# NAVAL POSTGRADUATE SCHOOL
## Monterey, California

DTIC
S ELECTE
JUN 21 1990
D

# THESIS

**Short Range Air Defense
Defense Planner**

by

Roger S. Dixon

June 1990

Thesis Advisor: Yuh-jeng Lee

90 06 21 010

# REPORT DOCUMENTATION PAGE

| 1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
|---|---|
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | Approved for public release; distribution is unlimited |

| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
|---|---|
| | |

| 6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School | 6b. OFFICE SYMBOL (if applicable) 52 | 7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School |
|---|---|---|
| 6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943 | | 7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943 |

| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School | 8b. OFFICE SYMBOL (if applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER - |
|---|---|---|

| 8c. ADDRESS (City, State, and ZIP Code) Monterey CA 93943 | 10. SOURCE OF FUNDING NUMBERS | | | |
|---|---|---|---|---|
| | PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO. |
| | | | | |

**11. TITLE** (Include Security Classification)
Short Range Air Defense Defense Planner

**12. PERSONAL AUTHOR(S)**
Dixon, Roger Steven

| 13a. TYPE OF REPORT Master's Thesis | 13b. TIME COVERED FROM 07/89 TO 06/90 | 14. DATE OF REPORT (Year, Month, Day) June 1990 | 15. PAGE COUNT 73 |
|---|---|---|---|

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

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Graphics workstation; Air Defense Artillery; simulation, defense planning; A* search, SB-Prolog; low-altitude avenues of approach; |
| | | | |
| | | | |

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

The Short Range Air Defense (SHORAD) Defense Planner, or SDP, is a prototype defense planning and simulation tool. It is designed to aid and train Army Air Defense Officers in the tasks of (1) planning the air defense for a given static asset; and (2) positioning short range air defense weapon systems in the best possible way. A prototype system has been constructed which allows the Air Defender to position his asset to defend, with the system then performing a heuristic search, and displaying four possible attack routes that could be used by attacking aircraft to reach the map region containing the asset. The user may then position Towed-Vulcan, Stinger, and Chaparral weapon systems where he feels they will provdide the best defense of the asset, or request the system to position four Vulcans, four Stinger, or four of each Vulcan and Stinger in defense of the asset. The user can then choose any vehicle that he, or the system, positioned and be able to see a 3D representation of what the gunner in that vehicle would see. He is also able to maneuver that vehicle over the 3D terrain to select the best possible defensive position from the gunner's point of view. Both the Towed-Vulcan, and the Chaparral weapon systems, have been modeled in 3D for use in the SDP prototype system.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED |
|---|---|
| 22a. NAME OF RESPONSIBLE INDIVIDUAL Yuh-jeng Lee | 22b. TELEPHONE (Include Area Code) (408) 646-2361    22c. OFFICE SYMBOL 52Le |

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

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

# SHORT RANGE AIR DEFENSE
## ·DEFENSE PLANNER

by

Roger Steven Dixon
Captain, United States Army
B.S., University of Tennessee at Knoxville, 1981

Submitted in partial fulfillment of the requirements for
the degree of

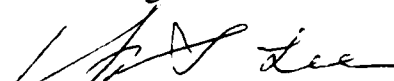## MASTER OF SCIENCE IN COMPUTER SCIENCE
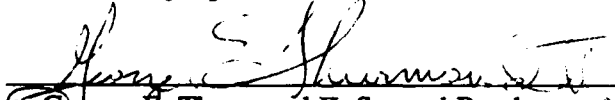
from the

## NAVAL POSTGRADUATE SCHOOL
June 1990

Author: _____
Roger Steven Dixon

Approved by: _____
Yuh-jeng Lee, Thesis Advisor

_____
George E. Thurmond II, Second Reader

_____
Robert B. McGhee, Chairman
Department of Computer Science

# ABSTRACT

The Short Range Air Defense (SHORAD) Defense Planner, or SDP, is a proto-type defense planning and simulation tool. It is designed to aid and train Army Air Defense Officers in the tasks of (1) planning the air defense for a given static asset, and (2) positioning short range air defense weapon systems in the best possible way. A prototype system has been constructed which allows the Air Defender to position his asset to defend, with the system then performing a heuristic search, and displaying four possible attack routes that could be used by attacking aircraft to reach the map region containing the asset. The user may then position Towed-Vulcan, Stinger, and Chaparral weapon systems where he feels they will provide the best defense of the asset, or request the system to position four Vulcans, four Stinger, or four of each Vulcan and Stinger,in defense of the asset. The user can then choose any vehicle that he, or the system, positioned and be able to see a 3D representation of what the gunner in that vehicle would see. He is also able to maneuver that vehicle over the 3D terrain to select the best possible defensive position from the gunner's point of view. Both the Towed-Vulcan, and the Chaparral weapon systems, have been modeled in 3D for use in the SDP protoype system.

iii

# TABLE OF CONTENTS

# I. INTRODUCTION

## A. AN INCREASING NEED FOR COMPUTER SIMULATION

As the rapidly changing world situation causes a scramble for the so-called "peace dividend", the United States military faces deep cuts in the defense budget. Projected cutbacks in large numbers of personnel, and lost funding for large training exercises, especially in the Army, create a need for good, inexpensive, three dimensional computer simulations. These simulators need to be able to realisticaly model the terrain, the weapon systems, and a combat environment, if they are ever to be capable of providing our leaders and soldiers with the same level of training value as a live exercise. Models need to be created for all weapon systems, and these models must be able to interact in order to simulate a true combined arms environment. Additionally, as the number of personnel is reduced, leaders should also have computer knowledge based expert systems to aid them in tasks such as terrain evaluation and defense planning.

## B. AIR DEFENSE SIMULATION

At the Naval Postgraduate School (NPS) in Monterey, California, a volume of work has been done over the past couple of years on creating just such simulations, using a number of land, sea, and aerial based weapon systems. Many of these simulators are networked together to allow weapon-on-weapon encounters. A set of

United States Army weapon systems, that up to this time had not been modeled in a NPS simulator, are those classed as Short Range Air Defense (SHORAD) systems. These weapons are the ones modeled in the SHORAD Defense Planner (SDP).

A number of the NPS simulators have incorporated knowledge based and/or expert systems, usually involving path planning, into the simulation. However, this type of work has always been done on a separate workstation from the main graphics portion of the simulator, and communication between the two workstations is via a local area network. The SDP prototype incorporates rule based systems, for both path and defense planning, on the same workstation (a Silicon Graphics, Inc. IRIS 4D/70GT) as the graphics interface, utilizing SB-Prolog (Debray&, 1989) modules.

## C. THESIS ORGANIZATION

The work done on this thesis can be broken down into three major areas: the graphical user interface, calculation of low-altitude avenues of approach, and SHORAD defense planning. Chapter II is an overview of the work done on simulations at NPS that led up to, or inspired, this study. Chapter III describes the graphical user interface, and how it was derived from an existing base model simulation. Chapter IV is a discussion of air defense weapon system modeling. Chapter V deals with the calculation of low-altitude avenues of approach. Chapter VI covers the issues of SHORAD defense planning. Chapter VII is the author's views regarding the limitations of this study, and possible areas for future research.

## II. HISTORICAL DEVELOPMENT

### A. INTRODUCTION

A great amount of work has been done at the United States Naval Postgraduate School (NPS) in the design and development of good, relatively inexpensive, moving platform simulators. These simulators have evolved, with each being an extension or specialization of those developed previously. The SHORAD Defense Planner (SDP) is a specialized prototype simulation/planning tool which uses the Moving Platform Simulator II (Strong&, 1989) as its base model. We present below those simulators that led to the development of MPS II, and some other simulators which add inspiration to the development of the SDP project.

### B. FIBER-OPTICALLY GUIDED MISSILE (FOGM) SIMULATOR

The FOGM simulator (Smith&, 1987) was developed on a Silicon Graphics, Inc. IRIS 3120 graphics workstation. The simulator presents a moving three-dimensional image as seen from the missile's perspective flying over a fixed 10 x 10 kilometer area of Fort Hunter-Liggett, California. The missile can target, track, and destroy any visible ground vehicles. Control of these vehicles is limited to the assigning of initial headings and speed. (NPS52-89-004, 1988, p. 4)

3

## C. VEHICLE SIMULATOR (VEH)

The VEH simulator (Oliver&, 1987) was an extension of the FOGM simulator, which was also developed on the SGI IRIS 3120. Improvements included allowing the user to drive and maneuver vehicles on the ground in real time, and increasing performance by only drawing the terrain in the field-of-view using a scanline Painter's algorithm. (NPS52-89-004, 1988, p. 5)

## D. FOGM/VEH NETWORKING SIMULATOR (FOGM/VEH NET)

FOGM/VEH NET modified the FOGM and VEH simulators, allowing them to communicate over an Ethernet local area network. Networking allowed the effects of driving a vehicle or flying a missile on one workstation to be seen by another user who was controlling another vehicle/missile on a second workstation. (NPS52-89-004, 1988, p. 5)

## E. VEH II VEHICLE SIMULATOR

The VEH II simulator, finished in June 1988, was an effort to enhance, and port, the VEH simulator from the IRIS 3120 to the IRIS 4D/70GT. The port allowed th: simulator to run under the MEX (SGI, 1987, p. W1) window management system, and the 4Sight (SGI, 1988) window management system. Enhancements included popup menus for user selections, the ability to add vehicles at any time, and the ability to save and reload vehicles to and from separate files. (NPS52-89-004, 1988, p. 5)

4

## F. MOVING PLATFORM SIMULATOR (MPS)

MPS (Fichten&, 1988) is a combination of the FOGM and VEH II simulators implemented on the SGI IRIS 4D/70GT graphics workstation. Improvements included (1) allowing the user to be able to select the 10 x 10 kilometer area of operation from the 35 x 35 kilometer terrain database, (2) a more efficient terrain drawing algorithm including distance attenuation, (3) the use of Z-buffering for hidden surface elimination, and (4) more user control over vehicle platforms. Broadcast networking allows the simulator to be run on numerous workstations simultaneously.

## G. MOVING PLATFORM SIMULATOR II (MPS II)

MSP II, completed in June 1989, was an extension of the MPS program that attempted to better meet the needs of the United States Combat Developments Experimentation Center (CDEC), Fort Ord, California. One enhancement to the MPS system included the capability to display the terrain at a user selected resolution in both two and three dimensional modes. Other enhancements included the locations of vehicles being converted and displayed in the Universal Transverse Mercator (UTM) coordinate system, a real-time simulator clock, more realistic displaying of terrain, and the ability to calculate and display line-of-sight (LOS) information. (Strong&, 1989, pp. 9-11)

5

## H. FORWARD OBSERVER SIMULATION TRAINER (FOST)

The FOST (Drummond&, 1989) system is a specialized prototype simulation/training tool for use by field artillery forward observers. It was developed on the SGI IRIS 4D/70GT graphics workstation, using the MPS program as a base model. This system showed that specialized training tools for specific weapon systems can be developed at a relatively low cost, while maintaining the "look and feel" of the base model (MPS) through the reuse of modularized code between the systems.

## I. AUTONOMOUS PLATFORM SIMULATOR (APS)

APS (Shannon&, 1989) combined the vehicle simulation/graphical capabilites of the MPS system with the concepts of optimal path planning and autonomous vehicle operations. APS was implemented utilizing a combination of the SGI IRIS 4D/70GT graphics workstation and the Symbolics 3600 workstation.

