ARMY RESEARCH LABORATORY

Warriors Edge Simulation and Gaming System: The Squad Simulation

by Mark Thomas and Gary Moss

ARL-TR-3564 **August 2005**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-3564 **August 2005**

Warriors Edge Simulation and Gaming System: The Squad Simulation

Mark Thomas and Gary Moss
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2005	Interim	1 February 2004–30 September 2004

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Warriors Edge Simulation and Gaming System: The Squad Simulation	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Mark Thomas and Gary Moss	P622783.Y10
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army Research Laboratory ATTN: AMSRD-ARL-CI-CT Aberdeen Proving Ground, MD 21005-5067	ARL-TR-3564
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In August 2004, the U.S. Army Research Laboratory (ARL) participated in a technology development effort to insert web-based information technologies into military operations. The exercise, sponsored by the Office of the Secretary of Defense, was the second year of the Horizontal Fusion portfolio. ARL was responsible for Army Urban Operations. Called Warriors Edge (WE), technologies showcased this year by ARL included micro robots, sensors, integrated soldier sensor/communications backpacks, mobile networking, and local-to-global information fusion and multilevel security. The WE exercise consisted of 15 soldiers representing a dismounted infantry platoon. The 15 live soldiers were comprised of a platoon headquarters section with a rifle squad. The adjacent squads were simulated using a computer game called SquadSim. The game software simulated a dismounted infantry rifle squad composed of a squad leader and two fire teams. The squad leader was played by a subject matter expert (SME). The SME controlled the squad leader using the computer interface to the SquadSim. The fire teams reacted to the squad leader's actions using computer-generated forces logic. This report describes the game software developed for the second year of the ARL WE.

15. SUBJECT TERMS

modeling and simulation, computer gaming, open source, dismounted infantry

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Mark Thomas
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	42	19b. TELEPHONE NUMBER (*Include area code*) 410-278-5011

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

In August 2004, the U.S. Army Research Laboratory (ARL) participated in a technology development effort to insert web-based information technologies into military operations. The exercise, sponsored by the Office of the Secretary of Defense, was the second year of the Horizontal Fusion portfolio. ARL was responsible for Army Urban Operations. Called Warriors Edge (WE), technologies showcased this year by ARL included micro robots, sensors, integrated soldier sensor/communications backpacks, mobile networking, and local-to-global information fusion and multilevel security.

The WE exercise consisted of 15 soldiers representing a dismounted infantry platoon. The 15 soldiers were comprised of a platoon headquarters section with a rifle squad. The other squads were simulated using computer-generated forces.

In the first year of the WE project, software was developed to translate simulation data packets into tactical objects (*1*). The software, DIStoTOS, provided the necessary message translation to insert simulated individuals and units into MIL-STD 2525B symbols. In addition, the Army standard OneSAF test-bed (OTBSAF) was used to provide individual soldiers, vehicles, and time-based scenario movements for the exercise. The combination of this software proved successful in the first year, but one deficiency was apparent. The OTBSAF interface for individual soldiers was too labor intensive for fine-grained live/virtual interaction. OTBSAF entities are commanded to perform explicit tasks. However, these tasks are nondeterministic as to time completion and movement patterns (soldiers would inexplicably wander off), and the single keyboard/mouse interface limited the number of entities which could be adjusted in real-time.

To supplement OTBSAF, computer gaming techniques were used to insert a man-in-the-loop (MITL) first-person shooter entity for adjacent squads. A squad simulation (SquadSim) was developed. The software simulated a dismounted infantry rifle squad composed of a squad leader and two fire teams. The squad leader was played by a subject matter expert (SME). The SME controlled the squad leader using the computer interface to the SquadSim. The fire teams reacted to the squad leader's actions using computer-generated forces logic.

This report describes the game software developed for the second year of the ARL WE.

2. SquadSim

The SquadSim was developed to put an MITL for better response to the platoon leader. The first year of the WE required OTBSAF fire teams to respond to platoon leader commands. The

OTBSAF interface made this cumbersome, and, in some instances, the troops did not arrive at the destination in time. SquadSim was developed to provide squad-level action, real-time response to orders, and MITL realism and decision making for the WE. Using open-source and commercial off-the-shelf (COTS) software, a first-person shooter game with computer-generated forces logic was developed.

SquadSim simulates a squad of dismounted riflemen. The keyboard operator is the squad leader, and the fire teams are computer-generated forces. The keyboard and mouse control speed, direction, and actions. The operator can fire the squad leader weapon, change the weapon, resupply, move across the terrain database, and change posture. The squad leader controls two fire teams using simple commands issued from the keyboard and mouse. The squad leader may command the squads to move to a location, move on a path, or follow him through a building. The fire teams respond to squad leader firing, enemy force firing, and other sound events (weapons detonations). Fire teams move in accordance with FM 7-8 (*2*). Fire teams may deploy in file, wedge, or echelon left/right formation. Fire-team formation parameters are modified by the operator using a separate graphical user interface called SquadSim Client Console (SS-CC).

Locomotion in the three-dimensional (3-D) mode imitates real human movement. Speed across the terrain is limited to appropriate speeds for human walking, running, or crawling. The player can assume a standing, kneeling, or prone posture. The player may jump over obstacles, walk through multistoried buildings, or walk along the terrain. Terrain collision is enabled by default. Multiple collision points on the squad leader avatar provide for accurate collisions with different-sized objects.

The first-person view displays the player's weapon, terrain and features, friendly and enemy entities, munitions effects, and simulated natural environment. SquadSim can display cloudscapes, fog, and time of day. Fully textured models for troops, vehicles, buildings, roads, and other terrain objects are included. The first-person view is shown in figure 1.

The squad may also be controlled using a two-dimensional (2-D) look-down view of the terrain. The 2-D view (shown in figure 2) allows the squad leader to lay down movement routes, see the entire squad at a glance, and monitor the formations. The 2-D mode allows for quick navigation through the terrain.

3. Technical Approach

The SquadSim design is modeled after an earlier project, Dismounted Infantry Simulation (DISim) (*3*). The DISim architecture is based on the ground combatant view of the scene from the ground up. The obstacles encountered, sounds heard, and events experienced all must be presented to the user in a timely and appropriate manner. The architecture provides the

Figure 1. First-person view.

Figure 2. Two-dimensional view of the simulated environment.

programmer with application programming interfaces (APIs) for graphics, sound, networking, entity management, and special effects. DISim was developed using the SGI Performer Graphics API. For SquadSim, however, an open source graphics solution was used.

The OpenSceneGraph (*4*) graphics library was used for developing the 3-D first person shooter view. OpenSceneGraph (OSG) is distributed in source code form and is portable across a number of operating systems such as Microsoft Windows and Linux. The choice of OSG was

made to foster cross-platform functionality. Source code provided the flexibility to make changes to the underlying software as required, and OSG had a rich set of features to include multiple database format importing (OpenFlight, 3-D Studio Max, etc.), window management, keyboard and mouse management functions, callback methods for fine control of graphics models and structures, and lots of sample source codes to help with learning how to use and extend the product. In addition, OSG has been integrated with other open-source libraries to provide an integrated platform for developing complex software.

Human figure animation is provided by the DI-Guy Application Programming Interface by Boston Dynamics (DI-Guy) (*5*). A COTS product, DI-Guy provides software level control of human avatars to simulate many movements of soldiers and other people and terrain objects. Using DI-Guy provided rapid insertion of human figures into the 3-D game engine, accelerating software development and product delivery.

