #1, horz, vert, elev, mobfac
   PSET (20 * horz - 10, 20 * vert - 10), clr(INT(elev / 10!) + 1)
  NEXT J
NEXT I
CLOSE #1
LOCATE 24, 1
PRINT "Hit any key to continue >": a$ = INPUT$(1)
COLOR 1, 7
END SUB
```

```
                              SOLDIER.BAS
'*********************************************************************
' The purpose of this program is to create or edit attribute files
' for each of the soldier (entities) with in the SPARTAN Combat Model.
' It will write the data to filename force.dat. If you want multiple
' entity files you will need to copy them over with DOS commands
' prior to rerunning the create option of this program.
'*********************************************************************

DEFINT A-Z
OPTION BASE 1
DECLARE SUB Createarray ()
DECLARE SUB Displayarray ()
DECLARE SUB Editarray ()

' This block describes the fields in the soldier records.

DIM SHARED x AS SINGLE          ' x,y,z are soldier's present coordinates
DIM SHARED y AS SINGLE
DIM SHARED z AS SINGLE
DIM SHARED xlast AS SINGLE      ' these are the soldier's last coordinates
DIM SHARED ylast AS SINGLE      ' mostly important for graphics
DIM SHARED size AS SINGLE        ' size should be between 1.6 - 2.0 meters
DIM SHARED speed AS SINGLE       ' range of speed is 0 - 40 units
DIM SHARED dir AS SINGLE         ' direction of travel in radians
DIM SHARED moving AS INTEGER     ' flag shows soldier's intent to move
DIM SHARED wpnrng AS SINGLE      ' max eff range of soldier's weapon
DIM SHARED ammo AS INTEGER       ' count of soldier's ammo remaining
DIM SHARED status AS INTEGER     ' flag shows if alive,dead or wounded
DIM SHARED posture AS INTEGER    ' indicates standing, crouched or prone
DIM SHARED incmd AS INTEGER      ' a 0 or 1  (1 indicates a sqd ldr)
DIM SHARED atkdir AS SINGLE      ' stores original movement direction
DIM SHARED tgteng AS INTEGER     ' target which is selected for engagement
DIM SHARED side AS INTEGER       ' identifies side the soldier is on

DIM SHARED movers(12, 20) AS SINGLE  'dimensions soldier's attribute
                                     'array
END


SUB Createarray
'*********************************************************************
'PURPOSE  to create a generic soldier attribute array that can be
'easily modified for specific scenario requirements in SPARTAN
'*********************************************************************
CLS
OPEN "force.exp" FOR OUTPUT AS #1

'this block creates array values for the blue soldiers
N = 0
FOR N = 1 TO 6
 x = 100 - 10 * N
```

```
 y = 400
 z = 10.5
 xlast = 0
 ylast = 0
 size = 1.8      ' meters in height
 speed = 20
 dir = 1.6       ' this is a direction of travel in radians
 moving = 1      'This is an integer value of 0 or 1 (1 is moving)
 wpnrng = 500    'Range in meters
 ammo = 20       'rounds of ammunition on individual
 status = 1      'This is an integer value of 0 or 1 (1 is alive)
 posture = 1!    ' 1 is erect   .5 is crouch   .25 is prone
 incmd = 0       ' a 0 or 1 which designates a squad leader
 atkdir = dir    ' maintains original movement direction
 tgteng = 0      ' identifies selected target
 side = 1        ' identifies the soldier's affiliation as blue
 WRITE #1, x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng,
& ammo, status, posture, obs1, obs2, tgteng, side

NEXT N

'this block creates the array values for the red soldiers

FOR N = 7 TO 12
   x = 610
   y = 10 + N * 10
   z = 10.5
   xlast = 0
   ylast = 0
   size = 1.8      ' meters of height
   speed = 20      ' meters per move
   dir = 4.7       ' this is a real value for direction in radians
   moving = 1      'This is an integer value of 0 or 1 (1 is moving)
   wpnrng = 600    'Range in meters
   ammo = 20       'rounds of ammunition on individual
   status = 1      'This is an integer value of 0 or 1 (1 is alive)
   posture = 1!    ' 1 is standing  .5 is crouch    .25 is prone
   incmd = 0       'a 0 or 1 which designates the squad leader
   atkdir = dir    'maintains original attack direction
   tgteng = 0      'identifies selected target
   side = -1       ' 1 is blue and -1 is red for force discrimination

 WRITE #1, x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng,
& ammo, status, posture, obs1, obs2, tgteng, side

NEXT N
CLOSE #1

' Display the new soldier files for review in unformatted form
' cannot use format and keep all records on screen.
```

```
OPEN "force.exp" FOR INPUT AS #1
FOR I = 1 TO 12
  INPUT #1, x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng,
& ammo, status, posture, obs1, obs2, tgteng, side

  WRITE x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng, ammo,
& status, posture, obs1, obs2, tgteng, side
NEXT I

PRINT "For your review ... Hit any key to continue ...": k$ = INPUT$(1)

CLOSE #1

END SUB


SUB Displayarray
'************************************************************************
' This routine provides an unformatted display of the text in ASCII text
' unformatted is the only way it will fit neatly on the screen.
'************************************************************************

OPEN "force.exp" FOR INPUT AS #1
FOR I = 1 TO 12
 INPUT #1, x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng, ammo,
& status, posture, incmd, obs2, tgteng, side

 WRITE x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng, ammo,
& status, posture, incmd, obs2, tgteng, side

NEXT I
 CLOSE #1
 INPUT " Hit any key to return to main menu", k$
 END SUB


SUB Editarray
'************************************************************************
'PURPOSE  this subprogram is an editor for the soldier attribute file
'************************************************************************

OPEN "force.exp" FOR INPUT AS #1
CLS

I = 0
DO UNTIL EOF(1)   'this block loads the file elements into a buffer
 I = I + 1
 INPUT #1, x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng,
& ammo, status, posture, incmd, obs2, tgteng, side
 movers(I, 1) = x: movers(I, 2) = y: movers(I, 3) = z
 movers(I, 4) = xlast: movers(I, 5) = ylast: movers(I, 6) = size
 movers(I, 7) = speed: movers(I, 8) = dir: movers(I, 9) = moving
```

```
 movers(I, 10) = wpnrng: movers(I, 11) = ammo: movers(I, 12) = status
 movers(I, 13) = posture: movers(I, 14) = incmd
 movers(I, 15) = obs2: movers(I, 16) = tgteng: movers(I, 17) = side
LOOP
CLOSE #1

'this block queries the user about modifying the attribute file
DO
PRINT "Do you wish to (a) modify all attributes of one individual "
PRINT "or (b) modify a single attribute on a group of soldiers ?"
PRINT "or (c) Exit the edit menu"
PRINT " Type (a ,b,or c) ":  which$ = INPUT$(1)

SELECT CASE which$

  CASE "a"
  INPUT "What is the number of the soldier to be modified ?", N

  PRINT "These are the soldier's present attributes"
  PRINT
  WRITE movers(N, 1), movers(N, 2), movers(N, 3), movers(N, 4),
  & movers(N, 5), movers(N, 6), movers(N, 7), movers(N, 8),
  & movers(N, 9), movers(N, 10), movers(N, 11), movers(N, 12),
  & movers(N, 13), movers(N, 14), movers(N, 15), movers(N, 16),
  & movers(N, 17)

  INPUT "Initial x  ", movers(N, 1)
  INPUT "Initial y  ", movers(N, 2)
  INPUT "Initial z  ", movers(N, 3)
  movers(N, 4) = 1
  movers(N, 5) = 1
  INPUT "Soldier size in meters  ", movers(N, 6)
  INPUT "Speed  ", movers(N, 7)
  INPUT "direction in radians  ", movers(N, 8)
  INPUT "Moving 1 is moving 0 is stationary.. ", movers(N, 9)
  INPUT "Weapon Range in meters  ", movers(N, 10)
  INPUT "Number of rounds of ammunition.. ", movers(N, 11)
  INPUT "Status '1' is alive '0' is dead  ", movers(N, 12)
  INPUT "Posture '.25' prone,'.5' crouched,'1.0' standing", movers(N, 13)
  INPUT "Command designator follower (0) leader (1)  ", movers(N, 14)
  movers(N, 15) = movers(N, 8)
  movers(N, 16) = 0

CASE "b"
 INPUT "Type <first,last> ID #'s for group to be modified."; first, last
 INPUT "Which attribute is to be modified (1 - 17) "; k
 INPUT "What should this new value be  "; newvalue!
  FOR I = first TO last
    movers(I, k) = newvalue!
  NEXT I
CASE "c"
  EXIT DO
```

```
CASE ELSE
  BEEP
  PRINT " either <a>, <b>, or <c> ": which$ = INPUT$(1)
END SELECT
LOOP

OPEN "force.exp" FOR OUTPUT AS #1   'writes changes into force.exp file
  FOR I = 1 TO 12
    WRITE #1, movers(I, 1), movers(I, 2), movers(I, 3), movers(I, 4),
& movers(I, 5), movers(I, 6), movers(I, 7), movers(I, 8), movers(I, 9),
& movers(I, 10), movers(I, 11), movers(I, 12), movers(I, 13),
& movers(I, 14), movers(I, 15), movers(I, 16), movers(I, 17)
  NEXT I
CLOSE #1
END SUB
```

```
'*************************************************************************
'*                                                                       *
'*                              UTIL.BAS                                 *
'*                                                                       *
'*************************************************************************

' This utility module has subprograms and functions that can be called
' from any of the other modules.  These routines do not require access
' to variables used in the other modules.

DECLARE SUB opening ()  '  draws out the presentation screen
DECLARE SUB frame (left%, right%, top%, bottom%, fore%, back%)

'Below is the syntax for using the Frame subprogram
' left% = 3: right% = 80: top% = 3: bottom% = 22: fore% = 5: back% = 0
' CALL Frame(left%, right%, top%, bottom%, fore%, back%)

END


SUB frame (left%, right%, top%, bottom%, fore%, back%) STATIC
'*************************************************************************
' This routine creates a framing box of any size  It takes six basic
' input parameters that control the dimensions and colors.
'*************************************************************************

COLOR fore%, back%

' ----- Draw the four corners

     LOCATE top%, left%: PRINT CHR$(201)
     LOCATE top%, right%: PRINT CHR$(187)
     LOCATE bottom%, left%: PRINT CHR$(200)
     LOCATE bottom%, right%: PRINT CHR$(188)

' ----- Draw the vertical lines

     FOR vert% = top% + 1 TO bottom% - 1
             LOCATE vert%, left%: PRINT CHR$(186)
             LOCATE vert%, right%: PRINT CHR$(186)
     NEXT vert%

' ----- Draw the horizontal lines

     horiz% = right% - left% - 1
     hline$ = STRING$(horiz%, 205)
     LOCATE top%, left% + 1: PRINT hline$
     LOCATE bottom%, left% + 1: PRINT hline$

END SUB
```

NOTE: This help subprogram is only an extract of the original subprogram. The original subprogram contains many screens of print text. This extract only contains an example of the coding used to create the user help menus.

```
SUB help
'************************************************************************
' PURPOSE  this routine is intended to provide the basic information
' necessary to use the STARTUP program.  Info provided is on how to
' create, modify and view the files needed to run the SPARTAN Combat
' Model. This routine allows the user to go directly to the
' information desired.
'************************************************************************

COLOR 1, 7
CLS
DO
LOCATE 4, 33
PRINT " HELP MENU "
LOCATE 6, 23
PRINT " SELECT ONE OF THE FOLLOWING TOPICS"
LOCATE 8, 27
PRINT " 1)  Operating SPARTAN"
LOCATE 9, 27
PRINT " 2)  Initial Event Datafile"
LOCATE 10, 27
PRINT " 3)  Terrain Datafile"
LOCATE 11, 27
PRINT " 4)  Soldier Attribute Datafile"
LOCATE 12, 27
PRINT " 5)  Exiting Help"

LOCATE 14, 1
PRINT " The STARTUP preprocessor program allows the user to modify three "
PRINT "key datafiles used by the SPARTAN Combat Model.  The 3 files are"
PRINT "an initial event list, a terrain file, and a soldier attribute file."
PRINT "The program allows the user to create a generic file for each, edit"
PRINT "each file and view the data in the files.  The help function is "
PRINT "intended to provide guidance on each function in the program and "
PRINT "should be read prior to working with the files."

CALL frame(1, 79, 2, 23, 1, 7)

LOCATE 22, 6: PRINT "Select Option Number  ": k$ = INPUT$(1)
SELECT CASE k$     'this select sends the user to the major category
CASE "1"
  GOSUB h500
CASE "2"
  GOSUB h200
CASE "3"
  GOSUB h300
CASE "4"
  GOSUB h400
```

```
CASE "5"
  EXIT DO
CASE ELSE
  BEEP
  k$ = INPUT$(1)
END SELECT
LOOP

GOTO 1000 'when the program finishes executing the help select loop, it
          'must jump over the subroutines and go to the end of the help
          'subprogram.  Otherwise, it will try to execute the subs and
          'cause an error.

'these subroutines contain groups of topics and allow the user to
'get specific details from within each major topic area.

h200:
CLS
COLOR 7, 1
LOCATE 2, 1
PRINT "SELECT A SUBTOPIC <1>CREATE<2>EDIT<3>REVIEW<4>PARAMETERS<5> EXIT "
COLOR 1, 7
LOCATE 4, 1
PRINT " The events module allows the user to create, edit and review the "
RETURN
                                    *
                                    *
                ****************************************
                The bulk of the help screen are formatted
                screens of information.  These were deleted
                from this text.  What remains of the help
                subprogram is just the structure of how the
                menu system was set up.
                ****************************************
                                    *
                                    *
1000 LOCATE 24, 2
PRINT "Returning to main menu"
END SUB
```

```
SUB opening
'************************************************************************
'PURPOSE  This routine calls the frame routine and draws a presentation
' screen for the model introduction.

' VARIABLES
' left%, right%, top%, bottom% are integer variables passed to frame
' that define the size of the box using rows and columns in text
' mode.
' fore% and back% are variables that define line and background colors
'************************************************************************

CLS
left% = 1: right% = 80: top% = 3: bottom% = 22: fore% = 1: back% = 7
CALL frame(left%, right%, top%, bottom%, fore%, back%)
left% = 9: right% = 72: top% = 10: bottom% = 16: fore% = 1: back% = 7
CALL frame(left%, right%, top%, bottom%, fore%, back%)
LOCATE 13, 25: PRINT "     SPARTAN COMBAT MODEL  "
LOCATE 14, 25: PRINT "          PREPROCESSOR   "
LOCATE 19, 22: INPUT "Press <Enter> when ready to continue", start
COLOR 1, 7
END SUB
```

Appendix C: SPARTAN Simulation Program Listing

This appendix contains the programming code for the SPARTAN simulation model. The code is written in QuickBASIC version 4.5.

The code is contained in three separate modules. The spartan.bas module contains the outer replication loop and the event scheduling algorithms that control the execution of the simulation. The combat.bas module contains all the subprograms and functions that model the combat processes. The utility.bas module contains subprograms and functions that support the processes within the other modules.

There are two special notes about this code. QuickBASIC does not use a line continuation feature, so ampersands [&] have been used in this text version of the code to indicate a line extension. In the help subprogram, all the screen text was deleted. Only an example of the help menu structure remains.