# III. THE GRAPHICAL USER INTERFACE

## A. INTRODUCTION

The concept we wished to achieve in the development of the graphical user interface on the SHORAD Defense Planner prototype was to maintain the "look and feel" of the MPS II simulator. We desired that any user of MPS II could run the SDP program, and with taking only a couple of minutes to learn the new menus, be able to effectively utilize the simulation. We also desired that any new modules or features created in SDP could be easily incorporated back into the MPS II simulator, thus extending the current system.

To start the SDP simulation, the user needs to go to the directory containing the SDP program files and type *sdp* at the command line. After adjusting the window to the desired size, the user will be presented with title screens as the terrain database is loaded. Messages and instructions to the user will appear on the right side of the screen. SDP is a menu driven system, and all selections are made by pointing to the option desired and clicking the right mouse button. An image will appear on the lower right of the screen if other control options are available at that point in the simulation.

## B. TERRAIN DATABASE

A number of the NPS simulators, including the SDP prototype, utilize a digital terrain database file provided to the school by the United States Army Combat Developments Experimentation Center (CDEC) at Fort Ord, California. This file is a special Defense Mapping Agency (DMA) digital terrain database (DTED), containing data for a 36 x 35 kilometer area of Fort Hunter-Ligget, California. It is a special file in that the data points it contains are spaced every 12.5 meters instead of the standard Level 1 DTED file which only has data points every 100 meters. Each Data point consists of 16 bits. The 3 most significant bits are vegetation data, and are not used by any NPS simulator to date. The remaining 13 bits, after conversion to decimal, represent the elevation in feet for that point. (NPS52-89-004, 1988, p. 9)

## C. MODIFICATION FROM MPS II

The MPS II program was written in the C programming language, using numerous small files, on the IRIS 4D/70GT graphics workstation. This design makes it a easy system to expand and modify. The first step in the development of the graphical user interface (GUI) for SDP was to remove or disable those features of MPS II that were not needed to implement the prototype. These features included the files for networking, the line-of-sight simulator modules, and the control structure to handle flying platforms. Also removed were the files for those unused vehicles including the tank, truck, wreck, observer, and fog-m missile. The files for the two

8

kinds of jeeps were retained for use in the SDP prototype. The open jeep models a Stinger weapon system as described in Chapter IV, and the covered jeep models a "command" or extra vehicle.

The first functional modification made to the MPS II code is in the presentation of the 35 x 35 kilometer map. In MPS II, the user is allowed a menu selection to specify a 10 x 10 kilometer area to operate in. However, if the user does not make that menu selection, a default 10 x 10 kilometer area is selected and presented anyway. We feel that the user should, especially in a defense planning application, explicitly select the area of operation. In the SDP prototype, this is provided for in that each time the 35 x 35 kilometer map is presented, the user must select the 10 x 10 kilometer area prior to seeing any other menus. Additional code had to be added to insure that both the 35 x 35 and 10 x 10 kilometer maps were correctly redrawn as the user switches between them, and changes areas of operation. Once the area of operation is selected, the user is presented the menu selections. All the selections presented to the user at this point will behave as their counterparts did in MPS II.

A second modification was done once the user is presented the 10 x 10 kilometer map, and has selected to add a platform from the menu. Here, one of the choices is to add a defended asset. If the user is doing a defense planning application, the asset to be defended should be placed first, but is not required in SDP. Once the user has made this menu selection, and placed the defended asset on the 10 x 10 kilometer map, the system will automatically make a blocking call to the Prolog module to calculate the low-altitude avenues of approach to the asset (see Chapter V). A "please

9

wait" message is displayed to the user while the avenues are calculated. The user may move the cursor at this point, but will not be presented any further menus until this task is done.

Once the defended asset is placed, and the avenues of approach are calculated, the user is returned to the menu normally presented with the 10 x 10 kilometer map. Now however, the user has an additional choice of activating the system defense planner (see Chapter VI). This option is only presented after an asset has been placed, and with this version of the SDP prototype, only one defended asset may be in the database at any one time. Selecting to activate the system defense planner will also cause a blocking call to the Prolog module designed to complete this task, and the user will be given a message to wait until those placement operations are done, and the locations of all the weapon systems are drawn.

The majority of the remaining modifications done in making the SDP prototype were those to correctly position, draw, and control the air defense weapon systems. All other menu selections function as did their counterparts in MPS II. The controls, both mouse and dial, for operating the ground weapon systems are the same as in the user interface of MPS II (Strong&, 1989, Appendix A).

In creating the SDP prototype, a total of 72 C language files were deleted and/or modified, with only 8 new files created to display the new weapon system models, and to do coordinate conversions to and from the Prolog modules. All additional programming for the prototype system was done using SB-Prolog. Over one half of the 72 C files that were deleted or modified in creating SDP were to remove those

features of MPS II not needed for our application. The author sees very few problems being caused by adding the features and new weapon systems developed in the SDP prototype directly to the MPS II simulator as an extension to that system.

# IV. MODELING OF WEAPON SYSTEMS

## A. INTRODUCTION

With any simulation tool, it is necessary to provide the user with a realistic image of the environment being simulated. This was done fairly well in MPS II with the modeling of terrain, time, and weapon systems. The user is allowed to select vehicles to operate, and then is presented an image from the perspective of the operator of that vehicle, seeing what they would see. This image includes a 3D view of the surrounding terrain, and any other weapon system that would be visible to the operator. This concept was extended in SDP, with the modeling of three additional short-range air defense weapon systems.

## B. M167 TOWED VULCAN GUN

The Vulcan towed air defense weapon system consists of a 6-barrel, 20-mm cannon and fire control system mounted on a 2 or 4-wheel trailer carriage. The system is capable of being towed at high speeds over improved roads, travel over rough terrain, and fording streams at a depth of 30 inches. The gun system was designed for deployment in the forward combat area to provide air defense against low-altitude air threat. It can be used against stationary or moving targets such as personnel, trucks, and light armored vehicles. The system is air portable by cargo aircraft and

helicopter, and can be air dropped. The Vulcan uses a linked feed system with a maximum rate of fire of 3000 rounds per minute. It has a combat loaded weight of 3150 pounds, with an overall width of 79 inches, length of 152 inches, and height in the Travel mode of 81 inches. The maximum planning range for aerial targets is 1200 meters, and ground targets is 4500 meters.(FM44-1, 1983, pp. B-2/3)

In the SDP system, the Vulcan is depicted as a 3-dimensional, scaled image consisting of 153 separate polygons. When selected as the vehicle to operate, the user will be presented with a 360 degree, 3D image as seen from the gunner's seat of the weapon system. The SDP prototype does not model the M561, 1 1/4-ton truck, which is normally used as the prime mover for the towed Vulcan gun.

## C. STINGER MISSILE SYSTEM

Stinger is a short-range, man-portable, shoulder-fired, infrared homing (heat-seeking) air defense guided missile system, firing a supersonic, surface-to-air missile. It is designed to counter high-speed, low-level, ground-attack aircraft. It also is lethal against helicopter, observation, and transport aircraft. Stinger is deployed to provide air defense to high-priority assets throughout the division/separate brigade/regiment areas of operations. The missile is sealed within the launcher, and is not removed except by firing. The launch tube is discarded after the missile is fired. The weapon weighs 34.9 pounds, and is 60 inches long. The weapon system has a range in excess of 4000 meters.(FM44-1, 1983, p. B-6) In the SDP prototype, a Stinger team is portrayed as an open jeep, identical to that used in MPS II.

## D. M48 CHAPARRAL MISSILE SYSTEM

The M48 system is composed of three major elements: launching station, carrier, and Chaparral missiles. It is a highly mobile susrface-to-air missile system designed to counter the high-speed, low-altitude air threat to organizations and critical assets in the forward areas. The launching station is a complete, self-contained weapon system, and may be separated from the carrier and operated in a ground-emplaced mode. The launching station may be sling-lifted by helicopter when separated from the carrier. The M48 system weighs 27508 pounds, and is 238 1/2 inches long, 105 3/4 inches wide, and 76 inches high. The system uses supersonic, surface-to-air, aerial intercept missiles. Each missile weighs 190 pounds, and is 9 1/2 feet long, 5 inches in diameter, having a wing span of 24.8 inches. Each Chaparral weapon system carries 12 missiles, with four on the launching station rails and eight stored. The Chaparral missile has a range of approzimately 5000 meters.(FM44-1, 1983, p.B-9)

In the SDP prototype, the Chaparral is depicted as a 3-dimensional, scaled image consisting of 206 separate polygons. When selected as the vehicle to operate, the user will be presented with a 3D image as would be seen through the tinted dome of the Chaparral gunner's turret. In the SDP system, the weapon system turrets do not at this time, rotate separately from the body of their carriers, but instead the user sees a view as if he was turning his head. It is therefore necessary to turn the whole vehicle to view a complete 360 degree area around the Chaparral.

## E. ICONS

As part of the user interface for the SDP system, 2-dimensional icons were developed for each weapon system. Each consists of a 16 x 16 bit image of the weapon. They are used to display the location of that type of weapon system on the 10 x 10 KM map, and used to select that weapon system for operation. In addition, each weapon system has associated with its icon a maximum possible engagement circle which is displayed with the icon on both the 35 x 35 KM and the 10 x 10 KM maps. This is done to allow the Air Defense planner to quickly identify any possible gaps in defensive coverage once all weapon systems are placed.

# V. CALCULATING LOW-ALTITUDE AVENUES OF APPROACH

## A. INTRODUCTION

In any defensive operation, one of the key considerations for the defending commander is the analysis of the terrain. In particular, he must be aware of the possible avenues of approach, into the sector he is defending, which could be exploited by enemy forces. A good avenue of approach must support rapid movement along its length. A good air approach, which the Air Defense commander must consider, "provides rapid access to the target area, together with terrain-masking from air defense radar and direct-fire air defense weapons" (FM100-5, 1986, p. 80). This normally involves the use of valleys, and other terrain features.

The Air Defense commander can use elevation and other map related data to conduct the terrain analysis in determining the low-altitude air avenues of approach into his sector. On a computer, this problem is ideal for some form of optimizing search operation. When analyzing the terrain for avenues of approach, one must think of the characteristics of the enemy that they will be facing. For an Air Defense application, the enemy consists of helicopters, and high speed attack aircraft. The key terrain feature of concern for this type of application is elevation. In either case, be it helicopters or high speed aircraft, it can be argued that because of the speed in which the terrain is navigated, it is unnecessary to think of terrain in 12.5 meter, or even in 100 meter intervals, but in most instances terrain elevation can be generalized

at 1 KM intervals for the purpose of determining avenues of approach. This is because the Air Defender does not need to know the exact optimum path across the terrain, but rather he can plan just as good a defense by knowing a more "generalized" direction of attack, or satisfying path across the terrain. After all, the enemy can not be counted on to always take what seems to be the "optimal" route to the location that is being defended.

## B. TERRAIN REPRESENTATION

In the SHORAD Defense Planner, once the user selects the location where he wishs to defend, the system will activate a compiled Prolog program which will calculate four of the possible low-altitude, air avenues of approach. One avenue of approach will be calculated from each edge of the 35 x 35 KM map into the sector that contains the defended asset. To determine the avenues of approach, a separate database was created to represent a generalization of the terrain elevation. Nine levels of elevation (0-8) were used, corresponding to the nine color or elevation levels depicted in the legend of the 35 x 35 KM map in the simulator. Each square kilometer of terrain is represented by the highest elevation within that square kilometer.

