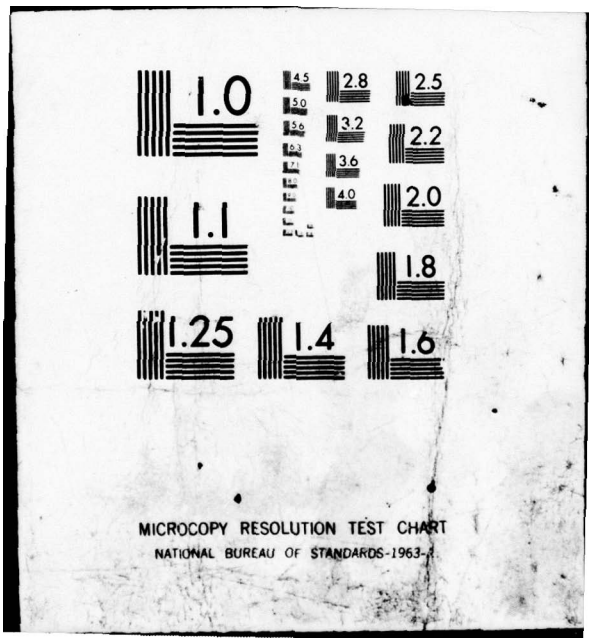
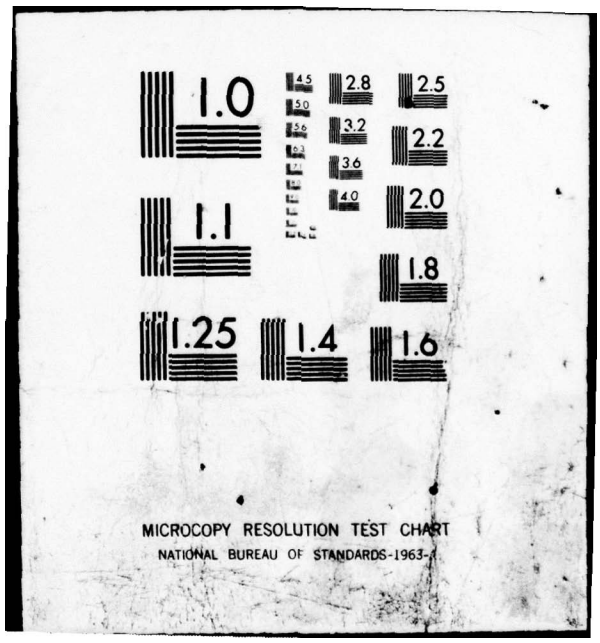
AD-A070 096 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
SYSTEMS ANALYSIS FOR THE INTERACTIVE SIMULATION WITH GRAPHICAL --ETC(U)
MAR 79 G S COKER, D R FORINASH

UNCLASSIFIED NL

1 OF 3
AD A070096

1 OF 3
AD A070096


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL

Monterey, California

LEVEL II

THESIS

SYSTEMS ANALYSIS FOR THE INTERACTIVE SIMULATION WITH GRAPHICAL DISPLAYS TO SUPPORT SIMULATION OF TACTICAL ALTERNATIVE RESPONSES (STAR)

by

George Sansing Coker

and

David Ralph Forinash

March 1979

Thesis Advisor: S. H. Parry

Approved for public release; distribution unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
(6) Systems Analysis For The Interactive Simulation With Graphical Displays To Support Simulation Of Tactical Alternative Responses (STAR).

5. TYPE OF REPORT & PERIOD COVERED
(9) Master's Thesis; March 1979

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
(10) George Sansing Coker
David Ralph Forinash

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

(11) 12. REPORT DATE
March 1979
13. NUMBER OF PAGES
218

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Naval Postgraduate School
Monterey, California 93940
(12) 219 p.

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

STAR
Interactive
Graphics
Simulation
Tactical
Operating System

79 06 20 065

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

A systems analysis of a computer system to support the STAR war gaming model is presented. The war game is described and the user's objectives are defined. Four conceptual methods for implementing the user's objectives are presented and a preferred solution is chosen. The characteristics of the preferred solution that are necessary to meet the user's objectives are described. Further software needed to implement the objectives is briefly discussed. The code for the current model is analyzed and a summary presented. ☆

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 | 1

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251 450

presented. Conclusions from this systems analysis are derived and future research areas are identified.

Approved for public release; distribution unlimited.

SYSTEMS ANALYSIS FOR THE INTERACTIVE SIMULATION WITH GRAPHICAL DISPLAYS TO SUPPORT SIMULATION OF TACTICAL ALTERNATIVE RESPONSES (STAR)

by

George Sansing Coker
Major, United States Marine Corps
B.S., Mississippi College, 1968

and

David Ralph Forinash
Captain, United States Army
B.S., United States Military Academy, 1970

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1979

Authors ---------- George S Coker

---------- David R Forinash

Approved by: ---------- S H Parry
Thesis Advisor

---------- Second Reader

---------- Chairman, Department of Computer Science

---------- William M. Tolles
Dean of Science and Engineering

ABSTRACT

A systems analysis of a computer system to support the STAR war gaming model is presented. The war game is described and the user's objectives are defined. Four conceptual methods for implementing the user's objectives are presented and a preferred solution is chosen. The characteristics of the preferred solution that are necessary to meet the user's objectives are described. Further software needed to implement the objectives is briefly discussed. The code for the current model is analyzed and a summary presented. Conclusions from this systems analysis are derived and future research areas are identified.

TABLE OF CONTENTS

I. INTRODUCTION

A computer simulation is the representation of a mathematical model in a manner that allows the user to examine the system being modeled and gain further insight into the inner workings of the system. Typically, simulations fall into five classes: the presentation of unobservable phenomena, operator training models, gaming models, design tools and models of systems with factors that preclude experimentation on the system itself [Rahe 1972].

Modern computers and graphics terminals have removed the last barriers that previously dictated all-digital simulations. These terminals can be used as exceptionally fast and versatile output devices. The computer may be equipped with a graphical input device such as a graphic tablet enabling the system to be used as a drafting device with unique properties [Newman and Sproull 1973].

The computer system intended for use should provide on-line, hands-on high-speed computation with excellent displays and interfaces to external hardware. The application of graphical support to war gaming adds a dimension previously not available to the modeler. The capability to visually monitor the simulation during execution provides insight into the system being modeled that tabular results obtained after the fact can never hope to attain. Interactive programming allows the development

of applications which can interact with a user and enable him to control the functions performed and the data operated on by the program. The modeler may now interactively participate in the simulation and at critical times make decisions that will create changes in the outcome of the simulation.

One important facet of interactive programming is the aspect of man-computer symbiosis. By achieving a very close coupling between the human and the computer, the computer may facilitate formulative thinking and allow the human member of the partnership to participate in making decisions and controlling complex situations without inflexible dependence on predetermined programs [Licklider 1960].

Any system devised to attain such an interactive simulation system with graphical support should not be hastily thrown together. "Add-on" support tends to overly complicate and reduce the efficiency of the existing system. For any system, the principles of simplicity and effectiveness are essential to the usefulness of the system. These two principles often compete with each other [Smith 1970]. In this light, it is critical that to support any existing or planned system, a careful systems analysis and design must be the first step toward implementing that support. This added effort should result in an efficient, effective system while maintaining maximum flexibility and simplicity. In the rush to implement a major system, the emphasis is generally placed on the application with little

consideration given the human factors when the interface between man and his program is implemented. Human interface is desirable at setup time when there are many displays, options and parameters for a user to control. If he is an experienced user he does not want to be forced to cycle through all the options. The user desires only enough prompting information to change those options necessary to setup and execute his run. No matter how well a system performs, it will be little used if it is difficult to setup. The user must be considered first and effective man-machine interface dialogue will become a major consideration, as important as the application itself.

The process of extending an existing war game to include interactive functions is so complex that programming productivity techniques must be planned and employed from the onset. These modern techniques have proven to be successful in control of software projects. Such techniques as structured design, top-down design, top-down programming, modularization, structured programming and walkthroughs, chief programmer team concepts and a scheduling methodology will be crucial to the development and maintenance of an acceptable model.

This study considers all possible software concepts and explores their applicability to the model. Some of these that could prove to be useful tools are: database management systems, SIMSCRIPT, graphic support languages and either general purpose or tailored versions of operating systems.

Various programming languages lend themselves to war gaming. The advantages of a high level language are well established and their usage is particularly significant in the area of programmer productivity. Because each of these language instructions translates into 10-20 lines of documented code, the daily productivity of the programmer is increased at least three fold. This increase has been demonstrated in numerous studies [Brooks 1975]. High level languages also contribute to lower application maintenance costs, improved documentation and design through their ability to allow self-documenting variable names, new constructs and easier implementation of algorithms (stack and queue manipulation for example).

SIMSCRIPT is an example of a high level language especially developed for simulations. The internal functions of SIMSCRIPT such as the event scheduling, queue manipulation and system defined variables enhance the desirability for using it. FORTRAN subroutines can be readily called from the language allowing program efficiency to increase by employing critical code in the form of FORTRAN subroutines. The current version of SIMSCRIPT II.5 was written in SIMSCRIPT II.5. This fact demonstrates the versatility of the language. In the last several years certain developments have improved the efficiency of SIMSCRIPT. Among these developments was the move from the generation of FORTRAN intermediate code to direct generation of machine code. Additionally, the number of steps required

to compile, link-edit and execute a SIMSCRIPT program was reduced, shortening compilation time.

This systems analysis parallels most systems analysis procedures but addresses the question, "What resources do I need to provide the desired capabilities given only the constraint of using SIMSCRIPT II.5 as a base simulation language" versus "How can I design this system to run on a particular computer". This is considered the appropriate approach to designing a system which truly meets the needs of the combat modeling community. Figure 1.1 depicts the systems analysis procedures followed.

UNCONSTRAINED SYSTEMS ANALYSIS STEPS

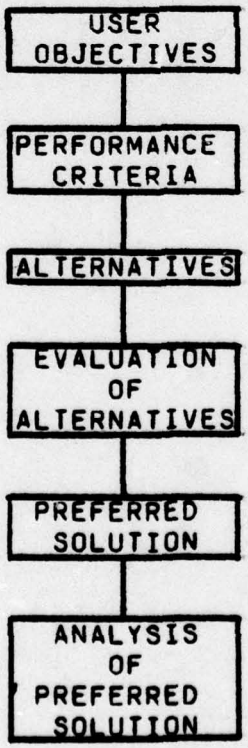


FIGURE 1.1

The first and key element of this systems analysis study was the identification of the user's objectives. Without this critical action the entire study would have been meaningless. The objectives fell into the three general areas of graphical display support, simulation of the battle area and the capability to determine the status of the battle or any of the components of the battle at any desired instant of time during the simulation.

Once the user's objectives were determined, the next step was to select a set of general performance criteria. Performance criteria are the key to measuring the degree of success in attaining the objectives of the user. System performance measures must be considered to produce an efficient and effective system for the entire life-cycle. The system must possess the ability to reflect changes in both friendly and enemy force structures and tactics. The data reflected by the display devices must be current at the time of display. This could mean that the simulation would have to be halted until a signal is generated by the completion of the display. The quality of the data base will be reflected by the level of data integrity needed. The size of the data base must be such that only necessary data items be stored. Any abuse of this will surface in the response time for any display invoked and the overall amount of secondary storage required for the data base. The system was designed with growth in mind so that additional capabilities may be accommodated as new technology develops.

As tactics and force structures change, the system must be capable of easily absorbing these needed changes. Response time, another performance measure, is considered to be from the time a user depresses a carriage return after a display request until that request is satisfied. From other studies [Sutherland 1966] response times of greater than 10 seconds are not desirable since the human mind will become impatient after such a long waiting period. A good response time is usually three to five seconds.

Alternative designs were conceived, each possessing the capability to provide all of the desired functions identified as user objectives within the performance criteria established. Alternative conceptual designs are defined as concepts arrived at through "dreaming" with objectives and restrictions in mind. In order to attempt to capture the latest hardware and software capabilities and philosophys and incorporate them into this system, a certain amount of general research was conducted in the areas of simulation, interactive graphics, database design and implementation, operating systems and systems analysis and design.

After the list of designs was consolidated into general concepts, each conceptual design was evaluated to insure that the design under consideration met the objectives. The advantages and disadvantages of each alternative were determined. This reduced the list to only feasibile solutions to the problem.

Chapter II briefly discusses the history of war gaming, the evolution of STAR, a current description of STAR and proposed enhancements with the support required to achieve them. This final version of STAR is the entire reason for this thesis, that is, the design of a system to reach this goal.

Chapter III discusses the alternative approaches to the design of a support system for STAR. The alternatives include a database management system, an operating system, a distributed system and graphical routines imbedded in the simulation program. These alternatives are described and evaluated as to their individual capability to implement the system. A prefered solution was then chosen.

Chapter IV continues with the analysis of the operating system needed to support the solution from chapter III. The basic features needed to support STAR are described and examples of their use are given. The modules of STAR not previously written are outlined to give guidance in future development activity.

Chapter V presents the results of the analysis of the current version of STAR. This chapter summarizes the findings of the analysis and presents modifications and recommendations that will lessen the storage requirements for STAR while speeding up the execution of the model.

Chapter VI contains conclusions from this thesis and recommends further courses of action to achieve the desired end product.

II. BACKGROUND

A. HISTORY OF WAR GAMES

War games are nearly as old as organized warfare itself. Evidence has been uncovered that indicates the use of games to simulate war in ancient Egypt. Progress in war gaming is marked by a series of improvements in support techniques available to the user.

During the latter half of the eighteenth century the Prussians developed an increased emphasis on warfare as a branch of applied mathematics. In 1780, Helwig, Master of the Pages for the Duke of Brunswick, invented a game quite similar to the modern commercial war game. The game used a modified chessboard. Terrain was represented by using combinations of 1666 small squares tinted in various colors. These small squares were grouped onto the board as terrain features. In 1795 Georg Vinturinus modified the game by constructing a map board of an actual piece of terrain.

In 1811 von Reisswitz, the Prussian War Counselor at Breslau, transferred the war game to a sand table with terrain modeled in sand to a scale of 1:2373. In 1824 Army Lieutenant von Reisswitz, Jr. modified his father's game by transferring the game to a realistic map-like chart with a scale of 1:8000. An umpire, detailed rules, and probability tables were also introduced by von Reisswitz, Jr. The size of the game was limited to approximately four square miles

of ground. The umpire not only monitored the play of the game for compliance with the rules but also imposed two minute time slices in the playing of the game. In this manner the game could be stopped, as desired, and a particular two minute round could be studied in detail.

Further improvements occured during the ninteenth century. In 1866 Lieutenant Wm. McC. Little suggested a game called "Naval War Game" that employed blackboards, sheets of paper or charts, or maps placed on tables to illustrate terrain. At about this same time celluloid sheets or overlays were introduced. Information drawn on these overlays was saved as a historical record that could be analyzed at a later date. This idea of overlays is attributed to the Naval War College.

Early in the twentieth century, new maps prepared especially for map maneuvers showed large tracts of actual terrain. The oldest map of American terrain made expressly for map maneuvers dates from 1906. This map of Fort Leavenworth, Kansas includes a tract approximately four miles in width by six miles in length with a scale of 1:5280 and a contour interval of ten feet [JCS 1969].

In the post WW-II era, the military use of war games became increasingly sophisticated and widespread. Sophistication is achieved through increasing computer technology. The computer allows large amounts of data to be stored and manipulated without tedious bookkeeping on the part of the user. There is some debate over the usefulness

of such computer simulations. The amount of data generated is so great that it can overwhelm the user, thereby undermining the very reason for the simulation.

Modern computer war games have seen evolution similar to manual games. One parallel development is in the terrain model associated with the game. Early games used flat (imaginary) terrain. Terrain advanced to idealized, easily constructed models that represented no identifiable terrain. The next step was to represent real terrain through the use of digitization at the expense of storage. The latest breakthrough is the use of parametric terrain capable of modeling any real terrain in a size never before imagined [Needles 1976].

B. EVOLUTION OF SIMULATION OF TACTICAL ALTERNATIVE RESPONSES (STAR)

A significant effort is currently underway at the Naval Postgraduate School to develop a mid-resolution combined arms model to determine both hardware and training measures of effectiveness. One of the primary goals of the model is to achieve an acceptable level of resolution while assuring that the model inputs and interactions are understood by the military decision-maker.

Five theses completed over the last two years form a basis for continuing the model development. A parametric terrain model was developed to provide a continuous macro-terrain representation. This representation has several

advantages over the classical approach of digitized terrain. Line-of-sight computations are made directly from mathematical relationships as opposed to the time-consuming iterative process required with digital terrain. Mobility is truly continuously as opposed to piecewise linear techniques used for digitized terrain. Terrain can be considered as a parameter of the combined arms analysis as opposed to a given. By appropriate selection of input parameters, any real section of terrain can be closely approximated by the parametric terrain model. Any size terrain sector can be easily generated without the storage constraints of digital terrain. A dynamic smoke module has been developed and operated in the parametric terrain model.

After development of the terrain model, two theses were devoted to a target servicing evaluation of blue artillery against a red ground threat. The result of this effort was a working model programmed in SIMSCRIPT which provides dynamic representation of the artillery missions down to the individual element level. This model forms the basis for future enhancements of the combined arms model. An ammunition supply model was developed to represent the effects of such parameters as interdictive enemy fire, RAM-D, truck trips per day to the ammunition supply point, truck replenishment rate, etc., on the number of rounds available to the combat vehicles over any sustained combat period.

Current efforts are underway to incorporate two-sided ground and artillery, with other systems as close air

support, minefields, cannon launched guided projectiles, advanced attack helicopters, air defense, etc. These enhancements will be made to the blue battalion versus red regiment model. In addition, a dynamic ammunition resupply model is being developed.

C. CURRENT DESCRIPTION OF STAR

The structure of STAR is truly hierarchial in that it is not confined to any specific unit size or configuration. The parent-child set structure of SIMSCRIPT, coupled with the flexible parametric terrain model, provides the required capabilities to realize a hierarchial representation. The level of resolution is prescribed by the requirements.

The first study application of STAR was in support of the 105/120mm ammunition stowed load requirements for the XM-1 tank. Initial production runs for the study were conducted for a blue battalion versus a red regiment in December 1978. This version of STAR represented all appropriate ground direct fire units, two-sided artillery, minefields and smoke. Upon completion of the Phase I battalion-level production runs (on a 10 x 10 km battlefield), Phase II was initiated. The result of Phase II was a brigade-level model versus a red division on a battlefield approximately 50 x 50 km. The Phase II model will be capable of simulating a multi-echelon red regimental attack on multiple avenues of approach in both the Covering Force Area (CFA) and the Main Battle Area (MBA). Extented

dynamic play of ammunition and POL resupply, as well as a significant enhancement of the tactical representation of battalion engagements, will result from the Phase II model. In addition, artillery units will be directly represented on the battlefield, allowing for specific play of counterbattery and counter air-defense fires. Finally, a dynamic air-to-air defense model is being developed for Phase II model representing two-sided air-to-air engagements for both fixed and rotary wing aircraft. All appropriate red and blue air defense systems will be played in the model.

The SIMSCRIPT language was selected for STAR because the language was designed for discrete event simulations. The many embedded features of the language give the programmer wide latitude in the construction of event flow. The language is English-like with regard to the construction of commands. The heart of the embedded simulation facilities is the timer, which is used with certain structural characteristics: entities, attributes, sets and events. These facilities greatly simplify the process of writing a simulation program and debugging the code. This is further enhanced by a compiler which provides error messages and trace-back routines similar to WATFOR and WATFIV in FORTRAN.

The structure of STAR begins with the concept of an entity. An entity is simply a representation or model of an item. In STAR the basic entity is a weapon system representing tanks, TOWS, artillery pieces, etc. Any of these entities may be brought into existance by a simple

phrase, which includes the name of the entity. For example, the phrase CREATE A TANK reserves a place in memory for the entity and its attributes which the programmer has chosen to call TANK. Associated with the word TANK is a pointer variable which points to the location in memory where TANK is stored. It is desirable to associate certain characteristics with entities after they have been created. These characteristics are referred to as attributes and are affixed to an entity by the internal bookkeeping procedures of SIMSCRIPT (the system) or are placed on the entity by the programmer. Attributes must be changed by the programmer as necessary to reflect changes in characteristics. Moreover, the system will change system-defined attributes as necessary. The concept of sets in SIMSCRIPT is very useful when it is necessary to group entities based on certain characteristics or in the construction of queues. In STAR sets have been used primarily to portray membership in organizations. The set structure mirrors organizational structure and enhances the programmer's ability to model unit tactics from a micro to macro level. An entity may belong to any number of sets and the entity acquires a membership attribute which facilitates identification of an entity's unit.

An extremely flexible method of filing allows entities to be ordered in a set by ranking of certain attributes or by a simple first-in-first-out basis. STAR uses the latter system for most applications. The set logic of SIMSCRIPT

allows this to be easily expanded to higher level organizations.

Each entity in STAR is modeled to reflect a flow of activities over time. In particular, each entity initiates or undergoes search, detection, target selection, firing or impact. These five events are scheduled dynamically based on the current tactical situation or an appropriate probability distribution. When an event is scheduled for an entity, the SIMSCRIPT timer makes a record of the time that the event is to occur (in terms of overall simulation time) and the entity for which the event has been scheduled. Other characteristics of the event may be recorded in a manner similar to the assignment of attributes. At the appropriate simulation time, the event is executed unless cancelled by some logic provided by the programmer. This event, when created, would be filed in an event set which contained, among other things, the time that the event is to take place, the entities involved in the event and the event location with respect to other scheduled events. When X seconds had elapsed from the current simulated time, the event would take place and the consequences of the logic written in the event routine would be executed. Event routines may in turn generate other events. FIRE, for example, causes the scheduling of an IMPACT event. IMPACT leads to the scheduling of other DETECT and TARGET.SELECT events.

Event routines are supported by a number of

computational subroutines in STAR. Subroutines are written in both SIMSCRIPT and FORTRAN which has given the simulation a great deal of flexibility. The difficult line-of-sight calculations, for example, are accomplished in FORTRAN because the terrain model was originally written in FORTRAN. It was this capability to call FORTRAN subroutines that made SIMSCRIPT an even more appealing language. Existing FORTRAN routines could be used with only minor modifications. Other routines more closely tied to the entity structure of SIMSCRIPT were written in that language. The routine that updates the list of detected targets for each TANK is written in SIMSCRIPT to take advantage of the dynamic dimensioning capabilities of the language and the pointer variable link listing techniques available. For large target arrays, these language features are extremely efficient in reducing memory requirements.

D. PROPOSED ENHANCEMENTS

The STAR combat model currently under development at the Naval Postgraduate School operates in batch mode on the IBM 360-67 computer in the W.R. Church Computer Center. It is difficult to build a simulation model in a batch processing environment. Batch processing consumes much of the time in developing the simulation. Interactive simulation is more economical as well as more effective in problem analysis. One of the primary goals of this project is to expand the model to operate in an interactive mode with graphical

support. Figure 2.1 depicts the goals of this project. The modeler's ability to debug a simulation is greatly enhanced by the interactive capabilities of a language. Since the simulation user is constantly engaged in upgrading his simulation, interactive capabilities are an important feature of any support system [Mills and Phil 1977].

EXTENDED STAR

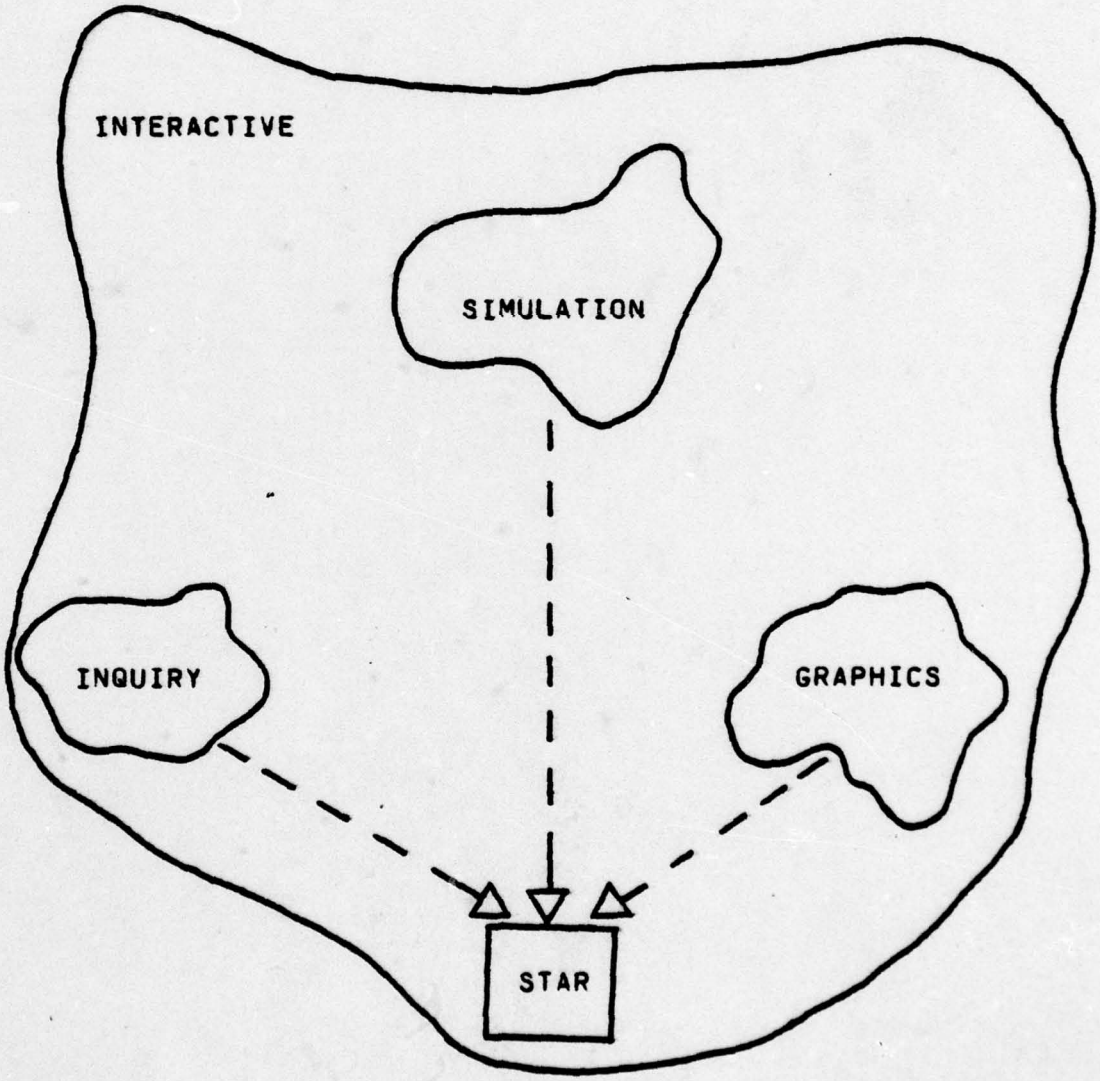


FIGURE 2.1

The ability to actively participate in this war game in an interactive mode would contribute to both the productivity and the flexibility of the model. In an interactive environment, the player or modeler would be able to input decisions that would approximate those made by the commander on the battlefield. Through this man-computer symbiosis, the ability of the model to more accurately reflect the actual outcome of a battle would be attained. The actual flow of the battle could be altered to reflect realism rather than rigid programming which could lead to unforseen, unrealistic circumstances.