```
'*********************************************************************
'*                                                                   *
'*              ******** SPARTAN COMBAT MODEL ********               *
'*                                                                   *
'*                        **** MAIN MODULE ****                      *
'*                                                                   *
'*********************************************************************

' The modular organization of this program is set up to have all key
' elements of the program in subprograms or functions.  This structuring
' means that the main program serves only to define which variables are
' accessible to which routines, and which routines will be called.  The
' program contains three modules.  In this main module, the purpose of
' the subprograms are to initialize the databases, operate the event
' scheduling routine and pass execution control to the various
' subprograms as necessary.  The combat module contains the subprograms
' that perform the combat processes.  While the utility module contains
' subprograms and functions that can be called from any of the modules.
'*********************************************************************

' This section declares all subprograms and functions in this module or
' used by this module.  To view the subprograms and functions hit F2
' and a menu will be presented.

DECLARE SUB search ()       ' uses continuous search algorithm
DECLARE SUB startmove ()    ' computes and updates next location
DECLARE SUB endmove ()      ' schedules next startmove events
DECLARE SUB initialize ()   ' Opens and loads all data files
DECLARE SUB linkempties ()  ' Initializes links of events list
DECLARE SUB event (e!)      ' transfers program control to next event
DECLARE SUB clock ()        ' calls next event off future event list
DECLARE SUB pause ()        ' used to insert pauses during debugging
DECLARE SUB remove (e!)     ' remove event e from event calendar
DECLARE SUB move (e!, fromlist, tolist!) 'move e from fromlist! to tolist!
DECLARE SUB showevents ()   ' prints top of event list and description
DECLARE SUB addevent (itype!, time!, entity) 'add event to event calendar
DECLARE SUB opening ()          ' draws an opening title screen
DECLARE SUB reacttofire ()      ' decides tgt reaction to being fired on
DECLARE SUB impact ()           ' determines outcome of engagement
DECLARE SUB tgtselect ()        ' decides if and at who firer will engage
DECLARE SUB directfire ()       ' computes time of flight and calls impact
DECLARE SUB plotterrain ()      ' plots relief map to represent elevation
DECLARE SUB killsoldier (tgt)   ' pulls events from FEL as soldier dies
DECLARE SUB oput ()             ' collects and prints battle results
DECLARE SUB help ()             ' online help utility for the user
DECLARE SUB soldierstats ()     ' displays current soldier attributes
DECLARE SUB battlestats ()      ' displays a sample of current battle stats
DECLARE SUB setup ()            ' allows the user to modify run parameters
DECLARE SUB frame (left%, right%, top%, bottom%, fore%, back%)
DECLARE SUB pottgtlist ()       ' shows a table of the target list values
```

```
' The 'events' COMMON variables are only shared in the SPARTAN module.

COMMON SHARED /events/ Maxevents, Firstevent, Firstempty, eventtype(),
& timeofevent(), backlink(), nextlink()

' The variables defined in this COMMON statement are
' accessible in the main and combat modules.

COMMON SHARED time, movers(), te, eventactor(), elevation(), mobility(),
& p1(), p2(), ph300(), ph400(), ph500(), ph600(), attenuation, activeblue,
& activered
COMMON SHARED bluecommand, redcommand, direct()
COMMON SHARED bluecount, redcount, bluerounds, redrounds

Maxevents = 99    ' This variable determines the size of the arrays used
                  ' in the event lists

DIM eventactor(Maxevents)    'eventactor(i)  = soldier index
DIM eventtype(Maxevents)     'eventtype(i)   = type
                             '               = 0 if empty
DIM timeofevent(Maxevents)   'timeofevent(i) = t ... if pending then time
                                               of this event

                             '               = i ... if empty then position
                             '                 in list of empties
DIM backlink(Maxevents):     'backlink(i)    = immediate predecessor
DIM nextlink(Maxevents):     'nextlink(i)    = immediate successor

' These arrays provide data files on the soldiers and terrain.
' They are loaded by subprogram 'initialize'

DIM SHARED movers(12, 20), elevation(50, 50), mobility(50, 50),
& direct(12, 9)

' These arrays contain probabilities for the search and direct fire
' routines
' p1 is probability of detection with infinite time (acquisition)
' p2 is probability of detection for a given time   (detection)
' ph is probability of hit
' sspk is probability of kill

DIM SHARED p1(10, 3), p2(10, 10), ph300(10, 3), ph400(10, 3), ph500(10, 3),
& ph600(10, 3)

'this section dimensions run parameters and defines their default
'values

DIM SHARED terminate, timestop, bluestop, redstop
LET terminate = 5000: LET timestop = 300!: LET attenuation = 1!
LET bluestop = 3: LET redstop = 4
```

```
'this section defines default values for switches that turn on command
'functions on each side.
LET bluecommand = 1:   LET redcommand = 1

CLS
    COLOR 7, 1
    CALL opening        ' Draws an opening title screen
CLS
    CALL linkempties    ' Initializes links in event list
    CALL initialize     ' Opens and loads data files
                        ' Loads initial events into event list
                        ' (must be in time sequence order from data file)
                        ' Sets system clock to 0.0

' Now the events(*,*) array looks like this:

'  Record #  actor   event type      refers back to    points forward to
'     1        1       1              1 (it's first)     3
'     2        0       0 (no event)   2 (it's first)     4
'     3        5       2              1                  5
'     4        6       0              2                  7
'     5        7       3              3                  6
'     6        4       4              5                  6 (it's last)
'     7        3       0              4                  8
'     8        9       0              7                  9
'     e       12       0             e-1                e+1

' This section turns on function keys (F1 - F5) for special information
' screens. The subroutines referenced are at the end of this module.

ON KEY(1) GOSUB hlp
KEY(1) ON
ON KEY(2) GOSUB soldierscreen
KEY(2) ON
ON KEY(3) GOSUB battlescreen
KEY(3) ON
ON KEY(4) GOSUB ground
KEY(4) ON
ON KEY(5) GOSUB targetlist
KEY(5) ON

SCREEN 9                    'provides the highest possible resolution
WINDOW (0, 0)-(1000, 1050) 'for EGA/VGA and sets the origin at the lower
                            'left hand corner of the screen.

active = 0: visual = 0 ' defines initial page values for the process of
                       ' of writing and viewing alternate screens

COLOR 7, 1 ' set foreground white and background blue
PRINT "Do you wish to modify the default settings <y> or <n> ":
& k$ = INPUT$(1)
IF (k$ <> "y" OR k$ <> "Y") THEN CLS
```

```
IF (k$ = "y" OR k$ = "Y") THEN PRINT "Do you want to see help first ?
& <h> help  <enter> continue ": helpkey$ = INPUT$(1):    CLS
IF (helpkey$ = "h" OR helpkey$ = "H") THEN GOSUB hlp
IF (k$ = "y" OR k$ = "Y") THEN CALL setup
COLOR 7, 0  'sets foreground white and background black
SCREEN 9, , active, visual  'returns the screen to original page before
                            'beginning to plot terrain

CALL plotterrain ' plots the initial relief grid on the screen (page 0)

' This is the master loop that keeps the simulation running. It
' continues to pull the next event off the calendar until some
' termination conditions are met such as
' event number (te) = 0 or a specified # of events

DIM SHARED timetostop AS INTEGER

OPEN "history.dat" FOR OUTPUT AS #1
PRINT #1, "    TIME        TYPE         ACTOR"

'this line sets the system counters to 0
quit = 0: timetostop = 0: activeblue = bluecount: activered = redcount
DO WHILE timetostop = 0
   CALL clock
   quit = quit + 1       ' This counter terminates the program after
                         ' a specified number of events are called

   LOCATE 1, 1
   PRINT USING " <F1> HELP  <F2> SOLDIER STATUS  <F3> SUM STATS
   & <F4> TERRAIN  <F5> TGTS ####.##"; time

   SCREEN 9, , active, visual  'returns to the battle screen after
                               'a function key has been used
   IF (quit >= terminate OR time >= timestop OR activeblue <= bluestop OR
   & activered <= redstop) THEN timetostop = 1
LOOP
CLOSE #1

CALL oput  'sends selected final output to the screen

END

hlp:                          'switches to page 1 and provides the user
 LET visual = 1               'with a layered help menu on how to work
 SCREEN 9, active, visual     'with the simulation
 CALL help
 COLOR 1, 0
 LET visual = 0
 RETURN
soldierscreen:       'switches to page 1 and displays soldier attributes
 LET visual = 1
 SCREEN 9, active, visual
```

```
 CALL soldierstats
 COLOR 1, 0: CLS 1
 LET visual = 0
 RETURN
battlescreen:          'switches to page 1 and displays battle statistics
 LET visual = 1
 SCREEN 9, active, visual
 CALL battlestats
 COLOR 1, 0: CLS 1
 LET visual = 0
 RETURN
ground:             ' plots the terrain relief on the screen when called
                    ' primarily to refresh the screen after a significant
 CALL plotterrain   ' number of the color points have been drawn over.

 RETURN
targetlist:   'switches to page 1 and displays a list of potential targets
 LET visual = 1
 SCREEN 9, active, visual
 CALL pottgtlist
 COLOR 7, 1: LOCATE 24, 1: CALL pause
 COLOR 1, 0: CLS 1
 LET visual = 0
 RETURN



SUB addevent (itype, time, entity)
'*********************************************************************
'PURPOSE  This routine adds an event to the active event list. It
'receives the type and time of event and takes the first empty location
'from the empty list. The routine then calls subprogram move to add the
'to the (pending) future event list.

' VARIABLES
' itype    an integer passed to the routine that specifies event type
' time     a real value passed to the routine that specifies event time
' Firstempty  the event number of the first event in the inactive list
' Grab     an integer buffer that stores the value for the event number
' eventtype()  the type attribute used in the future event list
' eventactor()  the index of the soldier that performs event
' timeofevent() the time attribute used in the future event list

' This routine calls move to take the event(Grab) from the inactive
' list and place it on the future event list
'*********************************************************************

' First, grab the lowest available record from the inactive list

Grab = Firstempty
```

```
' fill the new record with event information

  eventtype(Grab) = itype
  timeofevent(Grab) = time
  eventactor(Grab) = entity

' now move the event from the inactive to the pending list (FEL)

  CALL move(Grab, Firstempty, Firstevent)

END SUB


SUB clock
'*********************************************************************
' PURPOSE  This routine calls the next event off the active future
' events list.  The routine calls event which passes control of the
' simulation to the appropriate event module, and then removes the
' event from the active FEL and places it in the inactive file list.

' VARIABLES
' te     is an integer buffer variable that is set equal to Firstevent
' Firstevent  this integer variable is a pointer to the first position
'             in the FEL.

' event is called to transfer control to the event routine identified
'       by eventtype(te)
' remove is called to pull the first event from the FEL and move the
'        record to the inactive list
'*********************************************************************
'this line determines which event is the next to occur
 te = Firstevent

'this line calls the routine event which transfers program control
 CALL event(te)

' this line moves the event record off the FEL and over to the inactive
' list
 CALL remove(te)
END SUB
```

```
SUB event (te)
'*********************************************************************
' PURPOSE  This routine keeps the system clock (time) updated and
' transfers execution control to the event identified by eventtype(te)
' with the SELECT CASE function  (similar to a computed GOTO in FORTRAN)

' VARIABLES
' time   state variable that maintains the system attribute of time
' te     integer buffer that identifies next scheduled event
' timeofevent()  is the time that event te is scheduled to occur
' eventtype()  is the type of event represented by te
'*********************************************************************

'this line sends a copy of each event to a history file as it occurs
PRINT #1, timeofevent(te), eventtype(te), eventactor(te)

 time = timeofevent(te)   'this updates the system clock with the new
                          'event time

 SELECT CASE eventtype(te) 'transfers program control to the correct
 CASE 1                    'event subprogram
  CALL search
 CASE 2
  CALL startmove
 CASE 3
  CALL endmove
 CASE 4
  CALL tgtselect
 CASE 5
  CALL directfire
 CASE 6
  CALL impact
 CASE 7
  CALL reacttofire
 CASE ELSE
  BEEP
  PRINT " unknown event was sent to select in sub event"

END SELECT

END SUB
```

```
SUB initialize
'*************************************************************************
' PURPOSE  This routine has 3 primary functions which are to set the
' initial system time to zero; open and read in all data files to include
' initial events, soldier files, terrain files, and all probability
' tables; set any system constants.
' The user has the option of viewing all files.
' This subprogram does not call any other subprograms or functions
'*************************************************************************

LET time = 0!       ' This sets the initial system time at 0.0

' This block loads the initial events into the future event list (FEL)

OPEN "event.dat" FOR INPUT AS #1
PRINT "Do you want to see the data files being loaded?"
INPUT "If you do type (y). If not just hit return.  ", show$
CLS
LOCATE 2, 1
PRINT "            ***** JUST A MOMENT, INITIALIZING DATABASES *****  "
I = 0
IF (show$ = "y") THEN CLS : PRINT " Event type    Event time   Event actor"
LOCATE 3, 1
DO UNTIL EOF(1)
   I = I + 1
   INPUT #1, mytype, mytime, entity, entity2
   IF (show$ = "y") THEN PRINT , mytype, mytime, entity
   IF (I = 12 AND show$ = "y") THEN PRINT "hit any key to continue";
   & k$ = INPUT$(1): LOCATE 3, 1
   CALL addevent(mytype, mytime, entity)   'this enters initial events
                                           'from the file
LOOP
CLOSE #1

IF (show$ = "y") THEN CALL pause: CLS

' This block loads the terrain information into arrays mobility(),
' and elevation().

OPEN "board.dat" FOR INPUT AS #1
IF (show$ = "y") THEN PRINT "To view the next file, use scroll lock"
IF (show$ = "y") THEN PRINT "  x        y       elevation     mobility",
& k$ = INPUT$(1)
LOCATE 4, 2
k = 0
DO UNTIL EOF(1)
 k = k + 1
 INPUT #1, horz, vert, elev!, mobfac!
 elevation(horz, vert) = elev!
 mobility(horz, vert) = mobfac!
 IF (show$ = "y") THEN PRINT , horz, vert, elev!, mobfac!
LOOP
```

```
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

' This block loads the soldier data into the array movers()

OPEN "force.dat" FOR INPUT AS #1
IF (show$ = "y") THEN PRINT "Refer to appendices for list of soldier
& attribute names"
LOCATE 3, 1
J = 0
DO UNTIL EOF(1)
   J = J + 1
   INPUT #1, x, y, z, xlast, ylast, size, speed, dir, moving, wpnrng, ammo,
   & status, posture, incmd, atkdir, tgteng, side
   movers(J, 1) = x: movers(J, 2) = y:
   'soldier's elevation is set to his location
   movers(J, 3) = elevation(FIX(x / 20) + 1, FIX(y / 20) + 1)
   movers(J, 4) = xlast: movers(J, 5) = ylast: movers(J, 6) = size
   movers(J, 7) = speed: movers(J, 8) = dir: movers(J, 9) = moving
   movers(J, 10) = wpnrng: movers(J, 11) = ammo: movers(J, 12) = status
   movers(J, 13) = posture: movers(J, 14) = incmd: movers(J, 15) = dir
   movers(J, 16) = tgteng: movers(J, 17) = side
IF (show$ = "y") THEN PRINT USING "####.# ####.# ###.# ####.# ####.# #.#
& ##.# ##.## # ### ## # #.## ## ##.# ## ## "; x; y; z; xlast; ylast; size;
& speed; dir; moving; wpnrng; ammo; status; posture; incmd; dir; tgteng;
& side
LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

'this block records the original number of soldiers alive when the
' battle starts and the ammo available on each side
bluecount = 0: redcount = 0: bluebullets = 0: redbullets = 0
FOR I = 1 TO 6
  bluecount = bluecount + movers(I, 12)
  bluerounds = bluerounds + movers(I, 11)
NEXT I
FOR I = 7 TO 12
  redcount = redcount + movers(I, 12)
  redrounds = redrounds + movers(I, 11)
NEXT I

'This section loads probabilities of acquisition for the search module.

