1-1-1991

Functional Specification And Implemented Capabilities Of The IST Semi-automated Dismounted Infantry System

Mikel D. Petty

INSTITUTE FOR SIMULATION AND TRAINING

May 1991

Functional Specification and Implemented Capabilities of the IST Semi-Automated Forces Dismounted Infantry System

Mikel D. Petty · Clark R. Karr · David R. Van Brackle
Donald D. Cross · Robert W. Franceschini · Gilbert L. Gonzalez

IST

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 23826

University of Central Florida
Division of Sponsored Research

IST-TR-91-20

INSTITUTE FOR SIMULATION AND TRAINING

Functional Specification and Implemented Capabilities of the IST Semi-Automated Dismounted Infantry System

May 1991

IST-TR-91-20

Prepared by

Mikel D. Petty • Clark R. Karr • David R. Van Brackle
Donald D. Cross • Robert W. Franceschini • Gilbert L. Gonzalez

Reviewed by

Name

Institute for Simulation and Training • 12424 Research Parkway, Suite 300 • Orlando, Florida 32826

University of Central Florida • Division of Sponsored Research

Institute for Simulation and Training
Intelligent Simulated Forces

Functional Specification and Implemented Capabilities of the IST Semi-Automated Forces Dismounted Infantry System

IST Technical Report IST-TR-91-20

Mikel D. Petty
Clark R. Karr
David R. Van Brackle
Donald D. Cross
Robert W. Franceschini
Gilbert L. Gonzalez

1 June 1991

Table of Contents

1. Introduction

1.1 Purpose

This document is the interim technical report required as contract deliverable 8A under Workplan 4, dated 4 January 1991, of DARPA contract N61339-89-C-004 (Intelligent Simulated Forces: Evaluation and Exploration of Computational and Hardware Strategies). This report describes the Semi-Autonomous Forces Dismounted Infantry (SAFDI) project, which is a subtask of that contract.

1.2 SIMNET and the IST SAFOR Testbed

The U.S. Army/DARPA SIMNET is a well-known distributed interactive simulation system. In SIMNET, individual vehicle simulators are connected via a computer network, permitting them to coexist in a common, shared simulation environment and to interact (e.g. engage in combat) through the exchange of network packets. SIMNET is used to train tank and vehicle crews in cooperative team tactics. (The documentation of SIMNET is extensive; Thorpe[1987] and Pope[1989] are good examples.)

In a SIMNET exercise, the simulated battlefield opponent (or "opposing force") for the trainees can be provided in two different ways. One technique is for instructors to operate vehicle simulators similar to those used by the trainees. The instructors are trained to behave in a way that simulates threat doctrine. This method is expensive in terms of both manpower and equipment. The second technique is to provide a simulator node that generates and controls one or more simulation entities on the network with a computer system (and possibly one or more human operators). Such a computer generated opposing force is usually referred to as a semi-automated force (SAFOR). The SIMNET SAFOR system was implemented by BBN Systems and Technologies and is described in Downes-Martin[1990] and Crooks[1990]. The BBN SAFOR system uses two minicomputers and a single human operator to generate and control up to approximately 40 vehicles.

Under the Intelligent Simulated Forces (ISF) contract, the Institute for Simulation and Training (IST) has been conducting research in the area of SAFOR systems. IST has developed a SIMNET-compatible SAFOR testbed, which is a SAFOR system that connects to the SIMNET network and provides a mechanism for testing SAFOR control algorithms. Using the testbed, IST has tested the applicability of various artificial intelligence techniques to SAFOR systems. The IST SAFOR testbed is documented in Danisas[1990] and Gonzalez[1990b]. IST's artificial intelligence investigations are described in Coleman[1990] and Clarke[1991].

1.3 Semi-Automated Forces Dismounted Infantry

Dismounted infantry is conspicuously absent from the SIMNET battlefield. Although the SIMNET network protocols and image generators include the necessary features to represent and display dismounted infantry units, the BBN SAFOR system does not generate them. The manned dismounted infantry workstation developed by BBN Fraser[1990] is interesting but impractical for the large numbers

of infantry needed to populate the SIMNET battlefield realistically. Consequently, the SIMNET armor trainees train in an environment in which they are free to drive about the battlefield oblivious to the serious threat posed by infantry. Infantry are especially dangerous because they are are hard to see and are armed with powerful anti-tank guided missiles. The retraining necessitated by this unrealistic environment complicates the entire training process and reduces the effectiveness of the SIMNET environment as a training tool.

As a means to provide both a needed enhancement to SIMNET training and a demonstration of the capabilities of the IST SAFOR Testbed, IST was asked to produce a SAFOR system capable of generating and controlling semi-automated forces dismounted infantry (SAFDI) in the SIMNET battlefield. This SAFDI project was undertaken by IST within the context of the ISF contract.

The IST SAFDI system adds the functionality of dismounted infantry to the SIMNET battlefield. The SAFDI system is composed of two components: a Simulator component and a User Interface. The Simulator component consists of the programs and data structures that implement Dismounted Infantry within the SIMNET environment. The Simulator is a version of the IST SAFOR Testbed enhanced with many capabilities specific to dismounted infantry. The User Interface is a separate computer system with which an operator issues commands and instructions to the Simulator. The Simulator, in turn, is responsible for carrying out those instructions within the SIMNET environment.

Within the SIMNET environment, SAFDI fireteams generated by the IST SAFDI system have a substantial set of capabilities. They can:

- sight activity within Line of Sight
- report sighting to operator via the User Interface
- kill enemy infantry and vehicles
- mount and dismount APCs
- be seen according to visual range and posture
- be killed
- change visual appearance based on posture
- change movement speed.

The entities controlled by the SAFDI system represent a generic five man fireteam; the structure of that fireteam is given in Appendix A.2. These entities are fully functional in the SIMNET environment in that the SAFDI system generates and accepts standard SIMNET PDUs, allowing the SAFDI fireteams to interact with the entities on the SIMNET network.

In summary, SAFDI fireteams move, change posture, detect enemy entities, report enemy units to the operator, fire weapons, can be killed by enemy fire, and mount and dismount vehicles. These behaviors, with the exception of being killed, are initiated by commands from the operator via the User Interface. Each behavior may consist of several steps and decision points which are

performed automatically by the Simulator without operator intervention. Investigations are underway that address the issues of automating more complex behaviors and decision making activities.

As indicated earlier, the SAFDI project has two goals. First, the project serves as a demonstration and test application for the IST SAFOR Testbed. Second, the SAFDI project adds to the SIMNET environment a cost effective dismounted infantry component of computer generated and controlled forces. To be useful in the SIMNET environment, the SAFDI entities (fireteams) must behave realistically. This requires modeling human decision making capabilities and command, control, and communication mechanisms. This report documents the current ability to model such behaviors in the SIMNET environment.

1.4 Structure of this Report

Section 2 of this Technical Report describes the functional specification and implemented capabilities of the SAFDI system and its components, as of the report date. No technical knowledge is assumed or required for understanding Section 2.

Section 3 provides technical details of the implementation. It is intended as an more detailed discussion of the design and implementation issues of the system. The reader is assumed to have some knowledge of computer programming and algorithm design.

The Appendices contain supplemental technical detail, and a report of the initial functionality demonstration of the SAFDI system.

Section 2

SAFDI Overview

2. SAFDI Overview

This section provides a overview of the current capabilities (as of the report date) of the IST SAFDI system. The SAFDI system is composed of two major subsystems: the SAFDI Simulator and the User Interface. Section 2.1 describes the SAFDI Simulator. Section 2.2 describes the User Interface.

2.1 SAFDI Simulator Specification

The SAFDI simulation module provides the functional capabilities of SAFDI fireteams in the SIMNET battlefield.

2.1.1 SAFDI Implementation Levels

The implementation plan calls for SAFDI capabilities and commands to be implemented in four levels, each intended to provide a coherent set of functionality. The levels are: scouts, mechanized infantry, smarts, and units.

2.1.1.1 Scouts

SAFDI fireteams appear on the battlefield and can serve as scouts under the direct control of the operator. They are harmless and invulnerable. They serve to demonstrate the ability to create and move SAFDI fireteams in the SIMNET battlefield.

2.1.1.2 Mechanized Infantry

SAFDI fireteams can move, fire weapons, be killed, mount APCs, and dismount from them. They are fully functional but require an operator to initiate and control their behavior. This is the current level of functionality.

Two additional levels of functionality are planned for follow-on phases of this project.

2.1.1.3 Smarts

At this level, route planning, hiding, and limited recon behaviors are added to the SAFDI repertoire as automatic behaviors not requiring operator initiation and control.

2.1.1.4 Units

SAFDI platoons and companies may be defined. Orders given to higher level units are implemented as sets of orders which are passed to appropriate subunits. Orders given to units may be revised or expanded by the Simulator as they are passed to subunits.

2.1.2 SAFDI Capabilities

This section lists SAFDI capabilities and briefly indicates the program activities required for that capability. Note that some capabilities are partially implemented as stated. Follow-on phases of this project will deal with a more complete implementation of them. Unless stated otherwise, the following capabilities are implemented.

2.1.2.1 Basic Capabilities

Colonel L. Mengel (TSM SIMNET, Ft. Knox) listed the following capabilities as the minimum required in any usable SAFDI system.

- Report activity within Line of Sight (LOS).
  Using the SIMNET terrain database and a LOS algorithm developed at IST, each SAFDI fireteam determines which enemy entities can be seen. The Simulator generates sighting reports for sighted entities which are passed to the User Interface and displayed in the User Interface message area (in SALUTE format).

- Communicate with higher headquarters.
  Currently, only communication between the operator and individual fireteams is supported. Commands from the SAFDI operator are routed to the SAFDI fireteam. Messages from the fireteam are routed to the operator. In later implementation levels, commands from the operator to platoon and company levels units will be converted to the appropriate commands to the subunits and communicated to them automatically.

- Kill enemy infantry and vehicles.
  The Simulator selects targets for each fireteam from among the visible enemy entities based on that fireteam's rules of engagement and firing priorities. It calculates firing and impact probabilities, generates fire and impact messages (SIMNET PDUs), and tracks ammunition expenditure.

- Be killed.
  The Simulator receives and processes impact messages sent to individual SAFDI fireteams and assesses damage appropriately. SAFDI fireteams are suffer losses incrementally (i.e. man by man) based on the effectiveness of received fire. Currently only direct fire is considered. The effect of indirect fire is being implemented.

- Mount and dismount APCs.
  SAFDI fireteams mount nearby APCs when given mount commands. The mounted fireteams disappear from view and accompany the APC as it moves. Mounted SAFDI fireteams reappear on the battlefield at the location of the APC when they receive a dismount command.

- Be seen according to visual range and posture.
  Part of a SAFDI fireteam's appearance message is an indicator of the fireteam's posture; different icons are displayed by SIMNET image generators for each posture. Currently, a single icon represents a SAFDI fireteam.

- Change speed
  A SAFDI fireteam's rate of movement is based on its speed setting, the posture of the fireteam, and terrain being covered. An additional factor to consider is the exhaustion level of the fireteam. Currently, exhaustion is not a component of SAFDI rate determination.

2.1.2.2 Capabilities and Commands

The following table displays the capabilities and commands to be implemented at each level. The level 1 and 2 capabilities and commands have been implemented. A capability implemented at one level are available at all higher levels.

Table 1 - Capabilities and Commands

Levels:	1 Scouts	2 Mech Inf	3 Smarts	4 Units
Capabilities				
Report Activity	X			
Communicate	X			
Kill		X		
Be Killed		X		E
Mount and Dismount		X		
Be Seen	X			
Change Speed	X		E	
Troop quality			X	
Visual appearance				X
Commands				
Halt			X	E
Change Speed	X			E
Change Formation				X
Follow Vehicle			X	E
Resume			X	E
Resume All Subordinates				X
Rejoin Unit				X
Change Direction				X
Set Rules of Engagement		X		E
Resupply		X		E
Report Status	X			E
Mount		X		E
Dismount		X		
Change Posture		X		E
Set Firing Priorities		X		E
Select	X			E
Create	X			E
Remove	X			E
Beam	X			E
Reset	X			E
Clear	X			
Load	X			
Save	X			
Go To Avoiding Obstacles	X			E
Go To Using Cover			X	E
Go To Recon			X	E
Go To Hidden Position			X	E
Merge			X	

X: implemented at this level E: Enhanced at this level

2.2. User Interface Specification

This section describes the User Interface component of the IST SAFDI system.

2.2.1 Functional Specification

The User Interface provides the user with the ability to create and control simulated entities in the SIMNET environment. It does not actually control the entities--rather, the user's commands are communicated to the Simulator machine, which executes them. The User Interface machine acts purely as a front end to the Simulator machine. This design was chosen so that the User Interface could be developed somewhat independently of the Simulator, and so that much of the resource-consuming tasks associated with a user interface could be off-loaded from the Simulator, leaving more of its resources available for simulation.

2.2.1.1 Screen

The User Interface screen is divided into four distinct areas (panes): the Map Pane, the Entity Icon Pane, the Message Pane, and the Text Pane. Each pane has a different colored background so it may be identified by color.

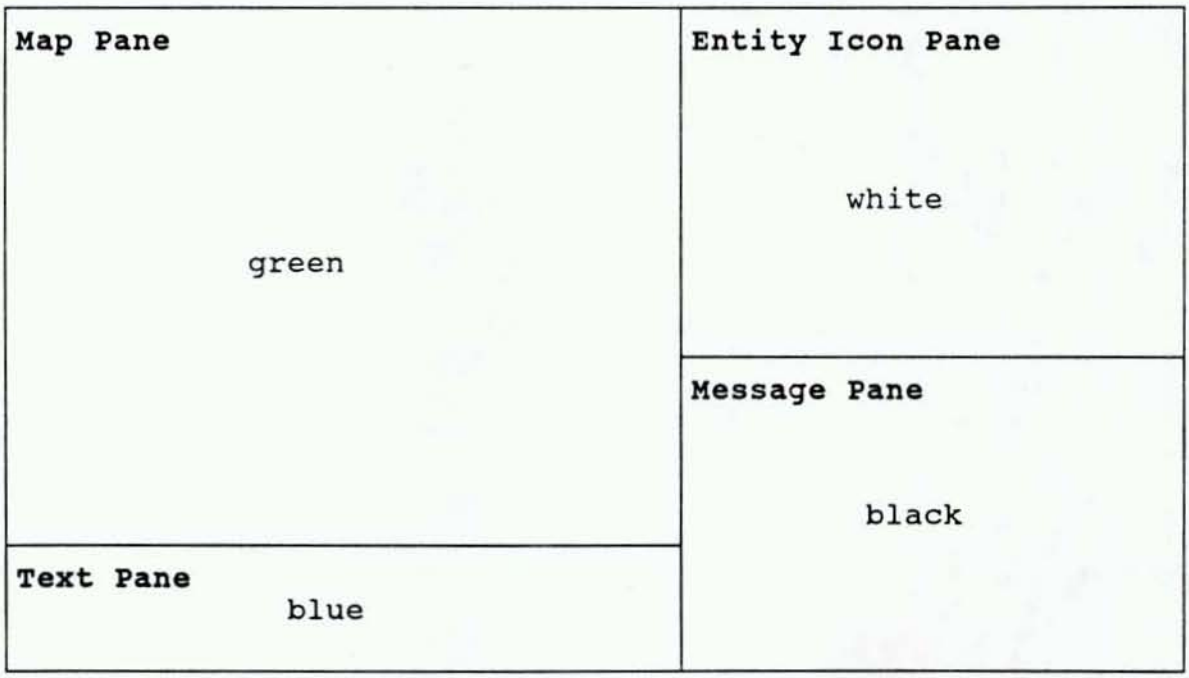


Figure 1 - Screen Layout

2.2.1.1.1 Map Pane

The Map Pane is a SIMNET plan view display showing a portion of the map of the area of operations. It starts in the upper left corner and takes up most of the screen. Its background is a green pattern. The Map Pane displays a rectangular section of the map represented in the SIMNET

terrain database. How large a section is displayed depends on the scale of the display which may be changed by the operator. The section of the map that is being displayed may be changed by the operator so that the entire area of operations may be seen. The Map Pane displays all terrain features and all entities within the area represented by the map. The displayed entities include all the fireteams controlled by the SAFDI system and any other entities on the network. In addition, it may optionally display map gridlines, contour lines and SIMNET database polygon boundaries.