A primary concern in the design of an interactive simulation system should be ease of user participation in the simulation during execution. To facilitate this ease of use, an appropriate interrupt facility is necessary to allow for suspension of the program at a given point in the program logic and for the transfer of control to the user. The interrupt handler should be flexible enough to process any appropriate request at any time and return control to the point of the interruption upon completion of the handling of the interrupt. This interrupt facility would not only allow input to the model and output from the model but also suspend the simulation to allow detailed examination, decision making and synchronization between simulation time and wall-clock time.

The model should operate in real-time if possible. The term real-time, in the modeler's sense of the term, is not

entirely critical. In modeling terminology real-time is in the sense of wall-clock time. One minute of clock time constitutes one minute of simulation. If simulation of a thirty-minute battle runs in thirty minutes of wall clock time, the simulation is said to run in real time. Real-time in the computer science vernacular is of utmost importance. A computer system is said to be running in real-time if there is a computer program and some other process running "in-step" in such a way that the associated process is not caused to run slower by the computer program. This could be exemplified by the simulation program and the graphics display process. If the simulation program sufficiently slows the interactive graphical input process that the inputs are received after they were needed for use in the simulation, then the computer system is not running in real-time. The interactive capability of the model will be useless if the inputs are not entered in sufficient time to effect the outcome of the battle. It is this real-time that the system must achieve.

Facilities are needed to save the state of the model at any time by executing a single command. Conversely, a command should be available to restore the simulation to a previously saved state and execution resumed from there. Such a capability provides the ability to save intermediate states of the simulation run for the purpose of returning to the previous points in simulation time. This technique is valuable in cases where an unexpected behavior enters the

simulation or an important behavior was bypassed before its presence was discovered [Sohnle 1973].

Since the primary purpose of the game is to function as an analytic tool, any support system must be capable of recording all interactive decisions for use in duplication of runs with alternate modeling parameters. The flow of the battle should also be recorded in order to allow in depth analysis at the conclusion of the simulation run if desired.

There has been little imaginative use of computer graphics as an input/output tool for simulations. Some use of simple plotting has been used but the power of interactive graphics is relatively untouched. Interfaces to graphics languages will permit the modeler to provide more meaningful displays for the simulation user. Input by way of graphics has been grossly underestimated. Special purpose graphics input is a natural means of getting input with today's interactive graphics devices.

The most fundamental capability of the proposed graphical support package is displaying maps of the battlefield. The standard map is a 10 X 10km contour map. This map would be plotted by referencing one of 25 standard map sections. These 25 standard map sections would cover the 50 X 50km battle area. By selecting one of the numbers from 1-25, the appropriate contour map would be drawn using the default contour interval of 100 meters. An alternate mode for plotting maps would be provided to allow the plotting of larger areas. Larger maps are required to monitor such

missions as counter-battery fire or long range surveillance. These larger plots would be called by referencing the lower left corner and the upper right corner utilizing four digit grid coordinates. Since the area to be plotted would be larger (or smaller if so desired) a contour interval must be supplied. Scaling will be performed as a function of maximum and minimum grid coordinates specified.

The contour maps provided are the background for several types of overlays. These overlays may be selected singly or in any combination. The plotting of excessive numbers of overlays may lead to cluttering of the display screen and should be avoided. One such overlay is in support of the dynamic smoke module included in the simulation. The smoke overlay will show the location of smoke or other obscuring elements such as fog or rain. Density of the smoke is indicated by the intensity of the displayed smoke. As the smoke dissipates the intensity decreases. The effect of wind on the smoke is shown by movement of the smoke display across the screen. Results of patrols and reconnaisance flights will be overlayed on the basic contour map. Targets detected by both ground and aerial observers are displayed while under observation. The results of active ground detection will be updated as required by movement of elements. The results of aerial reconnaisance are static in nature after the completion of the flight. Other overlays include the display of road networks, towns, obstacles, both natural and man made, animation of firing events, receipt of

enemy fire, nuclear planning aids and indirect fire.

Dynamic route and position selection will be supported from the graphics terminal. By utilizing a light pen, data tablet, cursor, track ball, joy stick, thumb wheels or some other type input device, the player should be able to interactively input changes or supplimental information into the model during execution. The graphical support package also provides for the monitoring of unit movement through the use of periodic updating of the current unit position. During execution the modeler can select the level of the unit to be plotted. If the modeler selects a unit level other than individual element, the plotting of the unit is performed using the standard military symbol for the unit.

Dynamic movement on the battlefield gives additional realism to the simulation and often discloses information that words cannot convey. Unit movement is represented as it occurs. The actual firing of weapons to include round flight and impact enhance the picture. Any combat introduced visual effects such as smoke from exploding rounds is depicted along with its dissipation and drift. One of the largest benefits of displaying unit movement is having a tool to debug the dynamic route selection module that will be developed for STAR. Unless the modeler has the capability to monitor the route taken by an element, he can never be certain of the performance of dynamic route selection or where that element is located.

Another analytic tool to be furnished the modeler is the

ability to draw line-of-sight (LOS) fans. These LOS fans are used to aid in selection of positions for weapons systems, radars and other LOS dependent systems. Analytically, these LOS fans serve to verify LOS calculations for firing events. The LOS fan will be represented by shading on the contour map, the heavier the shading the greater the visability. A second type of LOS fan will be offered, this being a compliment display that shades the area that cannot be seen.

The support system will incorporate an inquiry capability. Whenever a simulation creates significant output, the statistics collection capabilities of the simulation language may not provide the modeler with the information needed. Statistics collection by a simulation language is often too general and if more detail is required, a dump of the state changes of the model must be analyzed. This inquiry capability supports post execution analytic analysis by enabling specific information to be retrieved and thus avoid searching volumes of data to obtain a single data item. This feature of the war game allows the various players from staff sections to inquire and receive information concerning any data the model maintains that is normally available to that staff member. For example the G-1/S-1 needs information concerning command strength, losses, individual and unit replacements, friendly and enemy prisoners of war (POW), civilian personnel, safety, personnel services, graves registration, casulty reporting, awards and decorations, medical supply and maintenance,

straggler disposition and headquarters movement, security, operation, rear command post location and visitors. The G-2/S-2 is concerned with recommending essential elements of information (EEI), requests for target acquisitions, surveillance, reconnaisance, interrogation of enemy POWs, debriefing, captured enemy documents, signal intelligence, intelligence interpretation, weather, predicting NBC fallout, situation maps, counterintelligence, recommending proposed areas of operations to the G-3/S-3, intelligence training, aggressor forces if employed, civilian-military operations and camouflage. The G-3/S-3 is tasked with insuring the number and types of units assigned to support and accomplish the mission, attachment and detachment of units, organizing and equiping units, training, preparing the operational estimate, integration of fire and maneuver, basic and special loads for weapon systems, priorities for allocating critical resources, coordination and use of airspace, designation of bivouacing areas, recommending general location of the command post, electronic warfare activities, communications and maintaining a current estimate of the situation. The G-4/S-4 is concerned with matters of supply, monitoring the distribution of supplies, supervising the distribution of critical combat weapons and munitions, recommending prescribed loads, managing special weapons, procurement and storage of special weapons, collection and disposal of excess equipment, maintenance, repair parts, evacuation or retrograde of unservicable

equipment, transportation, refueling, construction, property control, food services, use of POWs and civilians and decontamination operations. This data is available in a tabular form similar to figure 2.2.

TABULAR DISPLAYS

"INGRES FLAVOR"

unit	position	#rds left	#rds fired	fuel	#tgts

FIGURE 2.2

The graphics package will include the capability to represent terrain in three-dimensional form. This facility enables the modeler to verify that the shape of the terrain used in the model is in reality a true representation of the actual terrain. Three-dimensional terrain plotting also serves to verify LOS calculations and aids in selection of routes and positions. Through the use of three dimensional terrain, aircraft flight simulation is possible. The viewing screen is capable of displaying terrain as seen from an aircraft. The display includes the terrain, vegetation and

all elements located on the terrain being traversed. Additionally this three-dimensioned terrain feature can be expanded to provide 360 degree scans from any point selected by the user.

The use of color in graphical displays was determined to be of extreme importance. The representation of red and blue forces is an obvious advantage. The ability to use color to represent vegetation lends realism to the display. The number of items to be represented is so large that without color it will be difficult, if not impossible, to distinguish features displayed.

A report writer is of practical importance to the analyst. This report writer will be highly formatted to provide statistics required by the analyst. The user will be able to specify the statistics to be displayed and the table will be generated automatically. Additional hard copy support will be available in the form of a copy device attached to the terminal that may be used to copy the current image on the display device.

Any graphical support system devised would not be complete without providing assistance to the user during execution. Instructions for use must be available, on call, at any time, with various levels of detail for various levels of experience in playing the game. This type of interactive counseling will be provided in the form of a help function. This help function will be provided for two levels of expertise, the novice and the experienced. The

novice user needs information concerning functions available, their formats and their commands. The expert, on the other hand, requires only a reference to the commands available since he is familiar with their formats.

Certain tutorials and guides must be supplied to the user. Directions for position selection using the line of sight fans must be readily available for use during setup of the simulation. A military symbol library should be included with the descriptions of the available symbols and the method of placement of the symbol of the overlay. This description includes details of how the computer decides where the automatically generated symbol is placed. Any graphical support provided for this system must provide for an interactive means with which the user can generate the parametric terrain and verify it against the actual terrain or a map. This terrain package must provide not only assistance in design of the terrain, but also the capability to record the parameters selected for later use in the simulation. It is highly desirable that the user have the means to obtain a hard copy of this terrain generation for this verification process.

E. TYPES OF GRAPHICAL DISPLAYS

The graphical support package will consist of three separate types of applications packages. The first type of support is provided in the form of monitors. These monitors are the devices that provide the selective terrain plots.

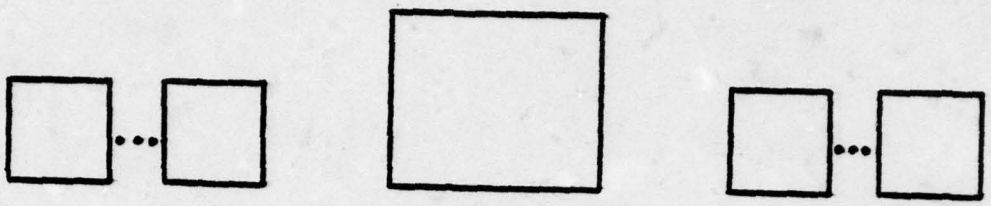
The terrain plots are selected by the user and displayed until the user elects to either terminate the display or request another sector. The display monitor reflects current unit movement located within the terrain sector. The user has limited control over functions of these monitors. These monitors have the basic function of displaying the contour maps. The user will be able to select from a standard set of 10 X 10km contour maps, or he may specify a sector using grid coordinates and the desired contour interval. The user may also specify the unit or element to be displayed. These monitors have a selective zoom feature to allow the user to focus on a given area in greater detail.

A second type of display incorporates the inquiry mode of the support package. These displays will be alpha-numeric terminals. These terminals enable the various staff players to inquire about personnel, logistical and other routine affairs of a unit. Levels of inquiry are controlled by the user. The terminal initiates a hierarchical search with the highest level unit available, giving the option of making the inquiry at this level of resolution or displaying subordinate units at a level one unit lower. If the user elects to make his inquiry, then action is initiated to determine the type inquiry from a menu of possible choices. The information is then displayed and execution continues. Should the user elect to traverse through the hierarchy in a downward direction, the monitor will display the subordinate

units and ask the user to indicate the unit in which he is interested. This is continued until the user reaches the level he desires and the inquiry is initiated.

A third type of display will provide the function of the master graphics console. This console is fully interactive and accomplishes all dynamic changes and selections in the program. This terminal is anticipated to be larger with greater resolution. The capability for drawing in three dimensions is realized at this terminal. All graphical requests are originated at this terminal with the possible exception of the inquiry functions. Inquiries may also be initiated at the alpha-numeric terminals. For ease in selection of the function to be performed, a menu selection technique is used. The user selects, by means of a light-pen or some curser positioning device, the desired function to be performed. This initial selection leads to the fullfillment of the user's request or the display of another menu. In addition to providing the executive routine which manages the execution of the simulation, the monitor provides the ability for the user to interact with the simulation program directly from the terminal. This allows the user to not only observe, verify and record data, but also interrupt the simulation to change parameter values or modify the model structure. Figure 2.3 depicts types of displays.

GRAPHICS DEVICES



MONITORS

INTERACTIVE CONSOLE

TABULAR DISPLAYS "INGRESS FLAVOR"

FIGURE 2.3

III. CONCEPTUAL SOLUTIONS

The problem at hand is to implement graphical support and inquiry capability to an existing war game without serious degradation of performance. This problem is generated by the merging of several capabilities. These capabilities include maintaining a real time environment, sharing of common data items without data redundancy, process synchronization, accurate and timely displays and interactive participation by both the user and programs. The puzzle is to fit these areas into a complete picture. This puzzle is the very essence of this thesis. The basic problem has been simplified by one assumption. The type of computer utilized is irrelevent providing it is capable of performing to the specified standards. The question of mini, maxi or micro implementation revolves around the state of the art at implementation time and the purpose of the computing system itself. If this simulation system is to become highly portable, then the preferred choice may be a microcomputer due to its low cost and portability. If this simulation system is to run "stand-alone" on a dedicated system, then it may be appropriate to develop the system on a minicomputer. If this simulation system is to share time with other programs in a multiprogramming environment, then the appropriate choice may be a large mainframe capable of both multiprogramming and multiprocessing. This thesis

leaves machine choice to the individual implementer.

Solutions to the puzzle fall into four general classes with various degrees of difficulty and performance standards. These choices include a database management system, a tailored operating system, a distributed system and graphic subroutines embedded within the simulation itself. Brief descriptions of the characteristics of these four approaches are discussed in the next four sections. Section E evaluates each approach's ability to implement the simulation system. The preferred solution is chosen in Section F.

A. DATABASE MANAGEMENT SYSTEM

One approach to the problem of supplying graphical support to the STAR model is through the use of a database management system. A database management system is a collection of software procedures, designed to facilitate access to a data base. This data base is shared by diverse users. In this approach the main simulation program, written in SIMSCRIPT, would act as a high-level language applications program. The user would interface with the applications program through the use of any standard alphanumeric terminal. The graphics routines would act as other high-level language applications programs. Since a database management system has the property of being able to present the same data to various users in differing formats, the applications programmers would have their own external

models of the data available to them [Curtice 1976]. This differing view would allow the graphics programs to be written in some language other than SIMSCRIPT since at this time there is no provision for graphical support in SIMSCRIPT.

The actual data that is being manipulated by the applications program is stored in a common area within the computer system. The function of the database management system is to allow the sharing of the data and create data independence to allow for different views of the data. It is the responsibility of the database administrator to develop and maintain the schemas allowing this mapping to occur [Martin 1976].

Database management systems must interface with the operating system in order to accomplish the actual mapping of data into memory. The operating system and the database management system must coexist, but this does not necessarily limit the usage of a database management system. Operating systems are available under which any given database management system may operate. There is an important relationship that must be discussed. Database management systems and operating systems provide the user with a variety of common functions. The database management system design must take into account the services provided by the operating system in order to minimize costly duplication. Some of the most common services provided by operating systems include process management, file-system

support, input and output support and usage measurement [Wiederhold 1977]. Some operating systems also provide facilities for sharing of data segments [Organik 1972].

The use of a database management system has several advantages. The database management system supports multiple users of the system at any given time. Data redundancy is reduced if not eliminated. Applications programs are data independent. The database management system provides internal safeguards for data integrity [Date 1977]. Post-execution analysis is also facilitated. The ability of a user to conduct inquiries is simplified by the adoption of a highly tailored data manipulation language.

A database management system offers a broad range of facilities for organizing, viewing and manipulating information. The creation of data tables by the user requires only a minimum knowledge of the system. Often new tables can be constructed automatically from existing tables on the basis of some format property of the original table [Fram et.al. 1977].

Typically, data manipulation languages are written in English-like commands. With minimal training, a novice user can do useful work on the database management system using the data manipulation language. Many of the more common commands may be invoked in a dialog form in which the system prompts the user for additional data or instructions.

One of the largest disadvantages is the addition of more overhead and hence additional execution time. Part of this

probem may be overcome by the design of a compiled database management system instead of an interpreted one [Wiederhold 1977]. In a typical database management system, a single user command may result in the performance of several physical input or output operations. Each input or output operation must be initiated by the central processor unit. In a multiprocessing environment, this requires the seizing and releasing of the central processor unit several times to carry out a user command, therefore a significant overhead due to task switching may be accrued. The order of increased complexity introduced to the system by the database management system concept must be considered. Database management systems are considered by many to be as complex as some operating systems and hence introduce an additional complexity factor over and beyond the operating system and the basic application programs. Figures 3.1-3.2 diagrams the roll of a Database Management System.

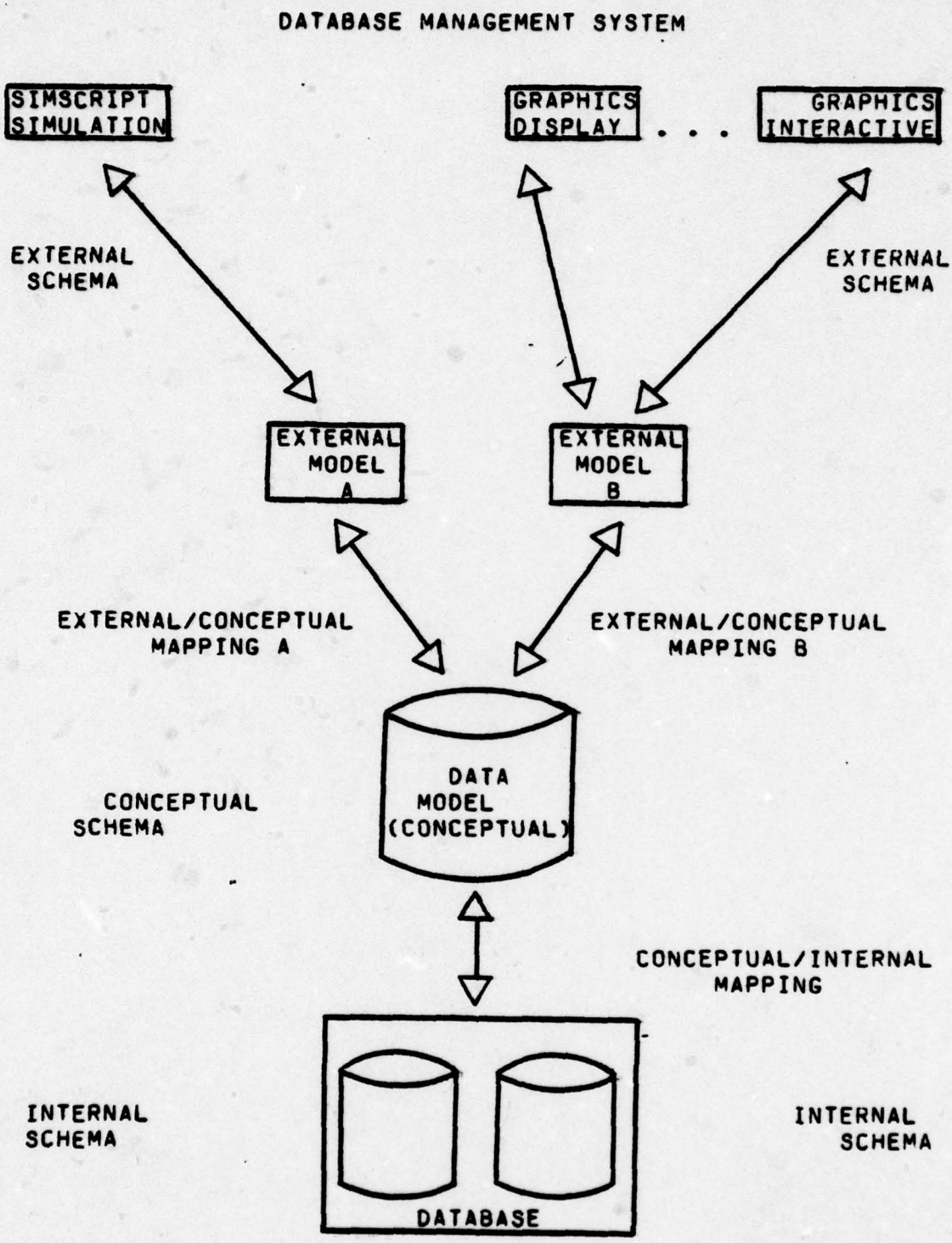

DATABASE MANAGEMENT SYSTEM
SIMSCRIPT SIMULATION
GRAPHICS DISPLAY
. . .
GRAPHICS INTERACTIVE
EXTERNAL SCHEMA
EXTERNAL SCHEMA
EXTERNAL MODEL A
EXTERNAL MODEL B
EXTERNAL/CONCEPTUAL MAPPING A
EXTERNAL/CONCEPTUAL MAPPING B
DATA MODEL (CONCEPTUAL)
CONCEPTUAL SCHEMA
CONCEPTUAL/INTERNAL MAPPING
INTERNAL SCHEMA
INTERNAL SCHEMA
DATABASE

FIGURE 3.1

DATABASE MANAGEMENT SYSTEM

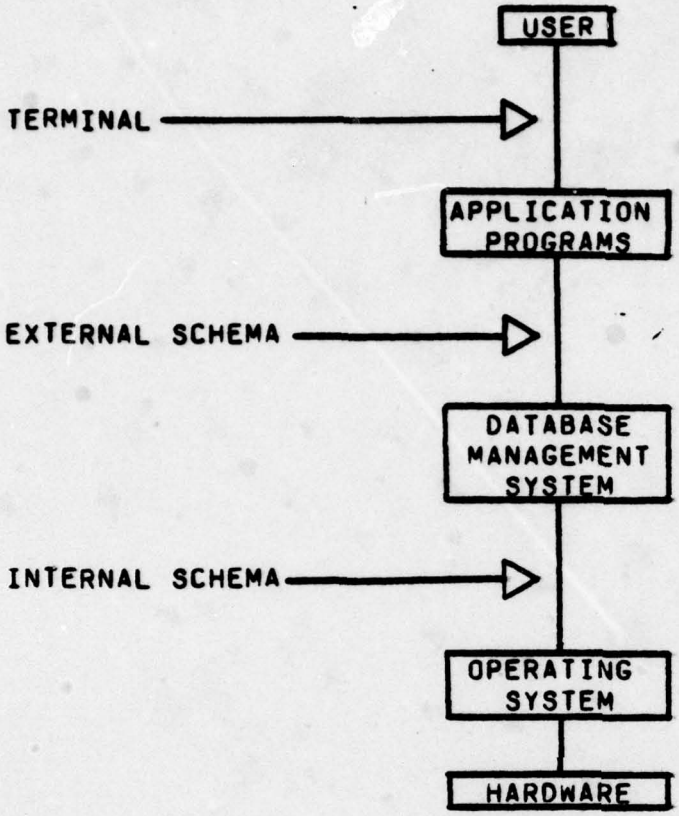


FIGURE 3.2

B. OPERATING SYSTEM

Modern computer hardware is very powerful and may be used for a variety of tasks. The hardware machine is difficult and awkward to use. In order to simplify usage of the bare computer, operating systems have been developed to provide a more hospitable interface with users. Operating systems have become so essential to efficient computer operation that many people view them as inseparable from the

hardware [Madnick and Donovan 1974].

An operating system is a collection of software modules within the computer system that control the operation of the computer. These modules simplify the use of the system, attempt to optimize performance and resolve conflicts within the system. The modules manage the processors, main storage, secondary storage, input/output devices and files. The operating system performs the task of scheduling the use of the computer. Sophisticated operating systems increase the efficiency and subsequently decrease the cost of using a computer.

Operating systems vary in complexity from simple monitor systems on microcomputers to sophisticated large scale systems capable of multiprogramming and multiprocessing while providing protection and interrupt hardware. Regardless of the complexity of the system, all operating systems provide binding for processors and memories. The operating system binds data to physical memory locations and output files to output devices. A process is bound to a processor. The ability to perform binding is fundamental to all operating systems.

Operating systems are somewhat distinct in their ability to provide protection mechanisms [Graham and Denning 1972]. The first level of protection provided by operating systems is common to all reliable systems. This is the protection of the operating system itself from destruction by tampering due to the executing program. This tampering may be

accidental due to the miscalculation of a subscript or some similar mistake, or it may be an intentional effort to sabatoge the system. Whatever the source of the tampering, the operating system must protect itself. A significant difference in operating systems is the ability to protect classified data within the computer system. Some computer systems are secure only in the dedicated mode where only classified material is allowed in the computer and the security perimeter is external to the machine. A more complicated but more useful type of security is the multilevel mode. In the multilevel mode the system may have various users with varying levels of security classification, all competing for system resources simultaneously [Whitmore et.al. 1973]. The security perimeter is internal to the computer and provided by a mechanism of the operating system. Access permission is determined by the operating system. This security mechanism may implement a descretionary or nondescretionary policy.