Sounds are an important aspect of first-person shooter games. Sound gives clues to enemy and friendly location, threats, and environmental ambience. The proper use of sound is important to the sense of realism and immersion that the game user experiences. OSG is integrated with the OpenAL (*6*) sound library. The OpenAL library provides 3-D sound capability and uses the OpenGL graphics library reference system for coordinate and angular specifications. This allows the software developer to use the same coordinates and angles for orienting the graphics model on the computer display and for sound location. Using the OpenAL provided a quick method for adding 3-D sounds to SquadSim for weapons fire, weapons detonation, footfalls, and environmental sounds.

Real-time physics in SquadSim are used to calculate the result of collisions between objects. Rigid body physics are computed using the OpenDynamicsEngine (ODE). The ODE implementation in SquadSim computes the results of collisions in the environment between objects and is used to move objects based on inertia such as a cylinder rolling downhill.

SquadSim has the look and feel of commercial games, providing the user with an instant familiarity with the controls, operation, and expectation of the game.

Distributed Simulation Capability

SquadSim uses the distributed interactive simulation (DIS) and high-level architecture (HLA) for networking. The DIS was chosen to rapidly integrate SquadSim with OTBSAF. In addition, using DIS made the software compatible with DIStoTOS, providing a message capability to operational C4I systems.

DIS capability gives SquadSim interoperability with OTBSAF, which was invaluable to software development. OTBSAF is mature and well documented and gives validity to systems which seek to be DIS interoperable. SquadSim can engage OTBSAF entities, and OTBSAF entities react to SquadSim entities.

In addition, SquadSim has a TCP/IP interface to the SS-CC. This link provides extended capability not provided by DIS. For instance, terrain database updates, such as the opening and closing of doors, are provided through the SS-CC. This provides a reliable method for terrain database consistency when multiple SquadSim stations are in use.

The SquadSim is composed of three components: (1) a 3-D viewer (Viewer), (2) the SS-CC, and (3) the SquadSim Server Console SS-SC.

4. The SquadSim Components

4.1 Viewer

The Viewer is the first person shooter view and acts as the window into the 3-D world. The viewer's capabilities have been previously described. Using Viewer, the user can play the function of a Squad Leader (SL) and see his fire teams fight according to his commands.

Because of the complexity of initializing the Viewer, setting up its run-time environment, and managing two fire teams at run-time, a separate helper application was developed. The SS-CC provides an easy-to-use interface to change system parameters without affecting the run-time performance of the Viewer.

4.2 SS-CC

This Java application is a tool for configuring, launching, and monitoring an instance of SquadSim. SquadSim runs in a separate window, either on the same PC desktop or on another monitor connected to the same computer as the SS-CC (given a multiheaded display capability for that PC). Before launching SquadSim, each instance of SS-CC must connect to a SquadSim Server Console (SS-SC) running somewhere on the local area network. By connecting to the same server, multiple instances of the client console are joined in a larger, distributed simulation (see the SS-SC description).

In addition to controlling an instance of SquadSim, the SS-CC facilitates a sharing of ground truth via bidirectional message passing (with SquadSim and the SS-SC) conducted over TCP/IP sockets using a private protocol (see appendix C for a packet-level description). Information passed in this manner includes the following:

- The state of building *portals* (is a given door or window open, closed, opening, closing, etc.).
- What squads have joined, and information about each squad member (all players including the SL) such as role, name, call sign, rounds of ammunition, and type of ammo.
- The posture of each player (standing, kneeling, or prone).

Some ground truth information such as damage to terrain (i.e., holes in walls or terrain and shattered windows) is already handled by DIS or HLA; however, this information is not saved. Therefore, a late joining client will only see damage that occurs after a connection has been established. It is planned that the SquadSim protocol will eventually maintain a history of detonations so that clients can be given an up-to-date ground truth when joining.

Configuration capabilities of the SS-CC can be categorized as follows:

- Details necessary to launch SquadSim.
- Network parameters identifying an instance of the SS-SC to connect to.
- Keyboard and mouse bindings to SquadSim commands.
- Environmental effects such as fog and time of day.
- Specification of fire team formations.

All settings can be saved for subsequent instantiations of this tool. A detailed description of configuration parameters, as well as the appearance and operation of the SS-CC, are documented in appendix A.

4.3 SS-SC

Also a Java application, the server console is a management tool that facilitates the joining of multiple client SquadSims into a distributed simulation or game as well as the eventual departure or resignation of each client. As such, the SS-SC must create the illusion that there is one persistent battlefield environment and that each instance of SquadSim immerses the SL/user in that virtual world. As mentioned in the last section, ground truth information is sent from client to server via a SquadSim-specific protocol. The SS-SC then takes care of propagating this information downward through all the client connections, thereby facilitating client-to-client communication or shared understanding. As mentioned earlier, histories of terrain damage, the state of building portals, and player postures are not currently maintained by the server, so ground truth is inconsistent for late-joining clients. This will be supported in a future release of the software.

There are configuration parameters to specify either DIS or HLA for communication between SquadSim clients and other simulations such as OTBSAF. Here, the operator selects between DIS and HLA, then configures particulars about the communication standard chosen. This facilitates central control over the various SquadSim clients as far as what standard to use and simplifies the joining of larger HLA federations or DIS exercises. Other details common to all clients are configured in the SS-SC, and all settings can be saved for subsequent instantiations of this tool.

The SS-SC also has monitoring capabilities. In addition to displaying errors and diagnostic messages, there is a 2-D plan view display which has simple graphical representations of all

entities in the distributed simulation. There is also a scoreboard identical to the one displayed in each SS-CC instance.

The details of configuration parameters, operation and appearance of the SS-SC are described in appendix B.

There were several challenges in developing SquadSim. Squad management, DI-Guy integration, and OpenDynamicsEngine implementation are discussed next.

5. Squad Management

Squad management is a complex issue. First, there are two fire teams (FT) with four men each. Each fire team has a formation and moves relative to the SL. Within each fire team, each soldier must move relative to his position in the FT, avoid obstacles, maneuver, and fight. When engaging a target, each FT member must select a target to shoot at, and firing must be coordinated for maximum firepower efficiency.

Each fire team must be able to move according to the needs of the squad. The lead fire team may have to move to a position on its own and wait there until commanded further. The SL can specify a destination by location or route. The fire teams may have to change formation due to terrain limitations. To accomplish this logic, a hierarchical class structure was developed.

The base class is the squad (*Squad*). The squad class initializes the two fire team classes and positions them in their initial location and formations. The fire team classes (*Fireteam*) initialize their individual members. Each individual member is assigned a position in the fire team formation and given a call sign and initial location. The individual member class (*FireTeamMember*) contains a reference to the parent fire team. This reference is used to retrieve the fire team location and mission parameters which are used to influence the individual member state. The class hierarchy is shown in figure 3.

The fire team class has methods to control movement and weapons fire. The fire team moves in relation to the SL. As the SL moves, the fire team updates its position and an internal bounding box. The bounding box may shrink or expand based on collisions with terrain objects. A small bounding box may trigger the formation to change to a line. A large bounding box may allow for a wedge or other formation. Changing the formation automatically changes the bounding box for the individual members to maneuver.