2.2.1.1.2 Entity Icon Pane

The Entity Icon Pane is in the upper right corner of the screen, and has a white background. It displays an icon for every entity created or pending creation. This is for use with the Mouse Interface; to give a entity an order, the user may "click" on its icon. The Entity Icon Pane is divided vertically into two halves--the left half is for entities controlled by the Simulator, and the right half is for entities pending creation. A row of icons occurs at the bottom of the Entity Icon Pane. These icons activate operations concerning the Map Pane and the List Pane and operations controlling the Simulator. These operations are described below in Section 2.2.1.4.2.

2.2.1.1.3 Message Pane

The Message Pane is a generic scrolling information window. It is in the lower right corner of the screen, and has a black background. The Message Pane is primarily for displaying messages from entities to the operator. This is where sighting reports and status reports appear.

2.2.1.1.4 Text Pane

The Text Pane is in the lower left corner of the screen, and has a blue background. The Text Pane is primarily for text interaction with the operator. In this pane appears the text of any command being formed by the user. Also in this pane are prompts for the next input, and certain error messages.

2.2.1.2 Entities

There are three classes of entities within the User Interface: Pending, Controlled and Other. (In the current SAFDI system, these entities may be dismounted infantry fireteams as well as any other vehicle or object type in the SIMNET protocol. The final version of the IST SAFDI system will be limited to dismounted infantry fireteams only.) The Pending and Controlled classes both describe entities controlled by the Simulator which is attached to the User Interface. The Other class describes entities controlled by other simulators. All entities appear on the map with an appropriate background color: blue for U.S., red for USSR, black for unknown

alignment. In addition, the vehicle type of each entity is indicated by a unique symbol.

2.2.1.2.1 Pending Entities

Pending entities are entities which are known only to the User Interface. They are not known to the Simulator or any other device on SIMNET. This class exists in order to allow all elements of a large scenario to be created simultaneously. The user may add or remove entities from the Pending List, and then may create them all simultaneously.

Pending entities appear in the Map with yellow highlighting. Their icons appear on the right side of the Entity Icon Pane, which is the Pending List side. Pending entities may NOT be given orders, since they have not yet been placed in the SIMNET environment.

2.2.1.2.2 Controlled Entities

Controlled entities are entities that are controlled by the Simulator to which the User Interface is attached. Once a Pending entity is created, it becomes a Controlled entity. Controlled entities normally appear on the map with white highlighting, although it may be red if the entity is firing, or grey if the entity is destroyed. Its User Interface name will always appear to the right of the entity's icon on the map. Controlled entity icons appear on the left side of the Entity Icon Pane. Controlled entities MAY be given orders by the User Interface.

2.2.1.2.3 Other Entities

Other entities are entities on SIMNET not under the control of the Simulator to which the User Interface is attached. They appear on the map with the same color scheme as Controlled entities (normally white, red if firing, grey if destroyed), but they can be distinguished from Controlled entities because they do not have names. Other entities do not appear in the Entity Icon Pane, and may NOT be given orders.

2.2.1.3 User Input

There are two main ways for the User to provide input to the User Interface: the keyboard and the mouse. There is a menu system to which the user may provide input by typing the name of a entity or command, or by clicking the mouse on a menu item. A single menu is visible at any one time. The menu contains all the options available to the User at that time. The menu changes as the User selects options. The text and mouse interfaces are integrated so that the user may use either at any time for any part of a command. There are also certain "Immediate Action Keys" which immediately perform functions outside of the structure of the menu system.

To give a entity an order, the user first specifies the name of the entity and then gives it a command. The name can be specified either with the keyboard or by clicking one of the entity's icons with the mouse. The user may click either the entity's icon in the Map Pane or its icon in the Entity Icon Pane.

In addition, there are three other icons representing Map commands, List commands and Simulator controller commands (called Master commands). Each of these command sets can be specified either by name using the keyboard or by clicking its icon with the mouse. The Map commands control the display of the Map. The List commands control the list of pending entities. The Master commands are powerful commands for scenario manipulation which an ordinary user shouldn't have access to; they require a password.

2.2.1.3.1 Keyboard Commands

SAFDI commands may be entered via the keyboard. The commands appear in the Text Pane as they are entered. All of the SAFDI commands have the following format:

<Entity> <Verb> <Parameter1> <Parameter2>...

The following are sample commands:

```
alpha Change Posture Kneeling
bravo GoTo AvoidObs ToLocation 40500 39975
charlie Status Report
delta Change Speed Double
```

Text typed at the keyboard is accepted character by character, and displayed on the screen in the Text Pane. The use of "Hot Keys" (see section 2.2.1.3.3 below) allows the user to specify entire words with individual keypresses. Hot Keys cause entire words to be displayed in the Text Pane. Spaces separate parts of a command and Return/Enter executes a command.

2.2.1.3.2 Mouse Commands

The mouse may be used in place of or in combination with the keyboard for entering commands. The basic mouse operation by the user is to move the pointer on top of the item to be selected or map location and clicking the mouse button. The mouse may be used in three distinct ways:

1. The mouse may be used to identify entities in the Map and Entity Icon Panes by moving the pointer on top of the entity's icon and pressing (clicking) the mouse button.

2. The mouse may be used to identify both points and directions on map by clicking within the Map Pane.

3. The mouse may also be used to identify verbs and options in the pop-up menus.

For example, a command can be fully given with the mouse in 3 steps:

(1) Click a entity in the Map or Entity Icon Panes.
(2) Click a verb from a menu.
(3) Click in the Map Pane to indicate location or direction.

Mouse clicks have different meanings in different states and different locations on the screen. The mouse interface is designed so that its use is quite intuitive--a "point and shoot" approach.

2.2.1.3.3 Hot Keys

A Hot Key for a menu option is a character which when hit will be accepted for that menu option. Each menu option has a hot key defined for it. The hot key for each option is the highlighted character within the option name on the pop-up menu. Use of the hot keys allows rapid command input by reducing the number of keystrokes required to enter a command.

2.2.1.3.4 Immediate Action Keys

The are also certain keys or key combinations which immediately perform functions outside the structure of the menu system. These immediate action keys are accepted and processed immediately. Thus, they can be used in the middle of typing commands. Hot keys include the arrows for moving around the map and ESC for halting the program.

2.2.1.4 User Commands

This section describes in detail the SAFDI command structure.

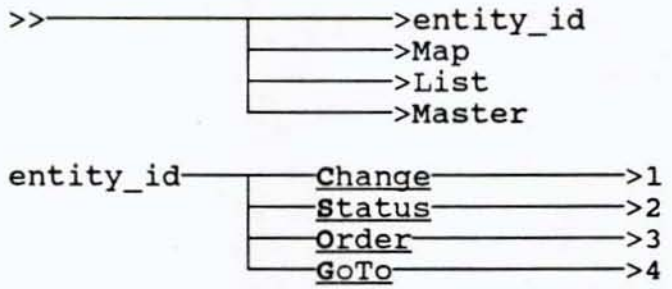
2.2.1.4.1 Command Structure

This section displays graphically the command structure built into the User Interface. The operator builds each command by repeatedly specifying a component of the command. A component can be specified via text input through the keyboard or by selecting options from a menu of options through mouse clicks or hot keys. The menu of options changes as the operator builds the command to reflect what options are available at the current stage of building the command. The User Interface provides context-sensitive submenus and prompts to assist the user in producing a valid command.

Key to Command Structure Diagram

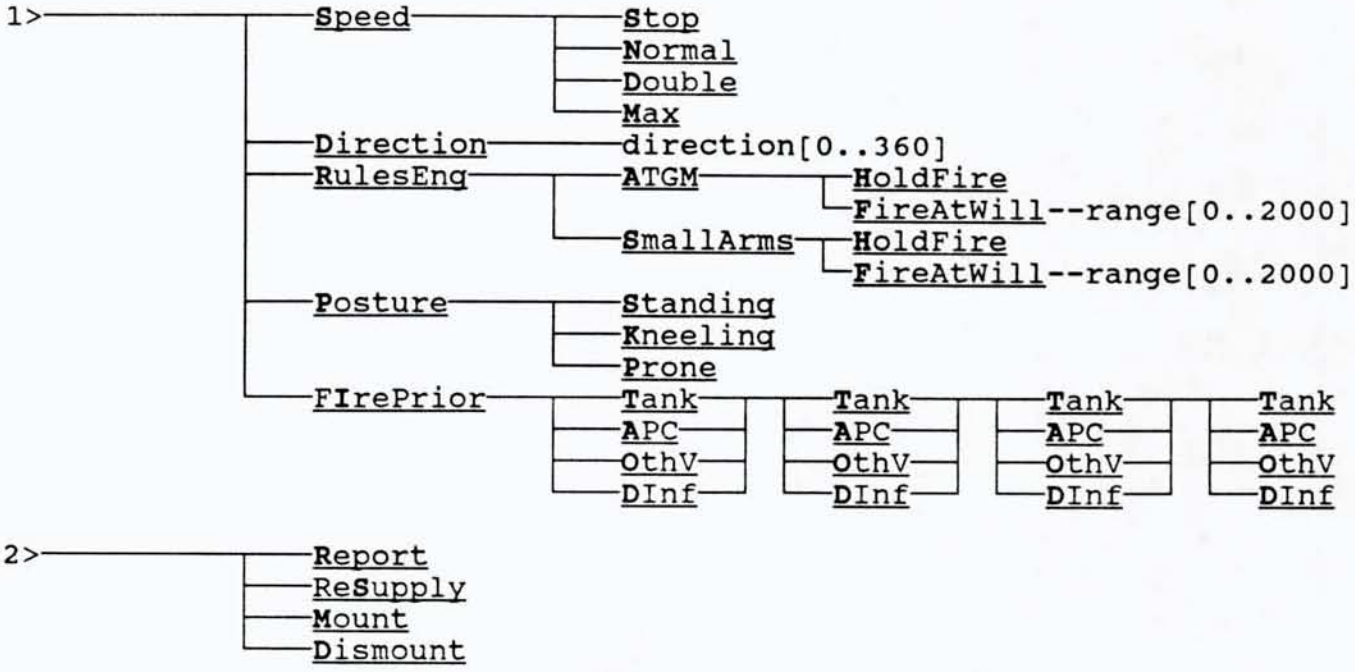
Change	menu option, selected by pressing bold character or mouse clicking menu option
entity_id	parameter
——>1	from-connector; continue this command at to-connector 1
1>——	to-connector; this command continued from from-connector 1
xxx[0..500]	valid values are integers in this range for option "xxx"
(8)	valid values are strings of this length without spaces
XMAX	maximum x coordinate of terrain map
YMAX	maximum y coordinate of terrain map
>>	prompt in Text Pane; ready for user input