OPEN "p1.dat" FOR INPUT AS #1    ' this is a 10 x 3 array
IF (show$ = "y") THEN PRINT " Probabilities of Acquisition(p1) based
& on range & posture."
IF (show$ = "y") THEN PRINT " range   prone     crouched     standing"
LOCATE 4, 1
I = 0
   DO UNTIL EOF(1)
     I = I + 1
```

```
    INPUT #1, p1(I, 1), p1(I, 2), p1(I, 3)
    IF (show$ = "y") THEN PRINT I * 100, p1(I, 1), p1(I, 2), p1(I, 3)
  LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

'This section loads probabilities of detection for the search module.

OPEN "p2.dat" FOR INPUT AS #1   ' this is a 10 x 10 array
IF (show$ = "y") THEN PRINT " This is a table of detection probabilities
& given "
IF (show$ = "y") THEN PRINT " that the targets can be acquired."
IF (show$ = "y") THEN PRINT "                    search time (in sec)"
IF (show$ = "y") THEN PRINT "range    .4    .8   1.2   1.6   2.0   2.4
& 2.8   3.2   3.6   4.0"
LOCATE 5, 1
I = 0
DO UNTIL EOF(1)
  I = I + 1
  INPUT #1, p2(I, 1), p2(I, 2), p2(I, 3), p2(I, 4), p2(I, 5), p2(I, 6),
  & p2(I, 7), p2(I, 8), p2(I, 9), p2(I, 10)
  IF (show$ = "y") THEN PRINT USING "#### -  .###  .###  .###  .###  .###
  & .###  .###  .###  .###  .### "; I * 100; p2(I, 1); p2(I, 2); p2(I, 3);
  & p2(I, 4); p2(I, 5); p2(I, 6); p2(I, 7); p2(I, 8); p2(I, 9); p2(I, 10)
LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

'This section loads probabilities of hit for the direct fire module.

OPEN "phit300.dat" FOR INPUT AS #1   ' this is a 10 x 3 array
IF (show$ = "y") THEN PRINT " Probabilities of hit(ph300) based on range
& & target posture."
IF (show$ = "y") THEN PRINT " range   prone      crouched      standing"
LOCATE 4, 1
I = 0
 DO UNTIL EOF(1)
   I = I + 1
   INPUT #1, ph300(I, 1), ph300(I, 2), ph300(I, 3)
   IF (show$ = "y") THEN PRINT I * 100, ph300(I, 1), ph300(I, 2),
   & ph300(I, 3)
 LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

OPEN "phit400.dat" FOR INPUT AS #1   ' this is a 10 x 3 array
IF (show$ = "y") THEN PRINT " Probabilities of hit(ph400) based on range
& & target posture."
IF (show$ = "y") THEN PRINT " range     prone      crouched      standing"
LOCATE 4, 1
I = 0
 DO UNTIL EOF(1)
```

```
   I = I + 1
   INPUT #1, ph400(I, 1), ph400(I, 2), ph400(I, 3)
   IF (show$ = "y") THEN PRINT I * 100, ph400(I, 1), ph400(I, 2),
   & ph400(I, 3)
 LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

OPEN "phit500.dat" FOR INPUT AS #1    ' this is a 10 x 3 array
IF (show$ = "y") THEN PRINT " Probabilities of hit(ph500) based of range
& & target posture."
IF (show$ = "y") THEN PRINT " range    prone      crouched      standing"
LOCATE 4, 1
I = 0
 DO UNTIL EOF(1)
   I = I + 1
   INPUT #1, ph500(I, 1), ph500(I, 2), ph500(I, 3)
   IF (show$ = "y") THEN PRINT I * 100, ph500(I, 1), ph500(I, 2),
   & ph500(I, 3)
 LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

OPEN "phit600.dat" FOR INPUT AS #1    ' this is a 10 x 3 array
IF (show$ = "y") THEN PRINT " Probabilities of hit(ph600) based of range
& & target posture."
IF (show$ = "y") THEN PRINT " range    prone      crouched      standing"
LOCATE 4, 1
I = 0
 DO UNTIL EOF(1)
   I = I + 1
   INPUT #1, ph600(I, 1), ph600(I, 2), ph600(I, 3)
   IF (show$ = "y") THEN PRINT I * 100, ph600(I, 1), ph600(I, 2),
   & ph600(I, 3)
 LOOP
CLOSE #1
IF (show$ = "y") THEN CALL pause: CLS

END SUB
```

```
SUB killsoldier (tgt)
'*************************************************************************
'PURPOSE  Eliminates active events for soldiers when each soldier dies.
' This routine searches the event list for events of the dead soldier
' and then calls the remove subprogram to transfer those records from
' the active to the inactive event list.

'VARIABLES
' Maxevents     the size of the event list
' I             is merely a counter
' tgt           the number of the dead soldier
' eventactor()  a vector stores the identity of the soldier scheduled
'               to perform each event.
'*************************************************************************

' this block looks at each event on the active list and determines if
' it belongs to the deceased soldier. If it does then it is removed
' from the active event list

 FOR I = 1 TO Maxevents
   IF (eventactor(I) > tgt - .1 AND eventactor(I) < tgt + .1) THEN
    CALL remove(I)  'places the event record (I) in the inactive file
   ELSE
   END IF
 NEXT I
END SUB


SUB linkempties
'*************************************************************************
' PURPOSE  This routine establishes links for the records in the active
' and inactive event lists.  It gives the first event a large value so it
' will never be called before all other events have been exhausted.  This
' feature keeps the model from having an empty list an any time.

' VARIABLES
  Firstevent = 1          'Firstevent is the active list top pointer
  eventtype(1) = 8        'designates the type of event
  timeofevent(1) = 9999   'designates the time the event is scheduled for
  eventactor(1) = 99      'designates which entity will perform the event
  backlink(1) = 1         ' predecessor link that points ahead
  nextlink(1) = 1         ' successor link that points behind
'*************************************************************************

  Firstempty = 2
  FOR I = 2 TO Maxevents
    backlink(I) = I - 1
    nextlink(I) = I + 1
  NEXT I
    backlink(2) = 2
    nextlink(Maxevents) = Maxevents
END SUB
```

```
SUB move (e, fromlist, tolist)
'*****************************************************************
'PURPOSE Move an event e from its current list (starts with fromlist)
'to its new list (starts with tolist).
'The event e time, actor and type data should be modified in place
'prior to the call to MOVE.  The variable e should not be one of the
'two pointers (fromlist, tolist).  FROMLIST and TOLIST will be
'Firstempty and Firstevent - which is which will depend on whether
'MOVE is called to get an event onto or off of the calendar.
'*****************************************************************

'First, delete event e from the list it is currently in

' IF e is the first event in the list THEN
'     change the id of the start of the list
'     make the backpath stop at the new top
'   ELSEIF e is the last event THEN
'     make the nextpath stop at the new end
'   ELSE
'    just cut e out of the middle of the list

  IF backlink(e) = e THEN
     fromlist = nextlink(e)
     backlink(fromlist) = fromlist  '"seal" the backpath at the new top
  ELSEIF nextlink(e) = e THEN
     nextlink(backlink(e)) = backlink(e) 'stop the nextpath at the new end
  ELSE
     nextlink(backlink(e)) = nextlink(e)  ' path forward now leaps e
     backlink(nextlink(e)) = backlink(e)  ' path backward now leaps e
  END IF

' Now, insert e into the new path by looking for its proper position
' starting at the top as identified by tolist.

' slip down the list until the 1st event that follows e is found
  putbefore = tolist
  WHILE timeofevent(e) > timeofevent(putbefore) AND nextlink(putbefore)
  & <> putbefore
    putbefore = nextlink(putbefore)
  WEND

' IF e is before at least one event THEN
'     IF e is before the old top of the list
'          put e first
'        ELSE
'          slip e in the middle of the list
'   ELSE
'     add e to the end of the list

  IF timeofevent(e) < timeofevent(putbefore) THEN
     IF putbefore = tolist THEN
       backlink(e) = e  ' event e comes from itself
```

```
        nextlink(e) = tolist  'event e leads to the old top of the list
        backlink(tolist) = e  'old top of list now points to new top (e)
        tolist = e            ' the new top of the list is event e
      ELSE
        backlink(e) = backlink(putbefore)  'link e and its predecessor
        nextlink(backlink(e)) = e
        nextlink(e) = putbefore            'link e and its successor
        backlink(putbefore) = e
      END IF
  ELSE
    nextlink(putbefore) = e                'add e to end of the list
    backlink(e) = putbefore
    nextlink(e) = e
  END IF

END SUB


SUB remove (e)
'*************************************************************************
' PURPOSE  This routine pulls an event from the top, middle or bottom
' of a linked list.  When passed the value e which indicates the event
' to be pulled, the routine zeroes out the event type and calls move
' to pass the event to the inactive file.

' VARIABLES
' e  is an integer value that is the identifier of the event of interest
  eventtype(e) = 0    ' dummy out the event type
  eventactor(e) = 0   ' dummy out the event actor
  timeofevent(e) = e ' set sequence number to the position in the array
'*************************************************************************

  CALL move(e, Firstevent, Firstempty)  'move event e from the pending
                                        'to the empty list
END SUB


SUB setup
'*************************************************************************
' PURPOSE  this routine queries the user about whether he desires to use
' default settings or not in the simulation and allows the user to modify
' certain run parameters if desired.
'*************************************************************************

'these constants determine whether the squad leaders can alter directions
CONST redincmd = 1, rednotincmd = 0
CONST blueincmd = 1, bluenotincmd = 0

again$ = "r"
DO WHILE again$ = "r"

 CALL frame(1, 79, 2, 23, 7, 1)
```

```
 LOCATE 4, 32
 PRINT " SIMULATION SETUP "
 LOCATE 6, 3: PRINT " Default settings are indicated in < >.
 & To modify hit <y> else <return>."
 LOCATE 7, 3: PRINT "================================================
 & ========================"
 LOCATE 8, 3: PRINT " Maximum number of events <5000>  Modify? (y/n)":
 & a$ = INPUT$(1)
 IF (a$ = "y" OR a$ = "Y") THEN LOCATE 8, 60: INPUT "Value "; terminate
 LOCATE 10, 3: PRINT " Maximum number of seconds for run <300>  Modify?
 & (y/n)": b$ = INPUT$(1)
 IF (b$ = "y" OR b$ = "Y") THEN LOCATE 10, 60: INPUT "Value "; timestop
 LOCATE 12, 3: PRINT " Terminate # level of Blue remaining <3>  Modify?
 & (y/n)": c$ = INPUT$(1)
 IF (c$ = "y" OR c$ = "Y") THEN LOCATE 12, 60: INPUT "Value "; bluestop
 LOCATE 14, 3: PRINT " Terminate at # of Red Remaining <4>  Modify?
 & (y/n)": d$ = INPUT$(1)
 IF (d$ = "y" OR d$ = "Y") THEN LOCATE 14, 60: INPUT "Value "; redstop
 LOCATE 16, 3: PRINT " Atmospheric attenuation coef. <1.0>  Modify? (y/n)":
 & e$ = INPUT$(1)
 IF (e$ = "y" OR e$ = "Y") THEN LOCATE 16, 60: INPUT "Value "; attenuation
 LOCATE 18, 3: PRINT " Should blue side respond to sqd ldr <yes> Modify?
 & (y/n)": f$ = INPUT$(1)
 IF (f$ = "y" OR f$ = "Y") THEN LOCATE 18, 60: INPUT " <y> or <n> "; g$:
 & IF (g$ = "n" OR g$ = "N") THEN bluecommand = bluenotincmd
 LOCATE 20, 3: PRINT " Should red side respond to sqd ldr <yes> Modify?
 & (y/n)": h$ = INPUT$(1)
 IF (h$ = "y" OR h$ = "Y") THEN LOCATE 20, 60: INPUT " <y> or <n> "; I$:
 & IF (I$ = "n" OR I$ <> "N") THEN redcommand = rednotincmd

 LOCATE 22, 4: RANDOMIZE  'queries the user to input an initial seed value.

 LOCATE 24, 2: PRINT "If done hit <return> else hit <r> to make changes ":
 & again$ = INPUT$(1)

CLS
LOOP
END SUB


SUB showevents
'*************************************************************************
' PURPOSE  This routine prints a list of the first twelve events on the
' future event calendar and a brief description of each event.
'*************************************************************************

CLS 1
COLOR 7, 1
PRINT "SEQ TYP LST  WHO    TIME <BK-NXT>"
e = Firstevent
ne = 0
I = 0
```

```
WHILE nextlink(e) <> e AND I <= 10
  I = I + 1
  ne = ne + 1
  LOCATE ne + 1, 1
  PRINT USING "##) "; ne;
  PRINT USING "## (##) ## at ###.## <##-##>"; eventtype(e); e;
  & eventactor(e); timeofevent(e); backlink(e); nextlink(e)
  LOCATE ne + 1, 35
IF (eventtype(e) = 1) THEN PRINT USING "Soldier ## will start searching at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 2) THEN PRINT USING "Soldier ## will start moving at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 3) THEN PRINT USING "Soldier ## will stop moving at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 4) THEN PRINT USING "Soldier ## will select a target at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 5) THEN PRINT USING "Soldier ## will fire at soldier ##
& at ####.##"; eventactor(e); movers(eventactor(e), 16); timeofevent(e)

IF (eventtype(e) = 6) THEN PRINT USING "Soldier's ## shot impacts on ## at
& ####.##"; eventactor(e); movers(eventactor(e), 16); timeofevent(e)

IF (eventtype(e) = 7) THEN PRINT USING "Soldier ## reacts to being fired
& on at####.##"; eventactor(e); timeofevent(e)

  e = nextlink(e)
WEND
  ne = ne + 1
  LOCATE ne + 1, 1
  PRINT USING "##) "; ne;
  PRINT USING "## (##) ## at ###.## <##-##>"; eventtype(e); e;
  & eventactor(e); timeofevent(e); backlink(e); nextlink(e)
  LOCATE ne + 1, 35
IF (eventtype(e) = 1) THEN PRINT USING "Soldier ## will start searching at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 2) THEN PRINT USING "Soldier ## will start moving at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 3) THEN PRINT USING "Soldier ## will stop moving at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 4) THEN PRINT USING "Soldier ## will select a target at
& ####.##"; eventactor(e); timeofevent(e)
IF (eventtype(e) = 5) THEN PRINT USING "Soldier ## will fire at soldier ##
& at ####.##"; eventactor(e); movers(eventactor(e), 16); timeofevent(e)
IF (eventtype(e) = 6) THEN PRINT USING "Soldier's ## shot impacts on ## at
& ####.##"; eventactor(e); movers(eventactor(e), 16); timeofevent(e)
IF (eventtype(e) = 7) THEN PRINT USING "Soldier ## reacts to being fired
& on at####.##"; eventactor(e); timeofevent(e)

LOCATE 14, 1
PRINT "**********************************************************************
& **********"
LOCATE 16, 26: PRINT " EVENT CALENDAR DISPLAY"
```

```
LOCATE 18, 1
PRINT " This display shows the next twelve events scheduled to occur in
& chronological"
PRINT " order.  The left side shows the type of event, its position in
& the event list"
PRINT " the event actor, the time of the event, along with the successor
& and "
PRINT " predecessor link pointers.  The right side is a short description
& of the event."
LOCATE 24, 2
CALL pause
END SUB
```

```
'*****************************************************************
'                                                                *
'                        ****** COMBAT MODULE ******              *
'                                                                *
'*****************************************************************

' This block contains the declare statements for the subprograms and
' functions that are in or called by this module.  Hit F2 for menu.