Some operating systems are capable of multiprogramming. In a multiprogramming environment, several user programs are allowed to compete for system resources simultaneously [Dennis 1965]. It is the function of the operating system to decide which job will be run at any given time. It may be possible for the user to define the priority of the job to the operating system. If this is the case the operating system does not have the problem of enforcing a descretionary priority policy. Determination of the

priority may be left to the operating system. Operating systems use various criteria for establishing priority. Methods include estimated length of execution, estimated storage requirements, estimated execution time and estimated output lines. An operating system may also use a combination of these two techniques. It is left up to the operating system to limit the number of programs the system will accept.

Some operating systems allow multiprocessing [Smith 1977]. In a multiprocessing environment the computer system has multiple processors, each capable of independent operation. It is the responsibility of the central processor unit (CPU) to coordinate or synchronize the functioning of the processors. In a system such as this it will be possible for one processor to be performing calculations while another processor is controlling output to the line printer.

The operating system not only provides memory management in the form of binding but also is capable of creating virtual memory. Virtual memory allows the address space of the program being executed to be either greater than or less the the physical memory of the machine. Utilizing this virtual memory, the user need not be limited by the physical size of the main memory [Denning 1970].

Another feature of operating systems is their ability to handle interrupts. Through the use of an interrupt handler, the operating system may accept an interrupt, process it

according to the instructions in the interrupt handler and return execution to the program in progress at the point of the interrupt. Operating systems have system defined interrupt handlers but many have provisions for the user to define his own interrupt handlers.

C. DISTRIBUTED SYSTEM

A distributed system is a computer system composed of multiple central processors that cooperate in problem solving. These CPU's may be spread over many miles or located in the same room. In order for the distributed system to function, coordination between the processors is accomplished by a distributed operating system.

The problem of extending the execution time of the model might be alleviated by the concept of a distributed computing system composed of one computer to process the simulation portion and maintain a master data base and a smaller computer providing the graphics and inquiry capabilities. A distributed system has the chacteristic that the functions are distributed or spread over multiple CPU's each designated to handle a particular function. This approach becomes advantageous by using a second processor to reduce the work load of the main CPU. Consider the case of a front-end processor connected to a main processor as indicated by figure 3.3.

DISTRIBUTED SYSTEM

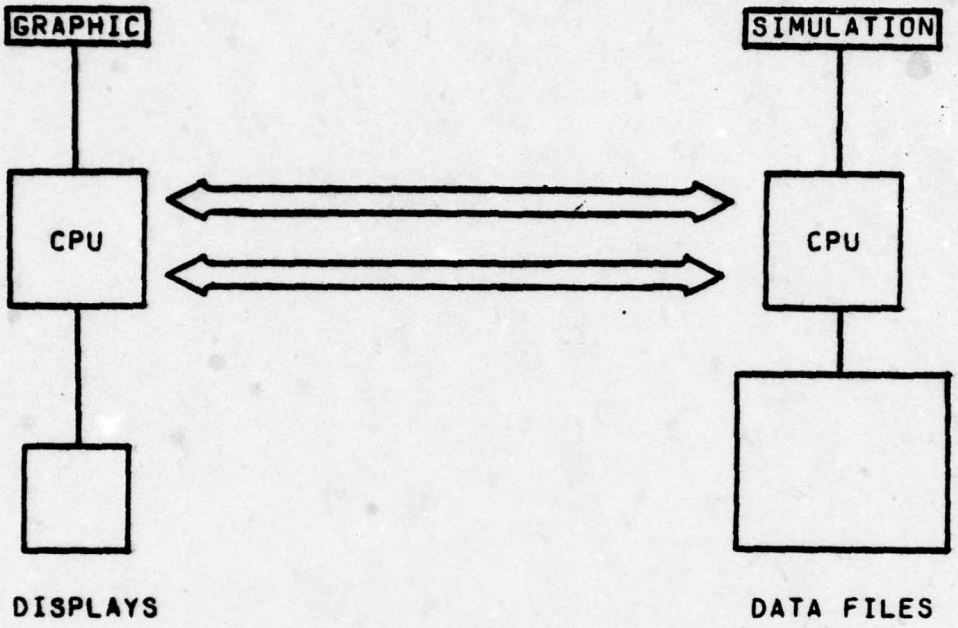


FIGURE 3.3

The front-end processor controls the interface with the user. Graphical displays and user command interpretation including editing are performed on the front-end processor. This allows the power of the main processor to be devoted to the computation bound simulation functions. In this case the main processor would have to pass the required display data to the graphics processor as needed. Assuming that this would take a small amount of time under CPU to CPU communication with a communication line of high transfer rate equal to the slower rate of the two processors, the

main processor could continue to simulate while the graphics processor generates the display list and causes the display to occur. This would have the effect of halting the simulation for only a minimum time frame and thereby not significantly degrade the system. Should it be desirable to stop the simulation for some update of information from the graphics processor, an interrupt could be generated by the graphics processor and sent to the main processor which would then halt the simulation to receive the update. The problem of maintaining data integrity emerges from the aforementioned situation. There is no way to determine if the data to be changed exists at the time of update or if the data displayed is current. This problem will have to be resolved by generating an update request, displaying the current status of the battle area and then updating the data. This must be handled through some automated means so that the user is not confronted with the problem, he has enough to be concerned with without the system complicating the situation for him.

The distributed system functions as follows. First there must be established priorities that each CPU follows. For instance, on the simulation CPU, communication between CPU's has a higher priority than the simulation and can be represented by the following pseudo code,"while (not communicating) do simulation". This action will give priority to CPU-CPU communication, allowing the user's inquiries to be answered more rapidly. The graphics CPU

continuously tests the terminals for the indicator that the user desires to perform some function. This can be also accomplished through an interrupt mechanism. Upon identification of the user's desired function, an argument list is constructed in conjunction with additional user supplied data, if required, and an interrupt is generated to the simulation CPU indicating that the graphics CPU desires to communicate. Upon receipt of the argument list the simulation CPU stops the simulation, stores the machine state and executes a "case" structure similiar to:

```
switch(function-id);
     {
     case(t): terrain information;
     case(i): inquiry;
     case(u): update;
      .
      .
      .
     }
```

passing back to the graphics CPU any resulting information. The graphics CPU now processes the information received from the host CPU and continues the function originally identified by the user, such as producing a display.

D. EMBEDDED GRAPHICS

The simplest and perhaps the most direct implementation of the desired capabilities is the execution of the graphic capability from a direct subroutine call from the SIMSCRIPT simulation program. Figure 3.4 depicts the embedded graphics approach.

SUBROUTINE CALL

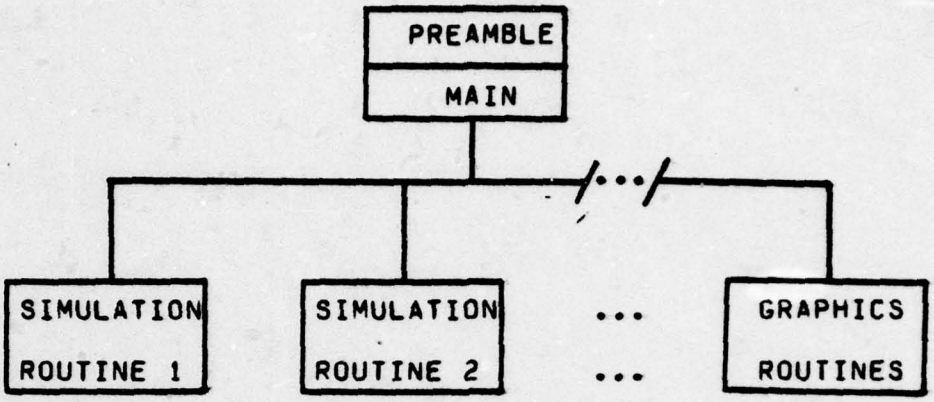


FIGURE 3.4

There are several advantages to this approach. The graphics package can be developed separately from the simulation model, keeping in mind that necessary parameters required by the display must be passed from the simulation program to the graphics subroutine. A simple driver could minic the functions of the call to the graphics routine during the development of the graphics package. In the same way, should the graphics subroutine provide the interface with the user for the interactive portions of the model, certain parameters would have to be returned to the simulation program. These parameters must define the type of function that the user desires to perform along with any function parameters required. The values of the parameters could be established through interpretation of light pen input from

the user. This interpretation could take the form of a CASE or nested IF type structure where parameter values would be established from an interpretation or series of interpretations of the light pen input. Once the parameter list is formed the graphics subroutine is called.

There are disadvantages to this concept. Due to the nature of subroutine calls, the action of the simulation program is at a standstill until the execution of the call is completed. This will increase the execution time of the simulation program and thereby increase the wall-clock time required to simulate any given battle. This problem ceases to retain great importance if it is desirable that the simulation be halted to allow any update information to be passed to the simulation process and thus maintain data integrity.

Because data integrity is a requirement of the system, the graphical displays must be capable of depicting the exact state of the simulation upon the display device. To change the route of march of a particular unit or element that item must be located in the depicted position at the time of the update or the exact position must be known to the user.

E. EVALUATION OF ALTERNATIVES

1. Common Considerations

There are several items that are common to all approaches. Included in these items are the use of color

displays, hard copy devices and the treatment of static data such as contour maps. Hard copy devices are required to record selective displays upon command of the user. Contour maps are thought of as any required data to enable a rapid draw of the desired area of terrain. Once the parametric terrain data is constructed, prior to the simulation execution and during some system initialization procedures, there is no need for the simulation process to have access to it since the simulation routines compute any required terrain data dynamically during execution. There is only the need to have this data available to the graphics display routines. The impact of a hard copy device upon the solution is seen as device dependent and therefore not of concern in the selection process. In the same manner the method of drawing contour maps is device dependent and the use of color is independent of implementation. These two factors are also omitted from the evaluation process. This evaluation focuses on the ability or inability of the alternative to support the simulation, evaluating failures on the lowest possible level.

2. Database Management System

In the database management system approach, the simulation program assumes the role of a high-level language applications program. All graphical routines except the inquiry program are also high-level applications programs. The inquiry program is an applications program written in a tailored data manipulation language.

A database management system is designed to allow the sharing of data and eliminate data duplication. Through the design of appropriate schemas, the database designer is able to present each applications programmer with an independent view of the data and allow the programmer to access any data within the data base. This approach presents a problem with multiple users attempting to write data simultaneously. This problem can be minimized through careful design and judiciously granting write access to shared data. Should two users desire to write data to the same file, the last copy written will prevail.

The recording of dynamic events presents a significant problem to the database management system approach. The ability of the system to accept and store input data from the user is routine to a database system. Anything that can be input and stored may also be recorded on secondary storage media. The significant problem arises when the machine state must be saved. Only the operating system has the capability to monitor and modify all the registers in the machine. For the database management system to save the machine state, close cooperation with the operating system is required. When it is desired to return to a decision point to resume execution with another decision, the database management system must relinquish control to the operating system while the operating system restores the values of the variables and the machine state.

Flexibility of play will require the database

management system to maintain some mechanism to generate the appropriate data structure for the level of play. The flexibility of play is not an insurmountable problem but the mechanism to achieve this goal may be quite complicated. The flexibility of display will be quite easily attained since display is dependent only on the data stored once the data structure for storing the data has been created. The ability to store various force structures must also be attained by some dynamic means since the actual size of the data structure required will not be known until run time.

The degree of interactive programming attained by the database management system will vary from routine to routine. The inquiry routines will have the full interactive characteristics of any database management system. The programs written in high-level languages are limited by the degree of interaction provided by the corresponding languages. The database management system has no means of incorporating interrupt driven processing. Interrupt driven processing requires action by the operating system and therefore a close relationship between the database management system and the operating system.

The database management system approach will adversely affect the real-time capability of the program. Overhead in a database management system is extensive. The desirable trait of data independence requires the additional cost of address translation. Various references to the same data element require the system to translate these

references into the same physical memory location. Additional overhead in execution time is required by the necessity to translate or compile input requests during execution.

The report writer is facilitated by the database management system approach. The database design allows for the user to request information in a standard format and have it displayed for him on the screen. Any information stored may be displayed as well as any combination of data items that may be created using relational calculus or standard boolean operators.

3. Operating System

Under the operating system concept, the simulation program becomes one of many in a multiprogramming environment. The graphics routines are organized into programs with related functions. These graphics programs become additional programs that will compete with all other programs for system resources.

The problem of sharing files is not significant to an operating system that uses segmentation. Any progran division that is important enough to be named may be created as a segment. In a system supporting this segmentation, any segment may be addressed by potentially any processor. By careful designation of the ability to read and write to a given segment, it is possible to allow a segment that is responsible for a file to create the file and to allow a segment that must use that data for display or other

purposes to access the data and use it. These segments that are sharing the data need not be in the same program. The programs need only be active at the time of sharing of the data. One potential problem may arise if more than one segment is allowed to write to any one data segment. In this case the last segment to write will be the segment that dictates the data value written. By carefull design of the programs involved, this problem may be made insignificant.

The ability to record dynamic events such as decisions by the user and simulation status present no significant problem for the operating system. At the time the user inputs his decision, the operating system needs to write the input data into the appropriate area in memory to affect the simulation. At this same time the operating system will make a copy of the decision information along with the machine state and any pertinate variable values before the decision is made. This additional information may be written to some secondary storage medium for use at a later time. At a later time when it is desired to return to a given decision point and change or modify the previous decision, the operating system has all the information needed to restore variable values and restore the machine state. Execution may then resume from the point of decision rather than requiring the entire simulation to be executed again.

In the area of flexibility, the operating system approach presents no problem. It is the normal function of

some operating systems to allocate storage for problem elements. At execution time the operating system will allocate storage as required by the simulation program.

The area of interactive programming is affected by the interactive capabilities of the programming language. These built in capabilities are the base level for the simulation. Further interactive capabilities may be provided by the operating system. For a system to be genuinely interactive, it is necessary for the system to be interrupt driven. In an interrupt driven system, the user generates an interrupt and the operating system then transfers control to the appropriate interrupt handler. The instructions in this interrupt handler dictate the response to the interrupt. Operating systems allow the user to write his own interrupt handlers to either supplement or replace the system provided handlers. In the event the user elects not to provide his own handler, the operating system provides default handlers. By anticipating the required types of interrupts and the appropriate responses, the user may effectively interrupt the execution, create a display or input data, and return to the point of interruption and continue execution.

The operating system approach may enhance the real time capability of the simulation. A multiprocessing environment allows the operating system to perform computation on one process while simultaneously performing input/output, paging or some other operation on another

process. This means the programs of the simulation may fully utilize the processors of the computer system and the processors need not be idle while a single program switches from one processor to another leaving the rest of the processors idle until the simulation needs its services. As a worst case, the operating system will add no more execution time to the program. The operating system is needed to provide user interface to the bare machine and hence is already present as overhead to any program run on the machine.

The post analysis report writer is still another program to exist in the multiprogramming environment. The operating system keeps the segments belonging to all simulation programs on call until the report writer completes its usage of the data. Once the report writer concludes execution, the system is allowed to free storage for other usage.

4. Distributed System

The envisioned solution would have two central processors. The simulation functions will reside on the main processor due to it's computation bound characteristics, while the graphics and inquiry functions will reside under the control of another possibly smaller processor.

Since the main CPU is charged with the responsibility of maintaining the master data base, there can be no sharing of data items between processors. The graphics processor must receive all data items that are dynamically changing.

It must also interpret all user commands and generate a CPU to CPU request for the data items necessary to fulfill the user's request. Any attempt to store these ever changing items is fruitless and will result in either duplicated files or excessive data transmission between the two CPUs. The request for data must be be generated on an interrupt basis to the main processor so that the excessive data transmission and data redundancy situations do not occur.

Dynamic event recording must be accomplished on both processors to enable system to be restarted at any specified state. The graphics processor will be required to store user commands and decisions, machine state and perhaps display generation data. The main CPU will be tasked with saving its machine state and all variable values for the simulation restart.

Flexibility of the system now spreads over the two processors. Not only must the implementer be concerned with the mechanism for level of play, level of display and size of forces represented but also the corresponding volume of data transmission between the two processors. This is an added concern in the distributed approach.

Since interactive play is desired, the graphics processor must interpret the user's commands in an interactive role and also generate an appropriate interrupt to the main processor for each type of request. This will require a user written interrupt handler on the main processor to decipher the interrupt and process it. The

interrupt handler will probably not be on the operating system level and thereby will cause additional delay prior to processing it.

The idea of distributing the STAR system functions to two processors was conceived to solve the real-time requirement problem. Certainly the distributed system would run closer to real-time than a subroutine call system. The required CPU to CPU communication will generate some overhead that other approaches do not. The user must see the current situation status displays and his participation must be received in time to accurately effect the outcome of the battle.

The problem of where to locate the report writer for post-execution evaluation arises. It should be capable of providing the user with his requirements at the location generating the displays. This means that the main processor must either continue to function only to pass data to the graphics processor for this purpose or create a file that is readable by the graphics processor during this phase. Once again additional overhead is required to accomplish both. In the first case additional execution time is required by the main to retrieve, process and transmit data to the graphics processor. In the latter case additional time is required to create the readable file should the two processors expect different file characteristics. The overhead of generating the file remains in either case and under all concepts discussed, however such record and file attributes as

header, trailer, inter-record gaps, blocking characteristics and character set will present a problem should two different hardware vendors be chosen for the two processors.

Since unchanging data items, such as terrain and road networks, exist during the execution of any given simulation, they are stored on the graphics side of the system originally and do not require the passing of large data structures as that of terrain representation. This is possible due to the use of parameterized terrain generation capability of the simulation to produce terrain elevations at any given point on the battle field.

5. Embedded Graphics

In this approach the simulation program is the center of control over all desired functions. The basic functions of graphical display, inquiry and update are fullfiled through calls to appropriate subroutines.

SIMSCRIPT uses a basic technique of executing subroutine calls from the timing-routine. This technique selects the next event to transpire from the event list. These events were previously scheduled by other subroutines in the simulation (internal) or received as input (external).

The sharing of information between the three basic functions (simulation, graphics and inquiry) presents no problem because the required common data can be stored in global variables or passed as arguments between the calling and called subprograms. Data integrity also presents no particular problem since only one subprogram may be in

execution at any one time unless some type of parallel processor is utilized. The same argument applies when one subprogram updates a variable value that another uses.

The recording of dynamic events can be easily accomplished except for retention of the machine state. Since machine state is important from the standpoint of restarting the process from a specified state, an assembler level subprogram is required to periodically save the necessary information on some secondary storage medium. Routines will also have to be developed to retain the decisions reached and periodic simulation state, however these can be written in the basic simulation language since all required information is defined to the simulation program.

Flexibility of the system for level of play, level of display and size of forces must be designed into the subprograms but may be realized through dynamic initialization of key execution parameters. The structure to allow this must be incorporated into the subprograms so that the maximum allowable force sizes can be allowed.

Interactive play can be achieved through careful design and implementation. A subprogram to periodically test display terminals for user participation is required. This subprogram will interpret the user's desired functions and schedule the compatable event which is stored in the timing-routine event list in time sequence. These events may be scheduled NOW, IN 10 MINUTES, or AT 1430 HOURS,

however, NOW does not imply instantaneous execution of the function since other previously scheduled events for the same time could exist with a higher statically defined priority.

Maintaining a real-time environment is not hindered by this approach when considering effect on the outcome of the simulation, however this approach will extend the execution time required for the battle. Should the user desire to update the simulation data during the execution, the simulation process is halted by the resident operating system until control is returned. The user need only schedule a display and an update NOW or at the same time with the display event having the next highest priority to the update event. During execution the display will be produced, showing the current simulation status and then the update event will execute.

The report-writer enhancement presents some difficulties particularly during post-simulation evaluation. Since the execution of the simulation has been terminated, a separate application program is required to process the saved data for the user.

Although the subroutine call approach is the simplest to implement, it does have the drawbacks indicated. This approach will have several beneficial side-effects. The graphic display is scheduled along with all other events and is placed in the event list in the appropriate time sequence. A display event can be generated with the SCHEDULE

A GRAPHICS.DISPLAY NOW, SCHEDULE A GRAPHICS.DISPLAY IN 10 MINUTES (DAYS or UNITS) or SCHEDULE A GRAPHICS.DISPLAY AT 1400 HOURS. This side-effect gives built-in flexibility to the scheduling of a display. The "GRAPHICS.DISPLAY" event includes initiation of inquiries and updates to the data base as well. Figure 3.5 depicts sample SIMSCRIPT code for this type of subroutine call.

```
PREAMBLE
  .
  .
  .
     EVENT NOTICES INCLUDE STOP.SIMULATION
          EVERY LOC.UPDATE HAS A VEHICLE
          EVERY LOS.UPDATE HAS A WHEEL
          EVERY   DETECT   HAS   A    WHOLE.TANK    AND    A
          DETECT.FOE.ENTIRE.TANK
          EVERY TARGET.SELECT HAS A FIRING.TANK
          EVERY   FIRE   HAS   A   SHOOTING.TANK    AND    A
          CORE.POINTER.OR.TGT.ID
          EVERY  IMPACT   HAS   A   TANK.THAT.SHOT   AND   A
          BLOCK.POINTER.OR.TGT.ID
          EVERY  GRAPHICS.DISPLAY  HAS  A   COMMAND.ID   AND
          ADDRESS.OF.PARAMETER.LIST
END


MAIN
       .
       .
       .
     CREATE A PARAMETER.LIST
          LET ATTRIBUTE1(PARAMETER.LIST) = value1
          LET ATTRIBUTE2(PARAMETER.LIST) = value2
            .
            .
            .
     SCHEDULE A GRAPHICS.DISPLAY NOW
          LET ADDRESS.OF.PARAMETER.LIST(GRAPHICS.DISPLAY)  =
          PARAMETER.LIST
END


EVENT GRAPHICS.DISPLAY
  .
  .
  .
     IF COMMAND.ID(GRAPHICS.DISPLAY) = 'I'
          PERFORM                INQUIRY                GIVEN
          ADDRESS.OF.PARAMETER.LIST(GRAPHICS.DISPLAY)
     IF COMMAND.ID(GRAPHICS.DISPLAY) = 'DT'
          PERFORM              TERRAIN.PLOT             GIVEN
          ADDRESS.OF.PARAMETER.LIST(GRAPHICS.DISPLAY)
       .
       .
       .
END
```

FIGURE 3.5

F. PREFERRED SOLUTION

In the selection of the preferred alternative, those items designated as common considerations play an insignificant role. The use of color to enhance the clarity of the displays is not envisioned to introduce an additional burden on any solution. The method used to rapidly produce a contour map is device dependent and not dependent on the preferred solution. Hard copy devices depend on machine interfaces and are therefore implementation independent.

It is possible for all alternative solutions to accomplish the sharing of date files. Database management systems are designed with this goal. Database management systems produce a data independence that allows each user to view the data in his own way. Operating systems that support segmentation are also capable of supporting the sharing of data. The operating system however shares the data in a format specific manner. The distributed approach allows sharing of data files between the central processors through the use of a distributed operating system that allows CPU to CPU communication. The subroutine call approach uses common data elements through parameter passing techniques and global variables.

Dynamic event recording is the first area in which the four approaches differ significantly in their ability to perform. All approaches are capable of recording decisions and saving the values of variables at the time of the decision. It is not a normal database management function

to record and later restore the state of the machine registers. This interface with the machine must be made with the cooperation of the operating system. The operating system approach, on the other hand, has the interface with the machine registers as part of its normal operations. The distributed approach is further complicated by the need to save the state of multiple CPU's and variable values in multiple machine memories. The subroutine call approach suffers from the inability to directly access machine registers in much the same manner as the database management system.

Flexibility in the level of display and the level of play must be discussed in two aspects. Level of display is a function of the level of play in the fact that the only possible items to be displayed are those items that are stored. The routines written to cause the display will be similar for all approaches. This leaves the area of flexibility as a function of level of play. The area of flexibility of play hinges on the ability to generate and store the appropriate data structure. This poses a problem to the database management system approach in that the system must dynamically generate the data structure at execution time. The generation of the maximum size data structure at every execution of the simulation would be wasteful of storage. The operating system approach is not hindered by the flexibility constraint. The operating system will automatically reserve storage for the data

structure specified by the simulation program. The distributed approach is similar to the operating system approach in that the operating system of the distributed system will allocate storage as required by the corresponding programs. The subroutine call approach is unaffected by the flexibility of play. By changing input parameters, the user may dictate the size and structure of the units participating in the simulation.

Interactive programming for the database management alternative is broken into two areas. The inquiry mode of the database system is limited only by the database designer. When the user asks and what he is allowed to ask are design considerations. All interactive capabilities other than the inquiries are limited by the programming language concerned. The subroutine call approach is also limited by programming languages. The operating system approach is capable of fully interrupt driven processing. The ability to interact is enhanced by the users ability to write interrupt handlers to respond to user generated interrupts. The distributed system's user programs are also limited by the chosen programming language.

Real-time processing is hindered by the database management alternative. The system must translate all data references into the appropriate addresses in order to complete the data references. The operating system approach does not affect the real-time ability of the simulation. The distributed system will slow the simulation due to the

time required for CPU to CPU communication. The greatest slow down will be for the subroutine call alternative since all processing must stop in the simulation while the subroutine creates the display data.

From the above discussion the operating system approach is the only alternative that satisfies all the requirements for the system. There are several other considerations that favor the operating system approach. In the operating system approach the key is the sharing of data. The key data to be shared is generated by the existing simulation program. Further program development to share this existing data may be done independently without adversely affecting the existing program.

Another consideration is the availability of a system to support the simulation. Both the database management system and the operating system approaches depend upon a complicated programming effort. A tailored database management system would have to be written and at best be an experimental system with unknown efficiency. On the other hand, general purpose operating systems capable of supporting this simulation system have been written and tested. These proven systems are available today.

It is anticipated that the simulation wil be run with classified input data. All solutions to the problem, except the operating system approach, delegate the problem of security to an external mechanism. Some operating systems are capable of providing security for the classified data

without depending on an external mechanism.

The last consideration is the complexity of the solution. The operating system approach places the entire burden on the operating system to perform tasks beyond the capability of the programming language. The database management system requires a complicated relationship between the database system and the operating system since the database management system is unable to fulfill all the requirements. The operating system is the only alternative to perform all requirements in a stand-alone basis.

IV. SOLUTION ANALYSIS

System analysis examines the existing and proposed software and produces the logical design for the system. It is in this phase that the inter-relationships between modules are examined , including determination of both coordination and synchronization of modules. The proposed simulation system is composed of four programs existing in a mutually cooperating environment. The first and most important of these four programs is the monitor program which is the master program that coordinates the use of the system and is capable of initiating any of the capabilities of the system. A second program is the simulation program itself. This program is the current version of STAR. The third program is a graphics program that produces all maps and overlays to the maps. The last program contains all administrative routines such as report writers, inquiry operations, all tutorials and assistance functions. The operating system coordinates these programs and converts isolated programs into the simulation system described in Chapter II.