Command Structure Diagram



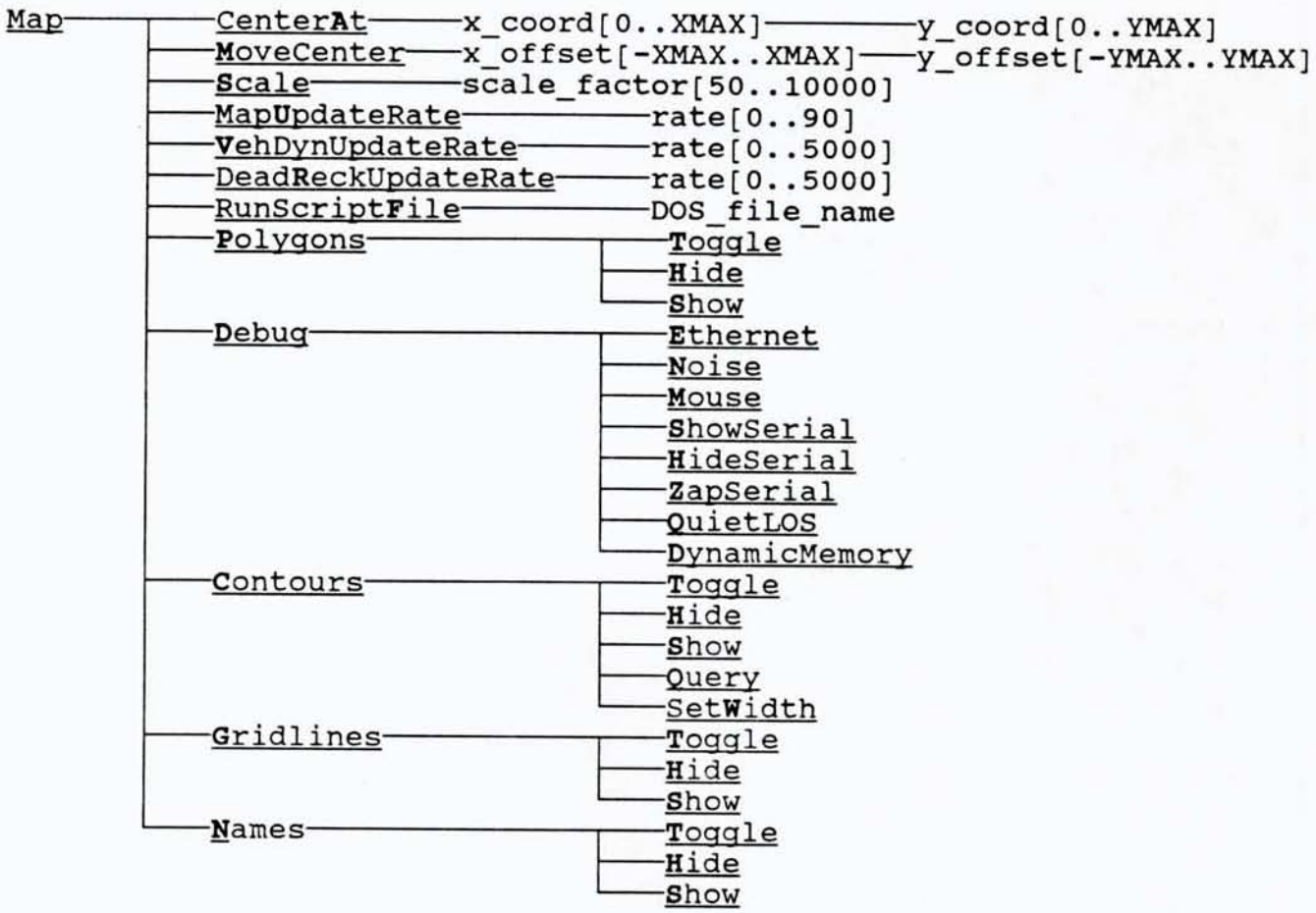
continued on next page

Command Structure Diagram continued



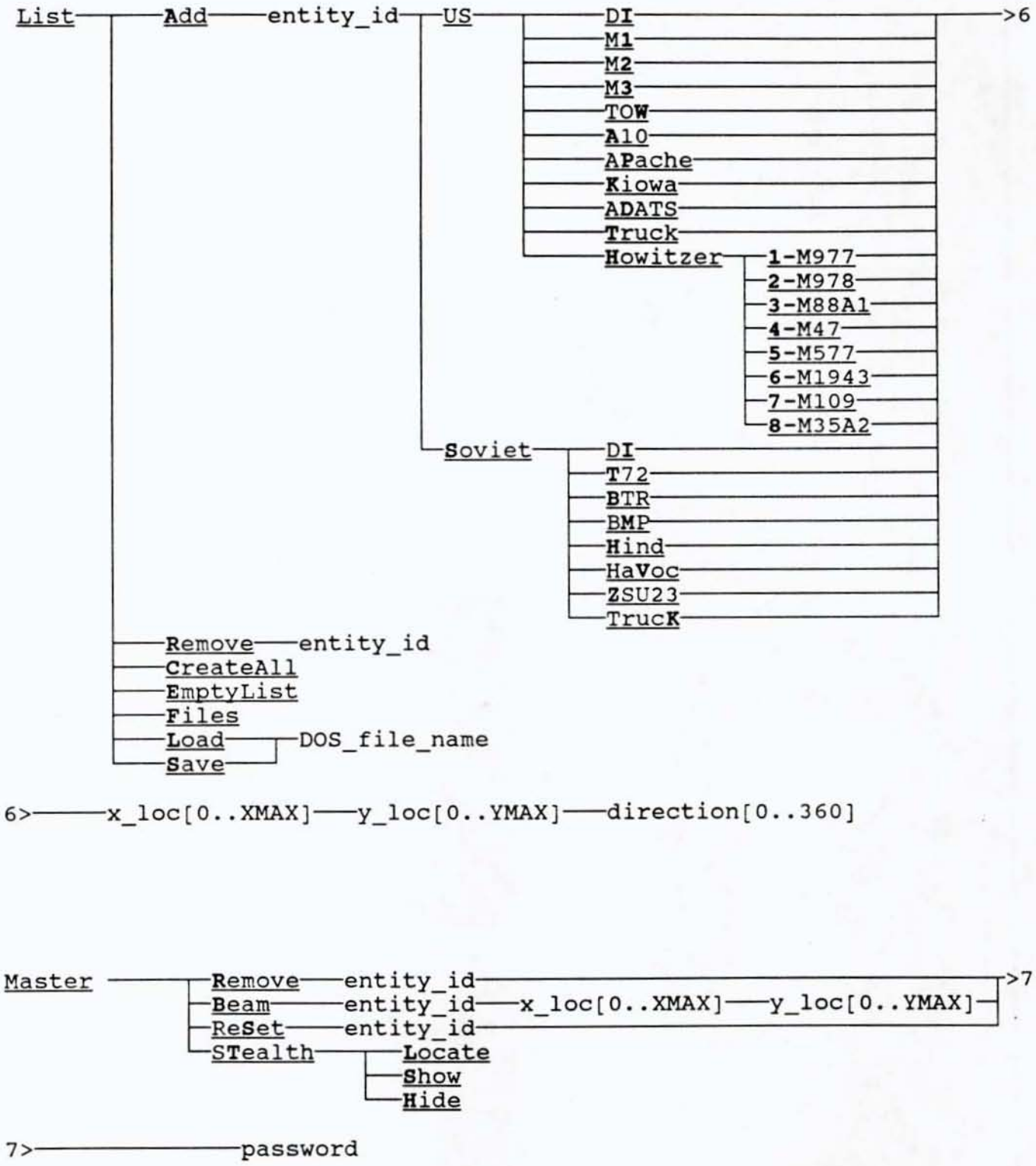
continued on next page

Command Structure Diagram continued



continued on next page

Command Structure Diagram continued



2.2.1.4.2 User Interface Commands

This section describes the User Interface commands and their implementation levels. Each command entry has these items:

UIC: User Interface command
IL: Implementation Level

Command format key:

In UIC: entity_id parameter
Change menu option, selected by pressing highlighted character

The symbols () enclose the elements of a set of options.

The symbol | is used to separate the elements of a set of options, exactly one of which should be chosen.

2.2.1.4.2.1 Entity Change Commands

Change Entity Speed

UIC: entity_id Change Speed (Stop|Normal|Double|Rush)
IL: 1

Sets the entity's speed in meters/second. The speed is found by accessing a 3D look-up table, with axes for speed setting, posture, and terrain. The Simulator should use the speed setting in the command, the posture of the entity identified by entity_id, and the terrain of that entity's current location to set the entity's speed.

Change Entity Direction

UIC: entity_id Change Direction heading[0..360]
IL: 1

Changes the heading of entity_id.

Change Entity Rules of Engagement

UIC: entity_id Change RulesEng (ATGM|SmallArms)
(HoldFire|FireAtWill) range[0..2000]
IL: 2

Sets entity_id's rules of engagement for the weapon selected.

Change Entity Posture

UIC: entity_id Change Posture (Standing|Kneeling|Prone)
IL: 1

Sets the DI fireteam's posture. Certain posture changes are implied by other commands and actions; when the Simulator performs these commands, the posture of the fireteam is also affected.

Change Entity Firing Order
UIC: entity_id Change FirePrior (Tank|APC|OthV|DInf) (Tank|APC|OthV|DInf) (Tank|APC|OthV|DInf) (Tank|APC|OthV|DInf)
IL: 2

Set entity_id's target priorities. A target type can appear only once in the list of priorities.

2.2.1.4.2.2 Entity Status Commands

Entity Status Report
UIC: entity_id Status Report
IL: 1

The Simulator returns to the User Interface a message containing all of the SAFDI fireteam attributes for entity entity_id that are not available in an Appearance PDU. These attribues include Posture, Strength, ATGM rounds, Small arms ammo, and Exhaustion level.

Entity Status Resupply
UIC: entity_id Status Resupply
IL: 2

Resupplies entity_id, if a supply source is available.

Entity Status Mount
UIC: entity_id Status Mount
IL: 2

Causes Simulator to mount fireteam entity_id in vehicle provided that fireteam is within legal mounting distance and the vehicle allows DI fireteams to mount them.

Entity Status Dismount
UIC: entity_id Status Dismount
IL: 2

Causes Simulator to dismount fireteam from vehicle.

2.2.1.4.2.3 Entity Order Commands

No entity order commands have been implemented at this stage of the project. The development plan calls for the implementation of entity commands in follow-on phases. Typical unit commands are: Halt movement, Resume movement, Follow vehicle, Separate subunit from unit, and Rejoin subunit to unit.

2.2.1.4.2.4 Entity GoTo Command

Entity Order Movement

UIC: entity_id GoTo x_loc[0..XMAX]
y_loc[0..YMAX]

IL: 1

Determines the route of travel using the current testbed route planning algorithm, with the following restrictions:

1. Buildings and non-fordable water are obstacles.
2. Treelines and fordable rivers are not obstacles.

2.2.1.4.2.5 Map Commands

Set Map Center

UIC: Map CenterAt x_coord[0..XMAX]
y_coord[0..YMAX]

IL: 1

Centers the Map display at (x_coord, y_coord). The unit of measure of x_coord and y_coord is meters.

Map MoveCenter

UIC: Map MoveCenter x_offset[-XMAX..XMAX]
y_offset[-YMAX..YMAX]

IL: 1

Moves the Map center x_offset horizontally and y_offset vertically. The unit of measure of x_offset and y_offset is meters.

Map Scale

UIC: Map Scale scale_factor[50..10000]

IL: 1

Sets the Map display scale to scale_factor. The unit of measure of scale_factor is meters.

Map Update Rate

UIC: Map MapUpdateRate rate[0..90]

IL: 1

Sets the Map display rate. The unit of measure of rate is IBM timer ticks which is 1/18.204 seconds or approximately .055 milliseconds.

Set Vehicle Dynamic Update Rate

UIC: Map VehDynUpdateRate rate[0..5000]

IL: 1

Sets the dynamic entity appearance update rate. The unit of measure of rate is IBM timer ticks which is 1/18.204 seconds or approximately .055 milliseconds.

Set Dead Reckoning Update Rate
UIC: Map DeadReckUpdateRate rate[0..5000]
IL: 1

Set the dead reckoning update rate. The unit of measure of rate is IBM timer ticks which is 1/18.204 seconds or approximately .055 milliseconds.

Run Script File
UIC: Map RunScriptFile DOS_file_name
IL: 1

Runs the sequence of User Interface commands stored in the file DOS_file_name.

Map Polygons
UIC: Map Polygons (Toggle|Hide|Show)
IL: 1

Controls the display of terrain polygons in the Map Pane. The Toggle option changes the current setting. The Hide option turns the display of terrain polygons off. The Show option turns the display of terrain polygons on.

Programmers' Tools
UIC: Map Debug (Ethernet|Noise|Mouse|ShowSerial|
HideSerial|ZapSerial|QuietLOS|
DynamicMemory)
IL: 1

Activates various programmers' debugging tools. This option will be removed in delivered systems.

Map Contours
UIC: Map Contours (Toggle|Hide|Show|Query|SetWidth)
IL: 1

Controls the display of elevation contours in the Map Pane. The Toggle option changes the current setting. The Hide option turns the display of elevation contours off. The Show option turns the display of elevation contours on. The Query option displays the current setting of the elevation scale. The SetWidth option sets the elevation scale which is the difference in elevation between contour lines in the map display.

Map Guidelines
UIC: Map Guidelines (Toggle|Hide|Show)
IL: 1

Controls the display of map 1 km gridlines in the Map Pane. The Toggle option changes the current setting. The Hide option turns the display of guidelines off. The Show option turns the display of guidelines on.

Map Names
UIC: Map Names (Toggle|Hide|Show)
IL: 1

Controls the display of names of Other SIMNET entities in the Map Pane. The Toggle option changes the current setting. The Hide option turns the display of names off. The Show option turns the display of names on.

2.2.1.4.2.6 List Commands

Add Entity
UIC: List Add entity_id(8) (US (DI|M1|...) | Soviet (DI|T72|...))
IL: 1

Adds an entity of the type selected to the Pending List.

Remove Entity
UIC: List Remove entity_id
IL: 1

Removes a single entity from a Entity list.

Create All Entities
UIC: List Create
IL: 1

The Entities on the Pending List are moved to the Controlled List. Causes the Simulator to create all the entities on the Pending List.

Remove Entire List
UIC: List EmptyList
IL: 1

Removes all entities from a Entity list.

List Files
UIC: List Files DOS_file_path
IL: 1

Displays the names of the Entity Lists (files with a ".lst" extension) saved to the disk under the subdirectory DOS_file_path.

Load Entity List
UIC: List Load DOS_file_name
IL: 1

Loads a Entity list from a disk file.

Save Entity List
UIC: List Save DOS_file_name
IL: 1

Save a Entity list to a disk file

2.2.1.4.2.7 Master Commands

Remove entity
UIC: Master Remove entity_id password
IL: 1

Removes entity from SIMNET network.

Change entity location
UIC: Master Beam entity_id x_loc[0..XMAX] y_loc[0..YMAX] password
IL: 1

The Simulator resets the x and y coordinates for entity entity_id to those given in the command. The entity's z value is reset to match the current Above Ground Level value. Any current GoTo command is cancelled. No other attributes, including speed, and direction are affected.

Reset entity attributes
UIC: Master Reset entity_id password
IL: 1

The attributes of entity entity_id are set to their respective reset values given in section 3.1.2.

Stealth commands
UIC: Master Stealth (Locate|Show|Hide)
IL: 1

The Locate option returns the current location of the Stealth unit. The Show option displays a Stealth icon in the Map Pane indicating the Stealth's heading. The Hide option turns off the display of the Stealth icon.

Section 3

SAFDI Technical Description

3. SAFDI Technical Description

This section describes in technical detail the SAFDI and User Interface Components.

3.1. SAFDI Implementation

This section describes in detail the SAFDI simulator component.

3.1.1 Definitions

The following terms are used in this section. They are defined here for reference.

vehicle_number:
Integer in the range [0..n], assigned sequentially to entities created by the Simulator. Applies to DI fireteams as well as actual vehicles.

vehicle_id:
SIMNET packet identifier consisting of three integers: site, host, and vehicle. Applies to DI fireteams as well as actual vehicles.

entity_id:
User Interface entity name, assigned by SAFDI operator. Consists of up to eight arbitrary characters, with spaces not allowed.

3.1.2 SAFDI fireteam attributes

SAFDI fireteams have the following attributes, or values which may change over the course of a simulation:

Attribute	Type and range	Initial/reset value
x position	Float[0..XMAX]	Given
y position	Float[0..YMAX]	Given
z position	Float[0..ZMAX]	Calculated
Roll	Float[0]	Constant
Pitch	Float[0]	Constant
Yaw	Float[0..2π]	Given
Posture	Char{s,k,p}	Given
Strength (# of men)	Integer[1..5]	5
ATGM rounds (Dragon)	Integer[0..2]	2
Small arms ammo (%)	Integer[0..100]	100
Exhaustion level (%)	Integer[0..100]	100

3.1.3 Sighting and Fireteam Destruction Reports

In addition to the messages from the Simulator to the User Interface produced in response to the User Interface commands, there are circumstances in which the Simulator may initiate a message. They are:

Sighting report:
When a SAFDI fireteam sights an enemy entity, the Simulator passes the information necessary to produce a sighting report. (See Section 3.1.3.2 for a discussion of how sightings are

determined.) The User Interface produces the sighting report in the message pane in SALUTE format (see Appendix A.1).