When looking at the threat, a high speed attack aircraft could be traveling at speeds between 500-600 knots while flying to and from a target. One knot is approximately one half meter per second, so one kilometer could be crossed in 3-4 seconds. In preparing an air defense plan, a general direction of possible attack is sufficient for planning purposes. Therefore, in finding the possible low-altitude avenues of

approach, the 35 x 35 kilometers were broken down into regions of 5 x 5 kilometers. This creates a new map consisting of 7 x 7 regions of terrain. Each region was given an elevation value equal to the sum of all the one square kilometer elevation values that comprised that region.

There were four regions of the map, however, where five or more of the square kilometers comprising that region were of the highest elevation level (8). With the speeds being traveled, a pilot would have to fly above the highest elevation in order to safely cross any of those four regions. This would expose the aircraft to being seen by all regions of the map for 15-20 seconds, which is sufficient time for air defense weapon systems to acquire and engage that aircraft. These four regions were therefore considered "undesirable" as far as a low-altitude avenue of approach, and an additional amount was added to the elevation value of these regions, thus increasing the cost of crossing them  A region and its elevation are represented in Prolog as a simple X,Y coordinate and an elevation value (i.e. elev([6,5],86)).

## C. SEARCH METHOD

To determine each avenue of approach, a modified version of a published A* search algorithm was used (Rowe, 1988, pp. 244-247). The modified algorithm does an optimum search from the asset being defended to the general goal of an edge of the map. This is accomplished by specifying the goal as only one coordinate, either x or y depending on the edge of the map desired as a goal, and the other becomes immaterial, since it is being unified with any value.

18

One possible method to find the least cost paths to the map edges is to add to the map two borders which are unseen by the user, but are used in the search process. The first border is one of extremely low cost values, which once encountered by the search algorithm, will insure that this border will be followed until the edge desired is reached. The second border contains high cost values, and acts as a barrier to insure the search does not attempt to access points outside the known database. This method will return the best low-altitude avenues of approach, but will not necessarily return one avenue to each edge of the visible map. This could occur any time the search leaves the visible map on a non-goal edge, and then travels a path along the invisible low-cost border to a point on the goal edge. This approach also increases the size of the database being searched, which in itself can decrement system performance as discussed below.

A second approach to implementing this type of search is to add a restriction to the algorithm where no successors which would be outside the visible map database are allowed to be generated. A successor in this type of application is one of eight possible neighboring regions to the region in which the search algorithm is currently reasoning about. This method forces the search to remain within the visible map boundries, and one optimal path is generated to each edge of the map. This also ensures that the database being searched is limited to the information representing the actual physical map. A drawback to this method is that if there is no real good avenue of approach to a given edge, the best possible path to that edge will be returned anywa. even though realistically the enemy would probably not come from that

direction. In the first method of implementation, a second path to another edge would be found if no good path existed in one direction (least resistance to the low cost border).

To insure that an avenue of approach is displayed from each edge of the visible map, and to maintain the smallest possible terrain database, the second method of finding paths to the edges was chosen for the SDP prototype. Four separate calls are made to the modified A* search algorithm, each specifying a different edge of the map as the goal. The heuristic function used is the estimated cost of moving to the region under consideration, plus the straight line distance to the edge of the map given as the goal. A separate distance function had to be written to get the straight line distance for each of the four edges. The cost function used for going from the current region to a new region is the elevation value of the new region under consideration, plus the distance from the current region to that considered region. Functions also had to be added, that convert each coordinate of the path from the 7 x 7 grid coordinate system, to an UTM coordinate which is utilized by the graphical interface of the SDP prototype. Information and code required to run SB-Prolog, and the Prolog code to find the avenues of approach, can be found in appendices A and B respectively.

## D. EFFECTS OF INCREASED DETAIL

We believe that the use of the generalized 7 x 7 grid regions in the search process provides satisfactory avenues of approach for planning purposes, especially if

the primary threat is high speed attack aircraft. If the primary threat is from helicopters, or if more detailed avenues are desired, then the user may wish to increase the detail of the map by doing the search process on the 35 x 35 square kilometer database, or on a database with even more detail. If this is done, however, the user must be prepared for the increase in time needed to complete the search on this larger database. Tests were conducted using a VAX 11/785, an ISI workstation, a Symbolics workstation, and the Silicon Graphics Iris 4D/70GT workstation, all running Prolog in the compiled and interpreted modes. On each system it was found that the time required to get four avenues of approach increased proportionally, relative to the size of database. As an example, on the Iris workstation, calculating four avenues of approach using the 7 x 7 grid (49 data points) takes approximately 2-5 minutes. When using a 30 x 30 grid (900 data points), four paths were returned anywhere from 30 to 100 minutes.

In addition, because we used Prolog with all the elevation data stored as facts, as we increased the size of the database above a 30 x 30 grid, we also experienced an increase in the occurance of system failure because of stack or heap overflows. This problem occured on all systems tested, and the increasing of the stack and/or heap size did not seem to significantly decrease the number of system failures. A partial solution to this problem was found in breaking the elevation database into smaller files, and read in only those portions of the database that were needed at any given time. On some systems, however, including SB-Prolog, this causes any facts already present with the same name to be overwritten.

# VI. AIR DEFENSE PLANNING

## A. AIR DEFENSE PRIORITIES

It is often the case that the Air Defender will find that he has been assigned more assets to defend than he has air defense weapon systems to defend them with. When this situation arises, the air defense artillery commander must sit down with the supported ground unit commander and work out a priority list of the assets to be defended. To establish this priority list, each asset in question must be evaluated in terms of it's "criticality, vulnerability, recuperability, and threat" (FM44-1, 1983, p. 4-7).

### 1. Criticality

To determine how critical the asset is, it must be evaluated on how essential it is to the overall success of the mission. It must be determined whether the loss of, or damage to, the asset will prevent the mission from succeeding; cause serious interference, now or later, to the mission execution; or only cause limited interference with mission execution. Those assets which are absolutely essential to the success of the mission are given the highest priority in terms of criticality.

### 2. Vulnerability

How vulnerable an asset is depends on its ability to survive on the battlefield. The vulnerability of a specific asset to air attack is determined by the asset's "hardness, its specific mission in the overall operation, the degree to which

22

the asset can disperse or displace to another position, the degree to which it can provide its own air defense, and the amount of protection afforded by passive air defense measures" (FM44-1, 1983, p. 4-8). Also, the size and maneuverability of the asset should be considered. Large, slow or nonmoving assets are easier to hit targets than small, highly mobile ones. Those assets which are less vulnerable in the above context should be a lower priority as far as allocating air defense weapons.

### 3. Recuperability

"Recuperability is the degree to which the asset can recover from inflicted damage in terms of time, equipment, and available manpower to again perform its mission" (FM44-1, 1983, p. 4-8).

### 4. Threat

To evaluate the air threat to any given asset, one must consider "targeting information provided by intelligence estimates, past enemy attack methods, and enemy doctrine" (FM44-1, 1983, p. 4-8). Also, the number and type of aircraft, both helicopter and fixed-wing, that the enemy has available to send into each asset's area of the battlefield, should be considered before deciding which assets require active air defense protection.

## B. AIR DEFENSE EMPLOYMENT PRINCIPLES

Once a priority list of assets has been determined, it is up to the air defense commander to allocate his limited air defense weapons to defend those assets, starting with the one with the highest priority. The air defense commander needs to follow,

23

as much as possible, the four air defense employment principles of "mass, mix, mobility, and integration" (FM44-3, 1984, p. 6-3). The air defense commander may be able to only accomplish one of the principles, but should try to meet all four.

## 1. Mass

The principle of mass deals with the assigning of a sufficient number of air defense weapon systems to effectively defend the asset. For SHORAD weapon systems, mass normally can not be achieved with less than a platoon sized unit, or a Stinger section. It may not be possible to achieve the principle of mass, as often only one platoon, or a section will be assigned to defend a battalion-size maneuver unit, which itself has a number of assets it's commander wants defended, or a large critical stationary asset.

## 2. Mix

The principle of mix works in conjunction with the principle of mass. Here, the Air Defender tries not only to achieve the principle of mass, but to do so using a mixture of complementary weapon systems (i.e. for SHORAD, a gun/missile mix). The concept is that an enemy may be able to circumvent one type of weapon system, but would be less likely to defeat multiple types, each with different characteristics and capabilities.

## 3. Mobility

The principle of mobility deals with the ability to quickly adapt to changing situations on the modern battlefield. The air defense units must be able to defend a moving convoy or maneuver unit as effectively as static, point assets. "SHORAD

units tasked with providing air defense to maneuver units should possess mobility at least equal to that of the supported unit" (FM44-3, 1984, p. 6-4). This lack of mobility equal to that of the Army's newer armor and mechanized infantry units, was one of the considerations that led to development and testing of the failed Sergeant York gun, and currently the Air Defense Anti-Tank System (ADATS) project.

### 4. Integration

The final air defense employment principle is integration. "Integration is the close coordination of effort and unity of action that maximizes individual air defense system operational effectiveness while minimizing mutual interference among operating forces" (FM44-3, 1984, p. 6-4). This coordination effort needs not only to be between air defense units, but also with the supported unit. It is critical that the air defense units get included in the overall plans of the supported unit, especially in terms of maneuver, intelligence, ground defense, nuclear biological and chemical (NBC) information, and logistical support.

## C. SHORAD EMPLOYMENT GUIDELINES

While the above employment principles apply to all air defense weapon systems, there are additional guidelines for the employment of SHORAD weapon systems. When planning his defense, the SHORAD commander should consider each of these guidelines, but must remember that they are just guidelines, and may not all be feasible in his current situation. In considering the guidelines, the size and shape of the asset, the number and type of weapon systems available, the terrain in the current

area of operation, and other tactical considerations all may limit the ability to follow those guidelines. The SHORAD employment guidelines cover the concepts of balanced fires, weighted coverage, early engagement, defense in depth, mutual support, and overlapping fires.

## 1. Balanced Fires

The concept of balanced fires states that "fire units are positioned to provide approximately equal firepower in all directions" (FM44-3, 1984, p. 6-4). This follows the idea that the modern battlefield is a 360 degree environment, and except for those cases where there are distinct channelling avenues of approach, air attack could come from any direction. Terrain can often prevent the employment of weapons systems to achieve perfect balanced fires. However, always using a symmetrical defense could aid the enemy in locating the defended asset, and other air defense weapon systems.

## 2. Weighted Coverage

With weighted coverage, weapon systems are positioned to cover the most likely air avenues of approach. It is employed when "enemy attack routes are known or when insufficient assets are available to provide balanced fires" (FM44-3, 1984, p. 6-5). Low-level air approaches into the defended area, such as valleys and rivers, should be defended by weapon systems under this concept.

## 3. Early Engagement

Defending weapon systems should be positioned away from the defended asset, in the direction of possible attack, in order to engage enemy aircraft

prior to their ordnance release. This guideline becomes harder to comply with as air-to-ground weapon systems become able to attack targets at greater ranges. "Early engagement is the most important SHORAD deployment guideline" (FM44-3, 1984, p. 6-6).

### 4. Defense In Depth

Defense in depth implies that the enemy aircraft should encounter an increasing volume of fire as it approaches the defended asset. Defense in depth is maximized by applying integration and coordination between all air defense weapon systems in the defense. (FM44-3, 1984, p. 6-6)

### 5. Mutual Support

Two weapon systems are in mutual support of each other if they can engage into the other weapon systems dead zone. Most air defense weapon systems normally have a dead zone in the area immediately surrounding the weapon system. If one weapon system comes under air attack, adjacent fire units can engage the aircraft.

### 6. Overlapping Fires

When two weapon systems are positioned so that their engagement zones overlap, they are considered to have overlapping fires. The defense planner uses overlapping fires when limitations exist that prevent mutual support between weapon systems. If two weapon systems are in mutual support, they automaticaly have overlapping fires. However the reverse is not always true.