A. OPERATING SYSTEM REQUIREMENTS

In order for an operating system to properly implement the simulation system, it must have a number of control features. These required features fall into the four functional areas of segmented memory, multiprocessing,

synchronization of processes and segment isolation.

1. Segmented Memory

The sharing of data between user programs is required to implement the STAR model. The simulation generates a data file that is displayed by the various graphics routines. This data file is accessed by the routines that display dynamic movement, animate weapon firing, display impact of rounds, and display unit positions as well as detected enemy positions. This data must not only be accessed but also be updated by the routines that enable dynamic route and position selection and support the inquiry functions. An operating system capable of segmented memory allows this sharing to take place [Daley and Dennis 1968].

A segment is a collection of information that is important enough to be given a name. A segment is the basic unit of sharing. Associated with a segment is a collection of attributes, including a unique identifier and an Access Control List. The Access Control List maintains information specifying the processes that may access that segment and whether the authorized access includes any combination of read, write or execute permission. Each segment may consists of up to six major parts. The text section contains the pure unmodified parts of the object code which would contain the program constants as specified in the SIMSCRIPT PREAMBLE. The definition section contains nonexecutable information used by system programmers in debugging and by the operating system in dynamic linking.

The linkage section contains the impure, modifiable parts of the object code and may be made up of two types of data. The links used to establish addresses at run time are the first type of entry and since the memory is demand paged, these addresses may change. The second type of data is the data items from the program that will be modified during execution. All variables will be stored here. The static section may also be used for storage on a per process basis alternately this storage may be included in the linkage section. A break map section contains information used by debuggers. The last section, the symbol section, contains anything generated that is not stored elsewhere [Honeywell 1975].

All segments that are competing for system resources are listed in a system-wide Active Segment Table. This table allows the operating system to know which segments are currently active and where they are located in memory. Table length is finite which requires the operating system to limit the number of segments capable of competing for system resources. This limiting of processes reduces the amount of time lost due to the switching of processors from one process to another.

2. Process States

A process is a set of related procedures and data undergoing execution and manipulation. Processes will generally contain one or more procedure segments, one or more data segments, a stack segment and a linkage segment

for each procedure segment included in the process. Processes within a computer fall into one of three execution states. A process is said to be running if it is currently executing on a processor. A process is ready if it would be running should a processor be available. A process is waiting if it cannot make immediate use of a processor since it is waiting for some external (to the process) event. The operating system keeps track of these processes in the system-wide Active Process Table. These three types of processes in the Active Process Table require synchronized use of shared segments.

3. Synchronization of Processes

In order to synchronize processes some method of communication between the processes must exist. This interprocess communication is monitored by a mechanism known as the traffic controller which functions as a general purpose supervisor for control of parallel operations [Daley and Dennis 1968]. Two of the more interesting mechanisms provided by the traffic controller are the block and wakeup functions. The block function forces a process to wait for an occurrance of an event generated by some other process while the wakeup function allows the process to be notified that the event has occurred and processing may resume. This blocking of a process is recorded in the Active Process Table.

Another mechanism used in synchronization is a condition handler. Users and processes may communicate with

processes through the use of condition handlers. Condition handling refers to an activity resulting from a hardware or software condition named by the user's program. The user may implicitly or explicitly identify the code to be executed in response to the condition. To initiate user interaction with the simulation, the programmer specifies the pressing of the ATTN key on the terminal keyboard as a hardware condition. In response to this condition, execution control is transferred to a condition handler which contains code to request the type of action required by the user. The user performs his desired interaction and the simulation resumes execution. Condition handling need not be a rigid response to the specified condition. The programmer has the option to specify condition handlers on a segment by segment basis since condition handlers are pushed onto a stack as they are defined and popped from the stack when the procedure segment for which they are defined is exited. In this manner, the programmer may specify a global condition handler as the general response to a given condition and redefine the response to the condition on a procedure by procedure basis should the response change for each particular environment. Should the response to the condition be the activation of another procedure, the condition handler then becomes another of the deliberately cooperating procedures.

Deliberately cooperating programs or procedures may interact through the use of interrupts which are synchronous

events internal to the machine. When one procedure desires to call another procedure, the calling procedure generates an interrupt. The key to flexibility in interrupt handling is to convert this interrupt to a wakeup. The interrupt is sent to the interprocess communication controller which in turn sends a wakeup to the called procedure. This called procedure is then activated and execution may resume. There is no problem with simultaneous use of a procedure by multiple calling procedures since all references to the called procedure are made via the calling procedure's linkage segment. Once the called procedure has completed execution, the called procedure places itself in a blocked status to await further use [Graham and Denning 1972].

Coordinated sharing of writable data segments may be handled through a lock mechanism provided by the operating system. This lock mechanism is applied to a data segment whenever that segment is being utilized by a process capable of modifying its contents. Further attempts to reference a locked segment will be denied until the segment is unlocked. The lock mechanism blocks the process that is attempting to use data segment and maintains the identification of the requesting process in a list structure. Once the segment has been updated, the locking mechanism sends a wakeup to the next process selected to use the data, unlocking the data segment for that process. This procedure is repeated until all requests are fulfilled and the data segment is unlocked to await future use.

4. Isolation Techniques

Interprocess isolation is accomplished through the Access Control List discussed in section A-1 of this Chapter. Intraprocess isolation is implemented through the use of concentric protection rings where a ring bracket is associated with each segment. The ring bracket is an ordered triple <R1,R2,R3> which specifies the access bracket <R1,R2> and the call bracket <R2,R3>. These two subsets of the ring bracket dictate the read, write, execute and call access for that segment. A procedure may execute in any ring from R1 to R2 inclusive and may be called by any procedure in rings (R2 + 1) to R3 inclusive, provided the calling procedure has sufficient security clearance. A data segment commonly has R2 equal to R3 since calling a data segment has no meaning. A procedure segment may write to a data segment in rings 0 to R1 inclusive and read from a data segment residing in the region 0 to R2 inclusive. Again, read and write access are denied to any segment in any ring that does not have sufficient access granted in the Access Control List.

A segment may also have a security level associated with it. This security level is composed of two parts, security classification and security category [Schiller 1975]. The security classification is a type of compartmentalization similar to the Department of Defense security classifications secret, confidential, etc. This security classification is a totally ordered set where

secret is strictly greater than confidential and so forth. The second part of the security level is the security category which is a modifier to the security classification and analogis to the Department of Defense categories of crypto, NATO, etc. In addition to access granted by the Access Control List, procedures must also have an appropriate security level in order to gain access.

B. ANALYSIS OF ENHANCEMENTS.

Certain design characteristics must be followed in the design of all software for the proposed system. All software developed should be easily portable, avoiding locally produced library functions since they may not exist at another installation. All modules should be human engineered to permit easy use. Operator inputs should be minimal and concise with default values provided for all modes, parameters and variables. These defaults lessen the burden on the user and facilitate the requirement for minimal input. Additionally, defaults reduce the number of user errors. Maintenance responsibility for the software must be charged to a specified individual or group of knowledgeable individuals. All graphics routines developed should comply with the new graphics standards under development by the American National Standards Institute [Newman and van Dam 1978].

In addition to the existing simulation program, several new modules and separate programs must be written to achieve

the type of interactive simulation described in Chapter II. These additional modules and programs include all graphical displays, inquiry, synchronization and report generator functions.

The choice of utilizing the operating system approach to implement the simulation system has facilitated the programming effort required. The area of flexibility of play no longer concerns the programmer. Storage for variables is automatically allocated as a normal function of the operating system. The programmer does not need to concern himself with an elaborate data structure that is capable of growth since this burden is assumed by the operating system. Flexibility of display becomes a problem of searching for all the elements of the unit being displayed and then using some appropriate weighting factor to properly position the unit symbol.

Programs may be developed independently of the simulation by using simulated data files and therefore not hinder the use of the simulation or require duplicate copies of the simulation program for developmental purposes. When the additional programs are tested and pronounced ready for use, the only action required is to inform the operating system to allow access to the shared data files. Synchronization of use of these shared files is required but mechanisms discussed in section A-3 of this chapter allow this synchronization to occur.

1. Interactive Programming

The area of interactive programming may be broken into two sections. The user may interact with the simulation program as well as the individual simulation programs interacting with each other. The case of the user interacting with the simulation may be satisfied by the use of condition handlers while individual programs cooperating with one another may be accomplished through the use of interrupt handlers.

An example of condition handling may be the implementation of dynamic route selection. When the light pen is activated, a hardware condition occurs and execution control is transferred to the condition handler. The condition handler sends a wakeup to an updating procedure. This updating procedure accesses the data and places a lock on the data segment. This lock prohibits the use of the data while it is being updated. When updating is completed, the lock is removed, the update procedure places itself in a blocked status to await its next wakeup message, the condition handler is exited and execution resumes.

Use of coordination by interrupt handling may be seen in the dynamic updating of positions. When the movement module changes the location of the unit, an interrupt is generated as the movement module is exited. This interrupt is converted into a wakeup that is sent to the display routine. The display routine creates the new display and then places itself in blocked status to await the next position change. This conversion of interrupts to wakeups

not only facititates display but also allows the procedure to be active only as required. Extraneous nonexecuting simulation routines need not be in primary storage until required for use and display routines need not continuously draw the same picture in order to prevent missing an event. Displays may now be made only as change occurs.

2. Real-time

Real-time must be approached from both the computer science and modeling definitions. The computer science real time is accomplished by the synchronization mechanisms discussed in section A-3. The various programs and procedures are forced to run in-step since they are called by wakeups from the main simulation on an as required basis. This system will have least overall detriment to the simulation in the modeling real-time sense. Memory management may prove to slow the simulation from paging activities but proper selection of the program's working set size will reduce these paging delays to a minimum [Denning 1968]. A clever operating system will have facitities for maintaining the current working set size as a program executes. Associative and cache memories will also speed up execution of the simulation execution due to high speed address translation and reference [Schroeder 1971]. The mechanisms to synchronize segment usage may cause some slowing while procedures are in a blocked status. This slowdown will be overcome since separate programs will be allowed to execute simultaneously in a multiprogramming and

multiprocessing environment. An example of synchronization of segment usage is the calculating line of sight while the display of current positions is being produced by another processor.

3. Monitor Program

The monitor program provides a master control facility for the modeler. This monitor program provides the main condition handler for the simulation. From the master console, any capability of the simulation system may be called at any time. Any privileged instructions available to the modeler but not the war gamer will be executable from this terminal.

The monitor program will be written as a separate program executing on a dedicated display device. All modules of the monitor program will be of sufficient priority to preempt any of the executing routines of the other programs of the system.

The monitor program is made possible through the use of condition handlers and shared data. The monitor program will also be capable of placing locks on data files since it is capable of writing to data files. Coordination with the remaining procedures will be accomplished through the block and wakeup mechanisms. The access rights and security levels of all procedures will be set by the monitor program.

4. Dynamic Event Recording

Dynamic event recording is used to save the execution state of the simulation at various decision making points.

This state of execution is composed of two distinct parts with distinct characteristics. The state of the simulation is the set of values of all simulation variables. Whenever a decision is input, the operating system needs to copy the data segments containing the appropriate variable values onto a secondary storage device. These values must be copied prior to any changes due to the input decision. The state of the machine is characterized by the values in the machine's internal registers. At the decision point, the operating system saves the register values in secondary storage. The operating system must then label these values and inform the user of the assigned label and resume execution. At some future time when the user desires to return to this point, he need only supply the label value of the decision point and the operating system will then restore the simulation and execution states, returning control to the user for his new decision.

5. Report Writer

The purpose of the report writer is to produce statistical reports based on stored historical data and current battle status. It should have the capability to produce graph summaries of the user's desired format. For instance, bar graphs may be desired over simple plots for certain items. For ease of implementation, this should be defined as a user requirement however the report writer can be developed with sufficient flexibility to allow interactive user format selection at the expense of larger

programs. If sufficient storage space (memory or secondary) exists and system execution is not sufficently degraded, then this flexibility should be pursued for the benefit of the ultimate user.

The report writer is a procedure included in the administrative routines program. Proper selection of ring brackets and delegation of access permission will allow the report writer to reference data and perform its required function. The report writer may be invoked from either the administrative program or the monitor program. The invoking program sends a wakeup to the report writer to initiate the procedure. Upon completion, the report writer places itself in a blocked status to await further use.

6. Inquiry Mode

The function of the inquiry mode is to allow commanders and their staff sections to question the simulation. Legitimate inquiries are those that each agency would normally initiate during an actual battle. For instance the S-1 would not be allowed to query directly the status of some area outside his cognizance but rather require him to communicate via the appropriate agency which has cognizance over the area in question. Chapter II, section D previously identified normal areas of cognizance for the staff sections.

Answers to any agency's request must reflect the accuracy and timeliness experienced in a true battle. This requirement should be handled in the simulation of inter-

organizational communication paths. Any accurate and instantaneous reply could be misleading to to the commander. Human factors must be considered and accounted for in the inquiry mode's operation to reflect realism.

7. Map and Overlay Generation

Initial terrain generation experiments were conducted on a PDP 11/50 using a TEKTRONICS 4014-1 display device as the display medium. The 4014 has several display modes including point and vector. A resulting tutorial was developed for use and is included as APPENDIX A.

Current methods used for generating contour maps from stored matrices of data could not be used to display the terrain for this simulation [Dayhoff 1963]. One of the advantages of parametric terrain is that terrain may be calculated rather than stored allowing large areas to be modeled. Using calculated terrain, the only altitude known is that of the current location. The decision was made to scan the area from South to North and from West to East to determine the location of contour lines. The resulting problem was that with a constant sampling step size, it could not be guaranteed that the step taken would in fact fall on a contour line. Next a point was accepted as being on a given contour line if it were within a specified tolerance from the contour line. This tolerance was measured in the vertical direction. The first attempt at displaying parametric terrain utilized the point mode of the 4014. The deficiency noted in this approach was the

inability to establish contour lines since the display image consisted of a series of dots. To accomplish the illusion of lines a sufficiently small delta x and delta y had to be used thus extending execution time for any given piece of terrain. If the display was to be magnified to any degree the line illusion became visible dots or points once again. This effect demonstrated the need to draw contour lines using the vector mode of the 4014.

In order to utilize the vector mode, a drawing algorithm was developed. The algorithm used to determine the direction of draw was quite simple. Once a decision to draw was made, all points adjacent to the current point in a North or East direction were calculated and the line was drawn to the point closest to the contour elevation. This approach produced contour bands rather than contour lines. These bands were acceptable on steep slopes but in the flatter areas they were quite wide. An attempt to reduce the number of line segments to be drawn was made by drawing to a given point only if the point immediately North of it was farther from the contour line than the current point. This produced contour lines that were shaded on the North side. The solution to this problem resulted in an additional condition that a point would not be considered for plotting unless it were closer to the contour line than both the adjacent points in the North and South directions. The resulting map was adequate but still retained shading along a hill's major access in the area of the contour line

plus and minus the tolerance. The shading was negligible near the peak of the hill, but quite distracting near the base. The shape given by the distribution is increasingly flatter as one proceeds outward from the center of the hill. The solution to this problem was in the calculation of a dynamic tolerance. This tolerance was calculated by determining the distance from hill center that the distribution reached three standard deviations or fell below some specified minimum altitude, whichever comes first. This method produces an acceptable map. The resulting algorithm follows:

1. Locate point closest to contour line.
2. Sample adjacent points to North and/or East.
3. Draw line to adjacent point closest to contour line.
4. Locate next point closest to next contour line.

The major drawback to this approach lies in consumption of time. This method is of complexity of order N squared meaning that doubling the size of the map to be displayed roughly quadruples the execution time. This routine is by no means real-time in nature. Experiments were conducted to speed up the drawing of this contour map. The map may be drawn in real-time if the calculations are not required every time the terrain is displayed. The solution requires that the commands to move and draw that are generated by the CPU and sent to the display device must be intercepted and written to a data file. Upon a request to draw a given area, only the actual move and draw commands need be executed. These commands may be created and stored

as the terrain is designed and referenced later during the simulation as required.

The ability to draw different sections of the battlefield is easily attained. By using a device such as the TEKTRONIX 4014 a virtual window may be defined to the display. This virtual window is mapped onto the physical display window and only the points within the virtual window are displayed. This virtual window may be dynamically created from inout parameters from the console. The contour lines may be stored in separate files per individual line elevation thus allowing combinations to be plotted and giving flexibility of selection of line interval.

Overlays will be plotted on the basic contour map. These overlays may be selected individually or in any combination of the available overlays. The use of color displays will make the overlays stand out from the map and avoid confusion when multiple overlays are requested. In order to avoid destruction of the contour map by overwriting from the overlays, the display terminal must have a selective erase capability. This will allow the removal of a specific overlay without disturbing any others that may be displayed at the time of removal.

8. Security of Classified Data

Physical security of the classified data during input or output remains the problem of the user. The terminals used must be in a secure environment to avoid compromise before data is entered into the computer and after the

classified output is generated. Remote use of the simulation will be possible if scrambling devices are placed on the communication lines. Computational security is provided by the operating system since properly specifying the security level of the simulation will cause the internal protection mechanisms to safeguard the data from tampering within the computer. Thus the simulation running classified data may be described with a security level of <clearance,STAR> and thereby refuse usage of the data to anyone regardless of classification without STAR access.

9. Library Routines/Tutorials Needed

In any interactive system certain library routines and tutorials make the system more convenient to use. These routines lie resident within the system and are capable of being called by the user. Several readily identifiable examples are discussed in the following sections.

a. Terrain Generation Package

The purpose of the terrain generation package is to allow the user to define any given terrain area in terms of the required parameters prior to the execution of the battle simulation. The terrain generation package uses the identical data items required by the elevation routine of the simulation. Once the user has defined the terrain to his satisfaction he can elect to creat a file containing the display device commands generated during the display phase.

The terrain generation package includes two major items, a highly interactive program and a tutorial

describing the program's use. The interactive program allows the user to define and display the terrain of the battle area. The definition phase includes capabilities to build, modify, add-to or delete-from a data file containing the parameters. Appendex A defines these allowable functions.

The terrain display phase allows the user to draw contour lines of his chosen level from the data. The user specifies bounds for the display in terms of maximum and minimum grid coordinates, the contour level and the step-size desired. The user specified bounds give the illusion of zooming-in or away from the terrain. Bounds smaller than the defined battle area will cause a smaller area to be displayed in the static display window creating a blown-up or zoom-in effect. Conversely, should bounds larger than the battle area be prescribed, a reduction or zoom-out illusion is created. All displays are generated without the need to store in memory an elevation for each <x,y> location.

The terrain generation program contains the following functions. The user is allowed to build the parametric data file under a filename of his choosing. He is allowed to add, delete and modify the file to reflect the current state of terrain generation. There should be a narrative portion which describes the function of the program and demonstrates effects of different input parameters. The user should also be allowed to familiarize

himself with the system by manipulating parameters and seeing the results displayed. The capability to plot contour lines of the user's desired contour level from the defined terrain is also provided.

b. Position Selection

The purpose of the position selection package is to allow the user of STAR to select defensive positions and routes-of-march for his forces. This package, like the terrain generation package, is primarily a pre-execution or set-up operation designed to facilitate planning of a battle. Execution during the battle will give the user insight as to possible new positions or tactics as the battle progresses. Since interaction between user and simulation is provided for, the user may desire to utilize information gleened from the position selection package to perform updates during the simulation.

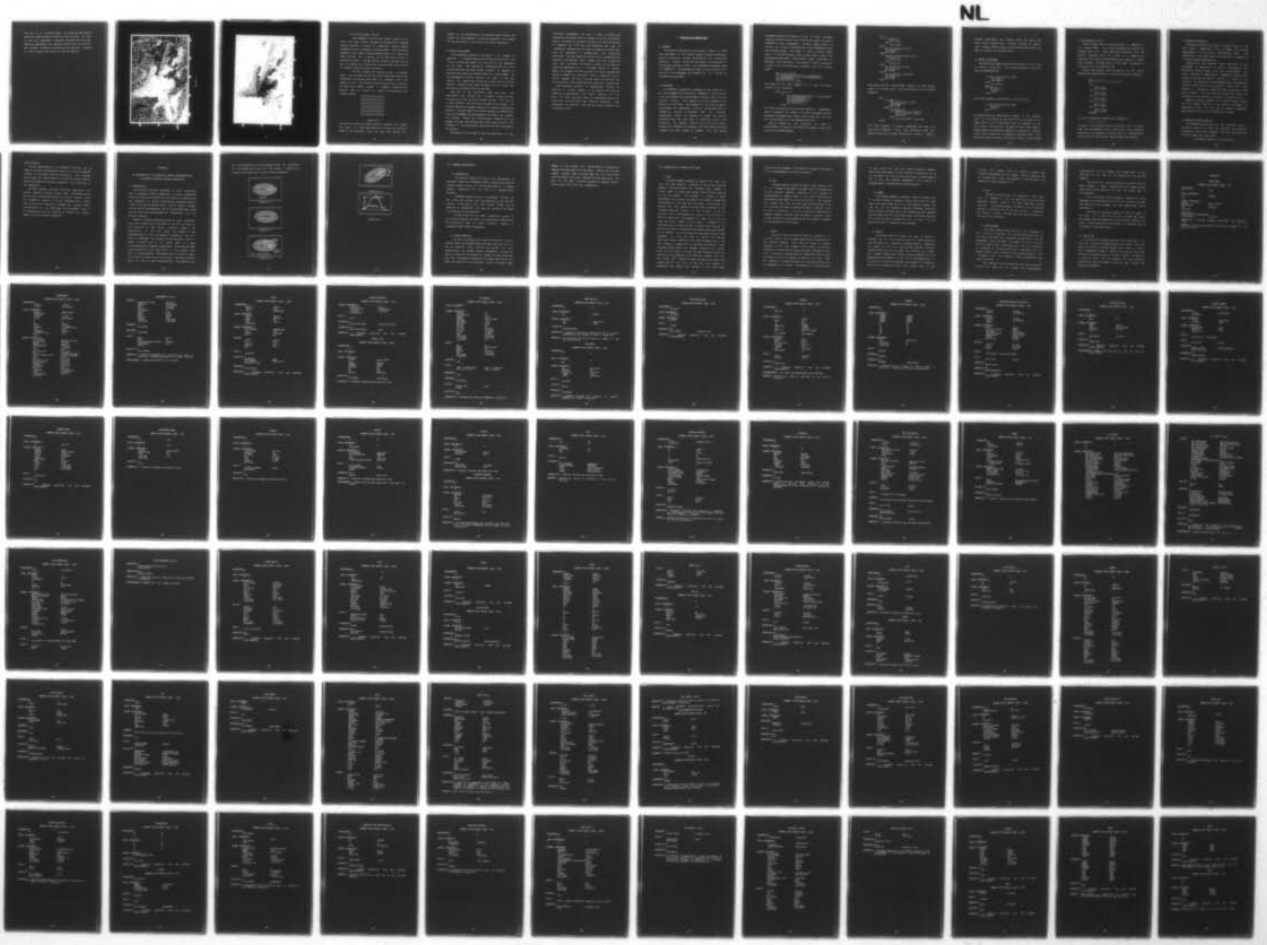
The position selection program should use the contour map generated by the terrain generation package and determine line-of-sight (LOS) fans for any selected position within the battle area. There are two approaches to the representation of LOS fans, each with it's own advantages and disadvantages. The first approach is to shade-in the visable area in the fan. This method gives the user a distinct impression of how much area the fan covers. However, specific terrain characteristics within the fan may be hidden from the user's view. To counter this degredation, the package should allow the user to display

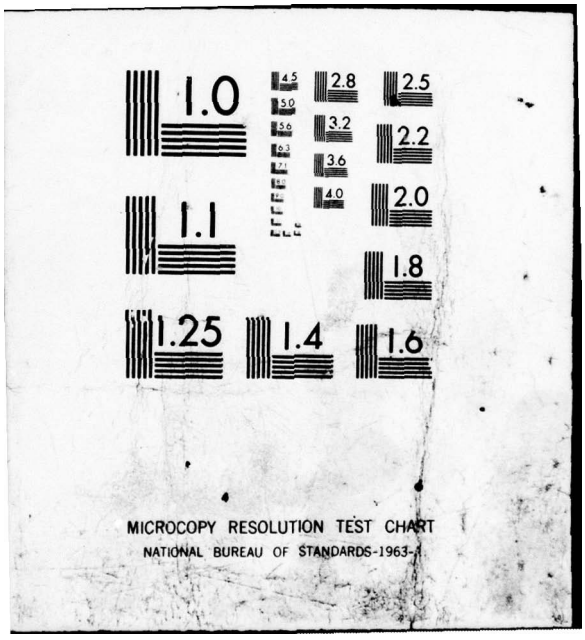
AD-A070 096 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
SYSTEMS ANALYSIS FOR THE INTERACTIVE SIMULATION WITH GRAPHICAL --ETC(U)
MAR 79 G S COKER, D R FORINASH

UNCLASSIFIED NL

2 OF 3

AD A070096




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

the fan in an inverted mode. This inverted mode should shade the areas outside the LOS fan thus allowing the user to see only geographic features defined within the fan. These two approaches when combined should allow the user all LOS related information concerning that position. Figures 4.1 and 4.2 depict the results of the two methods.