Fireteam destruction:
When a SAFDI fireteam is hit by fire and destroyed, that fireteam must be removed from the SAFDI operator's control. The Simulator must pass a message to the User Interface indicating the destruction of a fireteam and identifying the fireteam with the vehicle_number.

The reporting of sighted hostile entities is a critical SAFDI function. The procedure for determining sightings is described in the following sections.

3.1.3.1 Terrain Database

The SAFDI Line of Sight (LOS) algorithm determines point-to-point LOS using the SIMNET terrain database. The LOS calculation involves deciding whether the line between two points is blocked by a feature of the terrain database. For the purposes of the SAFDI system, we chose to consider three main features which can block the LOS: elevation of the terrain (e.g. mountains, hills, and valleys), treelines, and canopies.

The terrain database is organized into blocks which represent 500 meter x 500 meter squares called patches. Each patch is composed of 16 squares called grids. Both patches and grids can be referenced by computing indices based on the world coordinates of a point inside them.

Terrain is represented in the database as vertices, edges, and polygons in the following manner. Each vertex is an entry in a point array; the point array holds the coordinates of all vertices. Each edge is an entry in an edge array; the edge array holds the indices into the vertex array of the edges' vertices. Each polygon is an entry in a polygon array; the polygon array holds the indices into the edge array of the polygons' edges.

The elevation of the terrain is represented in the database by polygons; the three dimensional vertices of each polygon are used to compute the height of any point within the polygon. Polygons are organized by their patch and grid locations. Each edge structure has a code that tells which grids within the patch contain that particular edge. Therefore, to check whether the LOS is blocked by the elevation of the terrain, one must determine if the LOS intersects a polygon along the LOS. The check for intersection is done by determining if the LOS has a lower terrain height than the terrain height of any edge the LOS crosses at the point the LOS crosses the edge .

The way to find all points of intersection of a line with a polygon is to find the points of intersection, if any, of the line with each edge of the polygon. Because the terrain

database contains a list of edges of polygons indexed by grid location, it is not necessary to make reference to the list of polygons in order to compute LOS. Instead, we compute the intersections of all edges of polygons which are within the grids containing the LOS with the LOS. This removes a level of indirection and speeds the process of determining LOS.

3.1.3.2 General Line of Sight

The first task in determining Line of Sight (LOS) without the SIMNET environment is to determine which patches and grids to search for the possible polygon, treeline, and canopy intersections with the LOS. In the SAFDI system, the patch and grid indices are computed using Bresenham's algorithm (see [Foley,1982]) in the following way: First, the patch and grid indices are computed for the endpoints of the LOS. Then, Bresenham's algorithm is used to determine all (patch, grid) pairs which lie along the LOS. The result is a list of (patch, grid) pairs which is then used to determine intersections with the LOS.

3.1.3.2.1 Intersecting LOS with land polygons

The next step in determining LOS is to test whether the elevation of the terrain blocks the LOS. This is accomplished by testing each of the edges within the current patch and grid (the line intersection algorithm used is listed in the Section 3.1.3.3.2). The line intersection is computed using (x,y) coordinates only. Then, the z coordinates are calculated for the edge of the polygon and the LOS, respectively, at that point. These z coordinates determine whether the LOS is blocked. If the z value for the polygon edge is higher than the z value for the LOS, the LOS function immediately decides that the LOS is blocked, and returns to its caller without further processing.

3.1.3.2.2 Intersecting LOS with treelines

After the land polygons are checked, the treelines which are within the (patch, grid) pairs are checked to see whether they block the LOS. Treelines are represented by a linked list of three dimensional coordinates; the treeline extends between these points. Further, the treelines have a constant height above the ground. The approach used to determine whether the treelines block the LOS involves first computing the (x,y) intersection point of each treeline edge with the LOS. The height of this point in the treeline is computed as the height of the treeline plus the height of the terrain. This is compared to the height of the point along the LOS to determine whether the LOS is blocked. If the LOS is blocked, the LOS function returns to its caller without further processing.

3.1.3.2.3 Intersecting LOS with canopies

Canopies are represented in the terrain database as a combination of a treeline (the border) and polygons (the interior). So, the LOS test for a canopy involves combining

the steps outlined above for land polygons and treelines. Canopies are not organized by grids, just by patches. Therefore, all canopies located within a patch through which the LOS passes are checked.

3.1.3.3 SAFDI Line of Sight

The algorithm detailed above provides a method of determining the LOS between two points. In the SAFDI system, we need to determine the LOS between several entities. This is done by a function which provides the interface between the simulator and the LOS function.

There were two goals for the LOS calculation of the SAFDI system: efficient execution time and avoiding repeated sighting reports. An important goal of any LOS algorithm is to minimize the amount of execution time required. One way to accomplish this is to carefully monitor the number of times the function for determining LOS is called. In reality, sighting reports are only made once for a particular vehicle while that vehicle remains in sight. So, another goal of the simulation is to avoid repeated sighting reports.

Both of these goals are achieved in SAFDI using the concept of a sightings list. Each entity maintains a linked list of the other entities in the SIMNET battlefield which it has previously sighted. The algorithm detailed below takes advantage of this list in order to use a minimum amount of time in the LOS function and to avoid repeated sighting reports.

As a final note, each entity has an individual LOS update rate, which can be modified by the user. This controls the number of times that an update of the LOS list is performed on a particular entity.

3.1.3.3.1 Line of Sight Algorithm

The Line of Sight algorithm is as follows:

```
for each SAFDI fireteam i
    for each entity y in Si (current sightings list for i)
        calculate LOS from i to y;
        if LOS blocked & not seen for 60 seconds
            remove y from Si
        endif
    endfor

    X := set of entities Movable Objects Mgr is tracking
         using its dead reckoning model;
    X := X - all entities in X more than 2000 meters from i;
    X := X - all entities in X friendly to i;
    X := X - all entities in X that are destroyed;
    X := X - Si, the current sightings list for i
```

```
    for each entity x in X
        calculate LOS from i to x;
        if  LOS not blocked
            add x to S_i
            produce sighting message
        endif
    endfor
endfor
```

The above algorithm makes use of the LOS check algorithm to update the sightings list for a particular entity. There are two main loops.

In the first loop, the current sightings list is checked for entities which can no longer be seen. Entities which have not been seen for 60 seconds or more are removed from the current sightings list. However, entities which have been sighted in the last 60 seconds but are now unsighted are kept in the sightings list and marked as "unsighted-recently". This ensures that a new sighting report will not be generated for an entity which temporarily passes behind an object. Finally, the "unsighted-recently" mark is removed from those entities which were previously marked as unsighted (although not removed from the list) but are sighted on this update. In other words, they disappeared and then reappeared within 60 seconds, so the algorithm does not issue another sighting report.

In the second loop, all entities in the current simulation are placed in a list if they are within the range of vision of the entity doing the sighting. This range is taken to be 2000 meters in the SAFDI system. This list is then subjected to several filters. First, all entities are removed from this list which are friendly to the sighting entity (i.e. from the same country). Next, all entities are removed from the list which have been destroyed. Finally, all entities are removed from this list which are already in the sightings list. The LOS is computed for each entity remaining in this list, and the sightings list is updated appropriately. If a entity is sighted in this step, a sighting report is generated and sent to the user interface.

3.1.3.3.2 Line Intersection Algorithm

This algorithm determines whether the line segments l and m (defined by coordinates (lx1, ly1) --> (lx2, ly2) and (mx1, my1) --> (mx2, my2)) intersect. If so, the intersection point can be determined based on the values found as the result of computing parameters t1 and t2, as well as determinant D. If the lines are coincident, this fact is returned to be dealt with by the caller.

The Line of Sight Algorithm is as follows:

Method: Parameterize the line segments as follows:

```
l:  <x, y> = <lx1, ly1> + t1*<ldx, ldy>
m:  <x, y> = <mx1, my1> + t2*<mdx, mdy>

    where ldx = lx2-lx1,
          ldy = ly2-ly1,
          mdx = mx2-mx1,
          mdy = my2-my1,
          0 <= t1 <= 1,
      and 0 <= t2 <= 1.
```

Rewrite these equations into 2 equations with (t1, t2) as the solution by setting the x's and y's equal:

```
ldx*t1 - mdx*t2 = mx1-lx1
ldy*t1 - mdy*t2 = my1-ly1
```

Using Cramer's rule, define D, t1, and t2 as follows:

$$D = \begin{vmatrix} ldx & -mdx \\ ldy & -mdy \end{vmatrix}$$

$$t1 = \begin{vmatrix} dx & -mdx \\ dy & -mdy \end{vmatrix}$$

$$t2 = \begin{vmatrix} ldx & dx \\ ldy & dy \end{vmatrix}$$

where dx = mx1-lx1,
and dy = my1-ly1.

Then, the two line segments intersect if

```
0 <= t1 <= D and 0 <= t2 <= D  (if D > 0)
D <= t1 <= 0 and D <= t2 <= 0  (if D < 0)
```

The point of intersection can be calculated by using the original equations for l and m. However, it should be noted that the t1 and t2 values calculated above need to be scaled by D (t1 = t1/D and t2 = t2/D) in order to use them in the equations for l and/or m.

One final note: the lines are parallel if D = 0. They are coincident if D = 0 and t1 = 0 (t2 will also be 0).

3.1.4 Implied Posture Changes

The posture of a SAFDI fireteam may be changed explicitly by a Change Posture order from the SAFDI operator. In addition to an explicit order, certain simulation events cause a SAFDI fireteam to change its posture. A SAFDI fireteam must be kneeling to fire an ATGM. If a fireteam is ordered to fire an ATGM while not kneeling, the fireteam will assume a kneeling position during the entire fire ATGM sequence (see Section 3.1.5.1 below) and resume its initial posture at the end of the firing sequence. SAFDI fireteams assume a standing posture when mounting and dismounting vehicles.

3.1.5 SAFDI Combat

3.1.5.1 ATGM Firing

There are five phases in ATGM firing. First, the SAFDI fireteam must load the ATGM. Loading requires 60 seconds if the ATGM is not already loaded. Second, a target must be acquired; Section 3.1.3 discusses target sighting. Third, the SAFDI unit must stop movement and kneel. The time required to stop movement is dependent on the rate of travel. Fourth, the ATGM is fired if the target is still sighted. Firing an ATGM causes the creation of a missile entity within the Simulator. The flight dynamics of the missile entity are managed by the Simulator independent of the SAFDI fireteam firing it. Finally, a phase corresponding to controlling the ATGM during flight is entered. Firing and guiding an ATGM requires 10 seconds. The SAFDI fireteam may now return to the first phase; ie. loading an ATGM. If during phases 1 through 4 the SAFDI fireteam is destroyed, the ATGM firing process is stopped. If the SAFDI fireteam is destroyed after the ATGM is fired but before the ATGM strikes its target, the missile will not strike its target (no impact PDU will be generated).

3.1.5.2 Small arms firing

The basic procedure for resolving SAFDI small arms fire follows the SIMNET standard technique. In that technique, the firing entity determines whether a hit is scored and sends an impact PDU to the target entity. The target entity analyzes the impact PDU and determines what damage it has suffered. Thus, the firing SAFDI fireteam generates an impact PDU for every executed small arms attack; that PDU specifies the firepower intensity of the attack. The target SAFDI fireteam uses the firepower value in the PDU to assess casualties, which are taken incrementally. A single small arms attack may cause anywhere from zero to five casualties in the target fireteam.

When resolving small arms fire, a number of factors are considered. For the firing fireteam, they are:

- strength (# men) and weapons of firing fireteam
- range to target
- simultaneous firing/loading of ATGM
- movement of firing fireteam
- "surprise" or prepared fire

These factors, taken together, are used to produce the firepower value for the small arms attack.

For the target fireteam, the factors considered are:

- firepower value of attack
- posture of target fireteam
- strength (# men) of target fireteam

As a fireteam takes casualties, the weapons available to it decreases. The following table shows the weapons assumed to be present in a fireteam at each strength level.

Table 2 - Fireteam weapons availability

Fireteam strength (# men)	Weapons in fireteam
5	ATGM, SAW, M-16+M203, 3x M-16
4	ATGM, SAW, M-16+M203, 2x M-16
3	ATGM, SAW, M-16+M203, M-16
2	ATGM, SAW, M-16
1	ATGM, M-16

Note that the ATGM gunner is also armed with a M-16. The firepower of that M-16 is not counted in the fireteam's small arms firepower if the ATGM gunner is firing or reloading his ATGM weapon. See the following tables.

The relative firepower for each of the fireteam's small arms weapons is given in the next table.

Table 3 - Weapon firepower

Weapon	Range ≤ (meters) 50	100	200	300	400	500	750	1000
M-16	3	2	2	1	0	0	0	0
M-16+M203	6	4	2	1	0	0	0	0
SAW	10	8	6	5	4	3	2	1

The cumulative firepower of a fireteam is found by summing the individual firepower of the weapons available. The availability of weapons depends on the fireteam's strength and whether or not the ATGM gunner is participating in the small arms attack.

Table 3 - Fireteam cumulative firepower

Fireteam strength (# men)	Range ≤ (meters) 50	100	200	300	400	500	750	1000
5	25	18	14	9	4	3	2	1
5 - ATGM gunner	22	16	12	8	4	3	2	1
4	22	16	12	8	4	3	2	1
4 - ATGM gunner	19	14	10	7	4	3	2	1
3	19	14	10	7	4	3	2	1
3 - ATGM gunner	16	12	8	6	4	3	2	1
2	13	10	8	6	4	3	2	1
2 - ATGM gunner	10	8	6	5	4	3	2	1
1	3	2	2	1	0	0	0	0
1 - ATGM gunner	0	0	0	0	0	0	0	0

Small arms firing procedure

1. Based on the firing fireteam's strength, whether or not the ATGM gunner is in the process of firing an ATGM (see Section 3.1.5.1 above), and the range to the target, use the above table to get small arms attack firepower F.

2. Generate impact PDU with ImpactVariant.burst.quantity set to F. Generate a fire PDU.

3. Decrement fireteam small arms ammunition percentage by a constant amount that reflects the percentage of ammunition expended.

3.1.5.3 Receiving small arms fire

For a fireteam that is the target of a small arms attack, the probability of each man in the fireteam being "attrited" (killed or incapacitated) is given in the following table.