Individual movement is more complex. Within the fire team formation (file, wedge, echelon left/right), the individual must avoid obstacles, walk around corners, or follow the leader into a building. These modes are controlled by the SL using the SS-CC. During normal road marches,

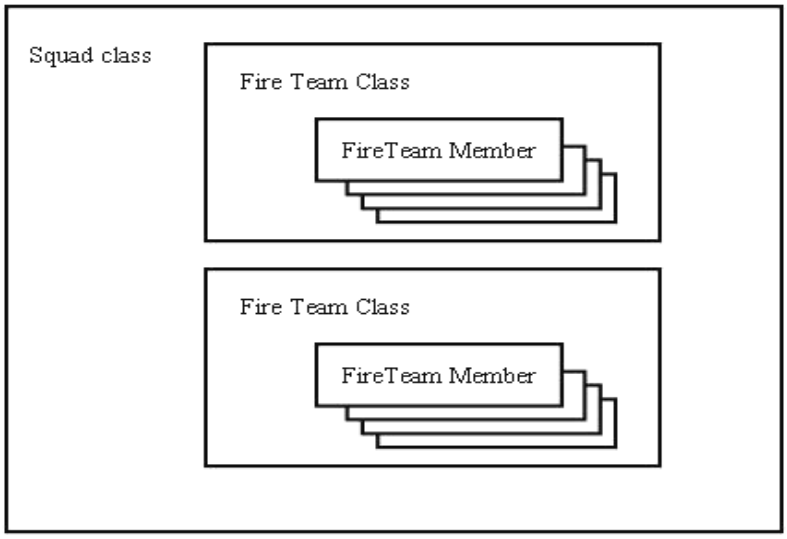

Figure 3. Class hierarchy.

fire team members (FTM) may occasionally have to walk around an obstacle. The FTM movement algorithm handles this by turning the FTM parallel to the intersecting object in the direction of the FT movement. FTMs avoid each other, jump over low obstacles, and do not stray too far from their assigned position in the fire team. This simple movement process works well in open terrain and provides autonomous movement without operator attention. It is not sufficient for close quarters such as urban environments. Therefore, another method is used to traverse urban canyons and perform building clearing operations.

To enter a building requires tighter spacing and exact movement to prevent anomalies such as walking through walls, entering the wrong room, etc. Therefore, a navigation map (*7*) is created from the SL movement. As the SL moves through the building, his position is stored in a waypoint list. The FT moves along this waypoint list until the end of the list is reached. When the final waypoint is reached, the FT deploys in its initial formation and awaits further action from the SL. This method provides building clearing functionality by computer-generated forces without extensive terrain database preprocessing.

Weapons fire is controlled per fire team. The SL may initiate FT attack by firing his own weapon. When the SL fires, the FT fire() method is called. The FT fire() method calls the individual member fire() method for any member whose health permits (dead members do not fire). When the individual member fire() method is called, the FTM will scan for a target and fire if one is acquired. FTMs fire until the target is dead. They then shift fire to another target if one is available or cease firing. FTMs do not fire at friendly troops. Weapons fire may damage buildings by blowing out windows and doors or creating breach holes in structures; figures 4 and 5 demonstrate this.

Figure 4. Undamaged building.

Figure 5. Building showing damage to window and wall breach.

6. DI-Guy Integration

DI-Guy was chosen as the soldier animation software because it is familiar to the author. The software has been used at ARL since 1996, and its capabilities, features, and programming interface are known and sufficient for this purpose.

The DI-Guy API comes in a variety of formats. These formats allow DI-Guy to be used with different graphics products such as OpenGL, the SGI Performer, and Windows. DI-Guy also has implementations for Windows, Linux, and SGI. The OpenGL implementation was used for SquadSim.

The DI-Guy OpenGL implementation must be run from a program thread which has a graphics context. In the OSG, there is an application thread, draw thread, and culling thread. The draw thread has the graphics context in OSG. The draw thread is where the actual scene is drawn to the screen. The culling thread is where scene geometry is pruned so that only the necessary parts of the scene are drawn. The application thread takes care of all other processing. Networking, keyboard and mouse commands, and other updates are processed in the application thread.

Each thread is an independent process. Threads share memory. An object updated in one thread is also updated in another. The problem with shared memory objects is concurrency. Unpredictable results occur when two threads are updating an object at the same time. Therefore, some data management needs to occur to prevent this. OSG uses node callbacks and mutexes to control memory management. Each OSG Drawable node can be given an update callback. In the update callback, data structures in the draw thread are updated by data from the application. Using this mechanism, DI-Guy was integrated using the following logic:

1. Create an OSG Drawable.
2. Supply an UpdateCallback() function.
3. Add the Drawable to an OSG Geode node.
4. Add the Geode to the scenegraph root.

Step 1 creates a derived class of the Drawable class. The new class contains a method named drawImplementation(). The drawImplementation() function is called every frame by the draw thread. The first invocation of drawImplentation() initializes the DIActor() class. All DI-Guy functionality is encapsulated in the DIActor() class. Instantiating the DIActor() class initializes the DI-Guy system. This must be done only once. Subsequent calls to drawImplementation() call the draw() method of the DIActor() class to draw the DI-Guy characters on the screen.

The UpdateCallback() function in step 2 copies data from the application thread to the data structures in the DIActor class.

The DI-Guy API defines all the data used to posture the DI-Guy avatar, locate it in the environment, and interact with it. Data passed from the application thread to the draw thread includes local position, azimuth, pitch, name, appearance string, DI-Guy shapeset, and DIS entity identification. This data is used to add an avatar to the display. To update the characters' appearance later, the local position, azimuth, pitch, speed, weapon fire mode, DIS appearance, DI-Guy appearance string, head azimuth, head elevation, gun azimuth, gun elevation, and range to the SL are passed. This data provides the necessary information for the DI-Guy motion library to position, move, and posture the avatar properly.

The DI-Guy API also provides functions to query data from the avatar, providing a feedback mechanism which is good for fine control of the avatar and effects on each individual entity. This query mechanism is used to place the squad leader avatar in the scene. The squad leader avatar in SquadSim must be processed differently from all other avatars. All other avatars are positioned exactly on the terrain according to their real-world posture information. The SL avatar is positioned slightly behind his real-world location to prevent unwanted avatar model artifacts from being displayed (figure 6). To do this, the drawing of the DI-Guy characters is done character by character vs. all characters at once. The DI-Guy draw is as follows:

1. Push the current projection matrix and attribute bits.
2. Set the OpenGL lighting, shading, texture, and culling modes.
3. For each character:

- If the character is the SL:
    a. Push the projection matrix.
    b. Translate by the negative of the head offset values.
    c. Draw the character.
    d. Pop the projection matrix.
- If the character is NOT the SL:
    a. Draw the character.

4. Pop the projection matrix.

Figure 6. Scene showing squad leader avatar. The helmet is an unwanted artifact.

7. Open Dynamics Engine (ODE)

ODE is an open source rigid body dynamics simulator. It is cross-platform, numerically stable, and has features for many real-world rigid body interactions. The ODE provides the simulator developer with a tool to model objects connected by joints, compute velocities from body mass and inertia, and detect collisions between bodies.

ODE is used in SquadSim to compute change in direction and speed of objects in the terrain. Cylinders, boxes, and spheres can be inserted into the scene to represent objects such as boxes, balls, trash cans, and grenades. The ODE computes the resulting motion of the objects according to their physical properties and environment interactions. A sample is shown in figure 7.

Object-to-object interaction is straightforward. Objects are added to the ODE world, and as the world is updated, it computes collisions, moves objects based on their inertia, and updates their velocity vectors and orientation parameters. The biggest challenge is terrain collision. Examples of OSG/ODE all have a flat terrain with set boundaries for walls. Flat terrain and walls in ODE are modeled by planes and described using the plane equation, $Ax + By + Cz + D = 0$. SquadSim terrain is not flat, however. The terrain has buildings, pitched roofs, and walls with portals. An object may bounce off a wall or go through an open window or door, so simple planes are insufficient for describing terrain in SquadSim. Therefore, each object in SquadSim

Figure 7. Physics-based objects in the simulated environment.

has two collision planes attached to it, one for the ground and the other for terrain objects in its direction of travel. The ground plane collider is aligned to the local ground plane normal. A sloping terrain, or pitched roof, will make a sphere roll. The front collision plane is sized to the bounding box of the object. This plane is used to collide with terrain objects. The front collider is aligned with the surface normal of the terrain object. This provides ODE with the proper data for calculating oblique hits. Using this method, objects can interact with the terrain without adding each polygon of the terrain to the ODE world.