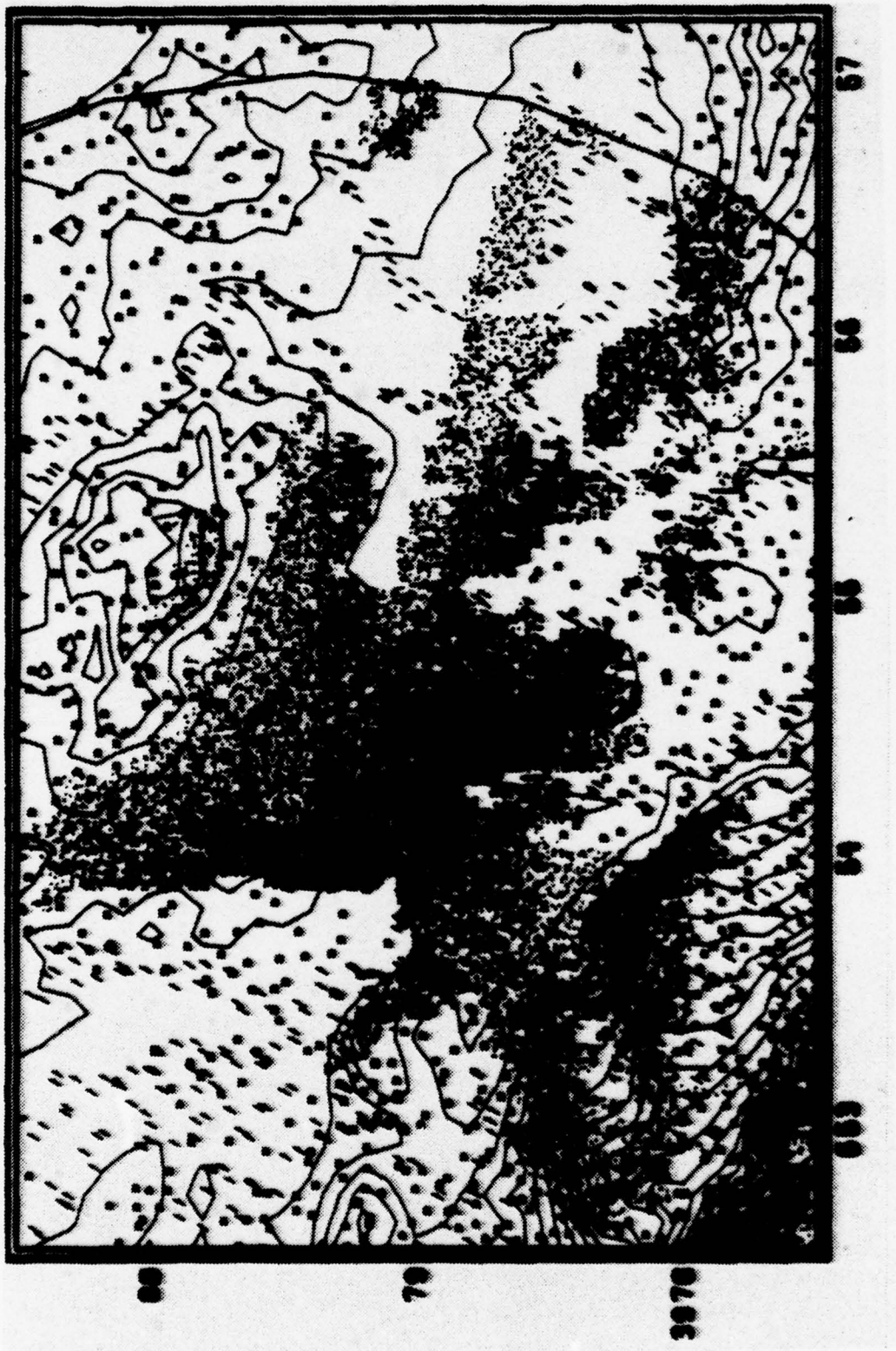


Figure 4.1

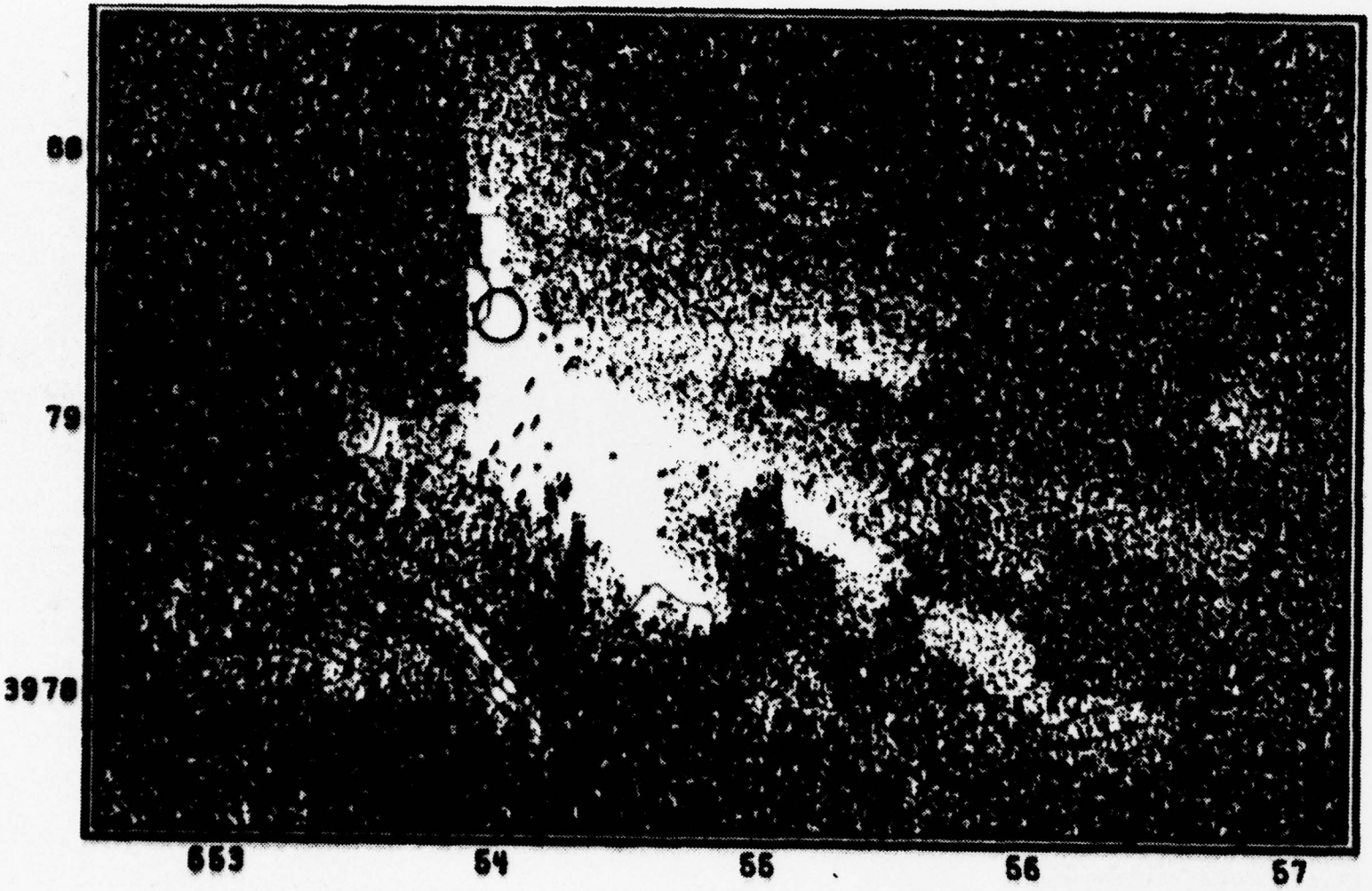


Figure 4.2

c. Military Symbol Library

The purpose of the military symbol library is to allow the user either through his interaction or program action to select a series of predefined display device commands to draw a standard military symbol. This functions somewhat like a table lookup procedure where the resulting table entries are a series of entries that define a particular symbol. The map overlay display programs must be able to access this library and retrieve these instructions for dynamic display during the battle.

Military leaders are innovative when a standard symbol has not been defined for some unique application and consequently establish some symbol to represent their new weapon or unit. The program should allow the user to define any additional symbols needed. A standard checker-board pattern, such as figure 4.3, should be provided on the display screen.

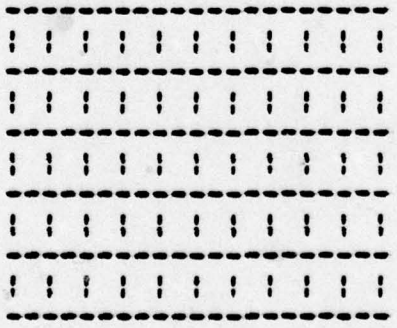


figure 4.3

The user can select any square and the program will shade that area. By continued selection, the user can define a symbol of his choosing. After the user has defined the

symbol to his satisfaction, the program should display the symbol for final approval of size and features. The symbol may now be stored in the library for future reference.

C. USE OF THE SYSTEM

The simulation system will be useful in all phases of modeling. A typical senario may follow this outline. The life of a simulation begins with the writing and debugging of the code. The simulation implementer may sit in his office and enter the code through a console as opposed to punching data cards and feeding them into a card reader. As the implementer finishes entering the code, he may now compile the program and correct any syntax errors from the console. Execution may reveal problems with his code that may also be corrected from his console.

The user of the simulation takes over after the implementer has finished and the model is ready for use. The user must first set up the model for use. This set up is facilitated by the tutorials provided by the system. The terrain generation package explains the method of generating terrain. The user uses this terrain package to generate and store the parameter values necessary to create and display the terrain. Using the line-of-sight capability of the system, the user then selects the initial positions for the elements on the battlefield using the LOS fans provided by the system.

The modeler is now ready to use the simulation. As the

simulation progresses, the user is able to monitor the simulation and decide when to interact with the simulation. Should he decide to interact, he may do so from his console by pressing the ATTN key and selecting the type of interaction desired from the menu of possible alternatives. Once the interaction is accomplished, the simulation continues. The user notes the effect of his interaction and wonders if the outcome would change had the interactive input been different. He elects to return to the point of input and experiment with a different course of action. Again, he presses the ATTN key and this time he chooses the option to repeat the simulation from a specified point. Having changed his input, the user elects to continue with the simulation. The simulation continues until termination.

Post execution analysis is accomplished through the creation of written reports by the report writer and the answering of questions by the inquiry routine. At this point the user may still elect to investigate behavior by returning to a given point and resuming execution. Upon completion of analysis, the user may elect to destroy the simulation files.

V. ANALYSIS OF CURRENT STAR

A. GENERAL

The analysis conducted on the current version of STAR was performed in the general areas of program structure, control structure, storage optimization and subroutine analysis. Certain guidelines were used in the analysis of the current version of STAR. The programming language used is SIMSCRIPT. The methodology used in the simulation will be left to the programmer and comments will be limited to programming techniques.

B. STRUCTURE

The SIMSCRIPT programming language has the capability to be both readable and structured. Readability and structure allow the program to be maintained by persons other than the original writers. This capability of the language is not being fully exploited. Good readability facilitates ease of maintenance and debugging. To attain the desired level of readability several principles must be followed. Indentation should be present to offset the control structure from the code that is contained within that structure. For example, should the "if" in an "if else" structure be indented five spaces, the "else" should also be indented the same number of spaces. Only one source

statement should be allowed to a line. All local variables should be defined rather than allowing the default values of SIMSCRIPT to take precedence. Variable names should be obvious as to their representation, since SIMSCRIPT allows long variable names. Many cases of short variable names occur in the STAR program either through ease of use or bad programming practices. These shorter names contribute to ambiguity and confusion. One example of lack of structure is seen in the following code extracted from the current model:

```
until jji = 1, do
let target(name(tank),2) = he.drag(tank)
let he.drag(tank) = 0 let mv.state(tank) = 1
let defnum(tank) = 1
let jji = jji + 1 loop
```

This code will be more readable if it were structured similar to the following:

```
until jji = 1,
     do
          let target(name(tank),2) = he.drag(tank)
          let he.drag(tank) = 0
          let mv.state(tank) = 1
          let defnum(tank) = 1
          let jji = jji + 1
      loop
```

This type of structure has two major benefits. Programmers easily recognise the scope of the UNTIL statement and the assignment statements are readily found since they are one per line and begin with the reserved word LET.

Another example of how structure may lead to understanding is found in the following segment of code from the routine CHG.SEC.SEARCH.

```
"d1"
     if css(a) = 1
          go to d11
     else
          go to d12
"d11"
     let pri.dir(a) = pi.c/2
     let css(a) = 0
     go to set
"d12"
     let zyx = uniform.f(0.,1.,1)
     if zyx ge .5
          go to d13
     else
          go to d14
"d13"
     let pri.dir(a) = pi.c/4
     let css(a) = 1
     go to set
"d14"
     let pri.dir(a) = 3*pi.c/4
     let css(a) = 1
     go to set
"set"
```

Upon examining this unstructured version of STAR source code, it is evident that it may be replaced by the following sequence:

```
"d1"
     if css(a) = 1
          let pri.dir(a) = pi.c/2
          let css(a) = 0
          go to set
     else
          let css(a) = 1
          if uniform.f(0.,1.,1) ge .5
               let pri.dir(a) = pi.c/4
               go to set
          else
               let pri.dir(a) = 3*pi.c/4
"set"
```

This code sequence is easily understood and has two additional benefits. First, this structured code will execute faster due to fewer transfers of control. Second,

storage requirements are reduced since this version has fewer lines of source code , thirteen instead of twenty-four, thereby reducing object code storage requirements and does not require the variable "zyx".

C. CONTROL STRUCTURES

The STAR model needs extensive optimization in the area of control structures. The following example is from the routine COMMO.PASS.TGT.

```
if pct.vis gt critical.value
     let lose = 1
     jump ahead
else
     let lose = 0
here
if lose eq 0
     let aim = 0
     return
else
```

This code sequence is equivalent to the following:

```
if pct.vis le critical.value
     let aim = 0
     return
else
```

The sequence may be reduced even further if the variable "aim" is initialized to zero since this is the majority of usage ("aim" is set to zero four times as opposed to one only once). This saving in storage of object code, ease of understanding and execution speed-up is attained by testing "less than or equal to" as opposed to "greater than". This particular type of control structure is used in other places

in the program as well.

Another common control structure abuse is observed in the routine RES4. This routine has seven "do loops", all indexed from 1 to 2. These loops may be combined into one control structure which will result in a reduction in execution time (counter maintenance) and a saving in object code storage. The combination of "do loops" is possible in the majority of the STAR routines.

Another optimization technique applies to the routine PRIORITY.AND.ROUND.SELECT with the follwing code sequence:

```
let i = i - 1
go to i0, i3, i6, i9, i12 per i

"i0"
     let i = 0
     go to bands
"i3"
     let i = 3
     go to bands
"i6"
     let i = 6
     go to bands
"i9"
     let i = 9
     go to bands
"i12"
     let i = 12
     go to bands
"bands"
```

A little "cleverness" reduces this sequence to

```
i = (i - 2) * 3
```

In fact, in this routine alone, fifty-four lines of source code may be reduced to four lines without loss of meaning. Benefits of the reduction include ease of understanding, more efficient execution and less storage requirements.

D. STORAGE OPTIMIZATION

Storage optimization can occur in several ways in the STAR model. Appendices C and D present the storage requirements of the current unstructured model. The "complexity" item under each routine analysis in Appendix B indicates storage dependencies.

Another area that deserves close monitoring is the assignment of subscripts in arrays. An example of this detrimental effect is seen in the array called TARGET. The actual variable is TARGET(321,2). SIMSCRIPT creates a data structure with 321 pointers to vectors of length 2. By reversing the subscripts giving TARGET(2,321) the SIMSCRIPT compiler stores this with 2 pointers of length 321. This reversal of subscripts saves 319 words of storage or 1278 bytes of memory. If at all possible, subscripts should be arranged with the smallest first and in increasing order.

Appendix D gives a summary of all static storage arrays and the storage required if an optimal assignment of subscripts is used. By this use of optimal subscripts approximately 12K bytes of storage may be saved.

E. SIMSCRIPT ROUTINE ANALYSIS

The SIMSCRIPT routine analysis was performed after a structured version of STAR was produced. All names are alphabetical within their category. For each subroutine the following items were identified:

1. Parameters passed to the subroutine.

2. Local variables defined to that subroutine.
3. Global variables accessed.
4. Variables read into the subroutine.
5. Variables written to print.
6. Data structures created.
7. Data structures filed into a predefined set.
8. Data structures removed from a set.
9. Data structures destroyed.
10. Vectors or matrix for which storage is reserved.
11. Vector or matrix for which storage is released.
12. Other subroutines called.
13. Subroutines that call the subroutine.
14. Events scheduled.
15. Subroutine that schedules the event.
16. Complexity of subroutine in regard to execution time and storage requirements.
17. Recommended improvements (excluding general improvements identified earlier).
18. Any remarks concerning the subroutine and values returned to the calling program.

VI. CONCLUSIONS AND RECOMMENDATIONS

The STAR war gaming model is written using sound modeling techniques. The implementation does have a serious drawback in maintainability. Without structure, this program is difficult, at best, to understand and will be extremely difficult for anyone other than the actual code writers to maintain. The current version of STAR is at this time undergoing extensive modification to give the program structure and to attempt to gain efficiency to both storage and execution.

This thesis is a preliminary design and therefore only a preferred solution was given. As the preferred solution is developed at a later time, more detail may be given as to the final implementation of the STAR model. The implementation of STAR using the operating system approach may be attempted here at the Naval Postgraduate School. The features described that are necessary to implement the proposed enhancements are found in the MULTICS operating system from Honeywell Information Systems, Inc. which is available on the ARPANET. This availability gives the school the opportunity for further study.

One of the most beneficial short term improvements that can be made is to obtain an interactive SIMSCRIPT compiler. The interactive compiler may be obtained free of charge from Consolidated Analysis Centers, Inc. under the university

grant program.

Further experimentation with graphical displays may be made at the Naval Postgraduate School utilizing the research computer in the school graphics laboratory. This computer may be used as a remote terminal to the W.R. Church Computer Center and graphics displays produced on the terminals in the laboratory.

The STAR wargame provides excellent opportunity for further computer science thesis work. Additional thesis material may include optimization of terrain display using the parametric method of terrain representation. Thesis work in the area of operating systems may include actual implementation of the STAR model on the ARPANET. Work in the database area will be required to develop the inquiry capability and the report generator.

APPENDEX A

AN INTRODUCTION TO INTERACTIVE TERRAIN REPRESENTATION UTILIZING PARAMETRIC TERRAIN GENERATION

I. INTRODUCTION

This tutorial has been developed to allow interactive use of parametric terrain generation. Currently mechanisms exist for building and modifying the input file required for the generation of terrain features and displaying contour lines of the user's desired level. Due to modular design, further terrain enhancements such as three-dimensional views from any point may be developed and incorporated into the existing program.

Parametric terrain generation uses as input parameters the x and y locations of the center of the hill (xc and yc), the elevation at that <xc,yc> location (peak), the eccentricity of the hill (ecc), the height of the parameterized hill (ht), the angle of rotation of the hill from an East-West axis (angle) and the spread of the hill (sprd). For each set of hill values there is a base altitude/elevation (basealt) associated. This base altitude is the elevation of the lowest point in the terrain area that is being modeled. The eccentricity of the hill (ecc) is the ratio of major axis to minor axis (major-axis/minor-axis) for the hill under consideration. The spread of the

hill is considered to be the distance along the major-axis for the elevation to drop fifty (50) meters. Figures A-1.1 through A-1.5 graphically explain these parameters.

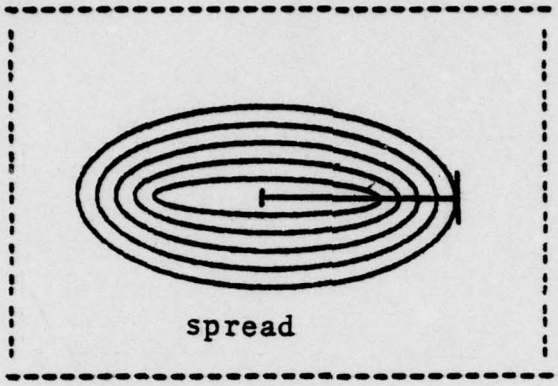


FIGURE A-1.1

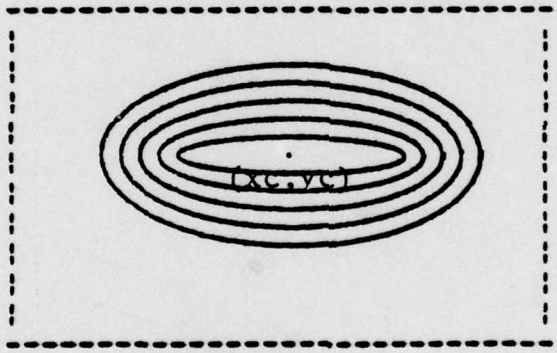


FIGURE A-1.2

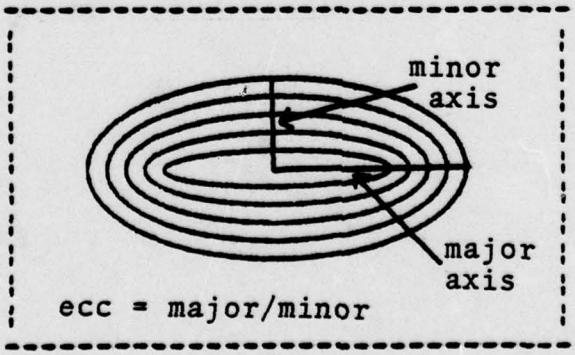


FIGURE A-1.3

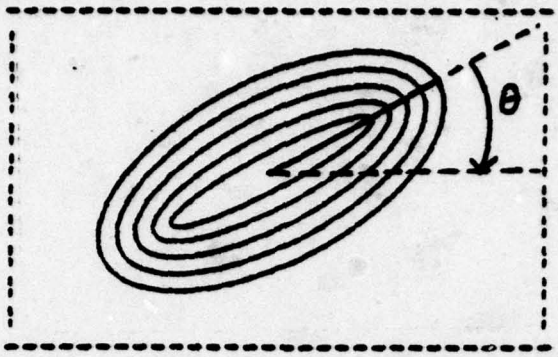


FIGURE A-1.4

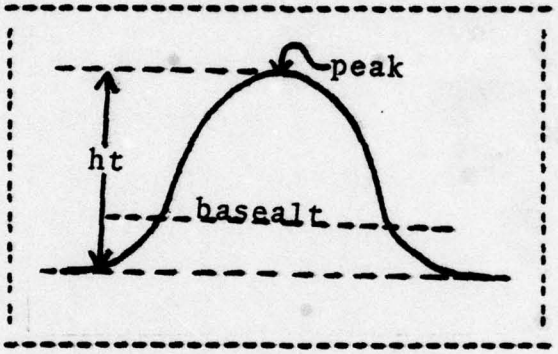


FIGURE A-1.5

II. HARDWARE REQUIREMENTS

A. INTRODUCTION

The display hardware utilized in the development is centered around the TEKTRONICS 4014-1 storage tube graphics display system. It has a 19 inch storage tube as a display medium. Associated with the 4014 is a standard ASCII keyboard.

A VERSATEC MATRIX printer is accessable through the PDP 11-50 and allows hard copy generation from the 4014 display. A hard copy of the 4014 display screen can be obtained by depressing the copy key located on the upper right portion of the keyboard.

The PDP 11-50 with the UNIX timesharing system is utilized and assumes the 4014 is a conventional alphanumeric CRT allowing the user to "login" normally. Section IV discusses these "login" procedures.

B. SYSTEM OPERATION

The 4014 is powered on by turning the off-on switch, located under the keyboard about one foot from the floor, to the on position. After turning the device on, wait approximately 30 seconds before proceeding to allow the device to warm up. The screen will appear a bright green and now must be erased by depressing the page reset key on the upper left portion of the keyboard. Should a residual image

appear on the screen, wait approximately 15 seconds and depress the page reset key once again. Insure that the mode toggle, located above the page reset key, is in the online position rather than local. Depress the carriage return and wait for the UNIX timesharing system to request "login". Follow normal PDP 11-50 login procedures.

III. DESCRIPTION OF PROGRAM FUNCTIONS

A. MAIN

The main program requests the name of the data file that the user desires to operate on during this terminal session. The user enters the filename, upon request in either upper or lower case letters, that he either desires to use or create. The filename is limited to eight characters but the user may specify any length filename and the program will only use the first eight, terminating all others. For this reason the first eight characters of any filename should be unique. If the filename is not an existing file, the program will create it for the user. If the file does exist, the program will open that file for read and write access to the user. A command list is displayed next to allow the user to select that function he desires and enter the appropriate command code when prompted by the program. The main program is developed around a CASE statement to allow the user to perform his desired functions. Control remains in this CASE statement until the user elects a code of "7" to terminate the session. When the user selects code 7 the program displays the current state of the data file that the user selected during the beginning of the session on the 4014 and updates the file for future use. Should the user not desire to retain the current copy of the file, he may exit the program by depressing the rub-out key located on the lower right

portion of the keyboard. (This method of program termination is not recommended as a short-cut however.)

B. ADD

The add function allows the user to add another set of hill values to the data file specified in the beginning of this terminal session. Addition of hill values is accomplished by adding them to the end of the data and incrementing the number of hills on the file. The program will prompt the user for each data item required for the hill and give the user the option of adding multiple hills or only a single hill. All hill values are floating point numbers, however, the user need not specify a decimal point if that value is a whole number since the program is capable of interpretation in this case.

C. BUILD

The build function gives the user the capability to initially build the file specified during the beginning of the terminal session. Care must be taken to prevent building a file that already exist since the building process will over-write all data previously resident on that file. Building is accomplished by requesting from the user the number of hills he desires to load to the file. Next the program will prompt the user as in the add function for all neccessary hill values. The program will allow the user

to only build one file per terminal session. However, should the user desire to build multiple files he may do so by building the first one and then enter a command code of "7" to exit the program. Subsequent file building is accomplished by repeating this same procedure.

D. CHANGE

The change process is actually a modify process since it allows the user to change or modify any single hill data item or all data items for a hill. The program prompts the user for the hill number he desires to change, allows the user to select the data item to be changed and then requests the new value that is to be substituted. The change function will allow the user to change one hill or many hills by asking the user if he has any more changes.

E. DELETE

The delete function allows the user to delete a complete set of hill values for the hill number specified from the data file. Deletion is accomplished by requesting the number of the hill that the user desires to delete, locating that set of hill values and shifting all higher hill number values down, overwritting the deleted hill and decrementing the number of hills. It is recommended that if multiple hills are to be deleted, the highest hill number be used first to prevent the user from losing track of the

current hill numbers since the deletion process also logically decrements the hill number. Multiple deletes may be accomplished since the program will ask the user if he desires to delete another hill.

F. PLOT

The plot function uses the parametric input data, which is located on the file specified by the user, and generates contour lines of the user specified contour interval. The program prompts the user for the minimum and maximum x locations, the minimum and maximum y locations, the contour interval desired and the minimum elevation in the area to be displayed.