Table 5 - Base P_k per man in target fireteam

Firepower of attack	Base P_k per man
30	.95
29	.94
28	.93
27	.92
26	.91
25	.90
24	.88
23	.86
22	.84
21	.82
20	.80
19	.78
18	.76
17	.74
16	.72
15	.70

Firepower of attack	Base P_k per man
14	.66
13	.62
12	.58
11	.54
10	.50
9	.46
8	.42
7	.38
6	.34
5	.30
4	.24
3	.18
2	.12
1	.06
0	.00

Small arms target procedure

1. Depending on the firepower value in the impact PDU, get base P_k from the table 5.

2. Set $P_k = P_k + (.02 * (\text{size of target fireteam} - 3))$.

3. If the target fireteam is kneeling, set $P_k = P_k * .5$.
   If the target fireteam is prone, set $P_k = P_k * .35$.

4. For each man in the target fireteam
   r = random number, $0 \leq r < 1$
   if $r \leq P_k$, decrement size of target fireteam.

3.1.5.4 Receiving other direct fire

Currently, SAFDI fireteams are not susceptible to fire from weapons other than small arms. A follow-on task in this project is to add susceptibility of SAFDI fireteams to other weapons. The following discusses the planned approach.

When a fireteam is attacked by direct fire from a weapon other than small arms (e.g. a M1 main gun, or a M2 automatic cannon), the resolution procedure is nearly the same as when receiving small arms fire. After deriving a firepower strength based on weapon type and range (see Table 6 below), the same P_k table and procedure is used.

Table 6 - Munition firepower

Vehicle and munition type	Range ≤ (meters) 250	500	750	1000	2000	3000
M1 munition_US_M392A2	16	16	16	16	16	12
munition_US_M456A1	16	16	16	16	16	12
M2 munition_US_M791	20	20	16	16	12	8
munition_US_M792	20	20	16	16	12	8
munition_US_M789	20	20	16	16	12	8
T72 munition_USSR_125HEAT	15	15	15	15	12	8
munition_USSR_125SABOT	15	15	15	15	12	8
BMP munition_USSR_73HEAT	16	16	10	10	4	0
munition_USSR_30SABOT	16	16	10	10	4	0
All others	8	4	2	1	0	0

The munition types in this table are defined in file mun_type.h. The munition type value will be found in the Impact PDU in field burst.projectile.

Other direct fire target procedure

1. Depending on the munition type and range in the Impact PDU, get the firepower value from the Munition Firepower table.

2. Using the firepower value derived, get the base P_k from the base P_k table in the previous section.

3. Follow the procedure in the previous section exactly, beginning with step 2.

3.1.6 Ammunition Resupply

SAFDI fireteams expend ATGM and small arms ammunition when firing, and may not fire if sufficient ammunition of the appropriate type is not available. SAFDI fireteams may resupply their ammunition to the initial levels (2 ATGMs and 100% small arms) under either of the following two conditions:

1. The SAFDI fireteam mounts a friendly APC. This resupply action is automatic, and need not be ordered by the SAFDI operator.

2. The SAFDI operator issues a Resupply command. In the next phase of the project, the distance from the fireteam to a friendly APC will determine if the resupply command can be executed. An additional restriction will be that the DI fireteam may not perform any actions other than LOS checks during the resupply period, which will be 2 minutes.

3.1.7 Finite State Machine Transition Diagrams

SAFDI behaviors are implemented within the Simulator as Finite State Machines (FSM). The following sections show the Finite State Machine diagrams for ATGM Firing, Load Small Arms, and the Mount and Dismount behaviors.

3.1.7.1 ATGM Firing

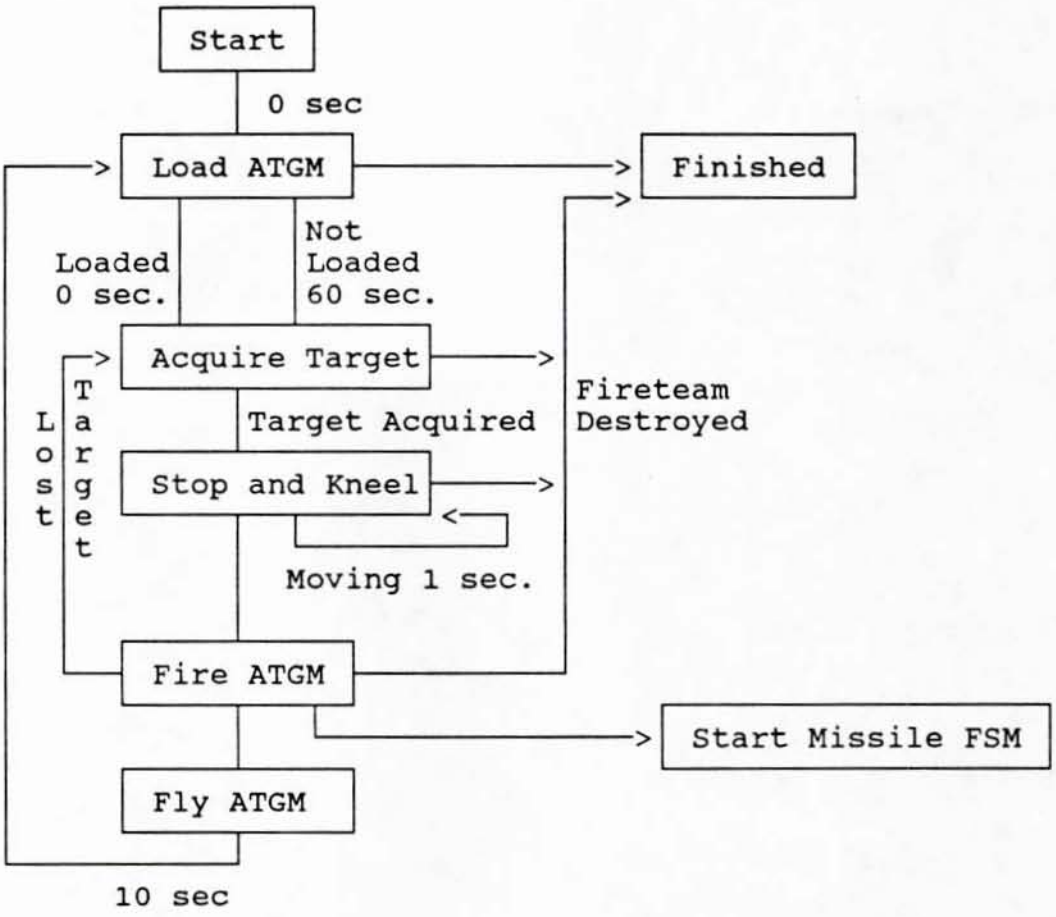


Figure 2 - ATGM Firing

3.1.7.2 Load Small Arms

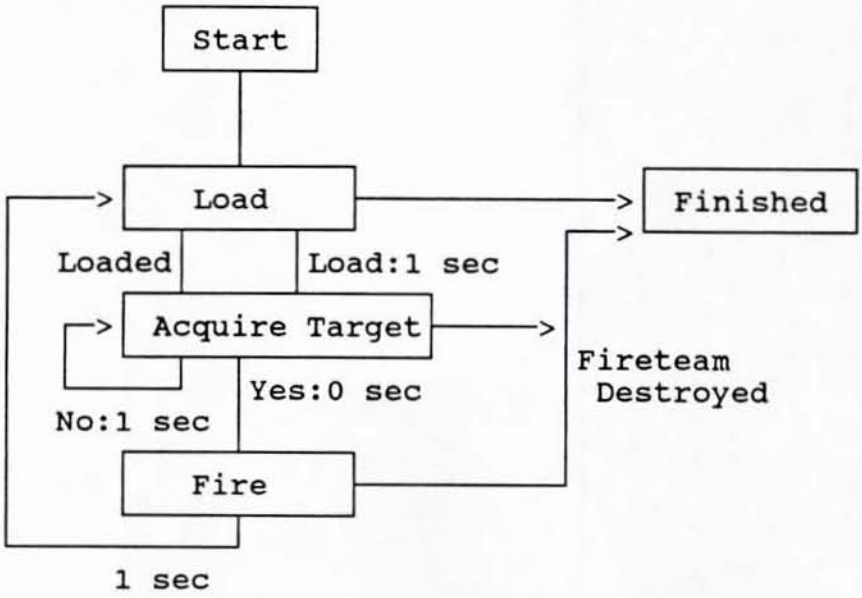


Figure 3 - Load Small Arms

3.1.7.3 Mount and Dismount

Mount:

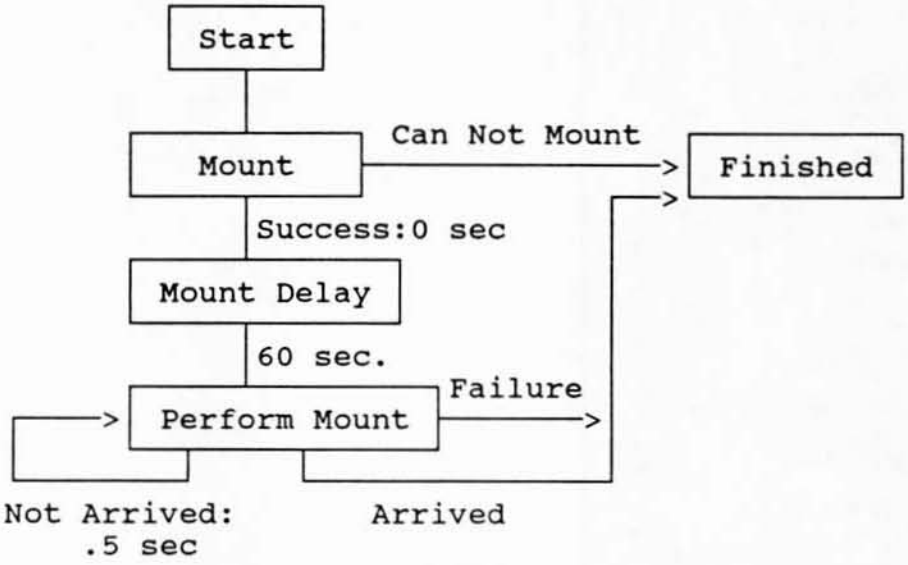


Figure 4 - Mount

Dismount:

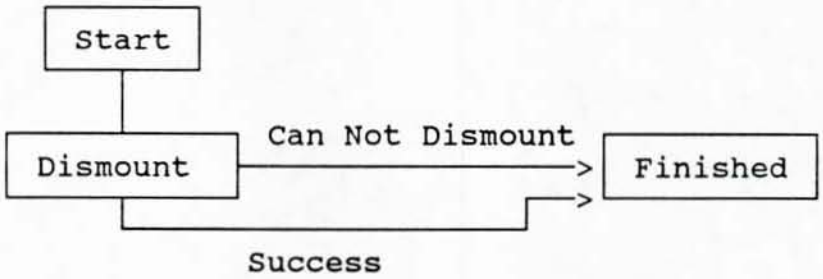


Figure 5 - Dismount

3.2 User Interface Specification

The SAFDI User Interface provides a user friendly interface to the operator who controls the SAFDI fireteams.

The User Interface uses SIMNET terrain databases to produce maps in the Map Pane. The SIMNET terrain databases are approximately 30 Megabytes each. The User Interface can be operated without a terrain database, but it would have no map in the Map Pane. Thus, the User Interface machine must realistically have a large capacity hard drive to accommodate the SIMNET terrain databases.

3.2.1 Data Structures

The following is a description of the major data structures involved in the major functions of the User Interface. C code is included wherever necessary.

Note: In the following technical discussion, the word "unit" refers to a SIMNET entity. For example, "UnitStruct" in the next section actually refers to an entity.

3.2.1.1 Entity

The following structure holds the information for any entity on the battlefield:

```
struct UnitStruct
{
  int ID, siteid, hostid, vehicleid;    //  SimNet ID
  double x, y;                          //  (x,y) position
  double vx, vy              //  (vx,vy) velocity
  double width;              //  width of object in meters
  int screenx, screeny,      //  Position of object on
       screenwidth;          //     map section of screen
  int class;                 //  Object's class
  int heading;               //  Object's heading 0-360
  int alignment;             //  US or SOVIET
  int needsredrawing;        //  True if entity has changed
                             //     & needs to be redrawn
  int deactivate;            //  True if entity is gone away
  int pending;               //  True if entity is pending
  int controlled;            //  True if entity controlled
  int left, top;             //  ScreenPos of bkgnd bitmap
  int namedrawn;             //  True if name was redrawn
  int redrawlevel;           //  Used in redrawing alg
  int mounted;               //  True if DI mounted on M2
  int destroyed;             //  True if entity is destroyed
  int msiteid, mhostid, mvehicleid; //  SimNet ID of M2
  int turretAzimuth;                //  TurretAngle 0-360
  unsigned long oldtime;            //  Last update time
  void *background;                 //  Background bitmap
  char *name, *displayname;  //  Hash & Display names
  char *type, code;          //  More type info
  int fireflag;              //  T if entity fired
                             //    since last update
};
```

3.2.1.2 Hash Table and Functions

The entities are put into a hash table using the following hash function on their name:

```
Hash(name) = [ sum of characters of name ] & 0xFF
```

Collisions are resolved with external chaining; that is, all entities for which Hash(name) = H are put in a linked list pointed to by HashTable[H]. If the entity was created

locally, then its name is given to it when it was created. If the entity was not created locally, then its name is built from its SIMNET ID as follows:

```
sprintf( name, "v%d.%d.%d", siteid, hostid, vehicle);
```

All access to entities must come through the Hash Table. The following functions manipulate the hash table:

```
struct UnitStruct *LocateUnit( char *unitname );

struct UnitStruct *GetUnit ( char *unitname );

void AddUnit( char *unitname,
              struct UnitStruct *UnitStuff);

void RemoveUnit( char *unitname );
```

The function EveryUnit(void (*func)(struct UnitStruct *Unit)) traverses the hash table structure and applies the function "func" to every unit in it. Thus, the hash structure may be changed without changing every piece of code which needs to traverse the structure.

3.2.1.3 Pending and Controlled Entity Lists

The IST SAFDI system simulates up to 12 fireteams at a time. The data structures reflect this limitation. The Pending List is an array limited to no more than twelve entities which are pending creation. It holds their name and type. The Controlled List is an array limited no more than twelve controlled entities. It holds their User Interface name, their network name, their type, and their SIMNET Vehicle ID information.

3.2.1.4 Maintaining the Map

The map of the terrain database is drawn and maintained largely by code which is part of the Simulator; it will thus not be discussed here. We have implemented new code to compute and draw contour lines, and a discussion of that code follows.

Contour lines are computed directly from the polygons in the terrain database. For any particular view on the map portion of the screen, the minimum and maximum values of the height of the terrain are determined. Then, all planes with heights between these minimum and maximum values (which are at a preset distance, contour_width, apart) are intersected with all polygons on the map. When any polygon contains intersection points with a plane, a line segment is drawn on the map which corresponds to this intersection. So, contours are constructed polygon by polygon instead of as a linked structure.

There are a few special cases to consider when using this

approach. First, if a polygon lies completely within one of the planes, then no action is taken; no contour line is drawn. If a polygon intersects a plane in exactly one point (a vertex of the polygon) then no action is taken. If, however, the intersection of a polygon and a plane is one of the sides of the polygon, then a contour line is drawn along this side.