8. SquadSim in Warriors Edge

The WE simulation suite was composed of two SquadSim stations, a OneSAF Test Bed Dismounted Infantry SAF (DISAF), a DIStoTOS station, and SquadSim-SC. The DISAF was used to supply troop movement and activity outside the McKenna* Military Operations in Urban Terrain (MOUT) site village, while SquadSim supplied troop movement and functionality inside the village. DIStoTOS provided the communications between the simulations and the C4I equipment, and the SquadSim-SC coordinated the runtime configurations of the two SquadSims.

SquadSim was used to play two squads. Two SquadSim stations running on laptop computers were used for this purpose. Each squad was given a call sign in the platoon and maneuvered according to the scenario script.

* Fort Benning, GA.

Follow-on work in this effort includes better movement algorithms for the fire teams in close quarters, engagement models, and mount/dismount capability with different troop carriers. Further integration with battle command language to provide multimedia communications with live C4I equipment is required. In addition, Spot Report generation through the SS-CC would provide command and control and intelligence for information fusion purposes.

9. References

1. Thomas, M. *The Warriors Edge Simulation System*; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, submitted for publication.

2. FM 7-8. *Operations*. Chapter 2. http://www.globalsecurity.org/military/library /policy/army/fm/7-8 (accessed May 2005).

3. Thomas, M. *Dismounted Infantry Visualization Research: The Dismounted Infantry Simulation (DISim)*; ARL-TN-193; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, December 2002.

4. The OpenSceneGraph. http://openscenegraph.org (accessed February 2004).

5. Boston Dynamics, Inc. *DI-Guy 5 User Manual*; Boston Dynamics, Inc: Cambridge, MA, 2004.

6. The OpenAL. http://openal.org (accessed February 2004).

7. DeLoura, Mark A. *Game Programming Gems*; Charles River Media, Inc.: Hingham, MA, 2000.

INTENTIONALLY LEFT BLANK.

Appendix A. User Guide to SquadSim Client Console (SS-CC)

A.1 Introduction

SS-CC is a Java front-end or client-side console for the SquadSim gaming station. This guide will walk you through how to configure the SS-CC to connect to the SquadSim Server Console (SS-SC) (see appendix B) and SquadSim prior to game startup, and to control the gaming station while a game is running. To begin, depending on whether you are on a Windows or a UNIX* platform, skip to the appropriate section on starting the console.

A.2 Starting SS-CC Under Microsoft Windows

Make sure there is a java.exe file in your execution Path. Locate the file named Client.bat in the SquadSim\Java folder. If it has not been taken care of during installation, check that the definition of ProjectDir on the first line of Client.bat is set to the folder where it is installed. Once the batch script is configured, execute it by either:

1. Double clicking on the Client.bat icon in Windows Explorer.
2. Starting up a DOS command window, navigating to the SquadSim\Java folder in the DOS window, and then typing *Client.*
3. Making a shortcut on your desktop to the batch script, then double clicking on the icon.

A.3 Starting SS-CC Under UNIX

Make sure there is a java.exe file in your execution Path. Locate the file named Client.sh in the SquadSim/Java directory. If it has not been taken care of during installation, check that the definition of InstallRoot on the first line of Client.sh is set to the directory where it is installed. Also check that the definition of CLASSPATH on the next line of the shell script is correct. Once the shell script is configured, execute it by either:

1. Double clicking on the Client.sh icon from a file browser.
2. Starting up a UNIX terminal emulation window, navigating to the SquadSim/Java directory, and then typing *sh Client.sh.*
3. Creating a launcher on your desktop which points to Client.sh, then double clicking on the icon.

* UNIX is a registered trademark of The Open Group.

A.4 Layout of the Client Console

Figure A-1 shows the SS-CC when it first initializes. The interface is divided into three distinct sections—a menu bar across the top, a column of buttons down the left side, and a panel comprising the remainder of the window. Each of the buttons, with the exception of Quit, selects which panel is displayed; only one panel is viewable at any given time. To change panels, click on one of the buttons on the left. The next sections describe the operation of each panel.

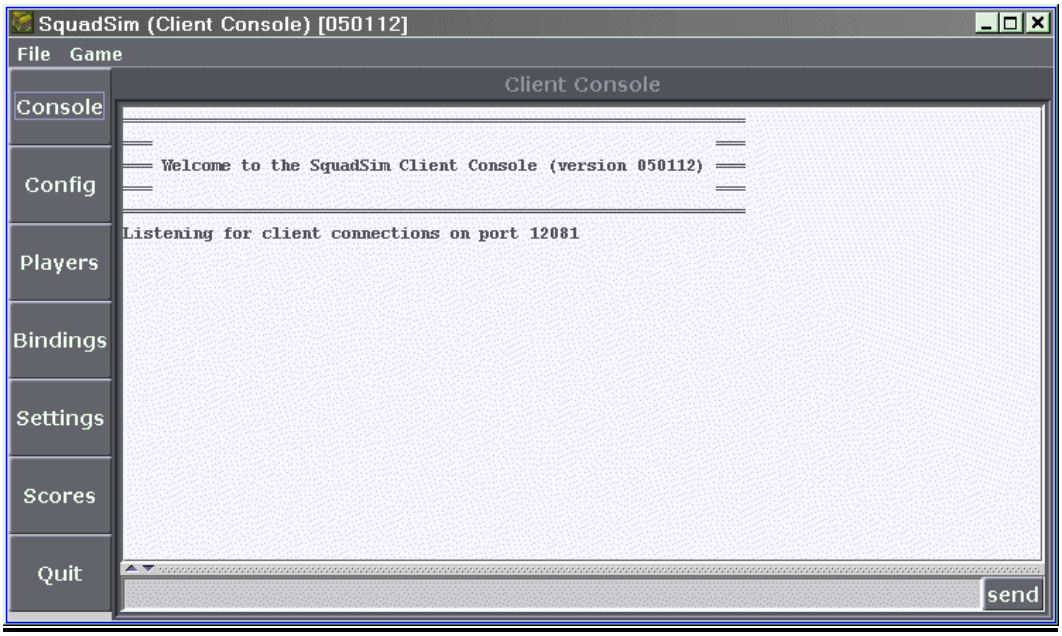

Figure A-1. SS-CC initialization (Console panel).

A.4.1 Console Panel Operation

In figure A-1, the Console panel is displayed. This panel displays diagnostic messages from the SS-CC, the SS-SC, and from SquadSim. It also has a one-line window at the bottom for sending chat messages. These messages will be displayed on the server as well as on each game station (SquadSim instance) that has joined. To send a chat message, type the message in this window and then click the Send button on the right.