G. MISCELLANEOUS

The first miscellaneous routine to be discussed is the "writefile" routine. This routine restores/writes the hill data from memory to a secondary storage (disk) file for subsequent use by the user. The number of hills is written first followed by all x locations (xc). The file structure is completed by writing the entire y locations (yc), hill spread values in the x direction (sx), spread values in the y direction (yc), the rotation values respectively and closing the file.

The "readfile" routine functions like the "writefile" function but loads the hill values from the secondary

storage (disk) file into memory. The filename used is that specified by the user at the beginning of the terminal session.

The "printfile" routine displays to the 4014 the hill data located in memory starting with hill number one and continuing to the number of hills that are being used in this session.

The "invalidcmd" routine produces appropriate error messages to be displayed on the 4014 for the user. The user must heed the error message and take appropriate action accordingly.

The "cmdlist" routine displays all function codes to the user. It is from this list that the user must select the command code corresponding to the function desired and enter it when prompted by the program. All commands must be followed by a carriage return.

IV. SYSTEM USE

To initiate the program execution after powering on the 4014 the following procedures must be followed. The first step is to login to the UNIX system with a name of "parry" and a password of "parry". (Note all entries are lower case letters.) The second step is to enter "terrain" followed by a carriage return and the program will begin execution prompting the user as needed to step the user through the required procedures.

APPENDIX B

AMMO.CHECK

NUMBER BYTES OBJECT CODE : 496

PARAMETERS :

a	rnd

LOCAL VARIABLES :

a	answer
rnd	

GLOBAL VARIABLES :

ap.tow	aw1.or.ms13
aw2.or.adm	c.1
c.2	he.drag
wpn.type	

CALLED BY :

priority.and.round.select	
t72.tactics	we.miss

COMPLEXITY : Constant storage requirement and execution time.

REMARKS : This routine returns the value of answer to the calling routine.

ARTY.IMPACT

NUMBER BYTES OBJECT CODE : 1312

PARAMETERS :

id.btry	id.fdc
id.fo	id.mission

LOCAL VARIABLES :

ans	estimate.of.time
i	id.btry
id.fdc	id.fo
id.mission	j
k	l
m	rg
time	time.1
time.2	time.3
time.4	time.5
time.6	time.7
within.tolerance	xx
yy	

GLOBAL VARIABLES :

caliber	debug
del.1	del.2
error.code	gsrs.code
gt.final.rg	last.fo.rg
miss.tolerance	msn.name
msn.time	my.radio
no.missions.fired	now.firing
num.adj.rounds	num.dpicm.left
rate.of.fire	rd.1.error
rd.2.error	st.firing
theta	time.v
volley	volleys.to.fire
which.volley	x.cur1
x.cur4	x.future.loc
y.cur1	y.cur4
y.future.loc	

WRITES :

id.fo	id.btry
id.fdc	msn.name
volleys.to.fire	num.adj.rounds
within.tolerance	time.v

ARTY.IMPACT (cont)

CALLS :

arctan.f	arty.time
assessment	dist
error	new.location
position.update	print
print1	tracer

SCHEDULES :

busy.radio.net	end.of.mission
guns.firing	open.radio.net

SCHEDULED BY :
guns.firing

COMPLEXITY : Execution is linear on volleys.to.fire but constant storage.

ARTY.TIME

NUMBER OF BYTES OBJECT CODE : 416

PARAMETERS :
a

LOCAL VARIABLES :

a	del.time

GLOBAL VARIABLES :

fa.time.deltas	rn.stream

CALLS :

normal.f	tracer
uniform.f	

CALLED BY :

arty.impact	
checking.guns.availability	
commo.attempt	end.of.mission
fdc.processing	update.cluster

COMPLEXITY : Constant execution time and storage.

REMARKS : Returns the value of del.time to the calling routine.

ASSESSMENT

NUMBER BYTES OBJECT CODE : 4528

PARAMETERS :

id.btry	id.fdc
id.fo	id.mission

LOCAL VARIABLES :

count	debug.count
difference	i
id.btry	id.fdc
id.fo	id.mission
j	k
l	m
pk	sig.x
sig.y	x.change
x.error	x.normal.error
xdif	xnew
y.change	y.error
y.normal.error	ydif
ynew	

GLOBAL VARIABLES :

alive.dead	ammunition.type
amt.of.hits	caliber
debug	defnum
displacement	d.radius
f.d	fired.at
fkill	foe
gt.final.rg	hit.state
kkill	largest.num.wpns
lethal.radius.array	level.of.damage
m.d	mfkill
mkill	msn.name
num.dpicm.left	num.guns
num.he.left	num.hit
p.punch	rd.offset
rn.stream	theta
time.v	wpn.type
x.current	x.future.loc
x.mpi	y.current
y.future.loc	y.mpi
z.current	

WRITES :

ammunition.type	caliber
defnum	difference
f.d	fired.at
fkill	guntube
hit.state	i
kkill	m.d
mfkill	mkill
name	ncase
num.hit	spd
time.v	x.current
y.current	z.current

RESERVES :

rd.offset

RELEASES :

rd.offset

CALLS :

abs.f	atrit
dist	loc
new.coordinate.system	normal.f
parameters	print
print1	

CALLED BY :

arty.impact

COMPLEXITY : Execution dependent on num.guns and tank in red.alive. Storage dependent on largest.num.wpns.

IMPROVEMENTS : Reverse subscripts on rd.offset.

ATRIT

NUMBER BYTES OBJECT CODE : 2256

PARAMETERS :

efkill	emkill
kaykill	sh.t
tgt.t	whocalled

LOCAL VARIABLES :

efkill	emkill
fnow	kaykill
mnow	pk
sh.t	tgt.t
whocalled	

GLOBAL VARIABLES :

alive.dead	damage.num
f.d	m.d
mfkill	mine.det
mkill	ph

WRITES :

efkill	emkill
f.d	fkill
kaykill	m.d
mfkill	mkill
pk	

CALLS :

uniform.f

CALLED BY :

assessment	geom
mrl.impact	pop.a.mine
red.arty.fires	

SCHEDULES :

final.death

COMPLEXITY : Constant execution time and storage requirements.

ATTRITION.CHECK

NUMBER BYTES OBJECT CODE : 496

GLOBAL VARIABLES :

b.pct.att	delta.t
n.blue.alive	n.red.alive
rc.count	r.pct.att
r.num.alive	

CALLS :

int.f

SCHEDULES :

attrition.check	stop.simulation

SCHEDULED BY :

attrition.check	main

COMPLEXITY : Constant execution time and storage requirements.

BASIC.LOAD

NUMBER BYTES OBJECT CODE : 2088

PARAMETERS :

a

LOCAL VARIABLES :

a

GLOBAL VARIABLES :

ap.tow	awl.or.msl3
c.1	c.2
capds	caseap
casehe	cheat
he.drag	hto
m1	m2
m3	name
op.rng	wpn.type

CALLED BY :

bl.create	red.create

COMPLEXITY : Constant storage and execution time.

BL.CREATE

NUMBER BYTES OBJECT CODE : 2824

LOCAL VARIABLES :

i

GLOBAL VARIABLES :

ap.tow	bn
b.num.alive	co
cocdr	color
defnum	dir.of.mvmt
guntube	list
mv.state	name
op.rng	pi.c
pi.hat	plt
pltldr	pointer
pri.dir	rc.count
r.con	rrrpoint
sec	target
veh.type	wpn.type
x.current	y.current
z.current	zh

READS :

bn	co
cocdr	dir.of.mvmt
name	plt
pltldr	pri.dir
sec	veh.type
wpn.type	x.current
y.current	

CREATES :

tank

FILES :

tank in blue.alive	tank in comp.unit
tank in plt.unit	tank in tanks

RESERVES :

list

RELEASES :

bl.create

CALLS :

basic.load	elev
hider	

CALLED BY :

main

COMPLEXITY : Storage and execution depend on num.alive.

BMP.TACTICS

NUMBER BYTES OBJECT CODE : 264

PARAMETERS :

a	b

LOCAL VARIABLES :

a	answer
b	

GLOBAL VARIABLES :

co	comp.unit
foe	tank

CALLED BY :
target.select

COMPLEXITY : Constant storage but execution time is linearly dependent on the number of tanks in comp.unit

REMARKS : This routine returns the value of answer to the calling routine.

BUG.CHECK

NUMBER BYTES OBJECT CODE : 1024

PARAMETERS :

a	b

LOCAL VARIABLES :

a	b
bc	

GLOBAL VARIABLES :

co	comp.unit
defnum	m2
mv.state	plt.unit
range	tank
t.spd	wpn.type

CALLS :
dr.mount

CALLED BY :
impact

SCHEDULES :
df.change

COMPLEXITY : Constant storage but execution in linearly dependent on tanks in comp.unit

BUSY.RADIO.NET

NUMBER BYTES OBJECT CODE : 240

PARAMETERS :
id.radio

LOCAL VARIABLES :
id.radio

GLOBAL VARIABLES :
state3

CALLS :
tracer

SCHEDULED BY :
arty.impact guns.firing

COMPLEXITY : Constant execution time and storage requirements.

CARDIO

NUMBER BYTES OBJECT CODE : 1280

PARAMETERS :

a	b
pct.vis	r
x	

LOCAL VARIABLES :

a	angle
area	at
b	bt
dd	denom
det.time	lambda
mt	p.sub.k
pct.vis	per.full.expo
r	rr
t.c.factor	tgt.element

GLOBAL VARIABLES :

b.area	blue
cbar	color
dir.of.mvmt	pi.c
pi.hat	pri.dir
r.area	red
spd	x.current
y.current	zl

CALLS :

abs.f	arctan.f
log.e.f	sin.f

CALLED BY :

step.time

COMPLEXITY : Constant execution time and storage requirements.

IMPROVEMENTS : All local variables need to be defined.

REMARKS : Returns the value of det.time to the calling routine.

CHARGE

NUMBER BYTES OBJECT CODE : 2152

PARAMETERS :

c

LOCAL VARIABLES :

avgxc	avgx5
avgx6	avgx7
avgyc	avgy5
avgy6	avgy7
c	cc
k	r
m	xc
x5	x6
x7	yc
y5	y6
y7	

GLOBAL VARIABLES :

bn	red.alive
tank	xc
x.current	yc
y.current	

CALLS :

get.up

SCHEDULES :

charge

SCHEDULED BY :

charge leave.check

COMPLEXITY : Execution time is linear on number tanks in red.alive. Storage requirements are constant.

CHECKING.GUNS.AVAILABILITY

NUMBER BYTES OBJECT CODE : 1824

PARAMETERS :

id.btry	id.fdc
id.fo	id.mission

LOCAL VARIABLES :

id.btry	id.fdc
id.fo	id.mission
time	

GLOBAL VARIABLES :

btry	debug
fire.dir.center	label
msn.name	msn.time
num.adj.rounds	num.missions
queue.size	queue.time
state	state1
st.firing	time.v

WRITES :

id.btry	id.fdc
id.fo	msn.name
state	time.v

FILES :

id.mission in howitzer.queue

CALLS :

arty.time	tracer

SCHEDULES :

guns.firing

SCHEDULED BY :

fdc.processing

COMPLEXITY : Constant execution time and storage requirements.

CHG.SEC.SEARCH

NUMBER BYTES OBJECT CODE : 1304

PARAMETERS :

a

LOCAL VARIABLES :

a	xyz
zyx	

GLOBAL VARIABLES :

blue	color
css	dir.of.mvmt
mv.state	pri.dir
pi.c	

CALLS :

set.sector	uniform.f

CALLED BY :

step.time

COMPLEXITY : Constant execution time and storage requirements.

IMPROVEMENTS : Needs to be rewritten to save 44 lines of source code.

COMMO.ATTEMPT

NUMBER BYTES OBJECT CODE : 1096

PARAMETERS :

id.fo	id.mission
id.radio	

LOCAL VARIABLES :

id.fo	id.mission
id.radio	time

GLOBAL VARIABLES :

error.code	idle
state3	time.v
wait.time	

FILES :

id.mission in msn.queue

CALLS :

arty.time	tracer

SCHEDULES :

commo.attempt	fdc.processing
open.radio.net	

SCHEDULED BY :

commo.attempt

COMPLEXITY : Constant execution time and storage requirements.

COMMO.PASS.TGT

NUMBER BYTES OBJECT CODE : 488

PARAMETERS :

a

LOCAL VARIABLES :

a	aim
baim	lose
r	

GLOBAL VARIABLES :

bwd.look	critical.value
foe	fwd.look
op.rng	pct.vis
plt	

CALLS :

dist	loc
sight	

CALLED BY :

t72.tactics

COMPLEXITY : Constant execution time and storage requirements.

REMARKS : This routine returns the value of aim to the calling routine.

COMPUTE

NUMBER BYTES OBJECT CODE : 3528

PARAMETERS :

f.pcvis	pc.vis
sh.t	tgt.t

LOCAL VARIABLES :

addefl	addel
deflbias	deflsig
elbias	elsig
f.pcvis	i
io	j
k	l
pc.vis	r
sh.t	tgt.t
vel	

GLOBAL VARIABLES :

accht	accke
accmb	accmsl
addon	bm.mov
ht.mov	projo
spd	wpn.type
x.current	y.current

WRITES :

i	k
l	

CALLS :

dist	geom
min.f	subcal
trunc.f	

CALLED BY :

impact

COMPLEXITY : Constant execution time and storage requirements.

CONVERT.BACK

NUMBER BYTES OBJECT CODE : 712

PARAMETERS :

a

LOCAL VARIABLES :

a	pi.int

GLOBAL VARIABLES :

c.bar	cbar
defnum	dir.of.mvmt
d.num	m.det
micro	mine.det
p.hat	pl.c
plow.cond	p.v
spd	time.v
t.spd	v.ms
x.ct	x.current
y.ct	y.current
z.ct	z.current
zh	

CALLS :

pop.a.mine

CALLED BY :

loc

COMPLEXITY : Constant execution time and storage requirements.

DECREMENT.AMMO

NUMBER BYTES OBJECT CODE : 768

PARAMETERS :

a	rnd

LOCAL VARIABLES :

a	rnd

GLOBAL VARIABLES :

ap.tow	aw1.or.ms13
aw2.or.adm	c.1
c.2	crf
he.drag	trf
wpn.type	

CALLED BY :

fire

COMPLEXITY : Constant storage and execution time.

DEFEND

NUMBER BYTES OBJECT CODE : 688

PARAMETERS :

a

LOCAL VARIABLES :

a

GLOBAL VARIABLES :

alive.dead	ap.tow
defnum	fa
mv.state	name
pi.c	pri.dir
spd	target
time.v	t.spd
wpn.type	

CALLS :

dismount.dragon	hider
set.sector	

CALLED BY :

loc

COMPLEXITY : Constant storage and execution time.

DETECT

NUMBER BYTES OBJECT CODE : 792

PARAMETERS :

a	b

LOCAL VARIABLES :

a	b
whocalled	

GLOBAL VARIABLES :

alive.dead	bwd.look
critical.value	defnum
fa	fip
fkill	fwd.look
line.of.sight.exists	pct.vis

CALLS :

list.update	loc
proximity.detect	sight
tracer	

SCHEDULED BY :

impact	step.time

COMPLEXITY : Constant storage and execution time.

IMPROVEMENTS : Delete the variable whocalled - declared but unused.

DF.CHG

NUMBER BYTES OBJECT CODE : 496

PARAMETERS :
a

LOCAL VARIABLES :
a c

GLOBAL VARIABLES :
alive.dead color
defnum spd

CALLS :
hider

SCHEDULED BY :
bug.check dr.mount
leave.check loc

COMPLEXITY : Constant storage and execution time.

DISMOUNT.DRAGON

NUMBER BYTES OBJECT CODE : 744

PARAMETERS :
a

LOCAL VARIABLES :
a

GLOBAL VARIABLES :
defnum he.dragon
m2 mv.state
name pi.c
plt plt.unit
pri.dir tank
target wpn.type
x.current y.current

CALLS :
hider loc
set.sector

CALLED BY :
defend

COMPLEXITY : Storage requirements are constant but execution time is linearly dependent on the number of tanks in plt.unit.

DIST

NUMBER BYTES OBJECT CODE : 144

PARAMETERS :

x1	x2
y1	y2

LOCAL VARIABLES :

distance	x1
x2	y1
y2	

CALLS :

sqrt.f

CALLED BY :

arty.impact	assessment
commo.attempt	compute
error	fdc.processing
fire	guns.firing
impact	new.location

COMPLEXITY : Constant storage and execution time.

REMARKS : Returns the value of distance to the calling routine.

DOING.CLUSTERS

NUMBER BYTES OBJECT CODE : 5048

PARAMETERS :

id.fo	name.priority
pri.value	

LOCAL VARIABLES :

a	angle
b	dir
i	id.fo
j	k
l	m
n	name.priority
prt.value	s
tank	total.clusters
x	y

GLOBAL VARIABLES :

alive.dead	b.org.alive
box.tolerance	clusters
c.number.array	debug
dir.of.mvmt	fo.max.range
fo.min.range	list
name	old.cluster.number
spd	state4
target	x.current
y.current	

WRITES :

clusters	i
id.fo	

CALLS :

abs.f	arctan.f
cos.f	dim.f
loc	sin.f

CALLED BY :

update.cluster

COMPLEXITY : Storage is constant but execution is dependent on the product of the size of the target list and the total number of clusters.

REMARKS : Returns the values of name.priority and pri.value to the calling routine.

DR.MOUNT

NUMBER BYTES OBJECT CODE : 1192

PARAMETERS :

a

LOCAL VARIABLES :

a	jji

GLOBAL VARIABLES :

ap.tow	defnum
fip	he.drag
m2	mv.state
name	plt
plt.unit	tank
target	time.v
t.spd	wpn.type
x.current	y.current

CALLED BY :

bug.check	leave.check

SCHEDULES :

df.chg

COMPLEXITY : Execution time increases linear with number tanks in plt.unit with wpn.type = 6 and he.drag(tank) gt 0 and fir(tank) ne 1. Storage is constant.

END.OF.MISSION

NUMBER BYTES OBJECT CODE : 2704

PARAMETERS :

id.btry	id.fdc
id.fo	id.mission

LOCAL VARIABLES :

estimate.of.time	id.btry
id.fdc	id.fo
id.mission	time
time	

GLOBAL VARIABLES :

amt.active.msns	amt.msns.fired
btry	debug
fist	
holding.msns	howitzer.queue
label	mission
msn.name	
msn.time	my.radio
new.location	num.adj.rounds
queue.size	queue.time
statel	st.firing
time.v	which.volley

WRITES :

fist	id.btry
id.fdc	id.fo
msn.name	time.v

FILES :

id.mission in msn.queue

REMOVES :

id.mission from howitzer.queue and holding.msns

CALLS :

arty.time	tracer

SCHEDULES :

fo.not.busy	guns.firing
open.radio.net	

SCHEDULED BY :

arty.impact	error

COMPLEXITY : Constant execution and storage requirements.

ERROR

NUMBER BYTES OBJECT CODE : 944

PARAMETERS :

ans	id.btry
id.fdc	id.fo
id.mission	

LOCAL VARIABLES :

ans	id.btry
id.fdc	id.fo
id.mission	sig.x
sig.y	tank
xdif	x.impact.point
x.normal.error	ydif
y.impact.point	y.normal.error

GLOBAL VARIABLES :

ammunition.type	caliber
error.code	gt.final.loc
rn.stream	theta
x.future.loc	y.future.loc

CALLS :

dist	new.coordinate.system
normal.f	parameters
position.update	tracer

CALLED BY :

arty.impact

SCHEDULES :

end.of.mission

COMPLEXITY : Constant execution and storage requirements.

FA.1.MAIN

NUMBER BYTES OBJECT CODE : 4936

LOCAL VARIABLES :

i	j
k	l
m	

GLOBAL VARIABLES :

amt.ammo.types	amt.blue.batterys
amt.calibers	amt.fa.time.deltas
amt.ffe.volleys	amt.mrl
amt.red.batterys	arty.pk.table
b.num.alive	b.org.alive
box.tolerance	color1
cutoff.time	debug
displacement	d.radius
fo.max.range	fo.min.range
fo.vehicle	fwd.obs.msn.tolerance
largest.num.wpns	
max.number.of.missions.per.fo	
max.range	miss.tolerance
n.battery	n.fdc
n.fo	no.range.bands
n.radio	num.dpicm.left
num.he.left	num.guns
p.punch	range.bands
rate.of.fire	red.1.constant
r.num.alive	rn.stream
r.org.alive	salvos
sigma.dpicm	tgt.acq.error
travel.time.array	tr.time
x.cur1	x.cur2
y.cur1	y.cur2

READS :

amt.ammo.types	amt.blue.batterys
amt.calibers	amt.fa.time.deltas
amt.ffe.volleys	amt.mrl
amt.red.batterys	arty.pk.table
box.tolerance	color1
cutoff.time	debug
displacement	d.radius
fo.max.range	fo.min.range
fo.vehicle	fwd.obs.msn.tolerance
largest.num.wpns	
max.number.of.missions.per.fo	
max.range	miss.tolerance
n.battery	n.fdc
n.fo	no.range.bands
n.radio	num.dpicm.left
num.he.left	num.guns
p.punch	range.bands
rate.of.fire	rn.stream
sigma.dpicm	tgt.acq.error
travel.time.array	tr.time
x.curl	y.curl

CREATES :

battery	fdc
fo	radio

RESERVES :

array.detect	arty.pk.table
clusters	c.number.array
displacement	
fa.time.deltas	fo.vehcle
lethal.radius	range.bands
red.planned.fires	sigma.dpicm
time.last.cluster.update	tgt.acq.error
travel.time.array	

RELEASES :

preplanned

CALLS :

preplanned

CALLED BY :

main

COMPLEXITY : Linear on n.fdc, n.battery, amt.ammo.types * amt.calibers, amt.blue.batterys * num.guns, amt.calibers * no.range.bands

IMPROVEMENTS : Reads num.dpicm.left then sets to 0.

FA.2.MAIN

NUMBER BYTES OBJECT CODE : 3472

LOCAL VARIABLES :

i	j
k	l

GLOBAL VARIABLES :

amt.ammo.types	amt.blue.batterys
amt.calibers	amt.fa.time.deltas
amt.ffe.volleys	amt.mrl
amt.red.batterys	arty.pk.table
b.org.alive	box.tolerance
cutoff.time	debug
d.radius	fa
fo.max.range	fo.min.range
fo.vehicle	fwd.obs.msn.tolerance
largest.num.wpns	lethal.radius
max.number.of.missions.per.fo	
miss.tolerance	n.battery
n.fdc	n.fo
no.range.bands	n.radio
n.tanks	pi.c
pointing.to	p.punch
rn.stream	r.org.alive
rrrpoint	type

WRITES :

amt.ammo.types	amt.blue.batterys
amt.calibers	amt.fa.time.deltas
amt.ffe.volleys	amt.mrl
amt.red.batterys	b.org.alive
box.tolerance	cutoff.time
debug	d.radius
fo.max.range	fo.min.range
fwd.obs.msn.tolerance	largest.num.wpns
max.number.of.missions.per.fo	
miss.tolerance	n.battery
n.fdc	n.fo
no.range.bands	n.radio
n.tanks	p.punch
r.org.alive	rn.stream

CALLS :
sqrt.f

CALLED BY :
main

SCHEDULES :
red.arty.fires update.cluster

COMPLEXITY : Linear on n.fo, amt.calibers * amt.ammo.types * 9, amt.red.batterys - amt.mrl

IMPROVEMENTS : Combine major loops.

FA.TGT.ERROR

NUMBER BYTES OBJECT CODE : 416

PARAMETERS :
a loc.error

LOCAL VARIABLES :
a loc.error

GLOBAL VARIABLES :
rn.stream tgt.acq.error

CALLS :
tracer uniform.f

CALLED BY :
new.location

COMPLEXITY : Constant execution and storage requirements.

REMARKS : This routine returns the value of loc.error to the calling routine.

FDC.PROCESSING

NUMBER BYTES OBJECT CODE : 3424

PARAMETERS :

id.fo	id.mission

LOCAL VARIABLES :

diff	i
id.btry	id.fo
id.mission	j
k	l
m	rg
time	type.ammo
volleys	xx
yy	

GLOBAL VARIABLES :

ammunition.type	amt.active.msns
amt.ffe.volleys	busy
debug	dpicm
error.code	fwd.obs.msn.tolerance
gt.final.rg	gt.initial.rg
mission	my.radio
msn.name	
msn.time	n.fdc
n.holding.msns	num.missions
process	queue.size
start	state1
status	theta
time.v	
volleys.to.fire	x.cur1
x.cur4	x.future.loc
y.cur1	y.cur4
y.future.loc	

WRITES :

i	id.fo
msn.name	num.missions
queue.size	state1
time.v	

FILES :

id.mission in holding.msns and msn.queue

CALLS :

arctan.f	arty.time
dist	tracer

SCHEDULES :
checking.guns.availibility
fo.not.busy

SCHEDULED BY :
commo.attempt

COMPLEXITY : Execution time is linear with n.fdc and storage is constant.

IMPROVEMENTS : Combine all "for" loops into one.

FINAL.DEATH

NUMBER BYTES OBJECT CODE : 1424

PARAMETERS :
a

LOCAL VARIABLES :
a

GLOBAL VARIABLES :

alive.dead	defnum
f.d	fired.at
fkill	guntube
hit.state	k.hit
kkill	m.d
mfkill	mkill
name	ncase
num.hit	spd
time.v	wpn.type
x.current	y.current
z.current	

WRITES :

defnum	f.d
fired.at	fkill
guntube	hit.state
k.hit	kkill
m.d	mfkill
mkill	name
ncase	num.hit
spd	time.v
wpn.type	x.current
y.current	z.current

CALLS :
tally.hit.state

SCHEDULED BY :
atrit

COMPLEXITY : Constant execution time and storage requirements.

FIRE

NUMBER BYTES OBJECT CODE : 2232

PARAMETERS :

a	id

LOCAL VARIABLES :

a	id
lose	r
stop.count	

GLOBAL VARIABLES :

alive.dead	blue
bwd.look	check.time
color	critical.value
defnum	foe
fip	fkill
fwd.look	mv.state
mzl.vel	op.rng
pct.vis	pointer
projo	range
sched	second.shot
tgt.scl	time.v
wpn.type	x.current
y.current	

CALLS :

decrement.ammo	dist
hider	loc
set.muzzle.vel	sight
stop.to.fire	tracer

SCHEDULES :

impact	target.select

SCHEDULED BY :

t72.tactics	target.select
we.miss	

COMPLEXITY : Constant execution time and storage requirements.