DECLARE SUB icon (who)    ' draws out the graphics of the soldiers
DECLARE SUB pause ()      ' used to allow the user to read screen
DECLARE SUB addevent (itype!, time!, entity) ' add events to calendar
DECLARE SUB endmove ()    ' plots soldier's new location
DECLARE SUB startmove ()  ' provides next location and time to move
DECLARE SUB search ()     'decides which enemy can be seen by observer
DECLARE SUB tgtselect ()  'selects a detected target to engage, if any
DECLARE SUB reacttofire () 'models soldier's action after being engaged
DECLARE SUB directfire ()  'performs firing and computes time of flight
DECLARE SUB impact ()      'determines hits and the results of a hit
DECLARE SUB killsoldier (tgt) 'removes future events of dead soldiers
DECLARE SUB plotterrain () 'plots the elevation on the graphics
DECLARE SUB oput ()        'provides final output at end of simulation
DECLARE SUB soldierstats () 'prints out a list of soldier attributes
DECLARE SUB frame (left%, right%, top%, bottom%, fore%, back%) ' frames
DECLARE SUB battlestats ()  'provides a summary of ongoing battle stats
DECLARE SUB pottgtlist ()   'displays each soldier's target list
DECLARE SUB reorient (who)  'reorients soldier direction on SL's command
DECLARE SUB cmddecision (who) 'changes soldier's directions when called
DECLARE SUB showevents ()   'a window showing next 12 events scheduled
DECLARE SUB hitcount (tgt)  'tracks total hits on each side for oput
DECLARE FUNCTION LOS! (entity1!, entity2!)  'checks LOS from obs to tgt
DECLARE FUNCTION triag (A!, R!, B!) ' provides RV of a triangular dist

' The variables in these COMMON's are globally defined
COMMON SHARED time, movers(), te, eventactor(), elevation(), mobility(),
& p1(), p2(), ph300(), ph400(), ph500(), ph600(), attenuation, activeblue,
& activered
COMMON SHARED bluecommand, redcommand, direct()
COMMON SHARED bluecount, redcount, bluerounds, redrounds

CONST pi = 3.14159
DIM SHARED tgtlist(12, 12) AS SINGLE
DIM SHARED pottgts(6, 2) AS SINGLE
DIM SHARED redhits AS INTEGER, bluehits AS INTEGER
END
```

```
SUB battlestats
'*********************************************************************
'PURPOSE  this routine computes and displays some simple data values
' during the course of the simulation
'*********************************************************************

CLS 1     'all of this block is formatting of a display screen
COLOR 1, 15
CALL frame(2, 78, 2, 23, 1, 15)
LOCATE 3, 30: PRINT "** BATTLE STATUS **"
LOCATE 7, 4: PRINT "Soldiers remaining"
LOCATE 9, 4: PRINT "Soldiers wounded"
LOCATE 11, 4: PRINT "Ammo remaining"

LOCATE 5, 25: PRINT "* BLUE FORCES *"
LOCATE 7, 28: PRINT USING "  ## "; activeblue
LOCATE 9, 28
bluewounded = 0
FOR l = 1 TO 6
IF (movers(l, 12) = 2) THEN bluewounded = bluewounded + 1
NEXT l
PRINT USING "  ##"; bluewounded
FOR I = 1 TO 6
  BLUEAMMO = BLUEAMMO + movers(I, 11)
NEXT I
LOCATE 11, 28
PRINT USING " ###"; BLUEAMMO

COLOR 4, 15
LOCATE 5, 50: PRINT "* RED FORCES *"
LOCATE 7, 53: PRINT USING "  ## "; activered
LOCATE 9, 53
redwounded = 0
FOR l = 7 TO 12
  IF (movers(l, 12) = 2) THEN redwounded = redwounded + 1
NEXT l
PRINT USING "  ##"; redwounded

FOR I = 7 TO 12
  REDAMMO = REDAMMO + movers(I, 11)
NEXT I
LOCATE 11, 53
PRINT USING "  ###"; REDAMMO

COLOR 1, 15
LOCATE 22, 3
PRINT "Do you wish to see event list ? ": k$ = INPUT$(1)
IF (k$ = "y" OR k$ = "Y") THEN CALL showevents
IF (k$ <> "y" AND k$ <> "Y") THEN LOCATE 24, 2: CALL pause

END SUB
```

```
SUB cmddecision (who)
'*********************************************************************
'PURPOSE this routine causes the soldiers to orient on the sqd
'leader's target or return to their original movement direction.
'*********************************************************************

'if the soldier is a sqd ldr and has identified a target then the
'soldiers in his squad will reorient on the enemy otherwise they
'will assume their original directions of movement

IF (movers(who, 14) > 0 AND movers(who, 16) > 0) THEN
  CALL reorient(who)
ELSE

  IF (movers(who, 17 = 1)) THEN firstone = 1: lastone = 6
  IF (movers(who, 17 = -1)) THEN firstone = 7: lastone = 12

  FOR I = firstone TO lastone
    movers(I, 8) = movers(I, 15)
  NEXT I
END IF

END SUB


SUB directfire STATIC
'*********************************************************************
' PURPOSE  This routine models a soldier that has identified a
' potential target to fire at. It  fires at the target and then calls
' the impact subprogram to determine whether the engagement was
' successful. This routine checks to ensure both actors are still alive
' at the engagement time, and line of sight still exists between the two.

' VARIABLES
 who = eventactor(te)       'this is the observer
 x = movers(who, 1)         'these are the coordinates and orientation
 y = movers(who, 2)         'of the observer
 dir = movers(who, 8)
 tgt = movers(who, 16)      'enemy to be engaged
 tgtx = movers(tgt, 1)      'enemy's  coordinates
 tgty = movers(tgt, 2)
 obsstatus = movers(who, 12)  'whether observer is dead or alive
 tgtstatus = movers(tgt, 12)  'whether the target is dead or alive
'*********************************************************************

'this check ensures that the target is alive
IF (tgtstatus > 0) THEN

 IF (LOS(who, tgt) = 1 AND movers(who, 11) > 0) THEN
   'Compute absolute distance between observer and target
    dist = SQR((obsx - tgtx) ^ 2 + (obsy - tgty) ^ 2)
```

```
   'Compute time of flight and impact time
   'this formula assumes a constant bullet velocity of 500 meters/sec
    impacttime = time + dist / 500!
    'this function draws out firing occurrences.
    LINE (x, y)-(tgtx, tgty), 4
    PLAY "MF00L64T64N45L32N12"  'this function creates the firing sound
                                'and delays the draw function long
                                'enough to observe it.
    LINE (x, y)-(tgtx, tgty), 0

    CALL addevent(6, impacttime, who)  'schedules impact of the bullet
    movers(who, 11) = movers(who, 11) - 1  'decrement ammo supply by one
 ELSE
    CALL addevent(1, time + triag(2!, 5!, 7!), who)
    'shooter looks for new target
 END IF
ELSE
    CALL addevent(1, time + triag(2!, 5!, 7!), who)
    'shooter looks for new target
END IF

END SUB



SUB endmove
'*************************************************************************
' PURPOSE The purpose of this routine is to determine the time for the
' beginning of the next movement.
' *************************************************************************

' VARIABLES
  who = eventactor(te)     ' who is a buffer for soldier's identity

  'this block computes a time for the next move to begin
  'and schedules a startmove to occur at that time.

  movetime = time + triag(8!, 10!, 12!)
  CALL addevent(2, movetime, who)

  'the duration of the time between moves was selected arbitrarily

END SUB
```

```
SUB hitcount (tgt)
'*************************************************************************
'PURPOSE  this routine counts the number of hits scored on each side
'for final output in the subprogram oput

'VARIABLES
' bluehit  is a counter for the number of hits on blue
' redhit   is a counter for the number of hits on red
'*************************************************************************

 IF (movers(tgt, 17) = 1) THEN bluehits = bluehits + 1
 IF (movers(tgt, 17) = -1) THEN redhits = redhits + 1
END SUB

SUB icon (who)
'*************************************************************************
' PURPOSE  This routine checks to see if the soldier is moving. If so,
' then it draws the soldier's symbol at the new location and overdraws
' the old location with the background color.

' VARIABLES
' who                       ' the soldier being drawn
x = movers(who, 1)          ' the new x,y coordinates for the soldier
y = movers(who, 2)
xlast = movers(who, 4)      ' the soldier's old x,y coordinates
ylast = movers(who, 5)
dir = movers(who, 8)        ' the orientation/movement direction
                            ' of the soldier
side = movers(who, 17)      ' the soldier's allegiance (1) blue (-1) red
'*************************************************************************

'this block determines the correct color for the icon
CONST blueicon = 9, redicon = 4
   IF (side > 0) THEN
      iconcolor = blueicon
   ELSE
      iconcolor = redicon
   END IF

'this block determines which location to be drawing in
IF (movers(who, 12) = 1) THEN

   'overdraw the soldier's old symbol with the background color
   CIRCLE (xlast - 3.6, ylast), 2, 0
   PSET (xlast, ylast), 0
   LINE -STEP(-10, -10), 0
   LINE -STEP(-10, 10), 0
   LINE -STEP(10, 10), 0
   LINE -STEP(10, -10), 0
```

```
   'draw the soldier's new symbol at the new location
   CIRCLE (x - 3.6, y), 2, iconcolor
   PSET (x, y), iconcolor
   LINE -STEP(-10, -10), iconcolor
   LINE -STEP(-10, 10), iconcolor
   LINE -STEP(10, 10), iconcolor
   LINE -STEP(10, -10), iconcolor

ELSE
   ' this block redraws the soldier's icon if he has not moved
   ' CIRCLE (xlast - 3.6, ylast), 2, iconcolor
   ' PSET (xlast, ylast), iconcolor
   ' LINE -STEP(-8, -8), iconcolor
   ' LINE -STEP(-8, 8), iconcolor
   ' LINE -STEP(8, 8), iconcolor
   ' LINE -STEP(8, -8), iconcolor

END IF

END SUB



SUB impact
'*************************************************************************
' PURPOSE  this routine determines if the bullet hits the target and
' whether it is wounded or killed.  The routine then modifies the
' targets attributes and calls the proper future events for
' both the firer and target based on the results

' VARIABLES
 who = eventactor(te)       'this is the observer
 x = movers(who, 1)         'these are the coordinates and orientation
 y = movers(who, 2)         'of the observer
 dir = movers(who, 8)
 tgt = movers(who, 16)       'enemy soldier being shot at
 xtgt = movers(tgt, 1)       'enemy's  present coordinates
 ytgt = movers(tgt, 2)
 xlasttgt = movers(tgt, 4)  'enemy's last coordinates
 ylasttgt = movers(tgt, 5)
 side = movers(tgt, 17)      'designates the allegiance of the target
 'activeblue                 indicates number of remaining blue soldiers
 'activered                  indicates number of remaining red soldiers
'*************************************************************************

 'this section computes absolute distance between observer and target
  dist = SQR((obsx - tgtx) ^ 2 + (obsy - tgty) ^ 2)

 'this section determines posture column to look under in p1 table
  IF (movers(tgt, 13) > .9) THEN tgtposture = 3
  IF (movers(tgt, 13) < .9 AND movers(tgt, 13) > .4) THEN tgtposture = 2
  IF (movers(tgt, 13) < .4) THEN tgtposture = 1
```

```
'this section determine p(hit) based on range and target posture

    IF (dist > 999) THEN dist = 899    'limits possible ranges to 1000m
                                       'since the tables are limited

'determines the correct p(hit) table depending on the soldier's
'weapon range.

IF (movers(who, 10) < 399) THEN
   phit = ph300(INT(dist / 100) + 1, tgtposture)
ELSEIF (movers(who, 10) < 499) THEN
   phit = ph400(INT(dist / 100) + 1, tgtposture)
ELSEIF (movers(who, 10) < 499) THEN
   phit = ph500(INT(dist / 100) + 1, tgtposture)
ELSE
   phit = ph600(INT(dist / 100) + 1, tgtposture)
END IF


IF (RND < phit) THEN

  CALL hitcount(tgt)

  'Determine probability of Kill
  IF (RND < .3) THEN     'arbitrary 30% chance of death

  'this section graphically portrays the bullet's impact
  'both explosion graphics and noise are created momentarily then erased
  'the basic idea for this came fr-    Microsoft DOS 5.0 demonstration
  'program entitled "GORILLA.BAS"

     FOR I = 1 TO 4
       PSET (xtgt - 3.6, ytgt), 14
       CIRCLE (xtgt - 3.6, ytgt), 1 + I, 14
       LINE (xtgt - 3.6, ytgt)-(xtgt - 3.6 + I * COS(4 * I / 3.14),
       & ytgt + I * SIN(4 * I / 3.14)), 4
       PLAY "MFL32OOEFGDFG"
     NEXT I
     FOR I = 1 TO 4
       PSET (xtgt - 3.6, ytgt), 0
       CIRCLE (xtgt - 3.6, ytgt), 1 + I, 0
       LINE (xtgt - 3.6, ytgt)-(xtgt - 3.6 + I * COS(4 * I / 3.14),
       & ytgt + I * SIN(4 * I / 3.14)), 0
     NEXT I

    ' set status to dead   posture to prone     movement to stopped
    movers(tgt, 12) = 0: movers(tgt, 13) = .25: movers(tgt, 9) = 0

    ' schedule the firer to resume searching
    CALL addevent(1, time + triag(2!, 5!, 7!), who)
```

```
    'overdraw the soldier's old symbol with the background color
    CIRCLE (xlasttgt - 3.6, ylasttgt), 2, 0
    PSET (xlasttgt, ylasttgt), 0
    LINE -STEP(-10, -10), 0
    LINE -STEP(-10, 10), 0
    LINE -STEP(10, 10), 0
    LINE -STEP(10, -10), 0

 CONST deadicon = 7  ' this set the color for a dead icon to lt gray

    'draw the dead soldier's symbol at the present location
    CIRCLE (xtgt - 3.6, ytgt), 2, deadicon
    PSET (xtgt, ytgt), deadicon
    LINE -STEP(-10, -10), deadicon
    LINE -STEP(-10, 10), deadicon
    LINE -STEP(10, 10), deadicon
    LINE -STEP(10, -10), deadicon