A.4.2 Config Panel Operation

This panel is for configuring static settings. These settings are only read at game startup and will not affect the game station operation while it is running. Figure A-2 shows what this panel looks like. All fields must be filled in, with the exception of the *HLA Configuration* section, which must be filled in only when *HLA Interface* is configured on the server. Each field is described next in the order that it appears:

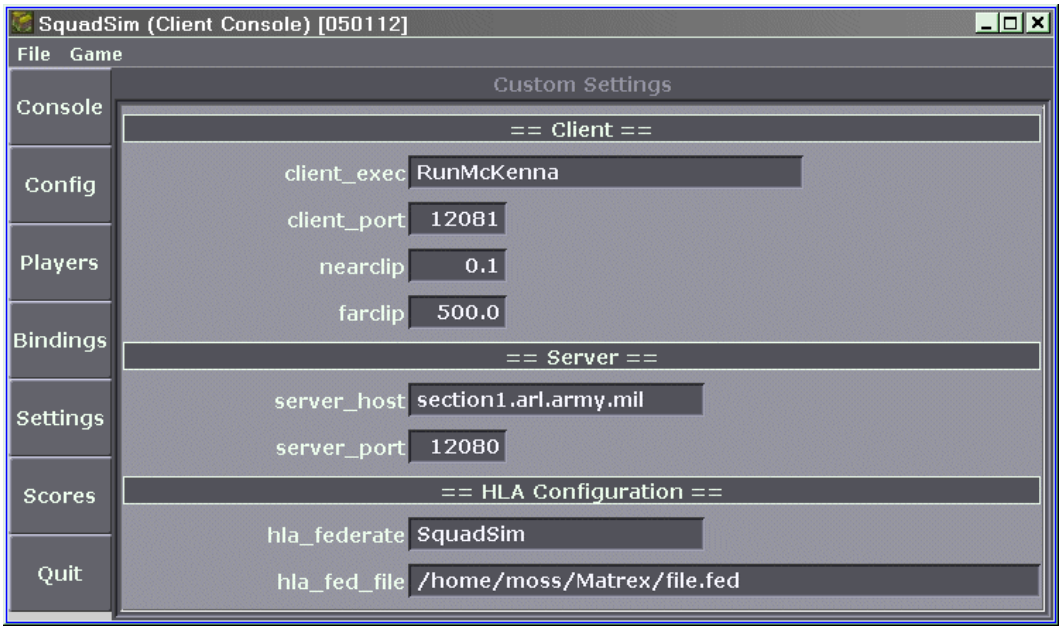

Figure A-2. Config panel.

== Client ==

- client_exec – name of SquadSim run script which needs to be installed in the InstallRoot directory. Note, if installed in a subfolder of this directory, the relative path name can be used.
- client_port – a port number to use for socket communications with SquadSim. This can be the same on every instance of SS-CC/SquadSim on a network because it is local to the computer where a particular gaming station is running. In the unlikely event that two gaming stations are running on the same machine, then these ports must be different.
- nearclip – distance in meters to the near clipping plane from the camera position.
- farclip – distance in meters to the far clipping plane from the camera position.

== Server ==

- server_host – the fully qualified host name or IP of the computer running the game server console (SS-SC).
- server_port – a port number to use for socket communications with SS-SC.

== HLA Configuration ==

- hla_federate – HLA federate name, typically name of application.
- hla_fed_file – absolute path name on local machine of HLA FED (.fed) file.

After modifying any fields, left click on the File menu button on the top menu bar and select Save Settings. All configuration settings are saved in a file called *userconfig.cfg* in the application directory. For UNIX systems, this is under *$HOME/SquadSim*, and for Microsoft Windows, it is under the standard application data folder (i.e., *C:\Documents and*

Settings\<username>\Application Data\SquadSim). See the example userconfig.cfg at the end of this appendix. The settings in this panel must be saved to be read by SquadSim.

A.4.3 Players Panel Operation

This panel is for configuring your squad (or team). This consists of selecting the unit type (i.e., US_DI_SQUAD, USSR_DI_SQUAD, or OTHER_DI_SQUAD) and specifying names and call signs for the team and each member of the team. Select the unit type with the drop-down menu at the top and fill in all fields with names and call signs that are unique across all stations. (See figure A-3 for an example configuration of this panel.) Once all fields are filled in, left click on the File menu button on the top menu bar and select Save Settings.

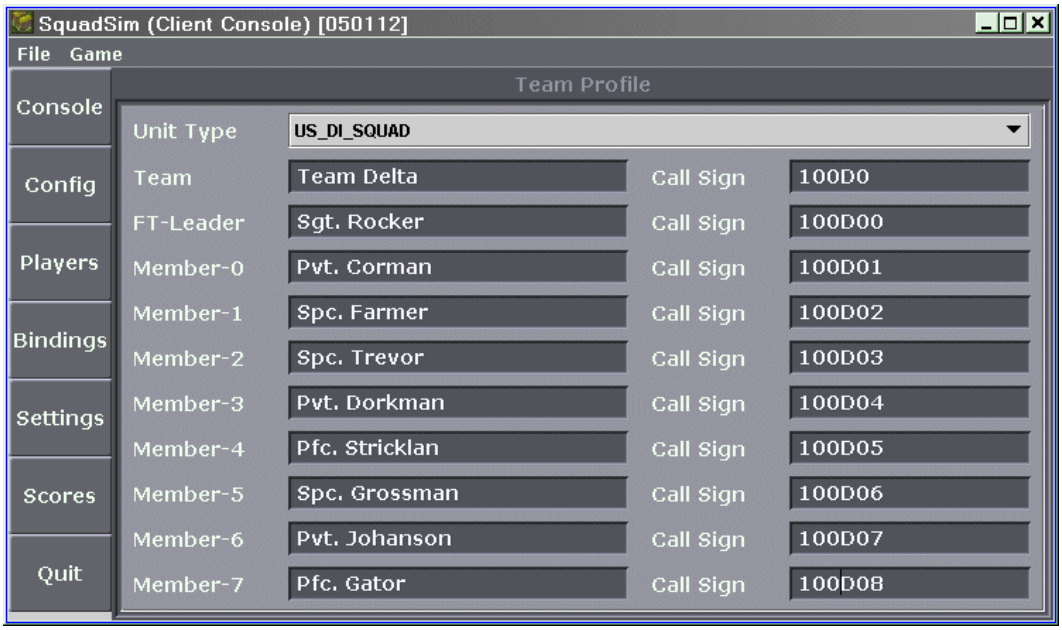

Figure A-3. Players panel.

A.4.4 Bindings Panel Operation

The Bindings panel is for configuring key bindings in SquadSim. To change a setting, click the left mouse button on the middle column of the associated row of the table. This will highlight that row and darken the middle column entry. Next, press the keyboard or mouse button that you would like to bind, and it will replace that entry. Note that if a particular key or button is already bound, you cannot assign it. If desired, you can change the binding to free up that key/button by changing it first. In figure A-4, the Bindings panel is shown and the user has just bound the right mouse button to the forward movement function. Once all fields are filled in, left click on the File menu button on the top menu bar and select Save Settings. Note that to have any effect, the bindings must be saved before SquadSim is started.