FIRST

NUMBER BYTES OBJECT CODE : 288

PARAMETERS :
a

LOCAL VARIABLES :
a

GLOBAL VARIABLES :
mu range
wpn.type

CALLS :
trunc.f

CALLED BY :
lay.load

COMPLEXITY : Constant execution time and storage requirements.

FO.NOT.BUSY

NUMBER BYTES OBJECT CODE : 408

PARAMETERS :
id.fo

LOCAL VARIABLES :
id.fo

GLOBAL VARIABLES :
amt.active.msns idle
status

SCHEDULES :
update.cluster

SCHEDULED BY :
end.of.mission fdc.processing

COMPLEXITY : Constant execution time and storage requirements.

GEOM

NUMBER BYTES OBJECT CODE : 8080

PARAMETERS :

addefl	addel
deflbias	deflsig
elbias	elsig
f.pcvis	pc.vis
r	sh.t
tgt.t	

LOCAL VARIABLES :

addefl	addel
aimdis	defdis
deflbias	deflmiss
deflsig	diswk
efkill	efpl
elbias	eldis
elmis	elmiss
elsig	emkill
empl	f.pcvis
fk	gamma
i	io
j	jo
k	kaykill
kaypl	kk
l	length
m	mk
mo	mp
n	pc.vis
r	ro
sh.t	size
tgt.t	turret
vel	whocalled
width	x.t
y.t	

GLOBAL VARIABLES :

alive.dead	damage.num
dir.of.mvmt	hnorm
jnorm	kkill
le11	le12
le31	le61
le71	le72
le81	le83
norseed	ph
projo	spd
tardim	veh.type
wpn.type	x.current
y.current	

GEOM (cont)

CALLS :

abs.f	arctan.f
atrit	cos.f
loadn	sin.f
trunc.f	

CALLED BY :

compute

COMPLEXITY : Constant execution time and storage requirements.

GET.UP

NUMBER BYTES OBJECT CODE : 576

PARAMETERS :

a	b

LOCAL VARIABLES :

a	b

GLOBAL VARIABLES :

ap.tow	bn
defnum	mv.state
name	red.alive
tank	target
wpn.type	

CALLS :

loc

CALLED BY :

charge

COMPLEXITY : Constant execution time and storage requirements.

GUNS.FIRING

NUMBER BYTES OBJECT CODE : 1768

PARAMETERS :

id.btry	id.fdc
id.fo	id.mission

LOCAL VARIABLES :

id.btry	id.fdc
id.fo	id.mission
rg	tof
type.ammo	wpn.type

GLOBAL VARIABLES :

adj.round	ammunition.type
caliber	debug
gt.final.rg	msn.name
my.radio	now.firing
time.v	travel.time.array
which.volley	
x.curl	x.future.loc
y.curl	y.future.loc

WRITES :

id.btry	id.fdc
id.fo	msn.name
time.v	which.volley

CALLS :

dist	tracer

SCHEDULES :

arty.impact	busy.radio.net
open.radio.net	

SCHEDULED BY :

arty.impact
checking.guns.availability
end.of.mission

COMPLEXITY : Constant execution time and storage requirements.

HIDE

NUMBER BYTES OBJECT CODE : 1536

PARAMETERS :

a	whocalled

LOCAL VARIABLES :

a	hold
whocalled	

GLOBAL VARIABLES :

alive.dead	defnum
mv.state	

CALLS :

hider	reload

SCHEDULES :

hide

SCHEDULED BY :

hide	reload
we.hit	we.miss

COMPLEXITY : Constant storage and execution time.

HIDER

NUMBER BYTES OBJECT CODE : 320

PARAMETERS :

a

LOCAL VARIABLES :

a	adef
aname	wtype

GLOBAL VARIABLES :

defnum	micro
name	wpn.type
zh	

CALLS :

mcov

CALLED BY :

bl.create	defend
df.chg	dismount.dragon
fire	hide
red.create	target.select
we.hit	we.miss

COMPLEXITY : Constant storage and execution time.

IFV.TACTICS

NUMBER BYTES OBJECT CODE : 352

PARAMETERS :

a	b

LOCAL VARIABLES :

a	answer
b	z

GLOBAL VARIABLES :

foe	plt
plt.unit	tank

CALLS :

uniform.f

CALLED BY :

target.select

COMPLEXITY : Execution is linear on tank in plt.unit and storage is constant.

IMPACT

NUMBER BYTES OBJECT CODE : 4488

PARAMETERS :

a	id
y	

LOCAL VARIABLES :

a	answer
dt	id
r	stopcount
whocalled	x
y	

GLOBAL VARIABLES :

alive.dead	bwd.look
check.time	co
critical.value	damage.num
dam.array	defnum
fa	f.d
fired.at	fip
fkill	foe
fwd.look	g.amm
guntube	hit.state
k.hit	killer
kkill	m.d
mfkill	mkill
mmm	mv.state
name	ncase
num.hit	pcb.vis
pct.vis	pointer
projo	range
spd	time.v
tow.kount	ttt
wpn.type	x.current
y.current	z.current

WRITES :

defnum	f.d
fired.at	fkill
g.amm	guntube
hit.state	k.hit
kkill	m.d
mfkill	mkill
name	ncase
num.hit	pcb.vis
projo	range
spd	time.v
ttt	wpn.type
x.current	y.current
z.current	

CALLS :

bug.check	compute
dist	list.update
loc	sector.check
sight	stop.to.fire
tally.hit.state	tracer
we.hit	we.miss

SCHEDULES :

detect	mrl.impact
new.fo	

SCHEDULED BY :
fire

COMPLEXITY : Constant execution time and storage requirements.

ITV.TACTICS

NUMBER BYTES OBJECT CODE : 352

PARAMETERS :

a	b

LOCAL VARIABLES :

a	answer
b	z

GLOBAL VARIABLES :

foe	plt
plt.unit	tank

CALLS :

uniform.f

COMPLEXITY : Execution time linear on tanks in plt.unit and storage is constant.

CALLED BY :

target.select

REMARKS : Returns the value of answer to the calling routine.

LAY.LOAD

NUMBER BYTES OBJECT CODE : 2136

PARAMETERS :

a	x

LOCAL VARIABLES :

a	time
x	

GLOBAL VARIABLES :

projo	wpn.type

CALLS :

first	max.f
normal.f	

CALLED BY :

t72.tactics	target.select
we.miss	

COMPLEXITY : Constant execution time and storage requirements.

REMARKS : Returns the value of time to the calling routine.

LEAVE.CHECK

NUMBER BYTES OBJECT CODE : 9768

PARAMETERS :

a

LOCAL VARIABLES :

a	b
bb	bc
cb	cc
ij	ik
im	

GLOBAL VARIABLES :

ap.tow	blue
bn	co
color	comp.unit
defnum	fa
hasty	m2
mv.state	name
plt	plt.unit
red	red.alive
tank	target
t.dead	time.v
t.spd	upper.lower
wpn.type	

CALLS :

dr.mount	loc
trunc.f	

CALLED BY :

tally.hit.state

SCHEDULES :

charge	df.chg

COMPLEXITY : Execution time depends on tanks in comp.unit * tanks in plt.unit and storage is constant.

LIST.UPDATE

NUMBER BYTES OBJECT CODE : 1840

PARAMETERS :

a	b
lose	whocalled

LOCAL VARIABLES :

a	b
count	flag
i	lose
size	whocalled

GLOBAL VARIABLES :

alive.dead	fa
list	temp.tgt

RESERVES :

list

RELEASES :

list

CALLS :

dim.f	min.f
uniform.f	

CALLED BY :

detect	impact
proximity.detect	target.select

SCHEDULES :

target.select

COMPLEXITY : Execution time and storage are linear on elements in list.

LOC

NUMBER BYTES OBJECT CODE : 1096

PARAMETERS :
tank

LOCAL VARIABLES :
tank

GLOBAL VARIABLES :

alive.dead	blue
co	color
defnum	list
mkill	m.red.alive
mv.state	name
plt	red
spd	target
wpn.type	

REMOVES :
tank from red.alive, comp.unit and plt.unit

RELEASES :
list

CALLS :

convert.back	defend
param.set	

CALLED BY :

assessment	commo.pass.tgt
detect	dismount.dragon
doing.clusters	fire
get.up	impact
leave.check	loc.update
mrl.impact	position.update
red.arty.fires	stop.simulation
t72.tactics	target.select

SCHEDULES :
df.chg

COMPLEXITY : Constant execution time and storage requirements.

LOC.UPDATE

NUMBER BYTES OBJECT CODE : 384

LOCAL VARIABLES :
tank

GLOBAL VARIABLES :
alive.dead delta.t

CALLS :
loc

SCHEDULES :
loc.update

SCHEDULED BY :
loc.update red.create

COMPLEXITY : Constant execution time and storage requirements.

MAIN

NUMBER BYTES OBJECT CODE : 5928

LOCAL VARIABLES :

cnum	i
j	pnum

GLOBAL VARIABLES :

area	b.area
bc.count	b.num.alive
b.pct.att	bttime
case	c.bar
cdtime	company.commander
constant	critical.value
d.num	dam.array
def.time	delta.t
ds1	ds2
guntube	hasty
i.dead	it.dead
jnorm	lin
lines.v	m.det
mmm	mu
ncase	n.company.commander
nnn	norseed
n.platoon.leader	pca.unc
pca.vis	pcb.unc
pcb.vis	p.hat
platoon.leader	p1.c
p.v	qq
r.area	rc.count
r.num.alive	r.pct.att
seed.v	s1.time
s2.time	steps
target	t.dead
temp.tgt	tgtscl
ttt	upper.lower
v.ms	w.k.c
x.ct	x.stop
y.ct	y.stop
z.ct	zh

READS :

b.num.alive	b.pct.att
cnum	delta.t
ds1	ds2
guntube	mu
ncase	pnum
r.num.alive	r.pct.att
seed.v	upper.lower
x.stop	y.stop

WRITES :

b.pct.att	guntube
norseed	r.pct.att
upper.lower	x.stop
y.stop	

CREATES :

every platoon.leader every company.commander

RESERVES :

bbbpoint	c.bar
dam.array	d.num
ds1	ds2
i.dead	it.dead
m.det	mu
pca.unc	pca.vis
pcb.unc	pcb.vis
p.hat	p1.c
p.v	qq
rrrpoint	target
t.dead	temp.tgt
tgtscl	v.ms
x.ct	y.ct
z.ct	zh

RELEASES :

fa.1.main	fa.2.main
res1	res2
res3	res4
res5	

CALLS :

b1.create	cos.f
fa.1.main	fa.2.main
initrd	loadn
res1	res2
res3	res4
res5	set.sector
vals.for.case	

SCHEDULES :

attrition.check	new.forces
step.time	stop.simulation
stop.simulation	

COMPLEXITY : Execution is dependent on the number of tanks. Storage is dependent on the number of platoon leaders, the number of company commanders and the number of elements in r.num.alive and b.num.alive.

REMARKS : Main routine starts the simulation.

MRL.IMPACT

NUMBER BYTES OBJECT CODE : 2568

PARAMETERS :

x.loc	y.loc

LOCAL VARIABLES :

id.btry	id.red.btry
no.mens.fired	pk
red.2.constant	type.ammo
x	x.loc
y.loc	

GLOBAL VARIABLES :

alive.dead	arty.pk.table
caliber	debug
defnum	f.d
fired.at	fkill
foe	guntube
hit.state	
k.hit	kkill
m.d	mfkill
mkill	name
ncase	no.msns.fired
num.he.left	num.hit
red.1.constant	spd
tank	time.v
wpn.type	x.current
y.current	z.current

WRITES :

caliber	defnum
f.d	fired.at
fkill	guntube
hit.state	k.hit
kkill	m.d
mfkill	mkill
name	ncase
num.hit	spd
time.v	type.ammo
x.current	y.current
z.current	

CALLS :

abs.f	atrit
loc	tracer
uniform.f	

SCHEDULED BY :

impact

COMPLEXITY : Execution time is linear on tanks in blue.alive and storage is constant.

REMARKS : Local variable no.mens.fired should be no.msns.fired.

NEW.COORDINATE.SYSTEM

NUMBER BYTES OBJECT CODE : 88

PARAMETERS :

angle	xold
yold	

LOCAL VARIABLES :

angle	xnew
xold	ynew
yold	

CALLS :

cos.f	sin.f
tracer	

CALLED BY :

assessment	error

COMPLEXITY : Constant execution time and storage requirements.

REMARKS : Yielding xnew and ynew

NEW.FO

NUMBER BYTES OBJECT CODE : 456

PARAMETERS :

a

LOCAL VARIABLES :

blue.alive	co
fa	plt.ldr
tank	type

SCHEDULED BY :

impact

COMPLEXITY : Execution time is linear on tank in blue.alive with co(tank) = co(a), until tank = pltldr(tank). Storage requirements are constant.

NEW.FORCES

NUMBER BYTES OBJECT CODE : 128

PARAMETERS :

start	stop

LOCAL VARIABLES :

start	stop

GLOBAL VARIABLES :

name

RELEASES :

bbbpoint	red.create
rrrpoint	

CALLS :

red.create

SCHEDULED BY :

main

COMPLEXITY : Constant execution time and storage requirements.

NEW.LOCATION

NUMBER BYTES OBJECT CODE : 1384

PARAMETERS :

a	del.time
id.mission	

LOCAL VARIABLES :

a	del.time
distance	err.1
err.2	err.3
err.4	err.5
err.6	err.7
err.8	id.btry
id.fdc	id.fo
id.mission	tank
x.1	x.2
x.3	xx

GLOBAL VARIABLES :

dir.apparent	direction
error.code	mission
spd.apparent	type
x.cur4	x.future.loc
y.cur4	y.future.loc

CALLS :

arctan.f	cos.f
dist	fa.tgt.error
position.update	sin.f
tracer	

CALLED BY :

arty.impact	update.cluster

COMPLEXITY : Constant execution time and storage requirements.

NEW.MISSION

NUMBER BYTES OBJECT CODE : 1432

PARAMETERS :

id.fo	m
name.priority	priority

LOCAL VARIABLES :

a	id.fo
m	name.priority
priority	

GLOBAL VARIABLES :

amt.in.clusters	clusters
direction	fist
fo.tgt.range	lcount
mission	msn.name
no.clusters	pointing.to
pri.of.cluster	speed
time.of.update	time.v
x.cur4	y.cur4

WRITES :

a	direction
id.fo	msn.name
speed	x.cur4
y.cur4	

CREATES :

mission

CALLS :

dist	tracer

CALLED BY :

update.cluster

COMPLEXITY : Constant execution time and storage requirements.

OPEN.RADIO.NET

NUMBER BYTES OBJECT CODE : 240

PARAMETERS :
id.radio

LOCAL VARIABLES :
id.radio

GLOBAL VARIABLES :
state3

CALLS :
tracer

SCHEDULED BY :

arty.impact	commo.attempt
end.of.mission	guns.firing

COMPLEXITY : Constant execution time and storage requirements.

PARAM.SET

NUMBER BYTES OBJECT CODE : 920

PARAMETERS :
a

LOCAL VARIABLES :

a	aname
weaponry	

GLOBAL VARIABLES :

c.bar	cbar
defnum	dir.mvmt
d.num	m.det
micro	mine.det
mv.time	name
p.hat	pi.hat
pl.c	plow.cond
p.v	spd
time.v	t.spd
v.ms	wpn.type
x.ct	x.current
y.ct	y.current
z.ct	z.current
zh	

CALLS :
move

CALLED BY :
loc

COMPLEXITY : storage requirements and execution time are constant.

PARAMETERS

NUMBER BYTES OBJECT CODE : 1128

PARAMETERS :

j	k
rg	type.ammo

LOCAL VARIABLES :

count	delta.1
delta.2	fraction
i	j
k	rg
sig.df	sig.rg

GLOBAL VARIABLES :

no.range.bands	range.bands

CALLS :

tracer

CALLED BY :

assessment	error

COMPLEXITY : execution time is linear depending on the number of range bands. Storage is constant.

REMARKS : This routine returns the value of sig.rg and sig.df to the calling routine.

POP.A.MINE

NUMBER BYTES OBJECT CODE :1760

PARAMETERS :

tnk	type

LOCAL VARIABLES :

efkill	emkill
jo	kaykill
tnk	type
whocalled	

GLOBAL VARIABLES :

damage.num	dam.array
defnum	f.d
fired.at	fkill
guntube	hit.state
k.hit	kkill
m.d	mfkill
minleth	
mkill	name
ncase	num.hit
plow.cond	
spd	time.v
wpn.type	x.current
y.current	z.current

WRITES :

defnum	f.d
fired.at	fkill
guntube	hit.state
k.hit	kkill
m.d	mfkill
mkill	name
ncase	num.hit
spd	time.v
wpn.type	x.current
y.current	z.current

CALLS :

atrit

CALLED BY :

convert.back

COMPLEXITY : Storage requirements and execution time are constant.

POSITION.UPDATE

NUMBER BYTES OBJECT CODE : 1104

PARAMETERS :

a	id.mission

LOCAL VARIABLES :

a	id.fo
id.mission	spd.bar
tank	x.bar
y.bar	

GLOBAL VARIABLES :

b.org.alive	c.number.array
direction	fist
name	no.clusters
red.alive	spd
speed	state4
time.v	t.position
x.current	x.cur4
y.current	y.cur4

CALLS :

cos.f	loc
sin.f	tracer

CALLED BY :

arty.impact	error
new.location	

COMPLEXITY : Storage requirements are constant. Execution is linear on tanks in red.alive.

PREPLANNED

NUMBER BYTES OBJECT CODE : 1008

PARAMETERS :

a	b
c	d
e	f

LOCAL VARIABLES :

a	b
c	d
e	f
i	j
k	l

GLOBAL VARIABLES :
red.planned.fires

CALLED BY :
fa.1.main

COMPLEXITY : Constant execution time and storage requirements.

PRINT

NUMBER BYTES OBJECT CODE : 336

PARAMETERS:
id.mission

LOCAL VARIABLES :

debug	id.mission
rad.err	x.cur4
x.future.loc	y.cur4
y.future.loc	

WRITES :
rad.err

CALLS :
dist

CALLED BY :
arty.impact assessment

COMPLEXITY : Constant execution time and storage requirements.

PRINT1

NUMBER BYTES OBJECT CODE : 1124

PARAMETERS :

id.mission	

LOCAL VARIABLES :

i	id.fo
id.mission	j
tank	

GLOBAL VARIABLES :

b.org.alive	c.number.array
debug	fist
name	no.cluster
rd.offset	
red.alive	state4
tank	x.cur4
x.current	x.future.loc
y.cur4	y.current
y.future.loc	

WRITES :

i	j
name	rd.offset
x.cur4	x.current
x.future.loc	y.cur4
y.current	y.future.loc

CALLED BY :

arty.impact	assessment

COMPLEXITY : Execution time is linear on tank in red.alive and storage is constant.

PRIORITY.AND.ROUND.SELECT

NUMBER BYTES OBJECT CODE : 1336

PARAMETERS :

a	b

LOCAL VARIABLES :

a	answer
b	i
j	p
rnd	threshold

GLOBAL VARIABLES :

blue	color
dsl	range
wpn.type	

CALLS :

ammo.check	trunc.f

CALLED BY :

target.select

COMPLEXITY : Constant execution time and storage requirements.

REMARKS : Returns the value of p and rnd to the calling routine.

PROXIMITY.DETECT

NUMBER BYTES OBJECT CODE : 864

PARAMETERS :

a	b

LOCAL VARIABLES :

a	b
whocalled	x.sample
y.sample	

GLOBAL VARIABLES :

alive.dead	blue
color	plt
plt.unit	tank
x.current	y.current

CALLS :

abs.f	list.update

CALLED BY :

detect

COMPLEXITY : Execution time is linear on tank in plt.unit and storage is constant.

RED.CREATE

NUMBER BYTES OBJECT CODE : 2408

PARAMETERS :

a	b

LOCAL VARIABLES :

a	b
i	

GLOBAL VARIABLES :

ap.tow	array.detect
bbbpoint	bc.count
bn	b.num.alive
c.number.array	co
cocdr	defnum
dir.of.mvmt	list
max.number.of.missions.per.fo	
mv.state	name
n.fo	op.rng
pi.c	plow.cond
plt	pltldr
pointer	pri.dir
spd	tank
target	time.v
wpn.type	x.current
y.current	

READS :

bn	co
cocdr	name
plt	pltldr
wpn.type	x.current
y.current	

CREATES :

tank

FILES :

tank in tanks, red.alive, comp.unit and plt.unit

RESERVES :

array.detect	c.number.array
list	

RELEASES :

array.detect	c.number.array

CALLS :

basic.load	hider
set.sector	

CALLED BY :

new.forces

SCHEDULES :

loc.update

COMPLEXITY : Execution is dependent on input parameters a and b, main loop will be executed b-a+1 times per invocation. Storage is dependent on bc.count, n.fo and max.number.of.missions.per.fo.

RED.ARTY.FIRES

NUMBER BYTES OBJECT CODE : 3368

PARAMETERS :

id.red.btry	iteration

LOCAL VARIABLES :

i	id.red.btry
iteration	pk
red.2.constant	type.ammo
x	

GLOBAL VARIABLES :

alive.dead	arty.pk.table
blue.alive	caliber
defnum	f.d
fired.at	fkill
foe	guntube
hit.state	k.hit
kkill	kount
m.d	mfkill
mkill	name
ncase	
no.msns.fired	num.dpicm.left
num.guns	num.he.left
num.hit	
red.planned.fires	rn.stream
salvos	spd
tank	time.v
wpn.type	x.current
y.current	z.current

WRITES :

caliber	defnum
f.d	fired.at
fkill	guntube
hit.state	k.hit
kkill	m.d
mfkill	mkill
name	ncase
num.hit	spd
time.v	type.ammo
x.current	y.current
z.current	

CALLS :

atrit	loc
tracer	uniform.f

SCHEDULES :

red.arty.fires

SCHEDULED BY :

fa.2.main	red.arty.fires

COMPLEXITY : Storage requirement is constant. Execution time is dependent on the product of salvos and tanks in blue.alive.

RELOAD

NUMBER BYTES OBJECT CODE : 2736

PARAMETERS :
a

LOCAL VARIABLES :
a

GLOBAL VARIABLES :

btime	c.1
c.2	capds
case	cda
cdh	cdtime
cheat	crf
defnum	def.time
guntube	init
r.con	s1.time
s2.time	veh.type

CALLED BY :
hide

SCHEDULES :
hide

COMPLEXITY : Constant execution time and storage requirements.

RES1

NUMBER BYTES OBJECT CODE : 232

GLOBAL VARIABLES :

hnorm	norseed

READS :
norseed

RESERVES :

hnorm	norseed

CALLED BY :
main

COMPLEXITY : Constant execution time and storage requirements.

RES2

NUMBER BYTES OBJECT CODE : 2000

GLOBAL VARIABLES :

accbm	accht
accke	accmsl
addon	bm.mov
bushbmp	dgnv
ht.mov	ke.mov
le11	le12
le31	le61
le71	le72
le81	le82
minleth	sovmg
tardim	usmg

RESERVES :

accbm	accht
accke	accmsl
addon	bm.mov
bushbmp	dgnv
ht.mov	ke.mov
le11	le12
le31	le61
le71	le72
le81	le83
minleth	sovmg
tardim	usmg

CALLED BY :
main

COMPLEXITY : Constant execution time and storage requirements.

REMARKS : Rearrangement of subscripts will provide more efficient storage utilizing fewer pointers.

RES3

NUMBER BYTES OBJECT CODE : 3336

LOCAL VARIABLES :

i	j
k	l
m	

GLOBAL VARIABLES :

le11	le12
le31	le61
le71	le72
le81	le83

CALLED BY :
main

COMPLEXITY : Constant execution time and storage requirements.

REMARKS : Rearranging loops to avoid duplication will cut the number of "for" loops from 11 to 6.

RES4

NUMBER BYTES OBJECT CODE : 1912

LOCAL VARIABLES :

i	j
k	l
m	

GLOBAL VARIABLES :

accht	accke
accmsl	addon
dgnv	ht.mov
ke.mov	tardim

CALLED BY :
main

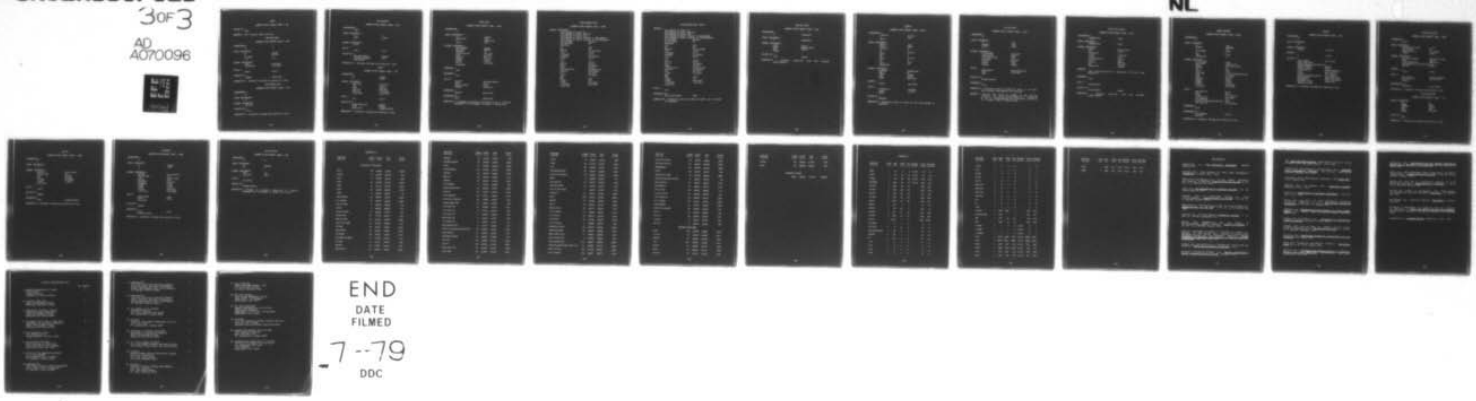
COMPLEXITY : Constant execution time and storage requirements.

REMARKS : Combine 7 "for" loops with i=1 to 2 into 1 loop.