    CALL killsoldier(tgt)  'removes dead soldiers active events

    'this section updates the number of active soldiers on each side
    'this is used in battlestats and as termination conditions in the
    'main module

    IF (side > 0) THEN
      activeblue = 0
      FOR I = 1 TO 6
        alive = 0
        IF (movers(I, 12) > 0) THEN alive = 1
        activeblue = activeblue + alive
      NEXT I
    ELSE
      activered = 0
      FOR I = 7 TO 12
        alive = 0
        IF (movers(I, 12) > 0) THEN alive = 1
        activered = activered + alive
      NEXT I
    END IF

'this section determines whether the dead soldier is a sqd leader. If so
'then all directions of movement are returned to original orientation.
    IF (tgt < 7 AND movers(tgt, 14) > 0) THEN
       FOR I = 1 TO 6
         movers(I, 8) = movers(I, 15)
       NEXT I
    ELSEIF (movers(tgt, 14) > 0) THEN
       FOR I = 7 TO 12
         movers(I, 8) = movers(I, 15)
       NEXT I
    ELSE
    END IF
```

```
  ELSE   'the target is only wounded and not killed

  'this section graphically portrays the bullet's impact
     FOR I = 1 TO 3
       PSET (xtgt - 2.83, ytgt), 14
       CIRCLE (xtgt - 2.83, ytgt), 1 + I, 4
       LINE (xtgt - 2.83, ytgt)-(xtgt - 2.83 + I * COS(4 * I / 3.14),
       & ytgt + I * SIN(4 * I / 3.14)), 0
       PLAY "MFL64O0EFGDFG"
     NEXT I
     FOR I = 1 TO 3
       PSET (xtgt - 2.83, ytgt), 0
       CIRCLE (xtgt - 2.83, ytgt), 1 + I, 0
       LINE (xtgt - 2.83, ytgt)-(xtgt - 2.83 + I * COS(4 * I / 3.14),
       & ytgt + I * SIN(4 * I / 3.14)), 0
     NEXT I

    ' set status to wounded   set movement to 0
     movers(tgt, 12) = 2:      movers(tgt, 9) = 0

     'schedule firer to reengage present target
     CALL addevent(5, time + triag(4!, 5!, 8!), who)

     'schedule target to react after being fired at
     CALL addevent(7, time + 2!, tgt)
  END IF
ELSE  'shot misses the target
 PLAY "MFL64T240o0b"
 'firer is scheduled to reengage the target after a short time required
 'to reload and reaim the weapon
 CALL addevent(5, time + triag(4!, 5!, 8!), who)

 ' target is scheduled to react after being shot at
 CALL addevent(7, time + 2!, tgt)
END IF
END SUB
```

```
FUNCTION LOS (entity1, entity2) STATIC
'*****************************************************************
' PURPOSE  This function when given an observer and target combination
' (entity1,entity2), returns a value of 1 (LOS exists) or 0 (LOS does
' not exist). It checks at 10 meter intervals whether the height at that
' point intersects LOS. Observer and target height are elevation + 2 m.

' VARIABLES
' obs and tgt    identify the entities for the LOS check
' elevation()     a 50x50 array that contains elevations of each grid
' NumberofChecks  an integer variable that is the possible # of checks
' LOSdist  a real variable that is the distance between obs and tgt
' LOSdir   a real variable that is the direction from obs to tgt (rdn)
' CheckX, CheckY, CheckZ    real variable values that represent the
' coordinates of the points being checked along the LOS path
' LOScheck   a buffer that holds the true value of LOS until the end of
' the function otherwise any references to LOS are assumed to be
' recursive calls by the compiler.
'*****************************************************************

obs = entity1: tgt = entity2

'Read in location coordinates of observer and target entities
obsx! = movers(obs, 1): obsy! = movers(obs, 2)
obsz! = movers(obs, 3) + movers(obs, 6)
tgtx! = movers(tgt, 1): tgty! = movers(tgt, 2)
tgtz! = movers(tgt, 3) + movers(tgt, 6)

'Compute absolute distance between observer and target
LOSdist! = SQR((obsx! - tgtx!) ^ 2 + (obsy! - tgty!) ^ 2)


IF (movers(obs, 17) <> movers(tgt, 17) AND LOSdist! <= 100 AND
& movers(tgt, 12) > 0) THEN movers(obs, 7) = 0: movers(obs, 13) = .5

dx! = 10 * (tgtx! - obsx!) / LOSdist!
dy! = 10 * (tgty! - obsy!) / LOSdist!
dz! = 10 * (tgtz! - obsz!) / LOSdist!

LOS = 1  'Initially set LOS to 1 until check determines otherwise
LOScheck = 1  'this buffer stores LOS value until end of function

'Determine number of checks need to be made along LOS path.
NumberofChecks = INT(LOSdist! / 10)
check = 0  'Initially set counter to 0
CheckX! = obsx!: CheckY! = obsy!:  ' designating observer location
                                   ' as the start point for LOS checks
CheckZ! = obsz!

' This loop will compare the intervening terrain elevation at 20 meter
' intervals with the LOS height (CheckZ)
```

```
DO    ' this loop will continue as long as LOS exists or until all
      ' intermediate points at 20m intervals are checked
   check = check + 1                    'keeps track of # of checks
   CheckX! = CheckX! + dx!              'CheckX is X coord for next check
   CheckY! = CheckY! + dy!              'CheckY is Y coord for next check

   'CheckZ is the elevation of the LOS at the check location

   CheckZ! = CheckZ! + dz!

   ' This check compares the height of CheckZ with the terrain height
   ' at the check location.  The checks continue until the terrain is
   ' at some check is high enough to break line of sight

   IF (CheckX! > 1000) THEN CheckX! = 999  ' this block prevents array
   IF (CheckY! > 1000) THEN CheckY! = 999  ' subscripts from exceeding
   IF (CheckX! < 0) THEN CheckX! = 0       ' their limits
   IF (CheckY! < 0) THEN CheckY! = 0

   IF (elevation(FIX(CheckX! / 20!) + 1, FIX(CheckY! / 20!) + 1) > CheckZ!)
   & THEN LOScheck = 0

   LOS = LOScheck  'If LOS does not exist then set LOS = 0

LOOP WHILE LOScheck = 1 AND check < NumberofChecks - 1

END FUNCTION


SUB oput
'*********************************************************************
' PURPOSE  This routine provides several screens of "end of battle"
' values.  The first screen provides some aggregated values for each
' side.  The next screen displays selected from the soldier attribute
' arrays, and the last screen shows the target list at end of battle.
'*********************************************************************

LOCATE 1, 1
PRINT USING "   END OF SIMULATION RUN   THE TIME IS ####.##.....
            "; time
PLAY "MFO0L16DEFGP0CCAA"
LOCATE 25, 2: INPUT " Do you wish to see final output <y> or <n> ", k$
IF (k$ = "y" OR k$ = "Y") THEN

 CLS
 COLOR 1, 15
 CALL frame(2, 78, 2, 23, 1, 15)
 LOCATE 3, 30: PRINT "** BATTLE STATUS **"
 LOCATE 7, 4: PRINT "Initial # of soldiers"
 LOCATE 9, 4: PRINT "Soldiers remaining"
 LOCATE 11, 4: PRINT "Soldiers wounded"
 LOCATE 13, 4: PRINT "Initial ammo"
```

```
LOCATE 15, 4: PRINT "Ammo remaining"
LOCATE 17, 4: PRINT "# of rounds fired"
LOCATE 19, 4: PRINT "# of hits on side"

LOCATE 5, 25: PRINT "* BLUE FORCES *"
LOCATE 7, 28: PRINT USING "  ## "; bluecount
LOCATE 9, 28: PRINT USING "  ## "; activeblue
LOCATE 11, 28
bluewounded = 0
FOR l = 1 TO 6
 IF (movers(l, 12) = 2) THEN bluewounded = bluewounded + 1
NEXT l
PRINT USING "  ##"; bluewounded
FOR I = 1 TO 6
  BLUEAMMO = BLUEAMMO + movers(I, 11)
NEXT I
LOCATE 13, 28: PRINT USING " ###"; bluerounds
LOCATE 15, 28: PRINT USING " ###"; BLUEAMMO
LOCATE 17, 28: PRINT USING " ###"; bluerounds - BLUEAMMO
LOCATE 19, 28: PRINT USING " ###"; bluehits

COLOR 4, 15
LOCATE 5, 50: PRINT "* RED FORCES *"
LOCATE 7, 53: PRINT USING "  ## "; redcount
LOCATE 9, 53: PRINT USING "  ## "; activered
LOCATE 11, 53
redwounded = 0
FOR l = 7 TO 12
 IF (movers(l, 12) = 2) THEN redwounded = redwounded + 1
NEXT l
PRINT USING "  ##"; redwounded

FOR I = 7 TO 12
 REDAMMO = REDAMMO + movers(I, 11)
NEXT I
LOCATE 13, 53: PRINT USING " ###"; redrounds
LOCATE 15, 53: PRINT USING " ###"; REDAMMO
LOCATE 17, 53: PRINT USING " ###"; redrounds - REDAMMO
LOCATE 19, 53: PRINT USING " ###"; redhits
LOCATE 24, 2
CALL pause

COLOR 1, 15: CLS
LOCATE 2, 2: PRINT "Selected values from the present soldier attribute
& files."
LOCATE 4, 2: PRINT "   x   /   y   / speed / moving / ammo / status /
& posture / tgteng"
FOR I = 1 TO 12
   PRINT USING " ####.#  ####.#    ##.#     ##      ##     ##       #.##
   &  ## "; movers(I, 1); movers(I, 2); movers(I, 7); movers(I, 9);
   & movers(I, 11); movers(I, 12); movers(I, 13); movers(I, 16)
NEXT I
```

```
 LOCATE 24, 2
 CALL pause
 CLS
 PRINT "              Potential Targets Array": PRINT
 PRINT "                        Targets "
 PRINT "  Observer 1   2   3   4   5   6   7   8   9  10  11  12"
 FOR I = 1 TO 12
   PRINT USING "   ##  - .## .## .## .## .## .## .## .## .## .## .## ";
   & I; tgtlist(I, 1); tgtlist(I, 2); tgtlist(I, 3); tgtlist(I, 4);
   & tgtlist(I, 5); tgtlist(I, 6); tgtlist(I, 7); tgtlist(I, 8);
   & tgtlist(I, 9); tgtlist(I, 10); tgtlist(I, 11); tgtlist(I, 12)
 NEXT I
 LOCATE 24, 2
 CALL pause
 CLS
 ELSE
 END IF
END SUB


SUB plotterrain
'************************************************************************
' PURPOSE  This routine places a color patch in each grid that
' identifies the elevation for that particular grid.
'************************************************************************
FOR I = 1 TO 50
  FOR J = 1 TO 50
    xspot = I * 20! - 10!
    yspot = J * 20! - 10!
    elev = elevation(I, J)
    IF (elev < 10 OR elev = 10) THEN       'color is light green
       spot = 10
    ELSEIF (elev < 20 OR elev = 20) THEN   'color is light cyan
       spot = 11
    ELSEIF (elev < 30 OR elev = 30) THEN   'color is light red
       spot = 12
    ELSEIF (elev < 40 OR elev = 40) THEN   'color is light magenta
       spot = 13
    ELSEIF (elev < 50 OR elev = 50) THEN   'color is light yellow
       spot = 14
    ELSEIF (elev < 60 OR elev = 60) THEN   'color is bright grey
       spot = 15
    ELSE spot = 6                          'color is brown
    END IF

    PSET (xspot, yspot), spot
  NEXT J
NEXT I

END SUB
```

```
SUB pottgtlist
'*********************************************************************
'PURPOSE  prints out a listing of the soldiers' target listings

CLS 1
COLOR 7, 1
CALL frame(2, 78, 2, 23, 7, 1)
LOCATE 5, 18: PRINT "             Potential Targets Array"
LOCATE 7, 18: PRINT "                      Targets "

LOCATE 8, 11: PRINT "  Observer 1   2   3   4   5   6   7   8   9  10  11
& 12"
FOR I = 1 TO 12
  LOCATE 8 + I, 11
  PRINT USING "   ##  - .## .## .## .## .## .## .## .## .## .## .## .## ";
  & I; tgtlist(I, 1); tgtlist(I, 2); tgtlist(I,3); tgtlist(I, 4);
  & tgtlist(I, 5); tgtlist(I, 6); tgtlist(I, 7); tgtlist(I, 8);
  & tgtlist(I, 9); tgtlist(I, 10); tgtlist(I,11); tgtlist(I, 12)
NEXT I

END SUB


SUB reacttofire
'*********************************************************************
' PURPOSE  This routine integrates a small portion of decision logic
' into the model.  After being shot at a soldier can take several
' courses of action which are basically random in this case.
'*********************************************************************

reaction = RND   ' value is used to determine the soldier's reaction

who = eventactor(te)    'the soldier's reaction maybe to fall to the
IF (reaction < .4) THEN 'ground and continue searching or to run away
                        'or even possibly to madly rush forward

 movers(who, 7) = 5: movers(who, 13) = .25
ELSEIF (reaction < .6) THEN
 movers(who, 8) = movers(who, 8) + 3.14
ELSE
 movers(who, 7) = 30: movers(who, 13) = 1
END IF

END SUB
```

```
SUB reorient (who) STATIC
'*************************************************************************
' PURPOSE  this routine will change the direction of a subordinate
' soldier's movement when it is called.  The soldiers will orient
' on the enemy soldier targeted by the squad leader.

' VARIABLES
tgt = movers(who, 16)
tgtx = movers(tgt, 1)
tgty = movers(tgt, 2)
'*************************************************************************

IF (who < 7) THEN
  FOR I = 1 TO 6
    IF (movers(I, 12) > 0) THEN   'if the soldier is alive
      IF (movers(I, 1) < tgtx AND movers(I, 2) < tgty) THEN
        movers(I, 8) = .8
      ELSEIF (movers(I, 1) < tgtx AND movers(I, 2) > tgty) THEN
        movers(I, 8) = 2.1
      ELSEIF (movers(I, 1) > tgtx AND movers(I, 2) > tgty) THEN
        movers(I, 8) = 3.9
      ELSE
        movers(I, 8) = 5.2
      END IF

    ELSE
    END IF
  NEXT I

ELSE
  FOR I = 7 TO 12
    IF (movers(I, 12) > 0) THEN 'if the soldier is alive
      IF (movers(I, 1) < tgtx AND movers(I, 2) < tgty) THEN
        movers(I, 8) = .6: movers(I, 7) = 40
      ELSEIF (movers(I, 1) < tgtx AND movers(I, 2) > tgty) THEN
        movers(I, 8) = 2.4: movers(I, 7) = 40
      ELSEIF (movers(I, 1) > tgtx AND movers(I, 2) > tgty) THEN
        movers(I, 8) = 3.8: movers(I, 7) = 40
      ELSE
        movers(I, 8) = 5.3: movers(I, 7) = 40
      END IF

    ELSE
    END IF

  NEXT I
END IF

END SUB
```

```
SUB search
'*************************************************************************
' PURPOSE  This routine will check all live enemy soldiers to see which
' are within range, and which have LOS with observer.  The routine then
' checks to see which targets can be acquired by the observer(p1) and of
' these that are acquired which are detected using probability (p2).
' The routine then places the candidates prob of det on the potential
' tgt list, and calls the target selection routine to decide which target
' may be engaged.