## D. SDP DEFENSE PLANNING

The SHORAD Defense Planner has a very simplified defense planning/weapons positioning implementation. The user may only designate one asset at a time to be defended in the 35 x 35 KM area of operation. The decision about what is the highest air defense priority is therefore made off-line by the user. The user also makes the decision regarding the principle of mix by selecting from a menu, the type of weapon systems to be positioned by the SDP system. The current protoype allows the choice of positioning a platoon of 4 Vulcan guns, a section of 4 Stinger teams, or a mixed defense consisting of a combination of 4 Vulcans and 4 Stinger. The principle of mass is ensured in that the asset to be defended is a static point asset, and the weapon systems are only positioned by the system's defense planner as platoons or sections. The principles of mobility and integration are not simulated in the current prototype.

Once the user has selected an asset to defend, and the four low-altitude avenues of approach are calculated and drawn, the user may select the menu option to allow the system to position weapon systems. The user selects the type of defense desired as described above, and then the system makes a blocking call to the Prolog defense planning module. The defense planning module takes as input the asset location in UTM coordinates, the type of defense desired, and the four avenues of approach to the region of the defended asset. The goal of the planning process is to select satis-factory positions for the weapon systems, while conforming to as many of the SHORAD employment guidelines as possible.

28

## E. DEFENSE PLANNER IMPLEMENTATION

The first thing accomplished by the system is to get the final approach direction of each of the four avenues of approach. This is done by calculating the changes in the X and Y coordinates between the region containing the asset, and the region on each avenue of approach just prior to the asset's region. This result is converted into a direction vector. Direction in the SDP system, which is used for vehicle platform headings and air defense system primary target lines (PTL's), is represented by eight values, in degrees, each being 45 degrees apart. It is these direction angles, pointing away from the asset and toward the incoming low-altitude avenues of approach, that are used by the rest of the planning module.

The SDP prototype positions one of each type of weapon system based on each of the four low-altitude avenues of approach. If it is desired to use a different number of weapon systems to represent a platoon/section (i.e. three Vulcans per platoon as in Light Divisions, or more Stinger teams per section), then a method to rank the avenues of approach should be developed. One such method would be to sum the elevation values for each region on the avenue of approach. The path with the lowest value then would be the most likely avenue of approach. Then, weapon systems could be positioned starting with the most likely avenue, and going in order until all available weapon systems are positioned.

In positioning each weapon, we wanted to maintain line-of-sight to the defended asset. To do this, the 35 x 35 kilometer elevation database was used, with checks being made to insure that the elevation of all square kilometers between the weapon

29

system and the defended asset were either equal to, or less than, the elevation of the square containing the asset. Since positioning of any one weapon system does not require the entire 35 x 35 elevation map, the database was broken into pieces containing overlapping 10 x 35 kilometers. Each piece is then loaded only when needed. The asset's location, and the location of the next point on the avenue of approach being worked on, determine which map section is called into the process. A position (1000 meters from the asset for Vulcan, and 2500 meters from the asset for Stinger) is checked, in the direction of the incoming avenue of approach, to see if it meets the line-of-sight requirement. If it does, the weapon system is placed at that position. If the position does not meet the line-of-sight requirement, then two more positions (+/- 45 degrees) are checked at the same distance from the asset. If none of these three positions meet the requirement, then the weapon system is placed at a default distance from the asset (250 meters for Vulcan, and 1500 meters for Stinger). The primary target line (PTL) given the weapon system is toward the incoming low-altitude avenue of approach currently being worked with.

As the system positions each new weapon, it checks to see if there already is a weapon system at the location desired. This could occur when two or more avenues of approach merge prior to nearing the defended asset. If another weapon system is already at the desired location, both the new weapon system, and the old weapon system at that location will be offset (200 meters for Vulcan, and 500 meters for Stinger) in both the X and Y coordinate directions. This creates a weighted coverage, with mutual support, along a now very high probability avenue of approach.

Once all the positions are determined for the number of weapon systems desired, the locations in UTM coordinates, and the primary target line for each fire unit is returned to the graphical interface for display. The user then can select and operate each of these weapons, further refining positioning based on the 3D view seen from the weapon system. The Prolog code for the defense planning module can be found in appendix C.

## F. RESULTS AND LIMITATIONS

When evaluating the results of the positioning of weapons by the SHORAD Defense Planner, it conforms well to the SHORAD employment guidelines. It is designed using projected low-altitude avenues of approach, and thus usually returns weighted coverage. However, balanced fires are possible since four avenues of approach, one to each edge of the map, are calculated. Since all weapons are positioned within range of the asset, overlapping fires is guaranteed, and mutual support between two or more weapon systems often occurs. Both defense in depth and early engagement can be accomplished if the user selects a mixed defense, and the elevation around the asset meets the line-of-sight requirement.

The major limitation in the current prototype is the elevation accuracy used in determining the line-of-sight calculation and positioning of the weapon system. The elevation value used represents the highest point in each square kilometer considered. The elevation of the exact point at which the weapon system is placed is not calculated, and thus the weapon system could be located in a depression, and itself not

have line-of-sight to the asset. This is why the user sould select each vehicle placed by the system, and check the 3D field of view, moving the vehicle if necessary.

The results, obtained using the simple weapons positioning module implemented in the SDP prototype, show that automated defense planning is possible. The more accurate and detailed the terrain data and weapon characteristics are modeled, the more useful the result for actual defense planning. However, as explained regarding the calculation of low-altitude avenues of approach, increased accuracy normally equates to increased computation time for a result. In a battlefield environment, the desire for greater accuracy versus decreasing time to act, must constantly be weighed.

# VII. CONCLUSIONS AND FUTURE WORK

## A. CONCLUSIONS

This work presents the specialization of previous moving platform simulation efforts done at the Naval Postgraduate School, for the purpose of providing a prototype defense planning and simulation tool to the U.S. Army Short Range Air Defense Artillery community. It provides three dimensional models of both the Chaparral missile system, and the Vulcan gun system. These may be incorporated into current simulators such as MPS II, or be used in future simulation projects. It also provides modules to calculate low-altitude avenues of approach into the area of operations, and to do defense positioning of air defense weapons systems. The principles used in these modules can easily be modified to support any type of weapon system.

The SDP prototype is the first system at the Naval Postgraduate School to incorporate rule based modules with the graphical interface, all of which are running on the same IRIS workstation. The modules were written in the public domain SB-Prolog, ported to the IRIS for this study. The porting of SB-Prolog, and the purchase of Common Lisp for the IRIS workstations in Spring quarter 1990, will allow future students at the Naval Postgraduate School to easily develop AI/graphic applications on the IRIS. We believe this will lead to improved, more realistic, moving platform and training simulations.

## B. SYSTEM LIMITATIONS

As a derivative of MPS II, the SDP prototype maintains the same performance limitations of that system in terms of the large amount of memory needed, and the reduction of system responsiveness as resolution is increased (Strong&, 1989, pp. 121-124). The introduction of the Prolog modules, with the overhead associated in the execution of Prolog, has also added to the memory and performance problems. As noted in Chapters V and VI, if the number of terrain data points considered were to be increased in the Prolog modules to a more realistic level for defense planning purposed, then the current prototype's performance would be degraded to an unexceptable level. Additionally, a bug in the current version of SB-Prolog forces a limit to the size, and makeup, of files being compiled to byte-code form. This causes added problems as the number of data points, and/or the size of the map increases.

## C. FUTURE WORK

We envisage the work on the SDP prototype evolving in two possible directions. The first is to incorporate the features of SDP back into MPS II, thus extending that system. The second is to extend SDP itself, improving the simulator so it may be a more effective planning and simulation tool. Regardless of which way the system evolves, there are a number of added features which should be explored.

One improvement needed on all the current simulations is to increase the realism of the turreted weapon system platforms. Control mechanisms need to be added

to allow the turrets to rotate independently of the rest of the weapon system, thus giving the appearance of tracking and engagement. Some work has been done in this area for the Chaparral and Vulcan systems. A separate class project titled ADASYS modeled the two weapon systems such that each translated as a unit, but the base and turret could rotate independently of each other. The Chaparral system was also capable of firing a missile. Time constraints prevented these features from being added to this version of the SDP prototype.

Any future version of SDP or MPS II should involve the modularization of the code. Though the small size of each file used in MPS II helped a little in modifying the code, the flow of control between the files has become tangled. Simple modifications which should have only involved one file, ended up involving a number of files. Software engineering techniques need to be used in a comprehensive design project for any future simulators derived from these two.

The next evolution of simulators should incorporate the presentation of cultural features such as vegetation and man made objects in the displays. DMA data files with this type of information are available at the school. Cultural features will have a major impact on any defense planning application.

The graphics workstations used at the Naval Postgraduate School have multiple processors. Studies should be made utilizing this resource in an effort to improve simulator performace. In SDP, both of the Prolog modules could be easily passed off to another processor.

Finally, as stated, the current prototype defense planning module consists of only a few simple rules. Additional work can be done to improve the robustness of this rule set. To improve the accuracy of the defense planning module, greater terrain detail should be looked at in the calculations. Not only should the terrain be considered at a higher resolution, but the number of directions checked should be increased from the current prototype. This would allow a greater chance of positioning the weapon system at maximum range, while decreasing the chances of not having line-of-sight with the defended asset. Also, additional rules could be added to handle such factors as weather considerations, battle damage projections, and logistics considerations.

# APPENDIX A

SB-Prolog is a public domain version of Prolog for machines utilizing the Unix operating system. It was originally developed at the State University of New York (SUNY) at Stony Brook from 1984 to August 1986. Since August of 1986, SB-Prolog has continued to be developed at the University of Arizona, Tucson, where the code is maintained on line. The system is written in C, and is based on an extension of the Warren Abstract Machine (WAM). It has a number of special features including: (1) compilation of object files; (2) dynamic loading of predicates; (3) full integration between compiled and interpreted code; and (4) a facility for the definition and expansion of macros that is fully compatible with the runtime system. (Debray&, 1989, pp. 2,44)

A copy of the SB-Prolog code was transferred from the University of Arizona to the Naval Postgraduate School for use in the development of the SDP prototype. A special patch file, also available at the University of Arizona, had to be transferred and installed to allow the WAM simulator to run on the Silicon Graphics Inc., IRIS workstations. Prior to using SB-Prolog, the user must modify his *.chsrc* file to include the command *setenv SIMPATH SBP/modlib:SBP/lib:SBP/cmplib* where *SBP* denotes the path to the root of the SB-Prolog system directories (Debray&, 1989, p. 3). This allows the WAM simulator to find, and dynamicly load, required files for system operation.