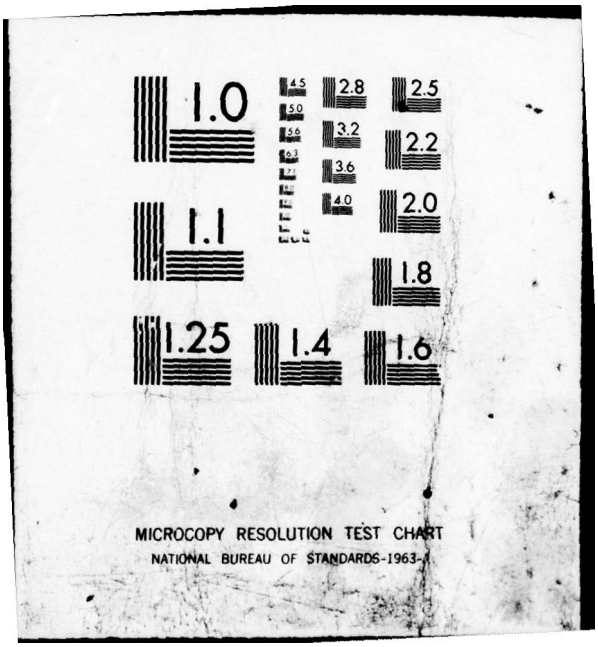
AD-A070 096 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
SYSTEMS ANALYSIS FOR THE INTERACTIVE SIMULATION WITH GRAPHICAL --ETC(U)
MAR 79 G S COKER, D R FORINASH

UNCLASSIFIED NL

3 OF 3
AD A070096



END
DATE FILMED
7-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

RES5

NUMBER BYTES OBJECT CODE : 48

CALLED BY :
main

REMARKS : This routine does nothing.

SECTOR.CHECK

NUMBER BYTES OBJECT CODE : 608

PARAMETERS :

a	b

LOCAL VARIABLES :

a	answer
b	c.left
c.right	r
x.t	y.t

GLOBAL VARIABLES :

area	constant
x.current	y.current

CALLS :

abs.f	sqrt.f

CALLED BY :

impact	step.time

COMPLEXITY : Constant storage and execution time.

SET.MUZZLE.VEL

NUMBER BYTES OBJECT CODE : 400

PARAMETERS :
a

LOCAL VARIABLES :
a

GLOBAL VARIABLES :
mzl.vel

CALLED BY :
fire

COMPLEXITY : Constant storage and execution time.

SET.SECTOR

NUMBER BYTES OBJECT CODE : 448

PARAMETERS :

tank	

LOCAL VARIABLES :

a	b
tank	width
x	y

GLOBAL VARIABLES :

pi.c	

CALLS :

cos.f	sin.f

CALLED BY :

chg.sec.search	defend
dismount.dragon	main
red.create	

COMPLEXITY : Constant storage and execution time.

SIGHT

NUMBER BYTES OBJECT CODE : 712

PARAMETERS :

a	aname
b	bname

GLOBAL VARIABLES :

bwd.look	fwd.look
hto	micro
name	pca.unc
pca.vis	pcb.unc
pcb.vis	x.current
y.current	z.current

CALLS :

los	min.f

CALLED BY :

commo.pass.tgt	detect
fire	impact
step.time	target.select

COMPLEXITY : Constant storage and execution time.

STEP.TIME

NUMBER BYTES OBJECT CODE : 1664

PARAMETERS :

a

LOCAL VARIABLES :

a	answer
bdet.time	lose
r	rdet.time
rn.b	rn.r

GLOBAL VARIABLES :

alive.dead	answer
bwd.look	defnum
delta.t	fa
f.blue.alive	fwd.look
name	op.rng
pcb.unc	pct.vis
steps	target
tgtscl	time.v
w.k.c.	x.current
y.current	

RESERVES :

list

RELEASES :

list

CALLS :

cardio	chg.sec.search
dist	min.f
sector.check	sight
tracer	uniform.f

SCHEDULES :

detect	step.time

SCHEDULED BY :

main	step.time

COMPLEXITY : Storage is constant. Execution time is linearly dependent on the number of tanks in red.alive.

STOP.SIMULATION

NUMBER BYTES OBJECT CODE : 4296

GLOBAL VARIABLES :

attributes of every fo	
attributes of every battery	
attributes of every fdc	
attributes of every mission in msn.queue	
attributes of every mission in holding.msns	
alive.dead	ap.tow
awl.or.msl3	bn
bc.count	
c.1	c.2
co	crf
defnum	dir.of.mvmt
f.d	f.hit
fired.at	fkill
he.drag	hit.state
k.hit	kkill
lin	
m.d	m.hit
mf.hit	mfkill
mkill	mv.state
name	nd.hit
n.blue.alive	n.red.alive
num.hit	plow.cond
plt	pri.dir
r.con	rc.count
sec	
spd	tank
time.v	trf
t.spd	wpn.type
x.current	y.current
z.current	

WRITES :

attributes of every fo	
attributes of every battery	
attributes of every fdc	
attributes of every mission in msn.queue	
attributes of every mission in holding.msns	
alive.dead	ap.tow
awl.or.msl3	bn
bc.count	
c.1	c.2
co	crf
defnum	dir.of.mvmt
f.d	f.hit
fired.at	fkill
he.drag	hit.state
k.hit	kkill
lin	
m.d	m.hit
mf.hit	mfkill
mkill	mv.state
name	nd.hit
n.blue.alive	n.red.alive
num.hit	plow.cond
plt	pri.dir
r.con	rc.count
sec	
spd	tank
time.v	trf
t.spd	wpn.type
x.current	y.current
z.current	

CALLS :

loc

SCHEDULED BY :

attrition.check	main

COMPLEXITY : Execution time is linear on tanks and storage is constant.

STOP.TO.FIRE.

NUMBER BYTES OBJECT CODE : 368

PARAMETERS :

a	stopcount

LOCAL VARIABLES :

a	stopcount

GLOBAL VARIABLES :

blue	color
projo	second.shot
spd	time.v
t.spd	

CALLED BY :

fire	impact

COMPLEXITY : Constant execution time and storage requirements.

SUBCAL

NUMBER BYTES OBJECT CODE : 4040

PARAMETERS :

f.pcvis	pc.vis
r	sh.t
tgt.t	

LOCAL VARIABLES :

efkill	efpl
emkill	empl
f.pcvis	i
io	j
jo	kaykill
kaypl	l
mo	n
nhit	pc.vis
phit	r
ro	sh.t
tgt.t	vel
whocalled	

GLOBAL VARIABLES :

bushbmp	damage.num
dgnv	ph
projo	sovmg
spd	tardim
usmg	wpn.type

WRITES :

projo	wpn.type

CALLS :

binomial.f	bushbmp
sovmg	trunc.f
usmg	

CALLED BY :

compute

COMPLEXITY : Execution time is linear on nhit and storage is constant.

T72.TACTICS

NUMBER BYTES OBJECT CODE : 1216

PARAMETERS :

a

LOCAL VARIABLES :

a	aim
answer	lose
result	time
x	

GLOBAL VARIABLES :

alive.dead	bwd.look
check.time	cocdr
critical.value	foe
fwd.look	pct.vis
pltldr	projo
sched	tank
time.a	time.v

CALLS :

ammo.check	commo.pass.tgt
exp.f	lay.load
loc	

CALLED BY :

target.select

SCHEDULES :

fire

COMPLEXITY : Execution time is linear on tank in plt.unit and storage requirements are constant.

REMARKS : Returns the value of answer to the calling routine. By initializing answer to 1 instead of 0, two "if" sequences may be eliminated resulting in 11 fewer lines of source code.

TALLY.HIT.STATE

NUMBER BYTES OBJECT CODE : 1856

PARAMETERS :

damage.num	tank

LOCAL VARIABLES :

damage.num	tank

GLOBAL VARIABLES :

blue	blue.alive
co	color
comp.unit	
f.hit	fired.at
k.hit	list
m.blue.alive	mf.hit
m.hit	name
no.hit	num.hit
plt	plt.unit
red.alive	target

REMOVES :

tank from blue.alive or red.alive, plt.unit and comp.unit

RELEASES :

list

CALLS :

leave.check

CALLED BY :

final.death	impact

COMPLEXITY : Constant execution time and storage requirements.

TARGET.SELECT

NUMBER BYTES OBJECT CODE : 2592

PARAMETERS :

a	

LOCAL VARIABLES :

a	aim
answer	engaged
i	id
old.range	oldp
p	r
rnd	time
whocalled	x

GLOBAL VARIABLES :

alive.dead	blue
bwd.look	check.time
color	critical.value
defnum	fa
fip	fire
fkill	foe
fwd.look	line.of.sight.exists
list	mv.state
name	pct.vis
pointer	projo
range	sched
target	time.a
time.v	wpn.type
x.current	y.current

CALLS :

bmp.tactics	dim.f
dist	exp.f
hider	ifv.tactics
itv.tactics	lay.load
list.update	loc
priority.and.round.select	sight
t72.tactics	xm1.tactics

SCHEDULES :

fire

SCHEDULED BY :

list.update	we.hit
we.miss	

COMPLEXITY : Constant storage and execution time.

TRACER

. NUMBER BYTES OBJECT CODE : 304

PARAMETERS :

a

LOCAL VARIABLES :

a

GLOBAL VARIABLES :

time.v	tr.time

WRITES :

a	time.v

CALLED BY :

arty.impact	arty.time
busy.radio.net	
checking.guns.availability	
commo.attempt	detect
end.of.mission	error
fa.tgt.error	fdc.processing
fire	guns.firing
impact	mrl.impact
new.coordinate.system	new.location
new.mission	open.radio.net
parameters	position.update
red.arty.fires	step.time
update.cluster	

COMPLEXITY : Constant storage and execution time.

UPDATE.CLUSTER

NUMBER BYTES OBJECT CODE : 2256

PARAMETERS :

id.fo	

LOCAL VARIABLES :

estimate.of.time	id.fo
id.mission	m
name.priority	pri.value
time.1	time.2

GLOBAL VARIABLES :

amt.active.msns	amt.msns.fired
busy	debug
last.clustered	msn.time
max.number.of.missions.per.fo	
state4	time.v

WRITES :

id.fo	time.v

CALLS :

arty.time	doing.clusters
new.location	new.mission
tracer	

SCHEDULED BY :

fa.2.main	fo.not.busy

COMPLEXITY : Constant storage and execution time

VALS.FOR.CASE

NUMBER BYTES OBJECT CODE : 712

GLOBAL VARIABLES :

ba	bh
capds	case
caseap	casehe
cda	cdh
cheat	guntube
init	

CALLED BY :

main

COMPLEXITY : Constant storage and execution time.

WE.HIT

NUMBER BYTES OBJECT CODE : 1328

PARAMETERS :
a

LOCAL VARIABLES :
a

GLOBAL VARIABLES :

ap.tow	aw1.or.ms13
aw2.or.adm	c.1
c.2	defnum
foe	he.drag
hitshot	missshot
r.con	r.count
wpn.type	

CALLS :
hider

CALLED BY :
impact

SCHEDULES :

hide	target.select

COMPLEXITY : Constant storage and execution time.

WE.MISS

NUMBER BYTES OBJECT CODE : 1808

PARAMETERS :

a	

LOCAL VARIABLES :

a	answer
r	time
x	

GLOBAL VARIABLES :

ap.tow	aw1.or.ms13
aw2.or.adm	c.1
c.2	check.time
defnum	foe
he.drag	hitshot
missshot	projo
range	r.con
sched	time.a
time.v	wpn.type
x.current	y.current

CALLS :

ammo.check	dist
exp.f	hider
lay.load	

CALLED BY :

impact

SCHEDULES :

fire.	hide
target.select	

COMPLEXITY : Constant storage and execution time.

XM1.TACTICS

NUMBER BYTES OBJECT CODE : 352

PARAMETERS :

a	b

LOCAL VARIABLES :

a	answer
b	z

GLOBAL VARIABLES :

foe	plt
plt.unit	tank

CALLS :

uniform.f

CALLED BY :

target.select

COMPLEXITY : Storage is constant. Execution is linearly dependent on the number of tanks in plt.unit.

APPENDIX C

ROUTINE	LINES	START	END	BYTES
SIMSCRIPT ROUTINES				
main	95	ba028	bb750	5928
hider	10	bb750	bb890	320
res1	3	bb890	bb978	232
res2	24	bb978	bc148	2000
res3	37	bc148	bce50	3336
res4	18	bce50	bd5c8	1912
res5	2	bd5c8	bd5f8	48
red.create	43	bd5f8	bdf60	2408
bl.create	46	bdf60	bea68	2824
new.forces	5	bea68	bebe8	128
sight	9	bebe8	beeb0	712
convert.back	17	beeb0	bf178	712
param.set	21	bf178	bf510	920
basic.load	27	bf510	bfd38	2088
vals.for.case	10	bfd38	c0000	712
reload	40	c0000	c0ab0	2736
bug.check	12	c0ab0	c0eb0	1024
dr.mount	18	c0eb0	c1358	1192
dismount.dragon	12	c1358	c1640	744
df.chg	14	c1640	c1830	496
defend	12	c1830	c1ae0	688
step.time	65	c1ae0	c2660	1664

ROUTINE	LINES	START	END	BYTES
detect	21	c2660	c2978	792
target.select	74	c2978	c3398	2592
fire	51	c3398	c3c50	2232
leave.check	112	c3c50	c6278	9768
charge	47	c6278	c6ae0	2152
get.up	12	c6ae0	c6d20	576
impact	89	c6d20	c7ea8	4488
loc.update	6	c7ea8	c8028	384
stop.simulation	27	c8028	c90f0	4296
cardio	55	c90f0	c95f0	1280
list.update	62	c95f0	c9d20	1840
proximity.detect	26	c9d20	ca080	864
commo.pass.tgt	21	ca080	ca2b8	488
t72.tactics	27	ca2b8	ca778	1216
ifv.tactics	9	ca778	ca8d8	352
xm1.tactics	9	ca8d8	caa38	352
bmp.tactics	6	caa38	cab40	264
itv.tactics	9	cab40	caca0	96
set.muzzle.vel	8	caca0	cae30	400
priority.and.round.select	49	cae30	cb368	1336
ammo.check	11	cb368	cb558	496
decrement.ammo	13	cb558	cb858	768
we.miss	39	cb858	cbfb8	1808
we.hit	28	cbfb8	cc4e8	1328
stop.to.fire	13	cc4e8	cc658	368
lay.load	35	cc658	ccea8	2136

ROUTINE	LINES	START	END	BYTES
first	5	ccea8	ccfc8	288
hide	23	ccfc8	cd4c8	1536
loc	23	cd4c8	cd910	1096
chg.sec.search	25	cd910	cde28	1304
tally.hit.state	43	cde28	ce568	1856
dist	4	ce568	ce5f8	144
set.sector	12	ce5f8	ce7b8	448
sector.check	10	ce7b8	cea18	608
attrition.check	7	cea18	cec08	496
compute	91	cec08	cf9c8	3528
geom	150	cf9c8	d1958	8080
subcal	79	d1958	d2720	4040
atrit	43	d2720	d2ff0	2256
pop.a.mine	20	d2ff0	d36d0	1760
final.death	14	d36d0	d3c60	1424
fa.1.main	83	d3c60	d4fa8	4936
fa.2.main	44	d4fa8	d5d38	3472
fo.not.busy	9	d5d38	d5fd0	408
update.cluster	46	d5fd0	d64a0	2256
commo.attempt	26	d64a0	d68e8	1096
open.radio.net	6	d68e8	d69d8	240
busy.radio.net	6	d69d8	d6ac8	240
fdc.processing	110	d6ac8	d7828	3424
checking.guns.availability	37	d7828	d7f48	1824
guns.firing	34	d7f48	d8630	1768
arty.impact	132	d8630	d9b50	1312

ROUTINE	LINES	START	END	BYTES
end.of.mission	59	d9b50	da5e0	2704
doing.clusters	114	da5e0	db998	5048
assessment	95	db998	dcb48	4528
error	28	dcb48	dcef8	944
red.arty.fires	75	dcef8	ddc20	3368
new.coordinate.system	8	ddc20	ddd18	88
new.mission	27	ddd18	de2b0	1432
parameters	33	de2b0	de718	1128
arty.time	24	de718	de8b8	416
fa.tgt.error	21	de8b8	dea58	416
new.location	54	dea58	defc0	1384
mrl.impact	53	defc0	df9c8	2568
preplanned	31	df9c8	dfdb8	1008
position.update	29	dfdb8	e0208	1104
print	10	e0208	e0358	336
print1	22	e0358	e0840	1124
tracer	8	e0840	e0970	304
new.fo	8	e0970	e0b38	456
FORTRAN ROUTINES				
move	145	e4c88	e5860	4048
initrd	73	e0e18	e1688	2414
setup	53	e70b8	e7508	1300
los	251	e3260	e4c88	7504
kover	14	ee220	ee3f0	684
elev	33	e2ea0	e31a0	988
dytunx	31	efa18	efaa0	1138

ROUTINE	LINES	START	END	BYTES
elevg	38	eebe8	eef78	1132
mcov	24	e2b88	e2dd8	866
PROGRAM TOTALS				
	3842	ba028	1177e0	382904

APPENDIX D

ROUTINE	TYPE	WDS	PTRS	OPT	REMARK	TOTAL	OPTIMAL
dgnv	i	16	3	3	(*/2)	19	19
usmg	i	48	16	16	(*/2)	64	64
sovmg	i	96	33	33	(*/2)	129	129
bushbmp	i	128	47	47	(*/2)	125	125
accmsl	r	224	35	23		259	247
ke.mov	r	420	83	39		503	459
accke	r	168	51	19		219	187
ht.mov	r	420	83	39		503	459
accht	r	224	67	23		291	247
addon	r	80	7	7		87	87
accbm	r	84	25	9		109	93
bm.mov	r	210	41	19		251	229
tardim	r	198	10	10		208	208
hnorm	r	1000	1	1		1001	1001
norseed	r	2	1	0		3	2
minleth	i	5	10	9		15	14
tgt.acq.error	r	12	3	3		15	15
tgtscl	r	18	1	1		19	19
mu	r	24	3	3		27	27
x.ct	r	2	1	0		3	2
y.ct	r	2	1	0		3	2
z.ct	r	2	1	0		3	2
v.ms	r	2	1	0		3	2

ROUTINE	TYPE	WDS	PTRS	OPT	REMARK	TOTAL	OPTIMAL
c.bar	r	2	1	0		3	2
pl.c	i	1	1	0		2	1
m.det	i	1	1	0		2	1
d.num	i	1	1	0		2	1
p.hat	r	2	1	0		3	2
pca.vis	r	2	1	0		3	2
pcb.unc	r	2	1	0		3	2
pcb.vis	r	2	1	0		3	2
pca.unc	r	2	1	0		3	2
p.v	r	2	1	0		3	2
zh	r	2	1	0		3	2
list	i	1	1	0		2	1
target	i	642	322	3		964	645
temp.target	i	150	1	1		151	151
ds1	i	360	16	16		376	376
ds2	i	362	10	10		372	372
t.dead	i	1	1	0	(*/4)	2	1
i.dead	i	1	1	0	(*/4)	2	1
it.dead	i	1	1	0	(*/4)	2	1
le11	i	1470	1165	691	(*/4)	2635	2161
le71	i	3675	2911	1659	(*/4)	6586	5334
le12	i	210	167	84	(*/4)	377	294
le72	i	525	416	249	(*/4)	941	774
le31	i	210	167	84	(*/4)	377	294
le61	i	210	167	84	(*/4)	377	294

ROUTINE	TYPE	WDS	PTRS	OPT	REMARK	TOTAL	OPTIMAL
le81	i	525	416	249	(*/4)	944	774
le83	i	525	416	249	(*/4)	944	774

BIBLIOGRAPHY

Brooks,F.P. Jr., The Mythical Man-month, Addison Wesley,1975.

Curtice, R.M., "The Outlook for Data Base Management," Datamation, p. 46-49, April 1976.

Daley, R.C. and Dennis, J.B., "Virtual Memory, Processes, and Sharing in MULTICS," Communications of the ACM, V. 11, No. 5, p. 306-312, May 1968.

Date, C.J., An Introduction to Database Systems, 2nd Ed, Addison-Wesley, 1977.

Dayhoff, M.O., "A Contour-Map Program for X-Ray Crystallography," Communications of the ACM, V. 6, No. 10, p. 620-622, October 1963.

Denning, P.J., "The Working Set Model for Program Behavior," Communications of the ACM, V. 11, No. 5, p. 323-333, May 1968.

Denning, P.J., "Virtual Memory," Computing Surveys, v. 2, No.3, p. 153-189, September 1970

Dennis, J.B., "Segmentation and the Design of Multiprogrammed Computer Systems," Journal of the ACM, V. 12, No. 4, p. 589-602, October 1965.

Fram, D., Castleman, P., Webb, F., Bilofsky, W., Zdonik, S., Beranek, B., and Newman Inc., "Mission: A Low-Cost Data Management System for the Biomedical Community," Annual Symposium on Computer Applications in Medical Care, 1st Proceedings, IEEE, New York, 1977.

Graham, G.S. and Denning, P.J., "Protection - Principles and Practice," Spring Joint Computer Conference, 1972.

Honeywell Information Systems, Inc., Multics Programmer's Manual - Subsystem Writer's Guide, September 1975.

JCS, Joint War Gaming Manual, JWGA-167-69, The Joint Chiefs of Staff, Joint War Games Agency, 1 July 1969.

Kiviat, P.J., Villanueva, R. and Markowitz, H.M., SIMSCRIPT II.5 Programming Language, 2nd Edition, Consolidated Analysis Centers, Inc., 1973.

Licklider, JC.R., "Man-Computer Symbiosis," IRE, Trans. HFE, p 4-11, March 1960.

Madnick, S.E., and Donovan, J.J., Operating Systems, McGraw-Hill, Inc., 1974.

Martin, J., Principles of Data-Base Management, Prentice-Hall, Inc., 1976.

Mills, R.E. and Phil, M., "The Interactive Extensible Simulation Capability of CML," 1977 Winter Simulation Conference, p. 331-334, 5-7 December 1977.

Needles, C.J., Parameterization of Terrain in Army Combat Models, M.S. Thesis, Naval Postgraduate School, Monterey, March 1976.

Newman, W.M. and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill, Inc., 1973.

Newman, W.M. and van Dam, a., "Recent Efforts Toward Graphics Standardization," ACM Computing Surveys, V. 10, No. 4, p. 365-387, December 1978.

Organick, E.I., The Multics System: An Examination of its Structure, MIT Press, 1972.

Rahe, G.A., "Simulation and Computer Graphics," Simulation Today No. 4, p. 13-16, August 1972.

Schiller, W.L., The Design and Specification of a Security Kernel for the PDP-11/45, ESD-TR-75-69, The Mitre Corporation, May 1975.

Schroeder, M.D., Performance of the GE-645 Associative Memory While MULTICS is in Operation, Harvard University, April 1971.

Smith, A.J., "Multiproccessor Memory Organization and Memory Interference," Communications of the ACM, V. 20, No. 10, p. 754-761, October 1977.

Smith, L.B., "The Use of Interactive Graphics to Solve Numerical Problems," Communications of the ACM, v. 13, n. 10, p. 625-634, October 1970.

Sohnle, R.C., Tartar, J., and Sampson, J.R., "Requirements for Interactive Simulation Systems," Simulation, p. 145-152, May 1973.

Sutherland, I.E., "Computer Graphics," Datamation, p. 22-27, May 1966.

Whitmore, J., Bensoussan, A., Green, P., Hunt, D., Kobziar, A. and Stern, J., Design for MULTICS Security Enhancements, ESD-TR-74-176, Honeywell Information Systems, Inc., December 1973.

Wiederhold, G., Database Design, McGraw-Hill, Inc., 1977.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center ATTN: DDC-TC Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
4. Professor Sam H. Parry, Code 55 PY Department of Operations Research Naval Postgraduate School Monterey, California 93940	5
5. Maj. George S. Coker 105 Kendallwood Dr. Fredericksburg, Virginia 22401	1
6. Cpt David R. Forinash Staff and Faculty Battalion United States Military Academy West Point, New York 10096	1
7. Office of the Commanding General U. S. Army TRADOC Attn: General Donn A. Starry Ft. Monroe, Virginia 23651	1
8. Headquarters U.S. Army Training & Doctrine Command Attn: ATCG-T (Col. Ed Scribner) Ft. Monroe, Virginia 23651	1

9. Headquarters 1
U.S. Army Training & Doctrine Command
Attn: Director, Analysis Directorage -
Combat Developments (Col. Tony Pokorney)
Ft. Monroe, Virginia 23651

10. Headquarters 1
U.S. Army Training & Doctrine Command
Attn: Director, Maneuver Directorate -
Combat Developments (Col. Fred Franks)
Ft. Monroe, Virginia 23651

11. Lt. General J. R. Thurman 1
Commanding General
U.S. Army Combined Arms Center
Ft. Leavenworth, Kansas 66027

12. Director 1
Combined Arms Combat Development Activity
Attn: Col. Reed
Ft. Leavenworth, Kansas 66027

13. Professor J. Hartman, Code 55HH 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93940

14. Dr. Wilbur Payne, Director 1
U.S. Army TRADOC Systems Analysis Activity
White Sands Missle Range, New Mexico 88002

15. Director 1
Armored Combat Vehicle Technology Program
Attn: Col. Fitzmorris
U.S. Army Armor Center
Ft. Knox, Kentucky 40121

16. Director 1
Studies Division - Combat Developments
Attn: Col. Burnett
U.S. Army Armor Center
Ft. Knox, Kentucky 40121

17. Col. Frank Day 1
TRADOC Systems Manager - XM1
U.S. Army Armor Center
Ft. Knox, Kentucky 40121

18. MG Dick Lawrence 1
Tank Forces Management Office
Room 1A-871, The Pentagon
Washington, D.C. 20310

19. Mr. David Hardison 1
Deputy Undersecretary of the Army
(Operations Research)
Department of the Army, The Pentagon
Washington, D.C. 20310

20. Director 1
U.S. Army Material Systems Analysis Activity
Attn: Mr. Will Brooks
Aberdeen Proving Grounds, Maryland 21005

21. Command and General Staff College 1
Attn: Education Advisor
Room 123, Bell Hall
Ft. Leavenworth, Kansas 66027

22. Headquarters, Department of the Army 1
Office of the Deputy Chief of Staff
for Operations and Plans
Attn: DAMO-2D
Washington, D.C. 20310