'VARIABLES
who = eventactor(te)       'this is the observer
x = movers(who, 1)         'these are the coordinates and orientation
y = movers(who, 2)         'of the observer
dir = movers(who, 8)
speed = movers(who, 7)     'observer's rate of travel
xlast = movers(who, 4)     'observer's last coordinates
ylast = movers(who, 5)
status = movers(who, 12)   'whether observer is dead or alive
side = movers(who, 17)     'whether the observer is a blue or red soldier
'*************************************************************************

Pthreshold = .3 / attenuation ' the minimum threshold level required
                              'to acquire a target. It has been set
                              'arbitrarily.  Attenuation is a parameter
                              'that can be set by the user to represent
                              'reduced visibility.  Its range of values
                              'should be (0.31 - 1.0). Less than .3 makes
                              'it impossible to acquire a target at any
                              'range.
nodetect = 1!
'This loop determines the side of the soldier and then performs a
'search routine on each of the possible enemy soldiers.
FOR I = 1 TO 12
  tgtlist(who, I) = 0 ' ensures the list only contains current entries.
  IF (movers(who, 17) <> movers(I, 17)) THEN  'so we only check enemy.
   lineofsight = LOS(who, I)     'calls for LOF check on each enemy

   IF (lineofsight = 1) THEN     'the search process continues only if
                                 'line of sight exists

    'distance between observer and target is computed.

    dist = SQR((movers(I, 1) - x) ^ 2 + (movers(I, 2) - y) ^ 2)

    'this section determines posture column to look under in p1 table
    IF (movers(I, 13) > .9) THEN tgtposture = 3
    IF (movers(I, 13) < .9 AND movers(I, 13) > .4) THEN tgtposture = 2
    IF (movers(I, 13) < .4) THEN tgtposture = 1

    IF (dist > 999) THEN dist = 899   'limits possible ranges to 1000m
                                      'since the tables are limited
```

```
    Pinf = p1(INT(dist / 100) + 1, tgtposture)  'pulls prob of acquistion
    IF (Pinf > Pthreshold) THEN                 'from table
       lo = .4: mode = 2: hi = 4!          'a search time is computed
       srchtime = triag(lo, mode, hi)      'using triangular distribution

       'a U(0,1) random # is compared to probability of detection(p2)
       IF (RND < p2(INT(dist / 100) + 1, INT(srchtime * 2.5) + 1)) THEN

    'this is a line draw function that displays detection occurences.
    LINE (movers(who, 1), movers(who, 2))-(movers(I, 1), movers(I, 2)), 1
    PLAY "p24"   'these functions produce the popping sound
    PLAY "p8"    'and delay the draw function long enough to observe it.
    LINE (movers(who, 1), movers(who, 2))-(movers(I, 1), movers(I, 2)), 0
    'this line puts the P2 value of each target in the observer's
    'target list for use by the target selection subprogram
    tgtlist(who, I) = p2(INT(dist / 100) + 1, INT(srchtime * 2.5) + 1)
    nodetect = 0!
        ELSE
        END IF
      ELSE
      END IF
     ELSE
     END IF
  ELSE
  END IF
NEXT I
IF (nodetect < 1!) THEN
 decisiontime = time + 5!     ' the decision time is a fixed 5 secs.
                              ' This was an arbitrary decision that was
                              ' intended to simplify the program.

 CALL addevent(4, decisiontime, who) 'schedules the tgtselect routine.
ELSE
 searchtime = time + triag(5!, 10!, 15!)
 CALL addevent(1, searchtime, who)
END IF

END SUB
```

```
SUB soldierstats
'************************************************************************
' PURPOSE  this routine displays current soldier attributes when called
' by the user throughout the simulation run.  It displays the blue force
' statistics in blue and red force in red.  The screen is presently full,
' so if different attributes are desired, it will be necessary to remove
' some that are on the screen.
'************************************************************************

CLS 1
COLOR 1, 15
CALL frame(2, 78, 2, 23, 1, 15)
LOCATE 3, 30: PRINT " COMBATANT STATUS"

LOCATE 4, 4
PRINT "Soldier #"
LOCATE 5, 4
PRINT "X Coord "
LOCATE 6, 4
PRINT "Y Coord "
LOCATE 7, 4
PRINT "Status "
LOCATE 8, 4
PRINT "Posture "
LOCATE 9, 4
PRINT "Moving "
LOCATE 10, 4
PRINT "Ammo (rnds) "
LOCATE 11, 4
PRINT "Direction"
LOCATE 12, 4
PRINT "Sqd Leader"

FOR I = 1 TO 6                  'blue force are soldiers 1 - 6
  LOCATE 4, I * 10 + 11
  PRINT USING " # "; I
  LOCATE 5, I * 10 + 9
  PRINT USING "####.# "; movers(I, 1)
  LOCATE 6, I * 10 + 9
  PRINT USING "####.# "; movers(I, 2)
  LOCATE 7, I * 10 + 9
  IF (movers(I, 12) = 0) THEN
    PRINT "Dead"
  ELSEIF (movers(I, 12) = 1) THEN
    PRINT "Alive"
  ELSE
    PRINT "Wounded"
  END IF

  LOCATE 8, I * 10 + 9
  IF (movers(I, 13) > .9) THEN
    PRINT "Standing"
```

```
    ELSEIF (movers(I, 13) < .9 AND movers(I, 13) > .5) THEN
      PRINT "Crouch"
    ELSE
      PRINT "Prone"
    END IF

    LOCATE 9, I * 10 + 9
    IF (movers(I, 9) = 1) THEN
      PRINT "Moving"
    ELSE
      PRINT "Stopped"
    END IF

    LOCATE 10, I * 10 + 9
    PRINT USING " ## "; movers(I, 11)
    LOCATE 11, I * 10 + 9
    PRINT USING "##.# "; movers(I, 8)
    LOCATE 12, I * 10 + 9
    IF (movers(I, 14) = 1) THEN PRINT " SL"
NEXT I

COLOR 4, 15                'red force consists of soldiers 7 - 12

LOCATE 14, 4
PRINT "Soldier #"
LOCATE 15, 4
PRINT "X Coord "
LOCATE 16, 4
PRINT "Y Coord "
LOCATE 17, 4
PRINT "Status "
LOCATE 18, 4
PRINT "Posture "
LOCATE 19, 4
PRINT "Moving "
LOCATE 20, 4
PRINT "Ammo (rnds) "
LOCATE 21, 4
PRINT "Direction"
LOCATE 22, 4
PRINT "Sqd Leader"

FOR J = 1 TO 6
   LOCATE 14, J * 10 + 11
   PRINT USING "##"; J + 6
   LOCATE 15, J * 10 + 9
   PRINT USING "####.# "; movers(J + 6, 1)
   LOCATE 16, J * 10 + 9
   PRINT USING "####.# "; movers(J + 6, 2)
   LOCATE 17, J * 10 + 9
   IF (movers(J + 6, 12) = 0) THEN
    PRINT "Dead"
```

```
    ELSEIF (movers(J + 6, 12) = 1) THEN
     PRINT "Alive"
    ELSE
     PRINT "Wounded"
    END IF

    LOCATE 18, J * 10 + 9
    IF (movers(J + 6, 13) > .9) THEN
     PRINT "Standing"
    ELSEIF (movers(J + 6, 13) < .9 AND movers(I, 13) > .5) THEN
     PRINT "Crouch"
    ELSE
     PRINT "Prone"
    END IF

    LOCATE 19, J * 10 + 9
    IF (movers(J + 6, 9) = 1) THEN
     PRINT "Moving"
    ELSE
     PRINT "Stopped"
    END IF

    LOCATE 20, J * 10 + 9
    PRINT USING " ## "; movers(J + 6, 11)
    LOCATE 21, J * 10 + 9
    PRINT USING "##.# "; movers(J + 6, 8)
    LOCATE 22, J * 10 + 9
    IF (movers(J + 6, 14) = 1) THEN PRINT "  SL"
NEXT J

COLOR 1, 15
LOCATE 24, 2: CALL pause

END SUB
```

```
SUB startmove
'*************************************************************************
' PURPOSE This routine accesses data on the soldier and the terrain at
' his location. It checks the status of the soldier and determines the
' soldier's next location and how long it takes to move to that location.
' The routine stores the present location in (xlast,ylast) and calls the
' endmove routine to occur at time + delta T.  The last step is a call to
' icon subprogram which redraws a soldier's symbol at his new location.

' VARIABLES
who = eventactor(te)
x = movers(who, 1)
y = movers(who, 2)
dir = movers(who, 3)
speed = movers(who, 7)
'*************************************************************************

IF (x > 1000) THEN x = 999        'This conditional block keeps the array
IF (x < 0) THEN x = 0             'superscripts called within the 0-1000
IF (y > 1000) THEN y = 999        'range  (should be temporary)
IF (y < 0) THEN y = 0

mobfac = mobility(FIX(x / 20) + 1, FIX(y / 20) + 1)

IF (x > 9991) THEN movers(who, 9) = 0
IF (x < 11) THEN movers(who, 9) = 0
IF (y > 9991) THEN movers(who, 9) = 0
IF (y < 11) THEN movers(who, 9) = 0

' This section adds an endmove event with new location to the event
' list.
' Store present location in the previous location positions
movers(who, 4) = x: movers(who, 5) = y

IF (movers(who, 9) = 1 AND movers(who, 7) > 0) THEN  'ensure soldier is
x = x + 20 * SIN(dir)  'compute new x,y coordinates  'moving (1 is yes)
y = y + 20 * COS(dir)

' computes the travel time and considers the soldier's speed, posture, and
' the mobility factor in the grid he started in.
stopmove = time + (100 * RND) / (movers(who, 7) * mobfac * movers(who, 13))
CALL addevent(3, stopmove, who)  'calls the    move event for this move.
ELSE
'calls an endmove for the soldier even if he is not presently moving, so
'if he is allowed to move again, he will still be in the movement process
stopmove = time + (100 * RND) / (1 + (movers(who, 7) * mobfac *
& movers(who, 13)))
CALL addevent(3, stopmove, who)  'calls the endmove event for this move.

END IF
```

```
' here we update the location records to the new values
movers(who, 1) = x: movers(who, 2) = y

IF (x > 1000) THEN x = 999    ' this conditional block is designed to keep
IF (x < 0) THEN x = 0         ' the subscripts of the array calls within the
IF (y > 1000) THEN y = 999    ' limits of the array  (should be temporary)
IF (y < 0) THEN y = 0

'updates the present elevation of the soldier
movers(who, 3) = elevation(FIX(x / 20) + 1, FIX(y / 20) + 1)

  CALL icon(who)  ' This subprogram draws the soldier's symbol at the
                  ' new location and overdraws the symbol at the old
                  ' location with the background color so that it is
                  ' erased and the soldier appears to be moving.

END SUB



SUB tgtselect
'*************************************************************************
' PURPOSE  This routine calls up the list of targets observered by the
' observer and determines which one the observer will engage if any. The
' detected targets are in the tgtlist array input by the search routine.

'VARIABLES
who = eventactor(te)        'this is the observer
'I                          'an index that indicates the target IDs
'tgtlist(who,I)             'this vector from the tgtlist array contains
                            'the P(det) for each target
                            'that has been detected by this observer.
' N  is the number of potential targets detected
' pottgts() is an array containing the P(det)s and an index number
'*************************************************************************

' we sum the values in the vector to get a total and to check whether
' there are current values in the array.

'resets values for each soldier when the selection cycle starts
'and ensures only current values are used
total = 0: N = 0: movers(who, 16) = 0

FOR I = 1 TO 12
   total = total + tgtlist(who, I)
   IF (tgtlist(who, I) > 0) THEN
     N = N + 1
     pottgts(N, 1) = I: pottgts(N, 2) = tgtlist(who, I)
   ELSE
   END IF
NEXT I
```

```
'this section simulates the observers decision of whether to engage
'any of the targets
IF (total < .2) THEN
  CALL addevent(1, time + triag(2!, 3!, 4!), who) 'schedules next search

                    'if the total is too small assume the soldier will
                    'elect to not fire at this time. Also protects
ELSE                'against dividing by zero
  FOR I = 1 TO N                                  'this loop standardizes the
    pottgts(I, 2) = pottgts(I, 2) / total         'values in the array, so
  NEXT I                                          'they sum to one.
  R = RND: runningsum = 0: N = 0
  DO WHILE (R > runningsum)                       'using a U(0,1) random
    N = N + 1                                     'variate we choose the
    runningsum = runningsum + pottgts(N, 2)       'target to be shot at
    IF (R < runningsum) THEN
     movers(who, 16) = pottgts(N, 1)
     tgt = pottgts(N, 1)
     'determines distance between the observer and target
     dist = SQR((movers(who, 1) - movers(tgt, 1)) ^ 2 + (movers(who, 2)
     & - movers(tgt, 2)) ^ 2)
     IF (dist < movers(who, 10)) THEN
      aimtime = time + triag(2!, 5!, 7!)    'determines aim time til shot
      CALL addevent(5, aimtime, who)        'is fired and calls a direct
     ELSE                                   'fire engagement
      CALL addevent(1, time + triag(2!, 5!, 7!), who) 'decides target is
     END IF                                                'out of range
    ELSE
    END IF
  LOOP
END IF

IF (who < 7 AND bluecommand > 0 AND movers(who, 14) > 0 AND
& movers(who, 16) > 0) THEN CALL cmddecision(who)
IF (who > 6 AND redcommand > 0 AND movers(who, 14) > 0 AND
& movers(who, 16) > 0) THEN CALL cmddecision(who)

END SUB
```

```
'*************************************************************************
'*                                                                       *
'*                          UTILITY.BAS                                  *
'*                                                                       *
'*************************************************************************

'This module contains subprograms and functions that do not require
'access to any data stored elsewhere in the model.  The routines in
'utility can be called by all other routines in SPARTAN.


DECLARE FUNCTION triag! (A!, D!, b!) 'provides a triangular random value
DECLARE SUB opening ()   'provides a presentation screen
DECLARE SUB frame (left%, right%, top%, bottom%, fore%, back%)
DECLARE SUB pause () 'provides a utility to pause the scrolling
'Below is the syntax for using the Frame subprogram
' left% = 3: right% = 80: top% = 3: bottom% = 22: fore% = 5: back% = 0
' CALL Frame(left%, right%, top%, bottom%, fore%, back%)

END

SUB frame (left%, right%, top%, bottom%, fore%, back%) STATIC
'*************************************************************************
' This module creates a framing box of any size  It takes six basic
' input parameters that control the dimensions and colors.
'*************************************************************************
COLOR fore%, back%

' ----- Draw the four corners

     LOCATE top%, left%: PRINT CHR$(201)
     LOCATE top%, right%: PRINT CHR$(187)
     LOCATE bottom%, left%: PRINT CHR$(200)
     LOCATE bottom%, right%: PRINT CHR$(188)

' ----- Draw the vertical lines

     FOR vert% = top% + 1 TO bottom% - 1
             LOCATE vert%, left%: PRINT CHR$(186)
             LOCATE vert%, right%: PRINT CHR$(186)
     NEXT vert%

' ----- Draw the horizontal lines

     horiz% = right% - left% - 1
     hline$ = STRING$(horiz%, 205)
     LOCATE top%, left% + 1: PRINT hline$
     LOCATE bottom%, left% + 1: PRINT hline$

END SUB
```

NOTE: The full help subprogram has not been included. Most of that subprogram consists of text screens which all have a similar format. The code listed here only shows an example of the coding structure of the menus without any of the help text.