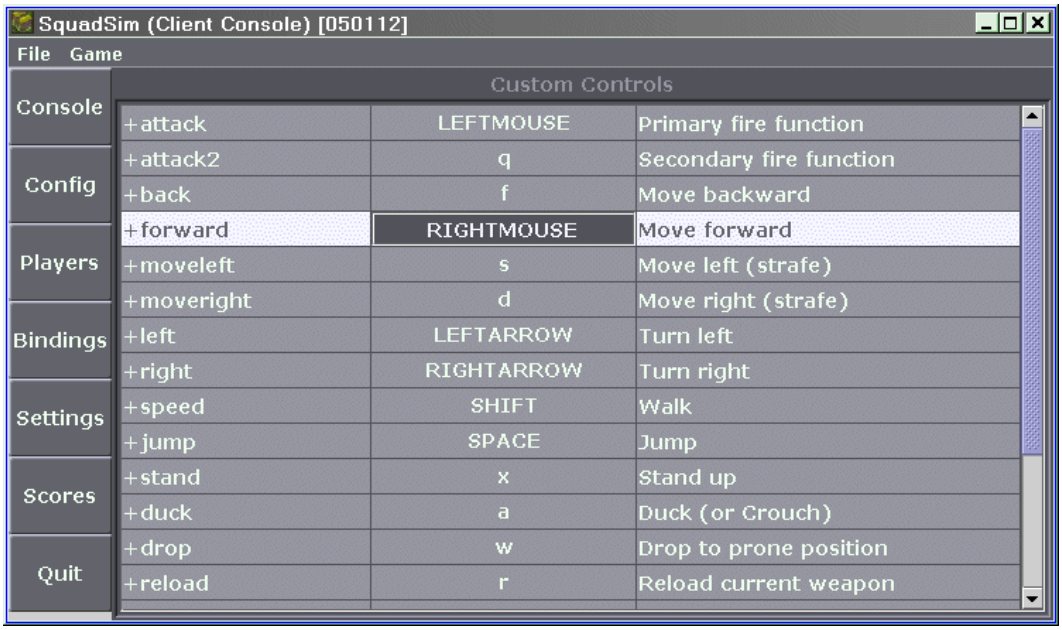

Figure A-4. Bindings panel (binding the right mouse button to forward movement).

A.4.5 Settings Panel Operation

This panel is for the dynamic control of environmental effects and team formations in SquadSim (see figure A-5). Moving the Fog Settings or Time of Day sliders has an immediate effect, so that you can see a continuous change as you operate the control.

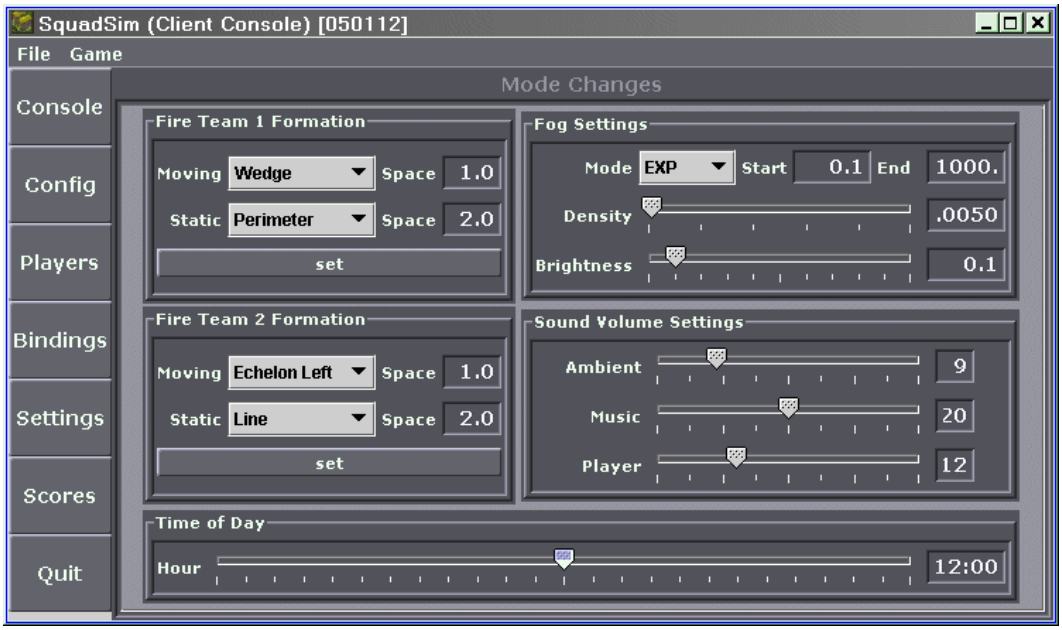

Figure A-5. Settings panel.

For the Fog Settings, Start and End parameters are only relevant when the Linear Mode is selected. When setting fire team formations, keep in mind that each team has an equal number of members (four at the time of this writing, but subject to change). There is a *base* team, called Fire Team 1, and a *rear* team, labeled Fire Team 2. The squad leader is not part of a formation, and typically is centered between the teams. To configure a team formation, select both the Moving and Static formations and adjust the spacing (to the right of the associated menu). Spacing refers to the distance in meters between adjacent team members and can be changed by typing in the text box labeled Space. Note that changes to a formation will not take effect until the associated set button is pressed. Select Save Settings from the File menu to preserve this configuration for the next time you run SS-CC or SquadSim, otherwise they will only effect the current session. Note that at the time of this writing, SquadSim does not support the Sound Volume Settings.

A.4.6 Scores Panel Operation

This panel is for information purposes only. It displays a table of squads (teams) at the top that contains the squad's team name, unit type, leader name, call sign, and latency. The latency is the number of milliseconds it takes for a network packet to make it to the game server and back. As depicted in figure A-6, selecting a row of the table will display that squad's member table at the bottom of the panel, including the number and type of rounds of ammo each has for the current weapon in use.

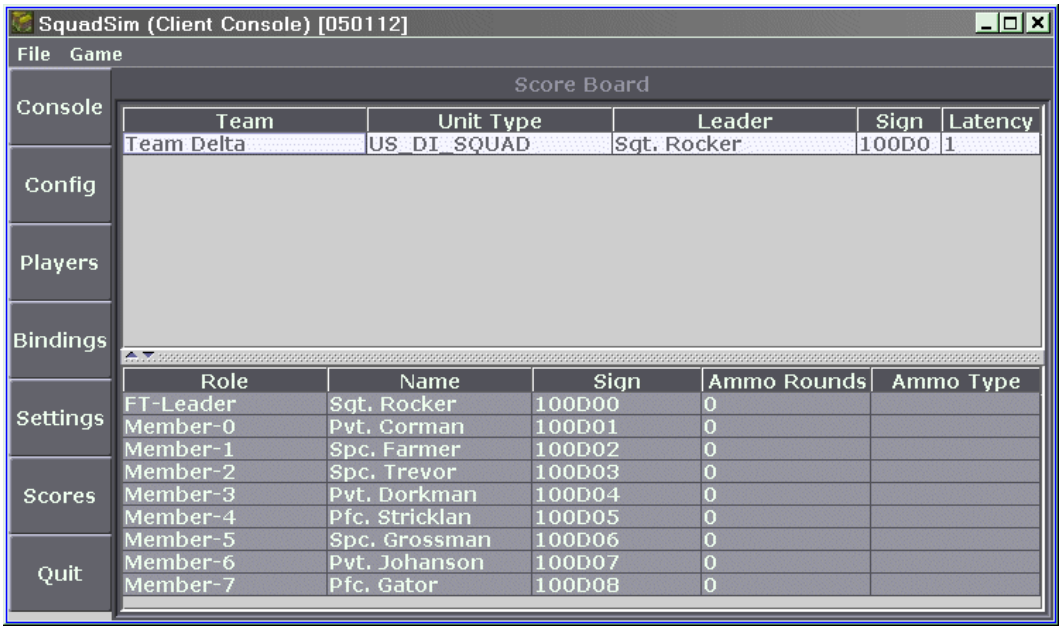

Figure A-6. Scores panel (Team Delta is selected).

A.5 Starting SquadSim

If you have read the previous sections and configured all settings, you are ready to start the game. Simply click on the Game menu in the menu bar at the top and select New. If you have configured things correctly, SquadSim should come up in a new window on your desktop. All standard error and standard out messages from SquadSim will appear in the Console panel. Please refer to separate documentation for instructions on the operation of SquadSim.