In the SDP prototype, the compilation option of SB-Prolog was used to create byte code files of both the module to calculate low-altitude avenues of approach, and the module to do the defensive positioning of weapon systems. This allows each of these modules to execute from within the SDP simulator without activating the SB-Prolog interpreter mode. To execute byte code files in this manner, the user manual indicates that the user must "include the undefined predicate handler '_$undefined_pred'/1 to the compiled file" (Debray&, 1989, p. 31). However, when developing the modules for SDP, we found that a number of additional predicates from the SB-Prolog system files were required to initialize and execute a separate SB-Prolog byte code files in the manner desired. The following code/predicates were added to each module developed.

```
% The following code was added to be the first calls made in both modules in
order to initialize the SB-Prolog system.

$init_sys,
$load_mod($io),
$load_mod($read),
$load($prorc),call($prorc),
$globalset('_$abort_cutpoint'(0)),
$globalset('_$break_level'(0)),

% The following code had to be added in support of the above system calls.

/* the undefined_pred interrupt handler: */
'_$undefined_pred'(Term) :-
    $functor0(Term,Pred),
    $arity(Term,Arity),
    ('_$nodynload'(Pred,Arity) ->
        fail ;
        (not(not('_$definable'(Term))),'_$call'(Term))).

'_$nodynload'(_,_) :- fail.
```

```
$init_sys :-
    $load($opcode),    /* THIS FILE MUST BE THE FIRST LOADED */
    $load($assert),
    $load($db),
    $load($dbcmpl),
    $load($bio),
    $load($bmeta),
    $load($name),
    $load($blist),
    $load($call),
    $load($glob),
    $load($funrel),
    $assert_union(loaded_mods(_),$loaded_mods(_)), /* nec for compile */
    $assert_abolish_i(defined_mods(_,_)),  /* nec for compile [a] */
    $globalset('_$nofile_msg'(1)).

$loaded_mods($buff). /* with readloop */
$loaded_mods($init_sys). /* with readloop */
$loaded_mods($assert).
$loaded_mods($db).
$loaded_mods($dbcmpl).
$loaded_mods($bio).
$loaded_mods($bmeta).
$loaded_mods($name).
$loaded_mods($blist).
$loaded_mods($call).
$loaded_mods($glob).
$loaded_mods($funrel).

'_$definable'(Term) :-
    $functor0(Term,Pred),$arity(Term,Arity),
/*  $telling(F),$tell(user),$writename('loading: '),$writename(Pred),
    $writename('/'),$writename(Arity),$nl,$tell(F), */
    defined_mods(Module_name,Pred_name_list), /* for each module.. */
    $init_membernv(Pred/Arity,Pred_name_list),
    /* is needed pred in this one? */
    !,                         /* yes! */
    $load_mod(Module_name),              /* go load it if necessary */
    $name(Module_name,Mod_name_chars),
    $append(Mod_name_chars,"_export",Exp_name_chars),
    $name(Exp_name,Exp_name_chars),
    $bldstr(Exp_name,1,Modcall),arg(1,Modcall,Explist),
    '_$call'(Modcall),
    $load_preds(Explist,Pred_name_list,Module_name).
```

39

```
'_$definable'(Term) :-          /* no module to define pred */
    $functor0(Term,Pred),
    $load(Pred),!,              /* file exists */
    (not($pred_undefined(Term))    /* file defines pred */
    ;
    $arity(Term,Arity),
    $telling(Fi),$tell(user),
    $writename(Pred),$writename('/'),$writename(Arity),
    $writename(' not defined by file'),$nl,$tell(Fi),fail
    ).
'_$definable'(Term) :-
    '_$nofile_msg'(1),          /* if message is to be displayed */
    $telling(Fi),$tell(user), $writename('No file to define '),
    $functor0(Term,Pred),$arity(Term,Arity),
    $writename(Pred),$writename('/'),$writename(Arity),$nl,$tell(Fi),fail.


$define_mod(Mod,Implist) :-
    defined_mods(Mod,Implist),!. /* no-op if there */
$define_mod(Mod,Implist) :-
    $assert(defined_mods(Mod,Implist)).


$load_mod(Module_name) :-
    loaded_mods(Module_name),!. /* already loaded, noop */
$load_mod(Module_name) :-
/*  $telling(Fi),$tell(user),$writename('load module: '),
    $writename(Module_name),$nl,$tell(Fi), */
    $load(Module_name),!,          /* now loaded */
    $assert(loaded_mods(Module_name)).
    /* $load_sub_mods(Module_name). */
$load_mod(Module_name) :-                  /* no such file */
    $telling(Fi),$tell(user),
    $writename('No file to define module '),
    $writename(Module_name),$nl,$tell(Fi),fail.

/* $load_sub_mods(Module_name) :-
    $name(Module_name,Mod_name_chars),
    $append(Mod_name_chars,"_use",Use_name_chars),
    $name(Use_name,Use_name_chars),
    $bldstr(Use_name,2,Modusecall),
    not($pred_undefined(Modusecall)),
    arg(1,Modusecall,M),arg(2,Modusecall,U),
    '_$call'(Modusecall),not(defined_mods(M,U)),!,
    $assert_union(defined_mods(_,_),Modusecall).
$load_sub_mods(_). */
```

```
$load_preds([],[],_) :- !.
$load_preds([],[_|_],Mod) :-
    !,
    $telling(Fi),$tell(user),
    $writename('Illegal use list for module: '),
    $writename(Mod),$nl,$tell(Fi),fail.
$load_preds([_|_],[],Mod) :-
    !,
    $telling(Fi),$tell(user),
    $writename('Illegal use list for module: '),
    $writename(Mod),$nl,$tell(Fi),fail.
$load_preds([Inname|Explist],[Inname|Pred_name_list],Mod) :- !,
    $load_preds(Explist,Pred_name_list,Mod).
$load_preds([Inname/Arity|Explist],[Outname/Arity|Pred_name_list],Mod) :-
    !,
    $bldstr(Outname,Arity,Outstr),$bldstr(Inname,Arity,Instr),
    ($pred_undefined(Outstr),!,$assert_union(Outstr,Instr),
        $load_preds(Explist,Pred_name_list,Mod)
    ;
        $telling(Fi),$tell(user),
        $writename('Attempt to redefine '),$writename(Outname),
        $writename('/'),$writename(Arity),$nl,
        $tell(Fi),
        $load_preds(Explist,Pred_name_list,Mod)).
$load_preds([Inname/Inarity|Explist],[Outname/Outarity|Pred_name_list],Mod)
:-
    $telling(Fi),$tell(user),
    $writename('Incorrect import arity: '),$writename(Outname),
    $writename('/'),$writename(Outarity),$nl,
    $tell(Fi),
    $load_preds(Explist,Pred_name_list,Mod),fail.

$init_membernv(Pa,[Tpa|_]) :- nonvar(Tpa),Pa=Tpa.
$init_membernv(Pa,Pn_list) :-
    nonvar(Pn_list),
    Pn_list=[_|Pn_tail],$init_membernv(Pa,Pn_tail).

/* the keyboard interrupt handler */

'_$keyboard_int'(Call) :- break,'_$call'(Call).

/* the general interrupt handler! */
```

```prolog
'_$interrupt'(Call,Code) :-
    Code =:= 0 -> '_$undefined_pred'(Call);
      (Code =:= 1 -> '_$keyboard_int'(Call)
        (Code =:= 2 ->
          $tell(user),$writename('Stack Overflow!!'),$nl,abort
          ;
          $writename('Illegal interrupt code'),halt)).


/****************************************************/
/* This is the dynamic load routine. It gets the SIMPATH global variable and
uses it to determine what files to try to load. */

load(File) :- $load(File).      /* for now */
$load(File) :-
    $buff_code(File, 0, 31 /*gb*/,0'/) -> $loadqual(File,0);
      not(not($nnload(File))).

$loadqual(File,Rc) :- '_$builtin'(11).

$nnload(File) :-
    $conlength(File,Flen),
    $readl_simpath(Dir),
    $conlength(Dir,Dlen),Wlen is Flen+Dlen+1,
    $alloc_heap(Wlen,Wname),
    $substring(0,Dlen,Dir,0,Wname,Dlen),
    $substring(0,1,'/',Dlen,Wname,Floc),
    $substring(0,Flen,File,Floc,Wname,Wlen),
    $loadqual(Wname,0),! .

$readl_simpath(Dir) :-
    $getenv_simpath(Simpath),
    $readl_getenvdir(Simpath,0,Dir).

$readl_getenvdir(Simpath,Loc,Dir) :-
    $subdelim(1,':',Dir1,Loc,Simpath,Nloc) ->
      (Dir = Dir1;
        $readl_getenvdir(Simpath,Nloc,Dir))
    /* else */;
      $conlength(Simpath,Len),Llen is Len-Loc,
      $substring(1,Llen,Dir,Loc,Simpath,_).

/* get environment variable value */

$getenv_simpath(X) :- '_$builtin'(12).
```

```
/* $buff **************************************************/
/* needed for dynamic loader */

$buff_export([$alloc_perm/2,$alloc_heap/2,$trimbuff/3,$buff_code/4,$symtype/2,
        $substring/6,$subnumber/6,$subdelim/6,$conlength/2,
        $pred_undefined/1, $hashval/3]).

$alloc_perm(Size, Buff) :- $alloc_buff(Size,Buff,0,0,0).

$alloc_heap(Size, Buff) :- $alloc_buff(Size,Buff,1,0,0).

/* Type 0: perm, 1: heap, 2: from Supbuff */
$alloc_buff(Size,Buff,Type,Supbuff,Retcode) :-
    $alloc_buff1(Size,Buff,Type,Supbuff,Retcode),
    (Retcode == 0 ->
        $writename('alloc failed'),$nl,fail;
        true).
$alloc_buff1(Size,Buff,Type,Supbuff,Retcode) :- '_$builtin'(76).

$trimbuff(Size,Buff,Type) :- '_$builtin'(79).
$trimbuff(Size,Buff,Type,Supbuff) :- '_$builtin'(79).

$buff_code(Buff,Offset,Disc,Term) :- '_$builtin'(77).

/* Type = 0: no ep, 1: dynamic, 2: ep to compiled code, 3: buffer */
$symtype(Term,Type) :- '_$builtin'(42).

$substring(Dir,NumBytes,Const,Locin,Buff,Locout) :- '_$builtin'(51).

$subnumber(Dir,NumBytes,NumCon,Locin,Buff,Locout) :- '_$builtin'(52).

$subdelim(Dir,Delim,Const,Locin,Buff,Locout) :- '_$builtin'(53).

$conlength(Const,Len) :- '_$builtin'(54).

$pred_undefined(Term) :-
    $symtype(Term,D),
    (D=:=0;
    D==0,D=:=3).

$hashval(Arg, Size, Hashval) :- '_$builtin'(43).
```

```
/* These routines put numbers into buffers in internet format */
$buff_putnum_n(Buff,Loc,Len,Num) :-
    Len =< 1 -> $buff_code(Buff,Loc,30 /*pb*/ ,Num)
    /* else */ ;
        Byte is Num/ 255,Rest is Num >> 8,
        Nlen is Len-1,Sloc is Loc+Nlen,
        $buff_code(Buff,Sloc,30,Byte),
        $buff_putnum_n(Buff,Loc,Nlen,Rest).


$buff_getnum_n(Buff,Loc,Len,Num) :-
    Len =< 1 -> $buff_code(Buff,Loc,31 /*gb*/ ,Num)
    /*else*/ ;
        Nlen is Len-1,Sloc is Loc+Nlen,
        $buff_code(Buff,Sloc,31,Byte),
        $buff_getnum_n(Buff,Loc,Nlen,Rest),
        Num is (Rest << 8) + Byte.


$globalset_a(Place,Value) :-
    integer(Value) ->
        ($opcode( getnumcon, GetNumOp ),
        $buff_code(Place, 0, 7, /*gepb*/ Buff),
        $buff_code(Buff, 4, 3, /*ps*/  GetNumOp /*getnumcon*/),
        $buff_code(Buff, 8, 2, /*pn*/  Value)) ;
    (real(Value) ->
        ($opcode( getfloatcon, GetFltOp ),
        $buff_code(Place, 0, 7, /*gepb*/ Buff),
        $buff_code(Buff, 4, 3, /*ps*/  GetFltOp /*getfloatcon*/),
        $buff_code(Buff, 8,27, /*pf*/  Value))).

/* for debugging */
$writename(X) :- '_$builtin'(133).
$nl :- '_$builtin'(26).

'_$tab_size'(17).

break :- '_$break_level'(Blevel),
    Nblevel is Blevel+1, $globalset('_$break_level'(Nblevel)),
    $readl_userio(I,O),
    $writename('[ Break (level '),$writename(Nblevel),
    $writename(') ]'),$nl,$readl_brklp1,
    $globalset('_$break_level'(Blevel)),
    $writename('[ End break (level '),$writename(Nblevel),
    $writename(') ]'),$nl,
    $readl_resetio(I,O).       /* should we reset here ? */
```

```
$readl_brklp1 :- repeat,'_$break_level'(Blevel),
    $writename(Blevel),$writename(': ?- '),
    $read(X, Vars),
    ($readl_stop(X),!;$readl_procq(X,Vars)).

abort :- '_$abort_cutpoint'(Cp), '_$cutto'(Cp), fail.

repeat.
repeat :- repeat.
```

Once the above code was added to each module, they compiled and executed as expected. A bug was discovered in the SB-Prolog system however, in which any time an attempt was made to compile a very large object file, the system would attempt to execute garbage collection to aid in the compilation. The garbage collection facility has not been correctly implemented in this version of SB-Prolog, and the result is a system crash. We contacted Saumya Debray at the University of Arizona about this problem, and we were assured that they were attempting to correct the bug for future versions.