3.2.2 Interface DFA

The Simulator to which the User Interface is attached is constantly changing. Thus, the addition and removal of commands from the User Interface command structure must be easy to understand and quick to do. This section describes the method chosen.

The acceptance of commands from the user is governed by a Deterministic Finite Automaton (DFA), or more simply a Finite State Machine. Each part of a command is represented as a state in the DFA. For instance, the command:

```
Map CenterAt 30000 40000
```

has four parts, and thus the User Interface goes through four states to input it: an object name entry state (Map), a command menu (CenterAt), an integer entry for X (30000), and an integer entry for Y (40000).

The User Interface is always in some interface state, and how any particular input character or mouse action is interpreted depends on the current state of the system. A state stack is maintained, so that if a user erases an input, he will properly end up in the previous state.

Each state has a Prompt, a Handler, a Gate, a Mouse State, and an Action (these are discussed in detail below). The Gate, Mouse State and Action may be NULL. In addition, there is a "Goto Table" which determines the next state for any given state and input. Note that "Branching" can only occur from a Menu state.

Within this structure, the addition of a command consists of adding states to the State table, adjusting the Goto table, and implementing the appropriate Action. Most new states can use existing Handlers, Gates and Mouse States. Adding and removing states is a very simple task. Thus, the command structure maintains a high degree of flexibility as well as functionality.

3.2.2.1 Prompt

The Prompt is a text string describing the expected input to the user. In the case of a Menu state, the Prompt describes the menu choices. The Prompt is displayed in the Text Pane in yellow. It is displayed immediately upon entering a state, and is erased upon exiting that state. If any character in the Prompt is preceded by an "at" sign (@), then it is highlighted in Light Cyan. If the current state is a Menu

state, then the highlighted letter is considered the "Hot Key" for that menu choice.

3.2.2.2 Handlers

A Handler is a special function which accepts characters based on the input expected. Each Handler takes one character as a parameter, and processes it accordingly - accepting or rejecting it, doing whatever display is necessary, and changing states if necessary. The handlers only accept one character at a time to maintain the asynchronous nature of the system; the user only types one letter at a time, and other processing must continue in between. The following handlers are implemented:

- UnitHandler — Accepts the name of the entity for which this command is given. it is always the first state.
- EndHandler — Special Handler for processing the action at the end of a command.
- IntHandler — Accepts an integer
- IDHandler — Accepts a string
- DoubleHandler — Accepts a double precision floating point number
- PasswordHandler — Special Handler for Passwords. Accepts a String, but displays blocks in the Text Pane instead of the string.
- MenuHandler — Special Handler for Menus. Accepts a single letter, but only one of the letters highlighted in the Prompt. Displays the entire choice which the letter indicates, not just the letter itself.

3.2.2.3 Gate

Gates are special functions which monitor the results of inputs and check them for accuracy. If a state has a gate, then the input quantity must pass the gate in order for the interface to proceed to the next state. Note that the gate is checked only when the interface attempts to advance to the next state. The following gates are implemented:

- AngleGate — Assures angles are between 0 and 360
- XGate — Assures X coordinates are on the map
- YGate — Assures Y coordinates are on the map
- UnitGate — Assures that commands are only given to entities which exist.

- PassGate — Checks for the proper password
- ContourGate — Assures that input contour widths are reasonable
- ListAddGate — Won't allow adding to a full pending list
- PriorityGate — Won't allow redundant firing priorities
- RangeGate — Assures that input ranges are reasonable

3.2.2.4 Mouse State

The Mouse State is a field in the State structure that enables the mouse to participate in the User Interface DFA in an integrated manner. Many of the prompts in the user interface are of a similar nature, such that the mouse can be used as a short-cut to using the keyboard. For example, in every state where the current state accepts an X-coordinate and the next state accepts a Y-coordinate, the state has a MouseState field of EnteringCoordPair. Whenever a mouse click occurs in this state, both the map object and the Entity Icon Pane object will respond by simulating the keystrokes necessary to enter the world-coordinates of the mouse click through the keyboard interface. In other words, if the mouse is clicked in a certain area in the map object, the actual battlefield coordinates are immediately fed to the user interface so that the user does not need to calculate and enter them manually. Similarly, when the mouse is clicked on a particular entity in the Entity Icon Pane, that entity's current location is fed to the user interface.

Here is a list of all the Mouse States currently used in the SAFDI User Interface:

- NothingSpecial:
  Tells the mouse code to ignore this state.
- EnteringCoordPair:
  Indicates that the user interface currently expects an X-coordinate, which should be followed by a Y-coordinate. If the user clicks on the map, the coordinates of the location on the map are fed to the keyboard part of the User Interface. If a mouse click on a entity in either the Pending List or the Controlled List of the Entity Icon Pane is made, the current coordinates of that entity are fed to the User Interface in the same manner.
- EnteringUnitName:
  Indicates that the user interface expects a entity name. The entity name can be selected by mouse either in the Map Pane or the Entity Icon Pane in the same manner as (x,y) coordinates can be specified.

- EnteringControlledUnitName:
  Same as EnteringUnitName, except that only entities in the User Interface Controlled List are accepted.

- EnteringPendingUnitName:
  Same as EnteringUnitName, except that only entities in the User Interface Pending List are accepted.

- EnteringNonpendingUnitName:
  Same as EnteringUnitName, except that only entities not in the User Interface Pending List are accepted.

- InsideMenu:
  Indicates to the mouse code that the current state uses a menu, and that the mouse code should create a mouse-controllable menu equivalent to the keyboard menu if a mouse is being used. (The SAFDI User Interface uses the mouse if it is installed, but is completely functional without a mouse.)

3.2.2.5 Action

The Action is a function which performs the action specified by the input command. It is only activated at the end of a command, but it can examine the components of a command by examining the state stack. The state stack holds the components of a command with the first component on the bottom of the stack and the last component on the top of the stack.

3.2.3 Main Task Loop

The main task loop provides the central functionality of the User Interface. It continually checks five separate areas to see if they need attention. The five areas are: Ethernet, Update, Serial Port, Keyboard and Mouse.

3.2.3.1 Ethernet

SIMNET packets are received along the Ethernet asynchronously and placed in a queue, but are not processed as they are received. Within the Main Task Loop, this queue is checked, and any packets in the queue are processed. Processing a packet consists of recording its effect on the battlefield and the simulated entities. For most SIMNET packets, this simply means changing the values in the fields of an entity's structure and allowing the Update and Display mechanisms to handle the rest. The only exception is the Impact PDU, which actually causes a visual signal on the screen.

3.2.3.2 Dead Reckoning and Screen Updates

The Dead Reckoning model for each entity which the User Interface knows about is updated on a regular basis. The Main Task Loop will check to see if the allotted amount of time has passed since the last update, and if it has, will perform the next update. This amount of time is initially 1/10 of a second, but can be changed within the User Interface. The update consists of updating the Dead Reckoning models,

determining which entities will have to be redrawn on the map, and redrawing them.

3.2.3.2.1 Dead Reckoning

The entities' Dead Reckoning models consist only of x & y coordinates of position and vx & vy coordinates of velocity. The other coordinate direction, z & vz, are not necessary since the User Interface Map is two-dimensional. The update consists of adding vx*(time_elapsed) to x and adding vy*(time_elapsed) to y, where time_elapsed is the time from the last update OR the last Vehicle Appearance PDU. A Vehicle Appearance PDU sets an entity's x and y coordinates to the x and y coordinates in the Vehicle Appearance PDU.

3.2.3.2.2 Redrawing Entities

In order to minimize "flashing", entities are redrawn on the map via a slightly complicated algorithm. First, it is determined which entities need to be redrawn. An entity needs to be redrawn if its position has changed by at least one pixel in any direction, or if it overlaps or is overlapped by such a entity. Of course, an entity does not have to be redrawn if both its old and new positions are not on the screen.

Groups of overlapping entities are collected into "Levels", and each level is redrawn separately. Redrawing a level has four steps:

1. All entities are erased (by writing their stored bit-mapped backgrounds on top of them).

2. Any entities which have been deactivated are removed from the hashing structure.

3. All remaining entities store a bitmap of the Map where they are going to be placed.

4. The remaining entities are drawn.

3.2.3.3 Serial Port, Keyboard, and Mouse

The Serial Port, Keyboard and Mouse are all sources of input which are polled by the Main Task Loop. If the Serial Port has characters waiting, they are received and processed by a special routine. If the Keyboard has a character waiting, it is read and passed to the Text Interface DFA. If there is a Mouse Click pending, it is processed by special Mouse routines.

3.2.4 Mouse Objects and Structures

There is a specialized module in the SAFDI user interface which implements object-oriented, event-driven entities for allowing the user to interact with the user interface using a mouse as an input device. Upon being initialized, this module creates a special object which represents the entire screen. This object

also acts as the root node of an n-ary tree, to which other objects are added in a hierarchy. This module also adds support for creating several primitive kinds of objects, such as push-buttons, boxes, and menus on the screen. In addition to these objects, the user can create ad hoc objects using the same protocols as the objects that are provided. The provided objects and the user-defined objects can then be combined to produce compound objects, which the Mouse Object engine can treat as single objects due to its hierarchical design. For example, the programmer can use a box object with two push-button objects as children to create a compound object which asks the user whether he/she really wants to perform a critical action. One push-button would contain the word "Yes" and the other push-button would contain the word "No". When the user places the mouse over either of these buttons and clicks the mouse button, a message is sent to that push-button by the actions of the Mouse Object engine, which causes the desired action to occur.

A primary design goal of the Mouse Object engine was to allow several objects to exist on the screen simultaneously in such a way that the CPU could handle all of them and still have time left to perform other actions. This is the reason an event-driven approach was used. Specifically, the Mouse Object engine operates primarily through a function called FocusAttentionOn(). FocusAttentionOn() is designed to be called frequently from the main task loop of the program using the Mouse Object engine. Every time it is called, it polls the mouse to determine if any mouse buttons have been pressed. If so, it reads the mouse to determine which button was pressed, and the coordinates of the mouse icon at the time the button was pressed. Using a depth-first traversal of the n-ary mouse object tree, the Mouse Object engine determines whether there is an object which existed underneath the location of the mouse click. If so, a message is sent to that object telling it that it has been selected with a mouse click. Each object in the hierarchy has a user-defined function attached to it (via a pointer to a function in its defining structure), called a Mouse Object Handler, so that each object can have unique behavior to any given message, including a mouse click message. There are other kinds of messages in addition to mouse click messages. All of the different kinds of messages are listed below, along with their meanings:

MOBJ_CREATE: This message is sent to an object when it is created. This message acts as a constructor message to an object, so that it can perform specific data and screen initialization operations. If the object has a screen representation, it should also save whatever is underneath the object and draw its image at this time. This is the same action as is expected when a MOBJ_REDRAW message is sent (see below). It is allowed for the newly created object to in turn create other subobjects by calling AddMouseObject() at this time, resulting in a single creation message yielding a complete, compound object.

MOBJ_REDRAW: This message is sent to an object to let it know that it should save whatever is underneath it and redraw its image on the screen. Objects which do not have a screen representation may ignore this message when it occurs.

MOBJ_UPDATE: This message is similar to the MOBJ_REDRAW message, except that the object does not need to save what is underneath it first. When this message occurs, the object can assume that its representation is already on the screen, but that it needs to be updated to reflect changes.

MOBJ_UNDRAW: This message is sent to an object whenever it should restore whatever was on the screen before it was most recently redrawn.

MOBJ_DESTROY: This message is sent to an object just before it is about to be destroyed by the Mouse Object engine. This message implements a destructor facility, so that each object can deallocate dynamic data structures and restore the screen to its original state, thus undoing whatever changes to the user interface its existence caused. When this message is received by a compound object, it should not destroy any of its child objects; the Mouse Object engine will do this automatically.

MOBJ_KEYBOARD: This message is sent to any object which has requested attention from the keyboard, whenever a key is pressed. The message contains the key that was pressed as part of the message. This allows objects to be activated either by the keyboard or by the mouse.

MOBJ_EVENT: This message is sent to any object when a mouse click occurs within the boundaries of the object's representation on the screen.

MOBJ_ATTN: Objects may request continuous attention from the Mouse Object engine, instead of having to wait for MOBJ_KEYBOARD and MOBJ_EVENT messages passively. Such objects will receive this continuous attention by receiving one MOBJ_ATTN message each time FocusAttentionOn() is called.

All of these messages are routed to the object via a pointer to a function, as mentioned previously. The function pointed to must have the following prototype as an interface:

```
int Handler ( int messageType,    // Type of MOBJ_ message
              int button,     // Button ID or key pressed
              int x, int y,  // Coord of button click
              MouseObjectNode *p,
                        // Pntr to object recing msg
              void *info );  // Ad hoc info for object
```

The function returns an integer which signifies whether the message was processed successfully or not. A return value of 0 signifies success, while a set of nonzero return values is used

for various possible failure conditions, such as running out of dynamic memory, etc.

To create a mouse object, the function AddMouseObject() (as defined below) is called.

```
int AddMouseObject ( int parent, //Handle of object's
                                 //  parent
    MouseFunction f, //Handler function for the object
    int x0, int y0,  //Object's upper left coordinate
    int dx, int dy,  //The object's width and height
    void *info );    //Ad hoc info for the object
```

This function allocates necessary memory, attaches it to the object n-ary tree appropriately, and then sends a MOBJ_CREATE message to the new object. The return value of the function AddMouseObject() is 0 if the object creation was unsuccessful, or a nonzero "handle" value if successful. The handle returned is a unique value which identifies the mouse object just created. Note that the 'parent' parameter passed to AddMouseObject() is also a handle, only it is the handle of the already-existing object which should become the new object's parent. Note that if 0 is passed for "parent", then the root object mentioned previously will be the new object's parent.

3.2.5 User Interface to Simulator mapping

This section describes the User Interface commands and their translation to Simulator commands. Each command entry has these items:

UIC: User Interface command
IL: Implementation Level
SC: Simulator command generated by USER INTERFACE
SR: Simulator response

Command format key:

In UIC:	entity_id	parameter
	Change	menu option, selected by pressing highlighted character
In SC:	i	constant
	entity_type	parameter

The symbols () enclose the elements of a set of options.

The symbol | is used to separate the elements of a set of options, exactly one of which should be chosen.