A.6 Exiting SS-CC

SquadSim should be shut down prior to exiting the Client Console. Once SquadSim has exited properly, click on the Quit button or select Exit from the File menu. This procedure ensures that all network communications, including the connection to the SS-SC, are terminated in an orderly fashion. Otherwise, the SS-SC and other services (i.e., the high-level architecture [HLA] Run-time infrastructure) can become inoperable.

A.7 Example userconfig.cfg File

```
// Filename: userconfig.cfg - generated by SquadSim (User Station) [050112].

// Player profiles
UnitType "US_DI_SQUAD"
Team "Team Delta" "100D0"
FT-Leader "Sgt. Rocker" "100D00"
Member-0 "Pvt. Corman" "100D01"
Member-1 "Spc. Farmer" "100D02"
Member-2 "Spc. Trevor" "100D03"
Member-3 "Pvt. Dorkman" "100D04"
Member-4 "Pfc. Stricklan" "100D05"
Member-5 "Spc. Grossman" "100D06"
Member-6 "Pvt. Johanson" "100D07"
Member-7 "Pfc. Gator" "100D08"

// Key Bindings
bind "LEFTMOUSE" "+attack"
bind "q" "+attack2"
bind "f" "+back"
bind "RIGHTMOUSE" "+forward"
bind "s" "+moveleft"
bind "d" "+moveright"
bind "LEFTARROW" "+left"
bind "RIGHTARROW" "+right"
bind "SHIFT" "+speed"
bind "SPACE" "+jump"
bind "x" "+stand"
bind "a" "+duck"
```

```
bind "w" "+drop"
bind "r" "+reload"
bind "e" "use"
bind "MIDDLEMOUSE" "mouselook"
bind "u" "messagemode"
bind "m" "messagemode2"
bind "p" "pic"
bind "`" "toggleview"

// Client Settings
set "client_port" "12081"
set "nearclip" "0.1"
set "farclip" "500.0"

// Server Settings
set "server_host" "section1.arl.army.mil"
set "server_port" "12080"

// HLA Configuration
set "hla_federate" "SquadSim"
set "hla_fed_file" "/home/moss/Matrex/file.fed"

// Team Formation settings
setf "team_1_move_formation" "Wedge"
setf "team_1_move_spacing" "1.0"
setf "team_1_stat_formation" "Perimeter"
setf "team_1_stat_spacing" "1.0"
setf "team_2_move_formation" "Wedge"
setf "team_2_move_spacing" "1.0"
setf "team_2_stat_formation" "Perimeter"
setf "team_2_stat_spacing" "1.0"

// Fog Settings
env "fog_mode" "LINEAR"
env "fog_density" "0.065"
env "fog_brightness" "0.3"
env "fog_start" "0.4"
env "fog_end" "4000.0"

// Sound Volume
env "volume_ambient" "10"
env "volume_music" "10"
env "volume_player" "10"

// Time Of Day
env "time_of_day" "14"
```

Appendix B. User Guide to the SquadSim Server Console (SS-SC)

B.1 Introduction

SS-SC is a server-side Java application that centrally manages one or more instances of SquadSim via communications with the associated SquadSim Client Consoles (SS-CCs) (see appendix A). The role of the server console is to manage connections from client consoles, facilitate interclient communication, maintain stats and ground truth information about a game, and facilitate administrative operations.

This guide will walk you through how to configure the server console, which must be running prior to starting any client consoles. To begin, depending on whether you are on a Microsoft Windows or a UNIX* platform, skip to the appropriate section on starting the server.

B.2 Starting SS-SC Under Microsoft Windows

Make sure there is a java.exe file in your execution Path. Locate the file named Server.bat in the SquadSim\Java folder. If it has not been taken care of during installation, check that the definition of ProjectDir on the first line of Server.bat is set to the folder where it is installed. Once the batch script is configured, execute it by either:

1. Double clicking on the Server.bat icon in Windows Explorer.
2. Starting up a DOS command window, navigating to the SquadSim\Java folder in the DOS window, and then typing *Server*.
3. Making a shortcut on your desktop to the batch script, then double clicking on the icon.

B.3 Starting SS-SC Under UNIX

Make sure there is a java.exe file in your execution Path. Locate the file named Server.sh in the SquadSim/Java directory. If it has not been taken care of during installation, check that the definition of InstallRoot on the first line of Server.sh is set to the directory where it is installed. Also check that the definition of CLASSPATH on the next line of the shell script is correct. Once the shell script is configured, execute it by either:

1. Double clicking on the Server.sh icon from a file browser.
2. Starting up a UNIX terminal emulation window, navigating to the SquadSim/Java directory, and then typing: *sh Server.sh.*

* UNIX is a registered trademark of The Open Group.

3. Creating a launcher on your desktop which points to Server.sh, then double-clicking on the icon.

B.4 Layout of the Server Console

Figure B-1 shows the SS-SC when it first initializes. The interface is divided into three distinct sections—A menu bar across the top, a column of buttons down the left side, and a panel comprising the remainder of the window. Each of the buttons, with the exception of Quit, selects which panel is displayed; only one panel is viewable at any given time. To change panels, click on one of the buttons on the left. The following sections describe the operation of each panel.

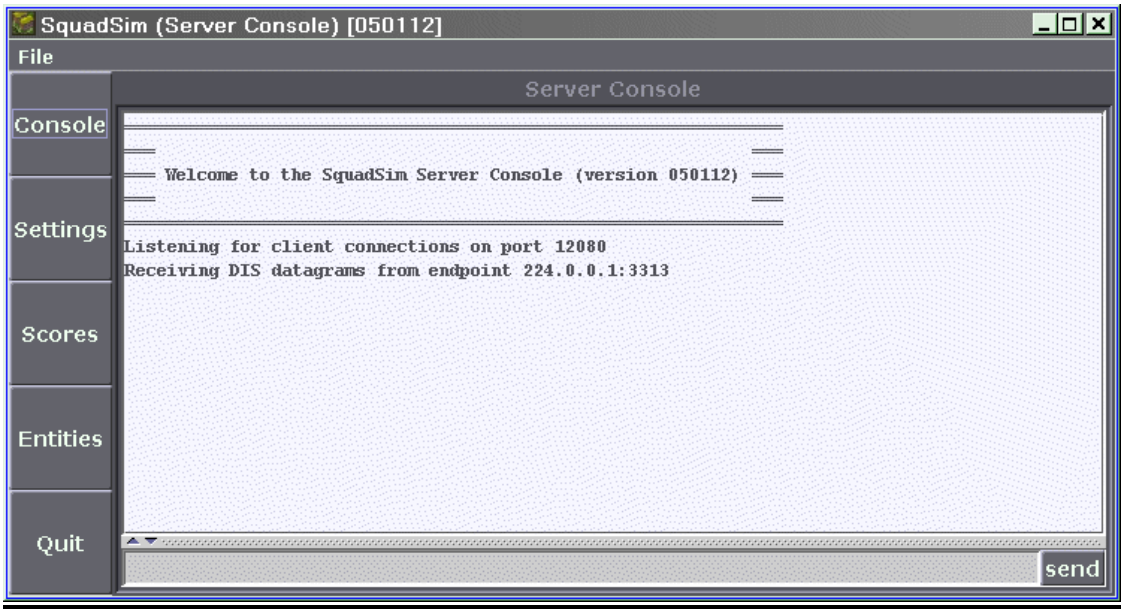

Figure B-1. SS-SC initialization (Console panel).