# APPENDIX B

```
% The following code is for the calculation of four low-altitude avenues
% of approach to the region of the map containing the defended asset. The
% additional code required to run this program as a compiled module under
% SB-Prolog has been omitted, and can be found in appendix A. The map
% elevation data has also been omitted. Any sized map may be used with
% this program by inserting elevation data in the form 'elev([X,Y],E).'
% and making the changes to the s_8, change_x, change_y, dist, and
% add_successors predicates.

sdp :-
    nodynload(agenda,4),nodynload(usedstate,2),
    nl,nl,
    see('asset.tmp'),
    read(Coordinate),display(Coordinate),
    seen,
    nl,
    write(' Stand by while fast-attack avenues are calculated. '),
    nl, nl,
    s_8(Coordinate).

member(X, [X|L]).
member(X, [Y|L]) :- member(X,L).


s_8(Initial_state) :-
    astarsearch(Initial_state,[7,Y],P1),
    astarsearch(Initial_state,[X,7],P2),
    astarsearch(Initial_state,[X,1],P3),
    astarsearch(Initial_state,[1,Y],P4),
    tell('paths'),
    output_path(P1,P2,P3,P4),
    told,
    tell('propaths.tmp'),
    write(P1),write('.'),nl,write(P2),write('.'),nl,
    write(P3),write('.'),nl,write(P4),write('.'),
    told.
```

```prolog
% Predicates to change the coordinates found by the search from the 7X7
% orientation used by this program to UTM coordinates to be used by the
% graphical user interface.

output_path(P1,P2,P3,P4) :-
    length(P1,N1),write(N1),nl,break_path(P1),
    length(P2,N2),write(N2),nl,break_path(P2),
    length(P3,N3),write(N3),nl,break_path(P3),
    length(P4,N4),write(N4),nl,break_path(P4).

break_path([]) :- !.
break_path([Point|Path]) :-
    point_out(Point),break_path(Path).

point_out([X,Y]) :-
    change_x(X,X1),change_y(Y,Y1),
    write(X1),write(' '),write(Y1),nl.

change_x(1,43500.0).
change_x(2,48500.0).
change_x(3,53500.0).
change_x(4,58500.0).
change_x(5,63500.0).
change_x(6,68500.0).
change_x(7,73500.0).

change_y(1,62500.0).
change_y(2,67500.0).
change_y(3,72500.0).
change_y(4,77500.0).
change_y(5,82500.0).
change_y(6,85500.0).
change_y(7,92500.0).


% Predicates to calculate both the actual and heuristic costs of points used
% in the A* search.

cost([X],0).
cost([X,Y|L],E) :-
    calculate_cost(Y,X,E2),
    cost([Y|L],E3),
    E is E2 + E3.
```

```prolog
calculate_cost([X1,X2],[N1,N2], Cost_X_N ) :-
    elev([X1,X2],E),
    D1 is ((X1-N1)*(X1-N1) + (X2-N2)*(X2-N2)),
    square(Dist_X_N,D1),Cost_X_N is Dist_X_N + E.

evalstate(State,Evaluation) :-
    elev(State,E),dist(State,Distance),
    Evaluation is Distance + E.

dist([X1,Y1],Dist_to_goal) :-
    goalreached([7,Y2]),var(Y2),
    D1 is ((7 - X1)*(7 - X1)),square(Dist_to_goal,D1).

dist([X1,Y1],Dist_to_goal) :-
    goalreached([X2,7]),var(X2),
    D1 is ((7 - Y1)*(7 - Y1)),square(Dist_to_goal,D1).

dist([X1,Y1],Dist_to_goal) :-
    goalreached([1,Y2]),var(Y2),
    D1 is ((X1 - 1)*(X1 - 1)),square(Dist_to_goal,D1).

dist([X1,Y1],Dist_to_goal) :-
    goalreached([X2,1]),var(X2),
    D1 is ((Y1 - 1)*(Y1 - 1)), square(Dist_to_goal,D1).


% Predicates to define what is a successor of any given point.

successor( X, Y ) :-
    neighbor(X,Y).

neighbor([X,Y],[X1,Y1]) :-
        X1 is X-1, Y1 is Y+1.
neighbor([X,Y],[X1,Y1]) :-
        X1 is X+1, Y1 is Y-1.
neighbor([X,Y],[X1,Y1]) :-
        X1 is X+1, Y1 is Y+1.
neighbor([X,Y],[X1,Y1]) :-
        X1 is X-1, Y1 is Y-1.
neighbor([X,Y],[X1,Y]) :-
        X1 is X+1.
```

```prolog
neighbor([X,Y],[X1,Y]) :-
        X1 is X-1.
neighbor([X,Y],[X,Y1]) :-
        Y1 is Y-1.
neighbor([X,Y],[X,Y1]) :-
        Y1 is Y+1.
```

% A* search code as given by Prof. Rowe, with modifications for open goal.

```prolog
astarsearch(Start,Goal,Path) :-
   asserta(goalreached(Goal)),
   cleandatabase, add_state(Start,[]),
   repeatifagenda, pick_best_state(State,Pathlist),
   add_successors(State,Pathlist), agenda(State,Goalpathlist,C,D),
   retract(agenda(State,Goalpathlist,C,D)),
   abolish(goalreached,1),reverse(Goalpathlist,Path),
   display(Path),nl.

reverse( L1, L2 ) :-
   eff_rev( L1, [], L2 ), !.

eff_rev( [ X | L1 ], L2, L3 ) :-
   eff_rev( L1, [ X | L2 ], L3 ).        % append implied
eff_rev( [], L, L ).

pick_best_state(State,Pathlist) :-
   asserta(beststate(dummy,dummy,dummy)),
   agenda(S,SL,C,D), beststate(S2,SL2,D2), special_less_than(D,D2),
   retract(beststate(S2,SL2,D2)), asserta(beststate(S,SL,D)), fail.
pick_best_state(State,Pathlist) :-
   beststate(State,Pathlist,D),
   retract(beststate(State,Pathlist,D)), not(D=dummy), !.

add_successors(State,Pathlist) :-
   goalreached(State), !.
add_successors([X,Y],Pathlist) :-
   X > 0, X < 8, Y > 0, Y < 8,
   successor([X,Y],Newstate),
   add_state(Newstate,Pathlist), fail.
add_successors(State,Pathlist) :-
   retract(agenda(State,Pathlist,C,D)),
   asserta(usedstate(State,C)), fail.
```

49

```prolog
add_state(Newstate,Pathlist) :-
   cost([Newstate|Pathlist],Cnew), !,
   agenda_check(Newstate,Cnew), !, usedstate_check(Newstate,Pathlist,Cnew),
   !,evalstate(Newstate,Enew), D is Enew + Cnew,
   asserta(agenda(Newstate,[Newstate|Pathlist],Cnew,D)), !.
add_state(Newstate,Pathlist) :-
   not(cost([Newstate|Pathlist],Cnew)),
   write('Warning: your cost function failed on path list '),
   write(Pathlist),nl, !.
add_state(Newstate,Pathlist) :-
   not(evalstate(Newstate,Enew)),
   write('Warning: your evaluation function failed on state '),
   write(Newstate), nl, !.

agenda_check(S,C) :-
   agenda(S,P2,C2,D2), C<C2, retract(agenda(S,P2,C2,D2)), !.
agenda_check(S,C) :-
   agenda(S,P2,C2,D2), !, fail.
agenda_check(S,C).

usedstate_check(S,P,C) :-
   usedstate(S,C2), C<C2, retract(usedstate(S,C2)),
   asserta(usedstate(S,C)), !, fix_agenda(S,P,C,C2).
usedstate_check(S,P,C) :-
   usedstate(S,C2), !, fail.
usedstate_check(S,P,C).

fix_agenda(S,P,C,OldC) :-
   agenda(S2,P2,C2,D2), replace_front(P,S,P2,Pnew),
   cost(Pnew,Cnew), Dnew is D2+C-OldC, retract(agenda(S2,P2,C2,D2)),
   asserta(agenda(S2,Pnew,Cnew,Dnew)), fail.

replace_front(P,S,P2,Pnew) :-
   append(P3,[S|P4],P2), append(P,[S|P4],Pnew), !.

repeatifagenda.
repeatifagenda :-
   agenda(X,Y,Z,W), repeatifagenda.

special_less_than(X,dummy) :- !.
special_less_than(X,Y) :- X<Y.
```

```prolog
cleandatabase :-
   checkabolish(agenda,4), checkabolish(usedstate,2),
   checkabolish(beststate,3), checkabolish(counter,1).

checkabolish(P,N) :-
   abolish(P,N), !.
checkabolish(P,N).

append([],L,L).
append([I|L1],L2,[I|L3]) :-
   append(L1,L2,L3).
```

% elevation data for 7X7 grid with each square being 5 sqr KM is omitted.