```
SUB help
'*************************************************************************
' PURPOSE  this routine is intended to provide the basic information
' necessary to use the SPARTAN program.  Information is provided on the
' setup of a run with some discussion of the methods used for modeling.
' This subroutine is intended to allow the user to go directly to the
' information desired.

CLS 1
DO
CALL frame(1, 79, 2, 23, 7, 1)
LOCATE 4, 35
PRINT " HELP MENU "
LOCATE 6, 25
PRINT " SELECT ONE OF THE FOLLOWING TOPICS"
LOCATE 8, 30
PRINT " 1)  Exit Help"
LOCATE 9, 30
PRINT " 2)  Setup Screen"
LOCATE 10, 30
PRINT " 3)  Terrain"
LOCATE 11, 30
PRINT " 4)  Movement"
LOCATE 12, 30
PRINT " 5)  Search"
LOCATE 13, 30
PRINT " 6)  Target Selection"
LOCATE 14, 30
PRINT " 7)  Engagement"
LOCATE 15, 30
PRINT " 8)  Reaction to Fire"
LOCATE 16, 30
PRINT " 9)  Output"

LOCATE 21, 3: PRINT "Select Option Number  ": k$ = INPUT$(1)

SELECT CASE k$     'this select sends the user to the major category
 CASE "1"
  EXIT DO
 CASE "2"
  GOSUB h200
 CASE "3"
  GOSUB h300
 CASE "4"
  GOSUB h400
 CASE "5"
  GOSUB h500
```

```
 CASE "6"
  GOSUB h600
 CASE "7"
  GOSUB h700
 CASE "8"
  GOSUB h800
 CASE "9"
  GOSUB h900
 CASE ELSE
  BEEP
  k$ = INPUT$(1)
 END SELECT
LOOP

GOTO 1000 'when the program finishes executing the help select loop, it
          'must jump over the subroutines and go to the end of the help
          'subprogram.  Otherwise, it will try to execute the subs and
          'cause an error.

'these subroutines contain groups of topics and allow the user to
'get specific details from within each major topic area.

h200:
CLS 1
COLOR 7, 0
LOCATE 2, 1
PRINT "SELECT A SUBTOPIC  <1>EXAMPLE  <2>DEFAULTS  <3>ATTENUATION  <4>EXIT"
COLOR 7, 1
LOCATE 4, 23
PRINT " SCREEN HEADING"
PRINT
PRINT     *** this print section would contain the main topic
PRINT         discussion
PRINT

LOCATE 23, 2
PRINT " Select a Topic  ": k$ = INPUT$(1)

DO
SELECT CASE k$     'this select sends the user to the major category
 CASE "1"
  CLS 1
  COLOR 7, 0
  LOCATE 2, 1
  PRINT "SELECT A SUBTOPIC <1>EXAMPLE <2>DEFAULTS <3>ATTENUATION <4>EXIT "

  COLOR 7, 1
  IOCATE 4, 23
  PRINT " SUBTOPIC HEADING "
  PRINT            ***  subtopic discussion would be listed here ***
  PRINT
  PRINT
```

```
  LOCATE 23, 2
  PRINT " Select a Topic  ": k$ = INPUT$(1)

CASE "2"
   .
   .
CASE "3"
   .
   .
CASE "4"
  EXIT DO
CASE ELSE
  BEEP
  k$ = INPUT$(1)
END SELECT

LOOP
CLS 1
RETURN

1000 LOCATE 23, 2
PRINT " Returning to main menu"

CLS 1
END SUB



SUB opening
'*************************************************************************
'PURPOSE  This routine calls frame and draws a presentation
' screen for the model.

' VARIABLES
' left%, right%, top%, bottom% are integer variables passed to the Frame
' subprogram that define the box size using rows and columns in text
' mode.
' fore% and back% are variables for the line and background colors
'*************************************************************************

CLS
left% = 1: right% = 80: top% = 1: bottom% = 21: fore% = 7: back% = 1
CALL frame(left%, right%, top%, bottom%, fore%, back%)
left% = 9: right% = 72: top% = 7: bottom% = 13: fore% = 7: back% = 1
CALL frame(left%, right%, top%, bottom%, fore%, back%)
LOCATE 10, 25: PRINT "**** SPARTAN COMBAT MODEL ****"
LOCATE 16, 22
PRINT "Press <Enter> when ready to continue", k$ = INPUT$(1)

COLOR 7, 1

END SUB
```

```
SUB pause
'************************************************************************
'PURPOSE  a utility that queries the user to continue when ready

PRINT "Press <RETURN> to continue ..."; k$ = INPUT$(1)

END SUB



FUNCTION triag (A!, D!, b!) STATIC
'************************************************************************
' PURPOSE  This routine uses a U(0,1) random variate to produce a random
' variate with a triangular distribution TRIAG(A,D,B) using the
' inverse transform method as discussed by Pritsker(Pritsker:713).
' VARIABLES
' A is the left end of the triangle
' B is the right end of the triangle
' D is the mode of the triangle which may be any point between (A,B)
' R is a uniform random variate between [0,1]
'************************************************************************

  R = RND      ' should have a variable seed eventually so that the
               ' random variate generator seeds can be controlled.

  IF (R < ((D - A) / (b - A))) THEN
    triag = A + SQR((D - A) * (b - A) * R)
  ELSE
    triag = b - SQR((b - D) * (b - A) * (1! - R))
  END IF