3.2.5.1 Entity Change Commands

Change Entity Speed
UIC: entity_id Change Speed (Stop|Normal|Double|Rush)
IL: 1
SC: vehicle_number (A|a) (S|s|N|n|D|d|R|r)
SR: Acknowledgement

Change Entity Direction
UIC: entity_id Change Direction heading[0..360]
IL: 1
SC: vehicle_number f heading
SR: Acknowledgement

Change Entity Rules of Engagement
UIC: entity_id Change RulesEng (ATGM|SmallArms) (HoldFire|FireAtWill) range[0..2000]
IL: 2
SC: vehicle_number (f|f|l|l) range
SR: Acknowledgement

Sets entity_id's rules of engagement as follows:
"(": ATGM fire at will ")": ATGM hold fire
"{": small arms fire at will "}": small arms hold fire

Change Entity Posture
UIC: entity_id Change Posture (Standing|Kneeling|Prone)
IL: 1
SC: vehicle_number (S|s) (s|k|p)
SR: Acknowledgement

```
Change Entity Firing Order
   UIC:  entity_id Change FirePrior (Tank|APC|OthV|DInf)
           (Tank|APC|OthV|DInf) (Tank|APC|OthV|DInf)
           (Tank|APC|OthV|DInf)
   IL:    2
   SC:   vehicle_number " (T|A|O|D) (T|A|O|D) (T|A|O|D)
           (T|A|O|D)
   SR:   Acknowledgement
```

3.2.5.2 Entity Status Commands

```
Entity Status Report
   UIC:  entity_id Status Report
   IL:   1
   SC:   vehicle_number ?
   SR:   Status report information

Entity Status Resupply
   UIC:  entity_id Status Resupply
   IL:    2
   SC:   vehicle_number =
   SR:   Acknowledgement

Entity Status Mount
   UIC:  entity_id Status Mount
   IL:    2
   SC:   vehicle_number + <siteid> <hostid> <vehicleid>
   SR:   Acknowledgement

Entity Status Dismount
   UIC:  entity_id Status Dismount
   IL:   2
   SC:   vehicle_number -
   SR:   Acknowledgement
```

3.2.5.3 Entity Order Commands

No entity order commands have been implemented at this stage of the project. The development plan calls for the implementation of entity commands in follow-on phases. Typical unit commands are: Halt movement, Resume movement, Follow vehicle, separate subunit from unit, and rejoin subunit to unit.

3.2.5.4 Entity GoTo Command

```
Entity Order Movement
   UIC:  entity_id GoTo x_loc[0..XMAX] y_loc[0..YMAX]
   IL:   1
   SC:   vehicle_number r ? x_location y_location
   SR:   Acknowledgement
```

3.2.5.5 Map Commands

Set Map Center
```
   UIC:  Map CenterAt x_coord[0..XMAX] y_coord[0..YMAX]
   IL:   1
   SC:   not passed to Simulator
   SR:   none
```

Map MoveCenter
```
   UIC:  Map MoveCenter x_offset[-XMAX..XMAX]
                        y_offset[-YMAX..YMAX]
   IL:   1
   SC:   not passed to Simulator
   SR:   none
```

Map Scale
```
   UIC:  Map Scale scale_factor[50..10000]
   IL:   1
   SC:   not passed to Simulator
   SR:   none
```

Map Update Rate
```
   UIC:  Map MapUpdateRate rate[0..90]
   IL:   1
   SC:   not passed to Simulator
   SR:   none
```

Set Vehicle Dynamic Update Rate
```
   UIC:  Map VehDynUpdateRate rate[0..5000]
   IL:   1
   SC:   u u rate
   SR:   Acknowledgement
```

Set Dead Reckoning Update Rate
```
   UIC:  Map DeadReckUpdateRate rate[0..5000]
   IL:   1
   SC:   m u rate
   SR:   Acknowledgement
```

Map Polygons
```
   UIC:  Map Polygons (Toggle|Hide|Show)
   IL:   1
   SC:   not passed to Simulator
   SR:   none
```

Programmers' Tools
```
   UIC:  Map Debug (Ethernet|Noise|Mouse|ShowSerial|
                    HideSerial|ZapSerial|QuietLOS|
                    DynamicMemory)
   IL:   1
   SC:   not passed to Simulator
   SR:   none
```

Map Contours
UIC: Map Contours (Toggle|Hide|Show|Query|SetWidth)
IL: 1
SC: not passed to Simulator
SR: none

Map Gridlines
UIC: Map Gridlines (Toggle|Hide|Show)
IL: 1
SC: not passed to Simulator
SR: none

Map Names
UIC: Map Names (Toggle|Hide|Show)
IL: 1
SC: not passed to Simulator
SR: none

3.2.5.6 List Commands

List Add Entity
UIC: List Add entity_id(8) (US (DI|M1|...) |
Soviet (DI|T72|...))
IL: 1
SC: not passed to Simulator
SR: none

Remove Entity
UIC: List Remove entity_id
IL: 1
SC: not passed to Simulator
SR: none

List Create All Entities
UIC: List CreateAll
IL: 1
SC: i c entity_type x_location y_location 0
SR: vehicle_id vehicle_number

Remove Entire List
UIC: List EmptyList
IL: 1
SC: not passed to Simulator
SR: none

List Files
UIC: List Files DOS_file_path
IL: 1
SC: not passed to Simulator
SR: none

Load Entity List
UIC: List Load DOS_file_name
IL: 1
SC: not passed to Simulator
SR: none

Save Entity List
UIC: List Save DOS_file_name
IL: 1
SC: not passed to Simulator
SR: none

3.2.5.7 Master Commands

Remove entity

UIC: Master Remove entity_id password
IL: 1
SC: i r vehicle_number
SR: Acknowledgement

Change entity location

UIC: Master Beam entity_id x_loc[0..XMAX] y_loc[0..YMAX] password
IL: 1
SC: vehicle_number b x_location y_location
SR: Acknowledgement

Reset entity attributes

UIC: Master Reset entity_id password
IL: 1
SC: vehicle_number ?
SR: Acknowledgement

Stealth commands

UIC: Master Stealth (Locate|Show|Hide)
IL: 1
SC: not passed to Simulator
SR: none

3.3. Terrain Database

A terrain database consists of a database header, a patch index list, a patch guard list, and a patch list. The database header gives general and statistical information about the database as well as some information about data area offsets. The patch index list is a list of patch offsets, where patch offset 0 is the offset from the beginning of the file to the beginning of data for patch 0 (n+1 offsets allow patch size calculations). A patch guard is a synopsis of patch data that is described as being used for fast inter-visibility calculations; there is a patch guard for every patch. A patch contains the data for a square block of terrain, where the whole terrain is divided up into many small patches (i.e. a 75,000m X 50,000m terrain is divided into 15,000 patches of 500m square).

Each patch of terrain contains vertices, edges, polygons (basically slope planes), trees, treelines, objects, canopies, and patch header information (to locate and describe the previous data objects). The polygons describe either soil, asphalt (road), or river terrain features. The vertices for polygons and edges are given as indices into the patch's vertex list (i.e. the coordinate of a vertex is stored once in the vertex list and indirectly referenced by all the polygons and edges that use it). Canopies share a similar strategy, using their own vertex list. Trees, treelines, and objects have their coordinate data directly within their structure. In addition to coordinate data, data for trees include type, height, and radius, and data for treelines include type, number of vertices, height, and extents. Data for objects are simply the vertices; they are considered flat. The canopies are logically sub patches of a patch, containing their own vertex and edge lists. One interesting feature of polygon data is that soil polygons cover the entire patch with river, road, and canopy polygons (trees, treelines and objects also) superimposed on top of them. The polygons seem to be sorted with river and road coming first followed by soil.

3.3.1 Terrain Database Interface

The Terrain Database Interface (TDBI) uses patch caching to provide quick access to terrain patch data. The TDBI requires initialization calls to set up the terrain database file (located on disk) and the patch cache (located in resident memory). Once these are set up, patches can be read from the terrain database file and kept in the patch cache for fast access. The patch caching system attempts to reduce the amount of disk reads through a Least Recently Used (LRU) scheme. The LRU scheme keeps in the cache patches that are active or that have been recently used, letting patches that have become stagnant (relatively) leave the cache to make room for new patches being read from disk.

Each simulated entity has a "region" which is a subset of patches that exist in the patch cache. Currently a region consist of the patch that contains the entity and the eight surrounding patches. An initialization call is required to set up an entity's region

with the appropriate patches. As the entity moves outside of its center patch, a request is made to update its region. Patches are released and acquired such that the region is filled with patches centered around the entity's new location. If the patches needed to fill the region are in the cache no disk read is necessary; otherwise, the patches are read from disk and added to the patch cache.

The TDBI is used by the Intelligent Simulated Forces (ISF) software to provide access to patch data. The immediate need for the TDBI is for determining the elevation of a simulated entity given its (x,y) coordinate. Given an (x,y) coordinate and the patch data, the vertices of the polygon containing the coordinate are used to define an equation of the plane of the polygon. Inserting the coordinate into the equation will yield the z component or the elevation. Modifications could be made to the database to incorporate the polygon plane equation, thus saving computation time.

A by-product of developing the interface was a plan view display of an entity's region. The plan view display shows the rivers, roads, trees, treelines, objects, and canopies of a region. Both scale and offset of the display can be changed, allowing zoom in and out capabilities.

3.4. SAFDI and User Interface Hardware Configuration

The SAFDI User Interface and the Simulator each require the following equipment:

- IBM PC compatible, with ISA (AT) bus
- Intel 80386 microprocessor
- Intel 80387 math coprocessor or equivalent
- 4 Mbytes of memory
- VGA graphics
- 3.5" diskette drive (720 or 1.44M)
- 80 Mbyte hard disk drive
- 3Com ethernet interface card
- Serial port

In addition, the following are required:

- 1 null modem cable compatible with the PC's serial ports.
- 1 Microsoft compatible mouse and driver on User Interface PC.

To connect the Simulator to the SIMNET Ethernet network requires:

- Thin Ethernet coaxial cabling
- 3 thin Ethernet T-connectors
- 2 50 ohm thin cable terminators
- 1 Thin Ethernet to Thick Ethernet converter box (eg. a Clear Signal 8043-3)

The User Interface is connected to the Simulator via serial ports. Currently, they communicate at 2400 baud, though this could safely be increased to as much as 19200 baud.

Overall system configuration:

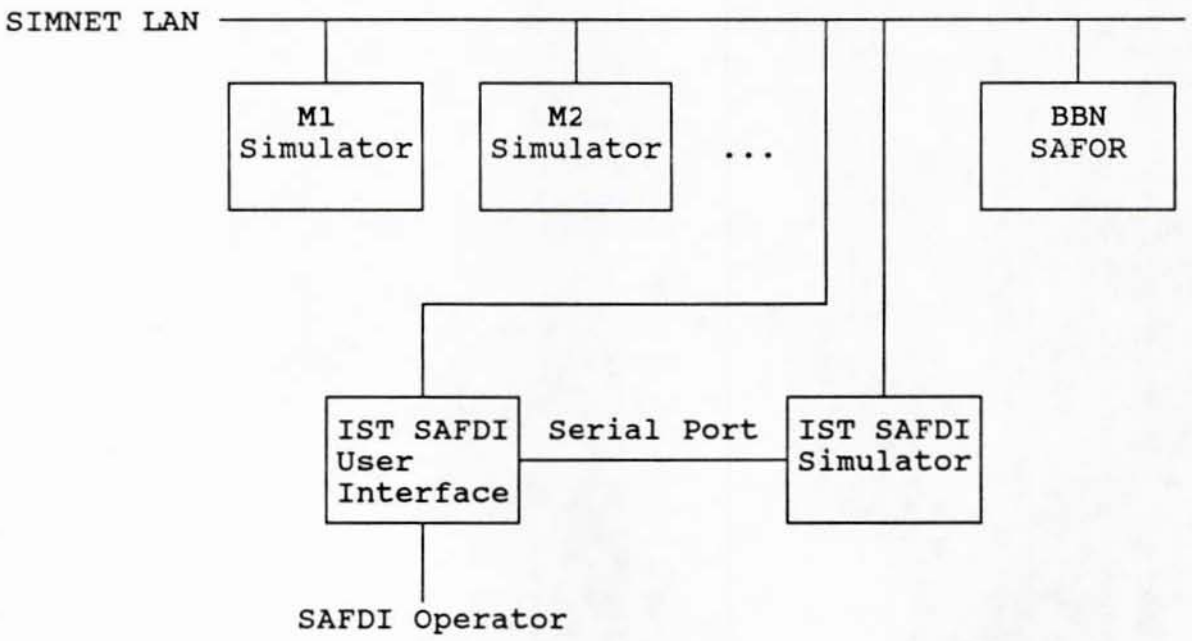


Figure 6 - System Configuration

3.5. Current Limitations

Known limitations of the current SAFDI implementation are listed in this section.

1. Workstation limit
   Because of the complexities associated with tracking mount and dismount operations across multiple SAFDI workstations, the SAFDI implementation is currently limited to a maximum of two SAFDI workstations simultaneously active on the network. Furthermore, if two workstations are active, they must be controlling forces of opposite alignment (i.e. one Blue and one Red). Of course, a single SAFDI workstation may be of either alignment.

   This limitation can be removed with considerable programming effort and some increase in network traffic. The correction involves SAFDI workstations transmitting non-SIMNET PDU packets over the network to signal mounts and dismounts to each other, so that each workstation can keep track of which M2s are occupied.

2. Number of fireteams supported
   The IST SAFDI system has not yet been stress tested with a large number of fireteams (i.e. more than 5 or 6), although the designed capacity is 12 fireteams (one Mechanized Infantry company).

3. Operator Intervention
   The operator must initiate most behaviors. There are no automatic complex or sophisticated behaviors supported at this stage. The automatic behaviors that are supported concern the activities involved in a single task. For example, upon being ordered to move a SAFDI fireteam can move from one location to another location avoiding obstacles automatically but will not react to being attacked along the way. The operator must intervene to cause the fireteam to react to the attack.

4. Unit command and control
   The individual fireteam is the only dismounted infantry unit currently modeled. Follow-on phases of this project will development higher level unit command and control behaviors.

3.6. Next Phase Tasks

SAFDI Phase 2 Changes and Enhancements

1. SAFDI fireteams should be able to mount any friendly vehicle, including helicopters, but not including fixed wing aircraft.

2. Dismount should require a 30 second delay. Under consideration are dismount left/dismount right/dismount rear battle drills.

3. Reduce mount range from 50 meters to 20.

4. Modify mount to mount at rear rather than mount at front.

5. New implied posture changes:
   a. prone at end of move
   b. prone after 5 seconds without moving.

6. Add SIMNET Id to display of Other Entities in User Interface.

7. Add other direct fire.

8. Enhance implied posture changes.

9. Correct the use of "unit" and "entity" throughout code and data files.

10. Design and begin implementation of SMARTS and UNITS.

11. Implement exhaustion component of rate determination.

12. Implement other movement types; eg. Using Cover, Go To Hidden Position, and Go To Recon.

13. Implement indirect fire.

14. Implement Change Firing Order command for small arms.

Section 4

References and Bibliography

4 References and Bibliography

Burg, J., Hughes, C. E., Lisle, C., Moshell, J. M., Carrington, J., and Li, X. (1991). "Behavioral Representation in Virtual Reality", Proceedings of the 2nd Behavioral Representation and Computer Generated Forces Symposium, Orlando FL, May 6-7 1991, pp. B 1-26.