## APPENDIX C

% The following code is for the positioning of the selected weapon systems
% to defend the given asset.  The additional code required to run this
% program as a compiled module under SB-Prolog has been omitted, and can be
% found in appendix A.  The 35 x 35 map elevation data files have also been
% omitted.

```
setup_def :-
    write('Defense Planner Active'),nl,
    get_def_wpns(Weapon_type),
    get_asset_data(Asset,SMAsset),
    get_propaths(Path1,Path2,Path3,Path4),
    get_directions(Path1,Path2,Path3,Path4,WD1,WD2,WD3,WD4),
    place_wpns(Weapon_type,Asset,SMAsset,P1,P2,P3,P4,WD1,WD2,WD3,WD4,[],Wlist_new),
    output_wpns(Wlist_new),
    write(Asset),nl,write(SMAsset),nl,
    write(Wlist_new).
```

% Initialization predicates to get data to plan the defense.

```
get_def_wpns(WT) :-
    see('def.tmp'),
    read(WT),
    seen.

get_propaths(P1,P2,P3,P4) :-
    see('propaths.tmp'),
    read(P1), read(P2),
    read(P3), read(P4),
    seen.

get_directions(P1,P2,P3,P4,WD1,WD2,WD3,WD4) :-
    direction(P1,1,WD1),
    direction(P2,2,WD2),
    direction(P3,3,WD3),
    direction(P4,4,WD4).

get_terrain([[X,Y]|[]]) :-
    get_data([[X,Y],[X,Y]]).
get_terrain(Path) :-
    get_data(Path).
```

```
get_data([[X,Y]|[[X1,Y1]|P]]) :-
    X >= 1, X =< 2, X1 >= 1, X1 =< 2,
    load(elev1).
get_data([[X,Y]|[[X1,Y1]|P]]) :-
    X >= 2, X =< 3, X1 >= 2, X1 =< 3,
    load(elev2).
get_data([[X,Y]|[[X1,Y1]|P]]) :-
    X >= 3, X =< 4, X1 >= 3, X1 =< 4,
    load(elev3).
get_data([[X,Y]|[[X1,Y1]|P]]) :-
    X >= 4, X =< 5, X1 >= 4, X1 =< 5,
    load(elev4).
get_data([[X,Y]|[[X1,Y1]|P]]) :-
    X >= 5, X =< 6, X1 >= 5, X1 =< 6,
    load(elev5).
get_data([[X,Y]|[[X1,Y1]|P]]) :-
    X >= 6, X =< 7, X1 >= 6, X1 =< 7,
    load(elev6).

get_asset_data(Asset,SMAsset) :-
    see('asset.tmp'),
    read(Junkasset),
    read(Asset),
    get_small_map(Asset,SMAsset),
    seen.

get_small_map([X,Y],[X2,Y2]) :-
    X1 is (X - 41000) / 1000,
    Y1 is (Y - 60000) / 1000,
    X2 is integer(X1),
    Y2 is integer(Y1).

direction([[X,Y]|[]],PN,Wpn_Dir) :-
    PN = 1, Wpn_Dir is 90.
direction([[X,Y]|[]],PN,Wpn_Dir) :-
    PN = 2, Wpn_Dir is 0.
direction([[X,Y]|[]],PN,Wpn_Dir) :-
    PN = 3, Wpn_Dir is 180.
direction([[X,Y]|[]],PN,Wpn_Dir) :-
    PN = 4, Wpn_Dir is 270.
direction([[X1,Y1]|[[X2,Y2]|P]],PN,Wpn_Dir) :-
    X is (X2 - X1),
    Y is (Y2 - Y1),
    facing(PN,X,Y,Wpn_Dir).
```

```prolog
facing(X,0,1,0).
facing(X,1,1,45).
facing(X,1,0,90).
facing(X,1,-1.0,135).
facing(X,0,-1.0,180).
facing(X,-1.0,-1.0,225).
facing(X,-1.0,0,270).
facing(X,-1.0,1,315).
```

% Top predicate to handle weapons placement based on menu choice of user.

```prolog
place_wpns(1,A,SMA,P1,P2,P3,P4,D1,D2,D3,D4,Wpnlist_old,Wpnlist_new) :-
    place_vul(A,SMA,P1,D1,Wpnlist_old,Wpnlist1),
    place_vul(A,SMA,P2,D2,Wpnlist1,Wpnlist2),
    place_vul(A,SMA,P3,D3,Wpnlist2,Wpnlist3),
    place_vul(A,SMA,P4,D4,Wpnlist3,Wpnlist_new).
place_wpns(2,A,SMA,P1,P2,P3,P4,D1,D2,D3,D4,Wpnlist_old,Wpnlist_new) :-
    place_sting(A,SMA,P1,D1,Wpnlist_old,Wpnlist1),
    place_sting(A,SMA,P2,D2,Wpnlist1,Wpnlist2),
    place_sting(A,SMA,P3,D3,Wpnlist2,Wpnlist3),
    place_sting(A,SMA,P4,D4,Wpnlist3,Wpnlist_new).
place_wpns(3,A,SMA,P1,P2,P3,P4,D1,D2,D3,D4,Wpnlist_old,Wpnlist_new) :-
    place_wpns(2,A,SMA,P1,P2,P3,P4,D1,D2,D3,D4,Wpnlist_old,Wpnlist1),
    place_wpns(1,A,SMA,P1,P2,P3,P4,D1,D2,D3,D4,Wpnlist_old,Wpnlist2),
    append(Wpnlist1,Wpnlist2,Wpnlist_new).
```

% Predicates to handle placement of 4 Vulcans.

```prolog
place_vul(Asset,SMAsset,Path,Dir,Wpnlist,Newlist) :-
    get_terrain(Path),
    position_vul(Dir,SMAsset,Asset,UTM_Loc),
    make_list_vul(UTM_Loc,Dir,Wpnlist,Newlist).

make_list_vul(UTM_Loc,Dir,Wpnlist,Newlist) :-
    wpnmember([2,UTM_Loc,Dir],Wpnlist),
    mod_list_vul([2,UTM_Loc,Dir],Wpnlist,Newlist).
make_list_vul(UTM_Loc,Dir,Wpnlist,Newlist) :-
    append(Wpnlist,[[2,UTM_Loc,Dir]],Newlist).

mod_list_vul([2,[X,Y],Z],Oldlist,Modlist) :-
    X1 is (X - 200.0), Y1 is (Y - 200.0),
    X2 is (X + 200.0), Y2 is (Y + 200.0),
    delete_wpn([2,[X,Y],Z],Oldlist,Changedlist),
    append(Changedlist,[[2,[X1,Y1],Z],[2,[X2,Y2],Z]],Modlist).
```

```
delete_wpn(X,[],[]).
delete_wpn(X,[X|L],M) :- delete_wpn(X,L,M).
delete_wpn(X,[Y|L],[Y|M]) :- not(X = Y), delete_wpn(X,L,M).
```

% Predicates to handle placement of 4 Stingers.

```
place_sting(Asset,SMAsset,Path,Dir,Wpnlist,Newlist) :-
    get_terrain(Path),
    position_sting(Dir,SMAsset,Asset,UTM_Loc),
    make_list_sting(UTM_Loc,Dir,Wpnlist,Newlist).

make_list_sting(UTM_Loc,Dir,Wpnlist,Newlist) :-
    wpnmember([1,UTM_Loc,Dir],Wpnlist),
    mod_list_sting([1,UTM_Loc,Dir],Wpnlist,Newlist).
make_list_sting(UTM_Loc,Dir,Wpnlist,Newlist) :-
    append(Wpnlist,[[1,UTM_Loc,Dir]],Newlist).

mod_list_sting([T,[X,Y],Z],Oldlist,Modlist) :-
    X1 is (X - 500.0), Y1 is (Y - 500.0),
    X2 is (X + 500.0), Y2 is (Y + 500.0),
    delete_wpn([1,[X,Y],Z],Oldlist,Changedlist),
    append(Changedlist,[[1,[X1,Y1],Z],[1,[X2,Y2],Z]],Modlist).
```