END FUNCTION
```

Appendix D: SPARTAN Operating Instructions

General Information

The purpose of this appendix is to provide a stand alone user's manual for the SPARTAN combat simulation model. These instructions are intended to enable someone with limited knowledge of IBM PC operating systems to use the model.

The SPARTAN combat model is a two-sided high resolution combat simulation between opposing infantry squads. The program uses an event step scheduling approach to representing time advance. The model is intended to illustrate the operation of an analytic type combat simulation as currently used by the US Army. All algorithms and data are intended to be representative of models presently in use by the military, but there is no intention of portraying this model as a fine tuned representation of reality. Most of the data is fabricated so it "looks about right" and no effort has been made to perform any validation of the simulation as an analytic tool. This is purely an instructional model.

The instructions that follow are organized into four main topic areas. Section I is a brief discussion of the various modeling processes. Section II describes an example scenario and the accompanying data files that have been provided with the model software. Section III provides a checklist for the set up and operation of the model. Section IV contains instructions on using the preprocessor to modify the scenario dependent data files. In addition, more information on the processes can be found in the SPARTAN thesis.

I. Model Description.

SPARTAN simulates soldier on soldier combat at the squad level. The maximum number of soldiers is six per side [Red or Blue]. Each soldier is armed with a semi-automatic rifle. The battlefield represented is a 1000m x 1000m area. Factors outside this region have no influence on the outcome of the battle.

The SPARTAN model represents primarily an attrition type simulation process with limited aspects of command and control, logistics, force structure, and environment.

The model uses an event scheduling technique to synchronize activities and maintain time representation within the model. Future events are maintained in a chronological list according to the time that the event is scheduled to occur. When one event is finished, it may generate additional events to occur in the future. These events are placed in time sequence order and executed when the time clock advances to the scheduled time. The simulation continues this process until the event list is empty or some other user defined termination condition occurs.

A. Battlefield Representation.

Terrain in the model is represented by a 50 x 50 system of square grid cells. Each grid has associated east-west coordinates (horz) and north-south coordinates (vert) along with the attributes of elevation (elev) and a trafficability factor (mobfac). These attributes enable the model to represent terrain relief features and to vary soldier movement speeds over different portions of the terrain. Relief is represented on screen by colored patches in each grid cell on the screen

where the colors vary with elevation. Table 1 in Section IV and the models online help screens provide a listing of the color codes. The terrain database can be easily modified to represent various types of geography. Section IV discusses using the preprocessor STARTUP to modify the terrain database. The only environmental effect modeled on the battlefield is atmospheric attenuation which can degrade the target acquisition possibilities.

B. Soldiers.

The basic entity of the simulation is an infantry soldier represented by a list of seventeen attributes. These attributes maintain location, status, and capability information on each soldier. Each soldier has the ability to move, search for and select targets, engage targets, and react to being engaged. Tables 2 and 3 in Section IV provide brief descriptions of the attributes.

1. Movement. When a soldier's attributes are set for movement, he will move from his initial location along his direction of movement at a speed commensurate with his speed attribute, his posture, and the trafficability factor of the grid within which he starts his move. The soldiers always move 20 meters per movement cycle although the time it takes to perform that move varies according to field conditions. The soldier's initial direction of movement changes only as a result of enemy contact. Since SPARTAN is a discrete event simulation, the continuous action of movement is modeled by a series of discrete jumps from one location to the next. Using this modeling technique there is a trade off between accuracy which would mean shorter jumps and

the increased workload of computing the extra jumps. In this case, after trial and error, 20 meters was chosen as a suitable distance.

2. Searching. The search process is a sequential series of checks that are performed to determine the possibility of successfully detecting an enemy soldier. SPARTAN uses a continuous search algorithm based, loosely, on equations developed by the US Army Night Vision Electro-Optical Laboratory [NVEOL]. Specific search sectors are not defined, so it is assumed that the soldier performs a 360 degree search during each search cycle. There is no distinction between searching on the move or at a halt. The only currently sensor available is unaided human vision. A successful detection involves meeting a number of conditions. The first condition is that line of sight must exist between the observer and the target. Next the probability of acquisition based on range and the posture of the target must exceed an optical sensor threshold which is unique for each sensor type and the level of atmospheric attenuation. If a target can be acquired then a random number is drawn to determine the time spent searching the target's sector. This time and range to target are used to determine a probability of detection. Now another random number is drawn and compared to this value to determine detection. Targets that are detected are placed on the observer's target list, and a blue line is momentarily plotted on the screen to indicate a successful visual detection.

3. Target Selection. If a target is successfully detected, it is added to a target list and each target is then rated based on the firer's targeting priorities. The detection probabilities for each potential target are normalized and a random number is drawn to select

the target to be engaged from the list. A direct fire event will be scheduled next unless the target is perceived to be outside the firer's weapon range or the target's detection probability is below an engagement threshold.

4. Direct Fire. The engagement process begins by ensuring the soldier still has ammunition available and line of sight still exists. When the trigger is pulled, the ammunition is decremented, a bivariate normal probability of hit for the conditions is drawn from a table and a time of impact is computed. The flight of the bullet is represented by a momentary flash of a red line drawn between the firer and the target along with a brief sharp sounding noise. If a bullet impacts, as a hit, then a wound or kill determination is made and the soldier reacts accordingly.

Each of the possible impact results has its own signature on the display. A miss only results in a dull popping sound. If the round missed, then the firer will attempt to reengage and the target soldier will react to being fired upon. If the soldier is wounded then the firer will continue to engage and the wounded soldier will react to fire. The only effect on a wounded soldier is to slow his movement. Wounds are not cumulative. A wounded soldier displays a short red flash with a louder popping sound. If a soldier is killed, an extended yellow flash occurs with an accompanying sound, and the soldier's icon is changed to gray.

5. Reaction to Fire. This process demonstrates some very simple decision logic. Either the soldier will charge his opponent,

drop to the ground, or move away from the opposition. In any case, he may continue to engage the enemy.

6. Attrition. When a soldier is killed, all future events on the scheduling calendar for that soldier are removed. This has the effect of canceling a soldier in place even as he was preparing to pull the trigger.

7. Output. SPARTAN provides a graphic representation of the battle along with a help menu and screen displays of soldier attributes, target lists, and summary statistics. Final output includes selected soldier attributes, target listings, summary values from the battle, and a history file of all events as they occurred.

II. Example Scenario.

A. General Situation.

The SPARTAN land combat model contains two homogeneous land forces fighting as combatants an unoccupied sector of terrain that is forward of the current line of troops for each side. Both forces consist of a six man dismounted infantry squad armed with semi-automatic .30 caliber rifles.

B. Mission.

The Blue squad is conducting a reconnaissance patrol into Red territory, and is considered successful if half the squad can reach the Red rear boundary in 500 time units [Their primary mission is to locate forces in the enemy rear and avoid combat unless engaged by the enemy.]. The Red squad is arrayed in a counter reconnaissance screen to stop the Blue squad, and is considered victorious if it can kill more than 50% of the Blue squad or if the Blue squad fails to reach the Red rear boundary in 500 time units. The Red soldiers move forward to occupy their positions after dark each night, so they have no prepared fighting positions. The Red soldiers are initially stationary.

C. Battlefield in the Example.

The environment is barren rolling hills. There is little or no tree growth; scrub and grasslands predominate (20:9-24). There is no cover or concealment other than the terrain features. Blue forces are arrayed in the west and Red forces to the east. There are no other forces in the area that can affect the outcome of the battle. Atmospheric attenuation is set to .6 to represent a moonlit night this is a constant factor that will not vary during the battle.

D. Soldier Equipment in the Example.

The only equipment consists of .30 Caliber semi-automatic rifles for each soldier and 20 rounds of ammunition. The maximum effective range of the weapons on each side is 400m. In this case, both sides have comparable weapons and ammunition stockages. The maximum rate of fire is 16-24 rounds per minute with a sustained rate of fire of 16 rounds per minute (20:16-7). These rates are approximated with the reload and aim times used in the model.

E. Datafiles Required for this Scenario.

There are three scenario specific datafiles. These are the terrain file [board.dat], the soldier attribute file [force.dat], and the initial event list [event.dat]. The files required for the example scenario are board.ex1, force.ex1, and event.ex1. Section III will provide instructions on using these files.

III. Set Up and Use of SPARTAN.

This section provides a detailed list of the steps necessary to operate the SPARTAN combat simulation, and a few suggestions on techniques for working with the model.

A. Equipment.

The only equipment required is an IBM compatible personal computer with a minimum of 512k of memory, and an EGA or better color monitor. All programs are in executable form, so no special software or run time libraries are required unless you desire to modify the text format code which is available. The language used to create this software was QUICKBasic version 4.5. This program code will not compile with earlier forms of BASIC.

B. Files.

The following files must be located within the same directory to operate the model. This directory can be on a floppy disk or a hard drive, but will run more rapidly if loaded on a hard disk.

filename	description
1) startup.exe	preprocessor program
2) spartan.exe	simulation program
3) event.dat	initial event data file
4) board.dat	terrain data file
5) force.dat	soldier attribute file
6) p1.dat	probability of acquisition table
7) p2.dat	probability of detection table
8) phit300.dat	probability of hit table for 300m eff weapon
9) phit400.dat	probability of hit table for 400m eff weapon
10) phit500.dat	probability of hit table for 500m eff weapon
11) phit600.dat	probability of hit table for 600m eff weapon

SPARTAN will only open datafiles with these exact names. Files for several scenarios may be available, but must be copied to the above listed names prior to execution. It is suggested that all datafiles are

stored with names different from those above. This ensures that files are not inadvertently overwritten. All datafiles created or worked on by the STARTUP preprocessor program have an exp ending on the file name extension [ie. board.exp, force.exp, event.exp]. STARTUP will be discussed in detail later, but is not required to operate the sample scenario.

B. Operating the Simulation.

The following instructions will guide the user through the steps necessary to run the example scenario on SPARTAN.

Step 1. Ensure the datafiles listed above are loaded in the same directory as spartan.exe.

Step 2. Set the default directory to that containing the files and at the prompt type: spartan

Step 3. A presentation screen will appear, then hit enter when ready to continue.

Step 4. You will be asked whether or not the datafiles should be displayed as they are loaded into the program. You will probably only need to view the files once since it does require a longer time to display and load as opposed to just loading the files. The program will require a short moment to load up datafiles.

Step 5. The next screen asks whether you desire to modify the default settings for the simulation. Changes may not always be necessary, but you will want to select the setup screen to modify some of the run parameters that are required for the example scenario. The run parameters include termination conditions for the run, an attenuation coefficient, and a designated seed for the pseudorandom number

generator. The example scenario requires three parameters to be modified. The run time should be set to 500 time units, the level of red remaining should be set to 0, attenuation to .6, and the blue command switch should be turned off. These changes will ensure the simulation runs until one of the victory conditions is achieved for either side. Turning off the blue command switch ensures that the blue soldiers will orient on their original direction of movement and not move to seek out and engage red forces. Remember, the blue mission is reconnaissance and self-defense. The last step in the set up screen is to enter an initial seed value. From this point forward, online help is available throughout the simulation run.

Step 6. Hit <enter> when finished with the setup screen and the simulation will begin its run. At this point, the user can no longer influence the outcome of the simulation. There are five function keys [F1 - F5] that provide help and information throughout the simulation. [Note: These keys do not function after the simulation terminates and can only be called from the graphics screen.]

Step 7. When the simulation terminates, you will have the option of viewing the final output. To print any of the tables, use the <print screen> key. Additional, output is available in a history.dat file which is produced with each run. The history file provides a chronological listing of all events as they occurred. It includes the event time, type and the event actor. To maintain copies of the history file, modify the file name or it will be overwritten during the next program run. This completes a run of the SPARTAN simulation model.

IV. Modifying the Datafiles.

The STARTUP program is a preprocessor that can be used to create datafiles for specific scenarios. The program is menu driven and enables the user to create, edit and review the terrain, soldier attribute, and initial event datafiles. Several representative hit probability tables [such as phit300.dat] have been provided for weapons with different ranges as well as the probability of acquisition [p1.dat] and probability of detection [p2.dat] tables. There is no provision in STARTUP to create or modify these files although all the files can be modified with a standard ASCII text editor.

A. Terrain Datafiles.

There are four attributes associated with each grid. These are the east-west coordinate (horz), the north-south coordinate (vert), elevation (elev), and the trafficability factor (mobfac). The coordinates are limited in values from 1 to 50. The elevation is in meters and has a range of 0 to 69 meters. The trafficability factor has a range of 0.1 to 1.0 with 1.0 being unimpeded movement. The color coding for the elevations is shown in Table 4.

Table 4 Elevation Color Coding

Elevation	Color
0 - 10m	light green
11 - 20m	cyan [turquoise]
21 - 30m	light red
31 - 40m	magenta [pink]
41 - 50m	yellow
51 - 60m	light grey
61 - 69m	brown

The terrain datafile module has an additional feature that plots out the terrain relief overlay for a visual representation of the data.

B. Soldier Attribute Files. Tables 5 & 6 provide a brief description of the seventeen attributes.

Table 5 Soldier Attribute Descriptions

Attribute	Description
1) x	present horizontal coordinate
2) y	present vertical coordinate
3) z	present elevation coordinate
4) xlast	last horizontal coordinate
5) ylast	last vertical coordinate
6) size	soldier's height in meters
7) speed	movement speed *
8) dir	soldier's movement direction/orientation
9) moving	indicates soldier's intent to move
10) wpnrng	identifies soldier's weapon type and range
11) ammo	maintains count of available ammunition
12) status	indicates whether alive, dead, or wounded
13) posture	whether standing, crouched, or prone
14) incmd	identifies a squad leader
15) atkdir	maintains original movement direction
16) tgteng	identifies a selected target
17) side	indicates affiliation of soldier

* movement units are relative on a scale of 0 - 40 units

Table 6 provides specific information on the acceptable values that should be used for each attribute.

Table 6 Soldier Attribute Variables

Attribute Name	Type	Range of Values
x	real	[0.1 - 1000.00]
y	real	[0.1 - 1000.00]
z	real	[0.0 - 69.0]
xlast	real	[0.1 - 1000.00]
ylast	real	[0.1 - 1000.00]
size	real	[1.5 - 2.0]
speed	real	[0.0 - 40.0]
dir	real	[0.0 - 6.28] (radians)
moving	integer	[0 stopped / 1 moving]
wpnrng	real	[300.0 - 600.0]
ammo	integer	[0 - 999]
status	integer	[0 dead / 1 alive / 2 wounded]
posture	real	[.25 prone/.5 crouch/1.0 stand]
incmd	integer	[0 subordinate / 1 squad leader]
atkdir	real	[0.0 - 6.28] (radians)
tgteng	integer	[0 - 12]
side	integer	[1 blue / -1 red]

The editor for the soldier attribute module of STARTUP allows the user to modify all the attributes of one soldier or a single attribute for a group of soldiers. At the present time, it does not allow for the modification of just a single value.

C. Initial Event File. The initial event file contains an initial search event [event type 1] and an initial movement event [event type 2] for each soldier. This file will usually not need to be modified to accommodate most scenarios. Two occasions when the user might choose to modify the file would be to delay the entry of soldiers into the battle or to remove selected soldiers from the battle. All soldiers in the battle must have both cited events. Even if a soldier is not expected to move, he needs the movement cycle to produce graphics. The event datafile for the example scenario has 24 events to start the simulation. Each event record has the fields in Table 7.

Table 7 Initial Event Attributes

Name	Range of values
event type	[1 - 8]
event time	[0.1 - 9999.0]
event actor	[1 - 12]

D. Performing Analysis with SPARTAN.

While SPARTAN is not an empirically valid model, it is still possible to experiment with the model to determine how the model will react to changes in weapon ranges, the attenuation factor, movement rates, tactical formations, or different types of terrain, etc.. An important consideration in modeling is to determine the sensitivity of the model to certain factors. This can be accomplished by varying just

one parameter of the model over a range of values while holding everything else constant.

The model design allows the random number stream to be set with new seeds for each run, so independent statistical results can be obtained using this model. This model has the potential to be used in other course project areas such as response surface analysis, variance reduction techniques, and multivariate statistics.

E. User Comments.

Since this is intended to be an educational tool, comments from personnel using the model will be greatly appreciated. Please approach this model with a critical view and provide feedback as necessary. Anything that presents problems to learning will be reexamined and modified if possible.

Bibliography

1. Bailey, H. H., L. G. Mundie, and H. A. Ory. Suggested Modifications to Optical Sensor Algorithms in JANUS. Contracts MDA903-90-C-0004 and MDA903-86-C-0059. Santa Monica CA: Rand Corporation, November 1990(RAND/N-3087-DR&E/A/AF).

2. Balci, Osman. "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-level Languages," Proceedings of the 1988 Winter Simulation Conference. 287-295. New York: IEEE Press, 1988.

3. Balci, Osman. "Credibility Assessment of Simulation Results: The State of the Art," Proceedings of the Methodology and Validation Conference. 19-25. San Diego: Society for Computer Simulation, 1988.

4. Banks, Jerry. and others. "Modeling Processes, Validation, and Verification of Complex Simulations: A Survey," Proceedings of the Methodology and Validation Conference. 13-18. San Diego: Society for Computer Simulation, 1988.

5. Battilega, John A. and Judith K Grange. The Military Applications of Modeling. Washington: Government Printing Office, 1984.

6. Biles, William E. "Introduction to Simulation," Proceedings of the 1987 Winter Simulation Conference. 7-15. New York: IEEE Press, 1987.

7. Bonder, Seth. "Mathematical Modeling of Military Conflict Situations," Proceedings of Symposia in Applied Mathematics. 25: 1-51. New York: American Mathematical Society, (1981).

8. Cohen, Jay W. and others. "Structured Modeling," Proceedings of the 1982 Winter Simulation Conference. 253-258. New York: IEEE Press, 1982.

9. Combat Modeling Briefing Slides, TRADOC Analysis Command, Monterrey, CA. July 1991.

10. Comptroller General of the United States. Guidelines for Model Evaluation (Exposure Draft). PAD-79-17, Washington: Government Printing Office, January 1979.

11. Comptroller General of the United States. Models, Data, and War: A Critique of the Foundation For Defense Analyses. PAD-80-21, Washington: Government Printing Office, 12 March 1980.

12. Department of the Army. Army Model Improvement Program. AR 5-11 (Draft). Washington: HQ USA, 11 April 1990.

13. Department of the Army. US Army TRADOC Analysis Center. CASTFOREM Methodology Manual. TRAC-WSMR-TD-4-88. Washington: Government Printing Office, March 1990.

14. Department of the Army. US Army TRADOC Analysis Center. JANUS (T) Documentation. Washington: Government Printing Office, June 1986.

15. Department of the Army. US Army TRADOC Analysis Center. JANUS(A) Supplement to JANUS Documentation. Washington: March 1991.

16. Department of the Army. US Army TRADOC Analysis Center. TRADOC Studies and Analysis. TRADOC Pamphlet 11-8 (Draft). Washington: Government Printing Office, 1991.

17. Derrick, E. Joseph. and others. "A Comparison of Selected Conceptual Frameworks for Simulation Modeling," Proceedings of the 1989 Winter Simulation Conference. 711-718. New York: IEEE Press, 1989.

18. Dudewicz, Edward J. and Zaven A. Karian. Tutorial: Modern Design and Analysis of Discrete Event Computer Simulations. Washington: IEEE Computer Society Press, 1985.

19. Dunnigan, James F. The Complete Wargames Handbook. New York: William Morrow and Company, 1980.

20. Engineering Design Handbook, Army Weapon Systems Analysis, Part One, DARCOM-P 706-101. US Army Material Development and Readiness Command, Washington: Government Printing Office, November 1977.

21. Evans, John B. Structures of Discrete Event Simulation. New York: John Wiley and Sons, 1988.

22. Gass, Saul I. and others. "An Assessment Procedure for Simulation Models: A Case Study," Operations Research. 39: 710-723 (September-October 1991).

23. Hartman, James K. Lecture Notes in High Resolution Combat Modeling. Unpublished Notes, 1985. Class handout distributed in OPER 775, Land Combat Modeling I. School of Engineering, Air Force Institute of Technology(AU), Wright-Patterson AFB OH, July, 1991.

24. Henriksen, James O. "One System, Several Perspectives, Many Models," Proceedings of the 1988 Winter Simulation Conference. 352-356. New York: IEEE Press, 1988.

25. Hughes, Bernard C. Jr. Target Selection Schemes. MS Thesis, Naval Postgraduate School, Monterey CA, March 1988 (AD-A194 657).

26. Joint Analysis Directorate, Organization of the Joint Chiefs of Staff. Catalogue of Wargaming and Military Simulations Modeling. JADAM 207-89. Washington: Government Printing Office, September 1989.

27. Kirby, Charles L., JANUS Proponency Director. Telephone Interview. JANUS Proponency Office, White Sands Missile Range NM, 5 September 1991.

28. Law, Averill M. and W. David Kelton. Simulation Modeling & Analysis. New York: McGraw-Hill Bool Company, 1991.

29. McCormick, Ernest J. and Mark S. Sanders. Human Factors in Engineering and Design. (Sixth Edition). New York: McGraw-Hill Publishing Company, 1987.

30. McCormick, William M. and Robert G. Sargent. "Comparison of Future Event Set Algorithms for Simulation of Closed Queueing Systems," Current Issues in Simulation. edited by Nabil R. Adam and Ali Dogramaci. New York: Academic Press, 1979.

31. Morris, William T. "On the Art of Modeling," Management Science. 11: B-707 -- B-717 (August 1967).

32. Nance, Richard E. and James D. Arthur. "The Methodology Roles in the Realization of a Model Development Environment," Proceedings of the 1988 Winter Simulation Conference. 220-225. New York: IEEE Press, 1988.

33. Overstreet, C. Michael and Richard E. Nance. "A Specification Language to Assist in Analysis of Discrete Event Simulation Models," Communications of the ACM, 28: 190 - 201 (February 1985).

34. Pritsker, A. Alan B.. Introduction to Simulation and SLAM II. (Third Edition). New York: John Wiley and Sons, 1986.

35. Rodriguez, Luis C. and others. "An Empirical Comparison of Advanced Event File Synchronization Structures," Proceedings of the 1982 Winter Simulation Conference. 189-194. New York: IEEE Press, 1982.

36. Sadowski, Randall P. "The Simulation Process: Avoiding the Problems and the Pitfalls," Proceedings of the 1989 Winter Simulation Conference. 72-79. New York: IEEE Press, 1989.

37. Shammas, Namir Clement. "The BASIC Revival," BYTE. 13: 295-300 (September 1988).

38. Sheridan, Robert E. "The Script Processing Technique in Modeling/Simulation and its Role in the Generation of Animated Computer Graphics," Proceedings of the 1986 Winter Simulation Conference. 819 - 824. New York: IEEE Press, 1986.

39. Seila, Andrew F. "SIMTOOLS: A Software Tool Kit for Discrete Event Simulation in Pascal," Simulation. 50: 93-99 (March 1988).

40. Tavares, MAJ Michael, Personal Interview. US Army Training Analysis Command, Scenario Development Center, 17 July 1991.

41. Thesen, Arne. "Writing Simulations from Scratch: PASCAL Implementations," Proceedings of the 1987 Winter Simulation Conference. 152-164. New York: IEEE Press, 1987.

42. Tremblay, J.P. and P.G. Sorenson. An Introduction to Data Structures with Applications. New York: McGraw-Hill Book Company, 1976.

43. Weiss, M. A. "Empirical Study of the Expected Running Time of Shellsort," Computer Journal. 34: 88-91 (February 1991).

44. Wiggins, Mike. "A Comparison of Computer Languages Pascal, C, Lisp and Ada," Journal of Pascal, Ada, and Modula-2. 7: 5-10 (January 1988).

Vita

Captain David Keith Cox was born on 2 December 1956 in Saint Louis, Missouri. He graduated from Webster Groves High School in Webster Groves, Missouri. He enlisted in the US Army in 1976 and, subsequently, attended the United States Military Academy Preparatory School and the United States Military Academy at West Point, New York. He graduated in May 1982 with a Bachelor of Science degree. He has been assigned as a platoon leader, squadron maintenance officer and troop executive officer in the 2d Squadron, 11th Armored Cavalry Regiment at Bad Kissingen, West Germany. He was subsequently assigned as the Company Commander of B Company, 1st Battalion, 64th Armor Regiment, 24th Infantry Division (Mechanized) and as Company Commander of Headquarters Company, 2d Brigade, 24th Infantry Division (Mechanized), Fort Stewart, Georgia. He is a graduate of the Armor Officer Basic and Advanced Courses. In August 1990, he was assigned to the Air Force Institute of Technology.

Permanent address: 458 Ivanhoe Place
Webster Groves, Missouri 63119