Clarke, T. L., and Otte, J. M. (1991). "Human Behavioral Modeling Using Catastrophe Theory", Proceedings of the 2nd Behavioral Representation and Computer Generated Forces Symposium, Orlando FL, May 6-7 1991, pp. C 1-8.

Coleman, V., Gonzalez, G., Petty, M., Smith, S., Vanzant-Hodge, A., Watkins, J., and Wood, D. (1990). "Alternative Implementations of Knowledge Representation and Acquisition Methods in Distributed Interactive Simulation: Investigations and Findings", Technical Report IST-TR-90-21, Institute for Simulation and Training, University of Central Florida, 113 pages.

Cox, B. J. (1986). Object-Oriented Programming, An Evolutionary Approach, Addison-Wesley, Reading MS, 274 pages.

Crooks, W. H., Fraser, R. E., Herman, J. A., Jacobs, R. S., McDonough, J. G., Bonanni, P., Harrison, B., Junot, A., and Kirk, J. (1990). "SIMNET Semi-Automated Forces (Version 3.x) Functional Specification", Technical Report PTR-4043-15-0200-4/90, Perceptronics.

Danisas, K., Smith, S., and Wood, D. (1990). "Sequencer/Executive for Modular Simulator Design", Technical Report IST-TR-90-1, Institute for Simulation and Training, University of Central Florida, 16 pages.

Downes-Martin, S. (1990). "Replacing the Exercise Controller with the Enemy: the SIMNET Semi-Automated Forces Approach", International Training Equipment Conference, UK, April 1990.

Downes-Martin, S. (1991). "The Combinatorics of Vehicle Level Wargaming for Senior Commanders", Proceedings of the 2nd Behavioral Representation and Computer Generated Forces Symposium, Orlando FL, May 6-7 1991, pp. D 1-19.

Fishwick, P. A., Petty, M. D., and Mullally, D. E. (1991). "Key Research Directions in Behavioral Representation for Computer Generated Forces", Proceedings of the 2nd Behavioral Representation and Computer Generated Forces Symposium, Orlando FL, May 6-7 1991, pp. E 1-13.

Foley, J. D., and Van Dam, A. (1982). Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading MA, 664 pages.

Fraser II, R. E., and Herman, J. A. (1990). "Integration of SIMNET DI Simulators with SAFOR Workstations, Functional Specification", Technical Report PTR-4043-06-0000-90/90, Perceptronics, 65 pages.

Gates, K., and Frantz, F. (1990). "Semi-Automated Force Simulation using a Blackboard", Proceedings of the 12th Interservice/Industry Training Systems Conference, Orlando FL, Nov 6-8 1990, pp. 295-301.

Gonzalez, A. J. (1990a). "Intelligent Adversaries in a SIMNET Simulation", unpublished, Institute for Simulation and Training, University of Central Florida, August 1 1990, 23 pages.

Gonzalez, G., Mullally, D., Smith, S., Vanzant-Hodge, A., Watkins, J., and Wood, D. (1990b). "A Testbed for Automated Entity Generation in Distributed Interactive Simulation", Technical Report IST-TR-90-15, Institute for Simulation and Training, University of Central Florida, 37 pages.

Grier, P. (1990). "Three Tracks for Simulation", AIR FORCE Magazine, August 1990, pp. 40-43.

Guibas, L. J., Hershberger, J., Leven, D., Sharir, M., and Tarjan, R. E. (1987). "Linear Time Algorithms for Visibility and Shortest Path Problems inside Simple Polygons", Algorithmica, Vol. 2, pp. 209-233.

Harmon, S. Y., Yang, S. C., Howard, M. D., and Tseng, D. Y. (1991). "A Behavior-Based SAFOR and Its Preliminary Evaluation", Proceedings of the 2nd Behavioral Representation and Computer Generated Forces Symposium, Orlando FL, May 6-7 1991, pp. G 1-14.

Huon, M. (1989). "Expert system for the simulation of tank platoon behavior in a synthetic scene", promotional literature, 11th Interservice/Industry Training Systems Conference, 9 pages.

Kornell, J. (1987). "Reflections on using knowledge based systems for military simulation", Simulation, Vol. 48, No. 4, April 1987, pp. 144-148.

Lavery, R. G. (1986). "Artificial Intelligence and Simulation: An Introduction", Proceedings of the 1986 Winter Simulation Conference, 1986, pp. 448-452.

Le, H. T. (1990). "On the Role of Distributed AI in Large Scale Network Simulation", Proceedings of the 12th Interservice/Industry Training Systems Conference, Orlando FL, November 6-8 1990, pp. 241-246.

Maruichi, T., Uchiki, T., and Tokoro, M. (1987). "Behavioral Simulation Based on Knowledge Objects", Proceedings of the European Conference on Object Oriented Programming, Paris France, June 17-18 1987, pp. 213-222.

Moshell, J. M., Hughes, C. E., and Petty, M. D. (1989). "Constraints as a Specification Mechanism for Automated Opposing Forces in Networked Simulators", Proceedings of the Interactive Networked Simulation for Training Conference, Institute for Simulation and Training, Orlando FL, April 26-27 1989, pp. 84-90.

Nelms, D. W. (1988). "SIMNET II", National Defense, July/August 1988, pp. 68-69.

Petty, M. D., Moshell, J. M., and Hughes, C. E. (1988). "Tactical Simulation in an Object-Oriented Animated Graphics Environment", Simuletter, Vol. 19, No. 2, June 1988, pp. 31-46.

Petty, M. D. (1990). "Experiments in Routing an Autonomous Land Vehicle with a Weakly Inductive Learning Algorithm", Proceedings of the Third Florida Artificial Intelligence Symposium, Cocoa Beach FL, April 3-6 1990, pp. 159-163.

Pope, A. R. (1989). "The SIMNET Network and Protocols", Report No. 7102, BBN Systems and Technologies, July 1989, 160 pages.

SOGITEC, "An Expert System for Tank Platoon Behavior", Interactions, No. 2, June 1989, pp. 6-7.

Stanzione, T. (1989). "Terrain Reasoning in the SIMNET Semi-Automated Forces System", Geo'89 Symposium on Geographical Information Systems for Command and Control, SHAPE Technical Centre, The Hague, Netherlands, October 1989.

Thorpe, J. A. (1987). "The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting", Proceedings of the 9th Interservice/Industry Training Systems Conference, Orlando FL, November 30-December 2 1987, pp. 492-501.

Wise, B. P., Miller, D., and Ceranowicz, A. Z. (1991). "A Framework for Evaluating Computer Generated Forces", Proceedings of the 2nd Behavioral Representation and Computer Generated Forces Symposium, Orlando FL, May 6-7 1991, pp. H 1-7.

Appendices

A. Appendices

A.1 SALUTE Report Format

S = Size of entity
A = Activity of enemy entity; if moving, give direction of movement.
L = Location of enemy entity (6-digit grid position)
U = Uniform
T = Time of sighting (DTG Date Time Group, e.g. 301630Z)
E = Equipment carried by enemy entity

Sample:

S: "Line alfa, platoon plus."
A: "Line bravo, moving south on road, in column."
L: "Line charlie, ES947859."
U: "Line delta, green camouflage."
T: "Line echo, 301630Z."
E: "Line fox, 4 bravo mike papa, 3 tango 7 2."

A.2 U.S. Mechanized Infantry Platoon

The dismount element of a mechanized infantry platoon are listed below. Soldiers who remain mounted in the Bradleys are not listed. (Information source: U.S. Army FM 7-7J March 1990)

BFV 1: Platoon Ldr, Platoon RTO, FO, Squad Ldr, Fireteam Ldr, SAW

BFV 2: Fireteam Ldr, Rifleman, Grenadier, 2x SAW, ATGM Gunner

BFV 3: Fireteam Ldr, Grenadier, 2x SAW, ATGM Gunner

BFV 4: Squad Ldr, Fireteam Ldr, SAW, ATGM Gunner, Medic, FO RTO

Total of all dismounted men: 23
Total without support troops: 19

Weapons
Platoon Ldr, Platoon RTO, FO, FO RTO, Squad Ldr,
Fireteam Ldr, Rifleman: M-16 5.56mm
SAW: 5.56mm
Grenadier: M-16 + M203 40mm grenade launcher
ATGM Gunner: Dragon ATGM + M-16 5.56mm
Medic:

IST SAFDI Generic fireteam
Strength: 5
Soldiers: 2x Rifleman*, Grenadier, SAW, ATGM Gunner

*Includes leaders.

The five men strength of the SAFDI generic fireteam reflects the typically non-combatant roles of the medic, Platoon RTO, and FO RTO.

A.3 Summary of April 10, 1991 SAFDI Demonstration

On April 10, 1991 the IST Intelligent Simulated Forces Lab demonstrated the IST Semi-Automated Forces Dismounted Infantry (SAFDI) system. In attendance were Col. Larry Mengel and SFC Hager (TSM SIMNET office), Col. Frank Smith and Mr. Russ Miller (Forces Command), Lt. Col. Thomas Mostaglio (U.S. Army TRADOC), Mr. Robert Paulson (PM Trade), Mr. Peter Bush (U.S. Army Infantry School), and IST personnel. The meeting was divided into two sessions.

The first session consisted of a presentation by Col. Mengel and Mikel Petty. Col. Mengel gave an overview of how the SAFDI system would be used in the training environment. Mikel Petty then discussed the design and implementation of the SAFDI system. He discussed the work plan outline, basic architecture and capabilities of the system, and the interface's command structure. He also went over some of the internal processing of the system, such as the algorithms used to calculate small arms fire. He emphasized that he was explaining this so he could get feedback from the attendees. Throughout the presentation, the attendees discussed the implications of the various design decisions.

The second session was a live demonstration of the IST SAFDI system seen with the BBN STEALTH. Capabilities demonstrated for DI included searching for and destroying targets, navigating through the terrain while avoiding obstacles, and mounting and dismounting from Bradleys. At the time of this demonstration, determination of line of sight was not completely functional.

IST's SAFDI testbed is fully compatible with SIMNET V6.0. It operates with two PCs, one that serves as a mouse-driven User Interface and a second one that simulates the entities.

SAFDI CAPABILITIES

This section lists the IST SAFDI capabilities as stated in the project plan. This list is based on Col. Mengel's minimum requirements for any usable SAFDI system that is part of SIMNET.

1. Using local terrain map and LOS algorithm, DI fireteams will determine sighted enemy entities, then generate sighting reports (in SALUTE format) to be displayed in the User Interface's Message Pane. (Radio communication will be assumed for all SAFDI fireteams.)

2. At this time, there is communication between operator and fireteams only. Future implementation levels will incorporate communications at the platoon or company level. Commands would then be passed to component fireteams automatically.

3. DI fireteams will be able to kill enemy infantry and vehicles. The simulation will perform firing probability calculations, generate fire and impact PDU's, create ATGM entities, and track ammunition expenditure.

4. DI fireteams can be killed by enemy's direct or indirect fire. The simulation will receive and process impact PDUs, and track incremental losses.

5. DI fireteams can mount and dismount APCs. When mounted, the fireteam is temporarily removed from the simulation and becomes associated with its APC. The dismount command places fireteam back on the battlefield.

6. DI fireteams can be seen according to visual range and posture. The simulation can send appearance PDUs of DI standing, kneeling, or prone, to produce the visual icon appropriate for that posture.

7. DI fireteams can change speed. The movement at fixed rates is appropriate for infantry, based on speed setting, posture, and terrain.

LAB DEMO

During the lab demonstration there were four demonstrations conducted by Mikel Petty.

The first scenario presented the capabilities of movement, route planning, postures, and user interface. The SAFDI fireteam automatically routed around obstacles. While the fireteam was moving, the operator changed the fireteam's speed directly with the User Interface commands. Changes in posture also resulted in changes in speed. The important points in this demonstration were:

- speed was changed directly with user interface
- change of posture caused changes in speed
- DI fireteams found routes around obstacles (route planning)

The second scenario demonstrated Mounting and Dismounting capabilities. The important points in this demonstration were:

- A SAFDI fireteam mounted a Bradley.
- A SAFDI fireteam failed to mount because the fireteam was too far from the Bradley. After the fireteam was given a command to move up closer, it was able to mount the Bradley.
- A Bradley holding a mounted fireteam was given the command to relocate, then the mounted fireteam was ordered to dismount. This showed that DI fireteams could be transported.

The third scenario demonstrated missile combat involving two U.S. fireteam versus two T72s and a ZSU-23. An interesting observation about this scenario was that the DI fireteams where nearly invisible from the enemy's perspective, even though they were in a standing posture and no effort was made to hide them. Therefore, DI fireteams will be very dangerous to tanks in the SIMNET battlefield.

The DI fireteams were given permission to fire under operator-specified rules of engagement. The DI fireteams then destroyed the targets with their Dragon missiles. The DI fireteams obeyed their firing priorities (tanks first and then other vehicles); even though the ZSU was closer, the DI fired at the T72's first because they were higher priority targets. The scenario was viewed from multiple perspectives, so that the missiles could be seen both incoming and outgoing. From the target's view, the missile blast was very visible, revealing the DI fireteams' positions. The important points of this demonstration were:

- DI fireteams were given permission to fire using Rules of Engagement
- DI fireteams destroyed the targets with Dragon missiles
- DI fireteams were virtually invisible beyond 200 meters
- DI fireteams obeyed firing priorities

The fourth scenario was a demonstration of a small arms ambush. Two Soviet fire teams could be seen advancing along the road, and then being ambushed by a U.S. fire team. The U.S. fireteam opened fire, killing or incapacitating all of the enemy fireteams. Each time the U.S. fireteam fired, a smoke cloud was visible.

The same scenario was run again, but this time, the enemy fireteams were given permission to return fire. In addition, all fire teams were permitted to start firing at a much longer range. Because they opened fire at a much longer range, their fire was less effective and all fireteams ran out of ammunition before they had killed all the enemy fireteams. The attendees then pointed out that the ammunition expenditure was too rapid. The important points of this demonstration were:

- fire teams were ambushed
- ammo expenditure was too rapid and will be corrected
- small arms fire generates a smoke cloud
- fire teams were taking losses one man at a time (incremental losses)

IST SAFDI FUTURE PLANS

The next SAFDI event will be a demonstration of IST SAFDI at Ft Benning. This will be conducted under SIMNET V6.6.1. The Ft. Benning demonstration date has not yet been specified, but will be July or later depending on when SIMNET V6.6.1 is installed at IST. All SAFDI development beyond the Ft. Benning demonstration is subjected to approval by PM TRADE.

SAFDI Phase 2 Demonstration

Location: Ft. Benning, GA.
Attendees: M. Petty, C. Karr, D. Mullally.
Equipment: 2 PCs and required ethernet connectors.