% Predicates to handle positioning of individual Vulcan systems.

```
position_vul(0,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    Y1 is Y + 1,
    elev([X,Y],E), elev([X,Y1],E1),
    E >= E1, UTMY is (AY + 1000.0).
position_vul(0,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X + 1, Y1 is Y + 1,
    elev([X,Y],E), elev([X1,Y1],E1),
    E >= E1,
    UTMX is (AX + 707.107),UTMY is (AY + 707.107).
position_vul(0,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X - 1, Y1 is Y + 1,
    elev([X,Y],E), elev([X1,Y1],E1),
    E >= E1,
    UTMX is (AX - 707.107),UTMY is (AY + 707.107).
position_vul(0,[X,Y],[AX,AY],[AX,UTMY]) :-
    UTMY is (AY + 250.0).
```

```prolog
position_vul(45,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X + 1, Y1 is Y + 1,
    elev([X,Y],E), elev([X1,Y1],E1),
    E >= E1,
    UTMX is (AX + 707.107),UTMY is (AY + 707.107).
position_vul(45,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X + 1,
    elev([X,Y],E), elev([X1,Y],E1),
    E >= E1,
    UTMX is (AX + 1000.0).
position_vul(45,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    Y1 is Y + 1,
    elev([X,Y],E), elev([X,Y1],E1),
    E >= E1,
    UTMY is (AY + 1000.0).
position_vul(45,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    UTMX is (AX + 176.7767),UTMY is (AY + 176.7767).
position_vul(90,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X + 1,
    elev([X,Y],E), elev([X1,Y],E1),
    E >= E1,
    UTMX is (AX + 1000.0).
position_vul(90,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X + 1, Y1 is Y - 1,
    elev([X,Y],E), elev([X1,Y1],E1),
    E >= E1,
    UTMX is (AX + 707.107),UTMY is (AY - 707.107).
position_vul(90,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X + 1, Y1 is Y + 1,
    elev([X,Y],E), elev([X1,Y1],E1),
    E >= E1,
    UTMX is (AX + 707.107),UTMY is (AY + 707.107).
position_vul(90,[X,Y],[AX,AY],[UTMX,AY]) :-
    UTMX is (AX + 250.0).
```

```
position_vul(135,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is (X + 1),
   Y1 is (Y - 1),
   elev([X,Y],E), elev([X1,Y1],E1),
   E >= E1,
   UTMX is (AX + 707.107),UTMY is (AY - 707.107).
position_vul(135,[X,Y],[AX,AY],[AX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   Y1 is (Y - 1),
   elev([X,Y],E), elev([X,Y1],E1),
   E >= E1,
   UTMY is (AY - 1000.0).
position_vul(135,[X,Y],[AX,AY],[UTMX,AY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X + 1,
   elev([X,Y],E), elev([X1,Y],E1),
   E >= E1,
   UTMX is (AX + 1000.0).
position_vul(135,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   UTMX is (AX + 176.7767),UTMY is (AY - 176.7767).
position_vul(180,[X,Y],[AX,AY],[AX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   Y1 is Y - 1,
   elev([X,Y],E), elev([X,Y1],E1),
   E >= E1,
   UTMY is (AY - 1000.0).
position_vul(180,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X - 1,
   Y1 is Y - 1,
   elev([X,Y],E), elev([X1,Y1],E1),
   E >= E1,
   UTMX is (AX - 707.107),UTMY is (AY - 707.107).
position_vul(180,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X + 1,
   Y1 is Y - 1,
   elev([X,Y],E), elev([X1,Y1],E1),
   E >= E1,
   UTMX is (AX + 707.107),UTMY is (AY - 707.107).
position_vul(180,[X,Y],[AX,AY],[AX,UTMY]) :-
   UTMY is (AY - 250.0).
```

```prolog
position_vul(225,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X - 1,
   Y1 is Y - 1,
   elev([X,Y],E), elev([X1,Y1],E1),
   E >= E1,
   UTMX is (AX - 707.107),UTMY is (AY - 707.107).
position_vul(225,[X,Y],[AX,AY],[UTMX,AY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X - 1,
   elev([X,Y],E), elev([X1,Y],E1),
   E >= E1,
   UTMX is (AX - 1000.0).
position_vul(225,[X,Y],[AX,AY],[AX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   Y1 is Y - 1,
   elev([X,Y],E), elev([X,Y1],E1),
   E >= E1,
   UTMY is (AY - 1000.0).
position_vul(225,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   UTMX is (AX - 176.7767),UTMY is (AY - 176.7767).
position_vul(270,[X,Y],[AX,AY],[UTMX,AY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X - 1,
   elev([X,Y],E), elev([X1,Y],E1),
   E >= E1,
   UTMX is (AX - 1000.0).
position_vul(270,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X - 1,
   Y1 is Y + 1,
   elev([X,Y],E), elev([X1,Y1],E1),
   E >= E1,
   UTMX is (AX - 707.107),UTMY is (AY + 707.107).
position_vul(270,[X,Y],[AX,AY],[UTMX,UTMY]) :-
   X =< 34, X >= 2, Y =< 34, Y >= 2,
   X1 is X - 1,
   Y1 is Y - 1,
   elev([X,Y],E), elev([X1,Y1],E1),
   E >= E1,
   UTMX is (AX - 707.107),UTMY is (AY - 707.107).
position_vul(270,[X,Y],[AX,AY],[UTMX,AY]) :-
   UTMX is (AX - 250.0).
```

```prolog
position_vul(315,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X - 1,
    Y1 is Y + 1,
    elev([X,Y],E), elev([X1,Y1],E1),
    E >= E1,
    UTMX is (AX - 707.107),UTMY is (AY + 707.107).
position_vul(315,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    Y1 is Y + 1,
    elev([X,Y],E), elev([X,Y1],E1),
    E >= E1,
    UTMY is (AY + 1000.0).
position_vul(315,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 34, X >= 2, Y =< 34, Y >= 2,
    X1 is X - 1,
    elev([X,Y],E), elev([X1,Y],E1),
    E >= E1,
    UTMX is (AX - 1000.0).
position_vul(315,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    UTMX is (AX - 176.7767),UTMY is (AY + 176.7767).


% Predicates to position individual Stinger weapon systems.

position_sting(0,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    Y1 is Y + 1, Y2 is Y + 2, Y3 is Y + 3,
    elev([X,Y],E), elev([X,Y1],E1),
    elev([X,Y2],E2),elev([X,Y3],E3),
    E1 =< E,E2 =< E1,E3 =< E2, UTMY is (AY + 2500.0).
position_sting(0,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2, Y1 is Y + 1, Y2 is Y + 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E1 =< E,E2 =< E1,
    UTMX is (AX + 1767.7669),UTMY is (AY + 1767.7669).
position_sting(0,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2, Y1 is Y + 1, Y2 is Y + 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E1 =< E,E2 =< E1,
    UTMX is (AX - 1767.7669),UTMY is (AY + 1767.7669).
position_sting(0,[X,Y],[AX,AY],[AX,UTMY]) :-
    UTMY is (AY + 1500.0).
```

```
position_sting(45,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2,
    Y1 is Y + 1, Y2 is Y + 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E1 =< E,E2 =< E1,
    UTMX is (AX + 1767.7669),UTMY is (AY + 1767.7669).
position_sting(45,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2, X3 is X + 3,
    elev([X,Y],E), elev([X1,Y],E1),
    elev([X2,Y],E2),elev([X3,Y],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMX is (AX + 2500.0).
position_sting(45,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    Y1 is Y + 1, Y2 is Y + 2, Y3 is Y + 3,
    elev([X,Y],E), elev([X,Y1],E1),
    elev([X,Y2],E2),elev([X,Y3],E3),
    E1 =< E,E2 =< E1,E3 =< E2, UTMY is (AY + 2500.0000).
position_sting(45,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    UTMX is (AX + 1060.6602),UTMY is (AY + 1060.6602).
position_sting(90,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2, X3 is X + 3,
    elev([X,Y],E), elev([X1,Y],E1),
    elev([X2,Y],E2),elev([X3,Y],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMX is (AX + 2500.0).
position_sting(90,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2,
    Y1 is Y - 1, Y2 is Y - 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E1 =< E,E2 =< E1,
    UTMX is (AX + 1767.7669),UTMY is (AY - 1767.7669).
position_sting(90,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2,
    Y1 is Y + 1, Y2 is Y + 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E1 =< E,E2 =< E1,
    UTMX is (AX + 1767.7669),UTMY is (AY + 1767.7669).
position_sting(90,[X,Y],[AX,AY],[UTMX,AY]) :-
    UTMX is (AX + 1500.0).
```

```prolog
position_sting(135,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2,
    Y1 is Y - 1, Y2 is Y - 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E >= E1,E1 >= E2,
    UTMX is (AX + 1767.7669),UTMY is (AY - 1767.7669).
position_sting(135,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    Y1 is Y - 1, Y2 is Y - 2, Y3 is Y - 3,
    elev([X,Y],E), elev([X,Y1],E1),
    elev([X,Y2],E2),elev([X,Y3],E3),
    E >= E1,E1 >= E2,E2 >= E3,
    UTMY is (AY - 2500.0).
position_sting(135,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2, X3 is X + 3,
    elev([X,Y],E), elev([X1,Y],E1),
    elev([X2,Y],E2),elev([X3,Y],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMX is (AX + 2500.0).
position_sting(135,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    UTMX is (AX + 1060.6602),UTMY is (AY - 1060.6602).
position_sting(180,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    Y1 is Y - 1, Y2 is Y - 2, Y3 is Y - 3,
    elev([X,Y],E), elev([X,Y1],E1),
    elev([X,Y2],E2),elev([X,Y3],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMY is (AY - 2500.0).
position_sting(180,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2,
    Y1 is Y - 1, Y2 is Y - 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E >= E1,E1 >= E2,
    UTMX is (AX - 1767.7669),UTMY is (AY - 1767.7669).
position_sting(180,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X + 1, X2 is X + 2,
    Y1 is Y - 1, Y2 is Y - 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E >= E1,E1 >= E2,
    UTMX is (AX + 1767.7669),UTMY is (AY - 1767.7669).
position_sting(180,[X,Y],[AX,AY],[AX,UTMY]) :-
    UTMY is (AY - 1500.0).
```

```prolog
position_sting(225,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2,
    Y1 is Y - 1, Y2 is Y - 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E >= E1,E1 >= E2,
    UTMX is (AX - 1767.7669),UTMY is (AY - 1767.7669).
position_sting(225,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2, X3 is X - 3,
    elev([X,Y],E), elev([X1,Y],E1),
    elev([X2,Y],E2),elev([X3,Y],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMX is (AX - 2500.0).
position_sting(225,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    Y1 is Y - 1, Y2 is Y - 2, Y3 is Y - 3,
    elev([X,Y],E), elev([X,Y1],E1),
    elev([X,Y2],E2),elev([X,Y3],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMY is (AY - 2500.0).
position_sting(225,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    UTMX is (AX - 1060.6602),UTMY is (AY - 1060.6602).
position_sting(270,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2, X3 is X - 3,
    elev([X,Y],E), elev([X1,Y],E1),
    elev([X2,Y],E2),elev([X3,Y],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMX is (AX - 2500.0).
position_sting(270,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2,
    Y1 is Y + 1, Y2 is Y + 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E >= E1,E1 >= E2,
    UTMX is (AX - 1767.7669),UTMY is (AY + 1767.7669).
position_sting(270,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2,
    Y1 is Y - 1, Y2 is Y - 2,
    elev([X,Y],E), elev([X1,Y1],E1),
    elev([X2,Y2],E2), E >= E1,E1 >= E2,
    UTMX is (AX - 1767.7669),UTMY is (AY - 1767.7669).
position_sting(270,[X,Y],[AX,AY],[UTMX,AY]) :-
    UTMX is (AX - 1500.0).
```

```
position_sting(315,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2,
    Y1 is Y + 1, Y2 is Y + 2,
    elev([X,Y],E), elev([X1,Y1],E1),elev([X2,Y2],E2),
    E >= E1,E1 >= E2,
    UTMX is (AX - 1767.7669),UTMY is (AY + 1767.7669).
position_sting(315,[X,Y],[AX,AY],[AX,UTMY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    Y1 is Y + 1, Y2 is Y + 2, Y3 is Y + 3,
    elev([X,Y],E), elev([X,Y1],E1),
    elev([X,Y2],E2),elev([X,Y3],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMY is (AY + 2500.0).
position_sting(315,[X,Y],[AX,AY],[UTMX,AY]) :-
    X =< 32, X >= 4, Y =< 32, Y >= 4,
    X1 is X - 1, X2 is X - 2, X3 is X - 3,
    elev([X,Y],E), elev([X1,Y],E1),
    elev([X2,Y],E2),elev([X3,Y],E3),
    E >= E1,E1 >= E2,E2 >= E3, UTMX is (AX - 2500.0).
position_sting(315,[X,Y],[AX,AY],[UTMX,UTMY]) :-
    UTMX is (AX - 1060.6602),UTMY is (AY + 1060.6602).


append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).


wpnmember(X,[X|L]).
wpnmember(X,[W|L]) :- wpnmember(X,L).


% Predicates to output weapon system locations for use by the graphics interface.

output_wpns(Wpnlist) :-
    tell('weapons.tmp'),
    length(Wpnlist,N),write(N),nl,
    break_list(Wpnlist),told.

break_list([]) :- !.
break_list([[N,[X,Y],Z]|Path]) :-
    write(N),write(' '),write(X),write(' '),
    write(Y),write(' '),write(Z),nl,
    break_list(Path).
```

# LIST OF REFERENCES

(Debray&, 1989) Debray, Saumya K., Dietrich, Suzanne, Pereira, Fernando, and Warren, David S., *The SB-Prolog System, Version 3.1, A User's Manual*, Department of Computer Science, University of Arizona, Tucson, AZ, December 1989.

(Drummond&, 1989) Drummond, William T. Jr., and Nizolak, Joseph P. Jr., *A Graphics Workstation Field Artillery Forward Observer Simulation Trainer*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.

(Fichten&, 1988) Fichten, Mark A., and Jennings, David H., *Meaningful Real-Time Graphics Workstation Performance Measurements*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

(FM44-1, 1983) Department of the Army, Field Manual 44-1, *U.S. Army Air Defense Employment*, Government Printing Office, Washington, D.C., May 1983.

(FM44-3, 1984) Department of the Army, Field Manual 44-3, *Air Defense Artillery Employment Chaparral/Vulcan/Stinger*, Government Printing Office, Washington, D.C., June 1984.

(FM100-5, 1986) Department of the Army, Field Manual 100-5, *Operations*, Government Printing Office, Washington, D.C., May 1986.

(NPS52-89-004, 1988) Naval Postgraduate School, Technical Report NPS52-89-004, *Meaningful Real-Time Graphics Workstation Performance Measurements*, Fichten, Mark A., Jennings, David H., and Zyda, Michael J., November 1988.

(Oliver&, 1987) Oliver, Michael R., and Stahl, David J., *Interactive, Networked, Moving Platform Simulators*, M.S. Thesis, Naval Postgraduate School, Montery, California, December 1987.

(Rowe, 1988) Rowe, Neil C., *Artificial Intelligence Through Prolog*, Prentice-Hall, Inc., 1988.

(SGI, 1987) Silicon Graphics Inc., *IRIS User's Guide, MEX Window Manager*, Mountain View, California, 1987.

(SGI, 1988) Silicon Graphics Inc., *4Sight User's Guide, Volume 1*, Mountain View, California, 1988.

(Shannon&, 1989) Shannon, Larry R., and Teter, William A., *An Autonomous Platform Simulator (APS)*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.

(Smith&, 1987) Smith, Douglas B., and Streyle, Dale G., *An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1987.

(Strong&, 1989) Strong, Randolf P., and Winn, Michael C., *The Moving Platform Simulator II: A Networked Real-Time Visual Simulator With Distributed Processing and Line-of-Sight Displays*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 0142 2
   Naval Postgraduate School
   Monterey, CA 93943-5002

3. Dr. Yuh-jeng Lee 50
   Naval Postgraduate School
   Code 52, Department of Computer Science
   Monterey, CA 93943-5100

4. Dr. Michael J. Zyda 2
   Naval Postgraduate School
   Code 52, Department of Computer Science
   Monterey, CA 93943-5100

5. CPT Roger S. Dixon 3
   4811 Idaho Ave.
   Nashville, TN 37209

6. Director of Combat Developments 1
   United States Army Air Defense Artillery School
   Fort Bliss, TX 79916

7. Director 1
   Headquarters, Department of the Army
   Artificial Intelligence Center
   ATTN: CSDS-AI
   Washington, DC 20310-0200