B.4.1 Console Panel Operation

In figure B-1, the Console panel is displayed. The console displays diagnostic messages. It also has a one-line window at the bottom for sending chat messages. These messages will be displayed on each instance of SS-CC that has connected. To send a chat message, type the message in this window and then click the Send button on the right.

B.4.2 Settings Panel Operation

This panel is for configuring the server console. These settings are only read at client console startup and will not affect the operation of the clients once they are running. Figure B-2 shows what this panel looks like. All fields must be filled in, except *HLA* and *DIS* settings are only relevant if the corresponding protocol is selected. Settings are described next in the order that they appear:

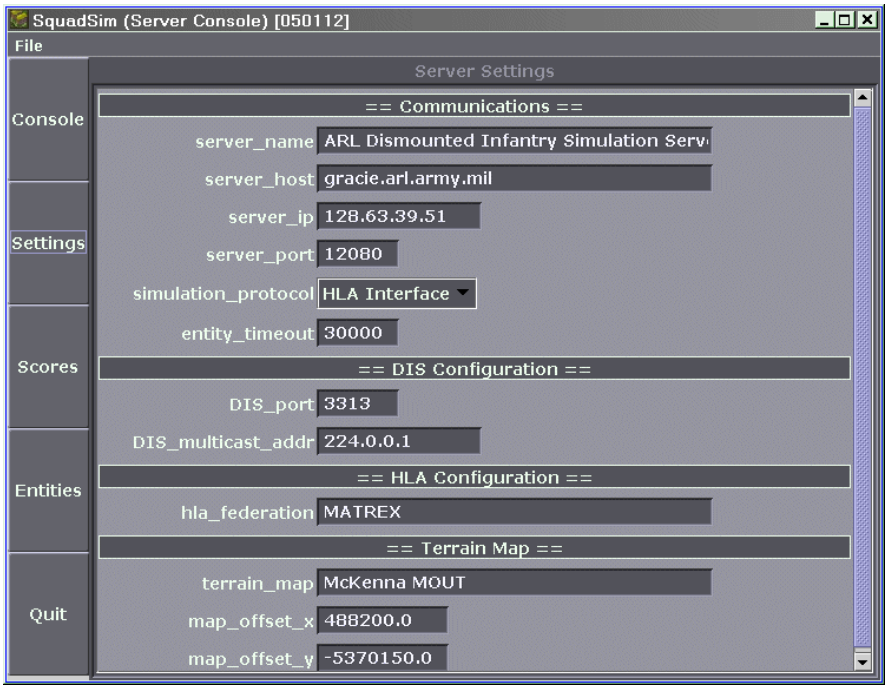

Figure B-2. Settings panel.

== Communications ==

- server_name – name of the server. Should be unique to the computer where it is hosted.
- server_host – the fully qualified host name or IP of the host computer.
- server_ip – the internet host IP (dotted-quad, e.g., 192.168.1.100).
- server_port – a port number to use for socket communications with SS-CC.
- simulation_protocol – drop down box, choices are *HLA Interface* and *DIS Protocol*.
- entity_timeout – entities will be removed if not heard from in this interval (specified in milliseconds).

== DIS Configuration ==

- DIS_port – port number to use for DIS multicast, must agree with *SquadSim* setting and any other relevant simulations.
- DIS_multicast_addr – multicast address to use (typically 224.0.0.1), must agree with other relevant simulations.

== HLA Configuration ==

- HLA_federation – name of HLA federation, must agree with other applications that are joining the federation (case sensitive).

== Terrain Map ==

- terrain_map – the name of the terrain data base (*SquadSim* uses *OpenFlt* format databases).
- map_offset_x – UTM offset (*X* coordinate) of the map.
- map_offset_y – UTM offset (*Y* coordinate) of the map.

Once all fields are filled in, left click on the File menu button on the top menu bar and select Save Settings. All configuration settings are saved in a file called *serverconfig.cfg* in the application directory. For UNIX systems, this is under *$HOME/SquadSim,* and for Microsoft Windows, it is under the standard application data folder (i.e., *C:\Documents and Settings\<username>\Application Data\SquadSim*). See the *serverconfig.cfg* example at the end of this appendix. Note that when a client (SS-CC) connects, a copy of this file called *remote-serverconfig.cfg* is downloaded to the client and is read by SquadSim. This enables the global control of high-level architecture (HLA) and distributed interactive simulation (DIS) configuration from SS-SC. Only two client-specific HLA settings are configured by SS-CC.

B.4.3 Scores Panel Operation

This panel is for information purposes only. It displays a table of squads (teams) at the top that contains the squad's team name, unit type, leader name, call sign, and latency. The latency is the number of milliseconds it takes for a network packet to make it from the server to the client and back. As depicted in figure B-3, selecting a row of the table will display that squad's member table at the bottom of the panel, including the number of rounds of ammo each has for the current weapon in use.

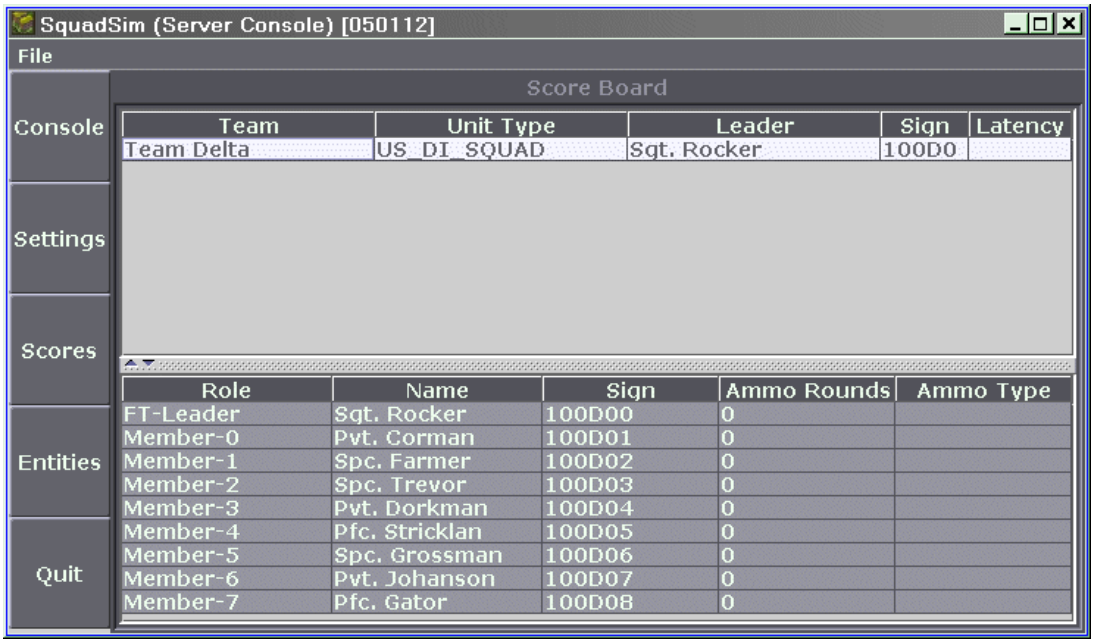

Figure B-3. Scores panel (Team Delta is selected).

B.4.4 Entities Panel Operation

This panel is a two-dimensional display of relative player positions. Each player is displayed as a simple arrowhead-shaped graphic which points in the direction the entity is heading. The color of each graphic indicates the unit type (i.e., blue is for US, red for USSR, and yellow for OTHER). The extents of the area being displayed are shown at the bottom in local coordinates.

When the mouse cursor is inside the Entities panel's display area, the cursor changes from the default pointer shape to a "+" shape.

To zoom out, click the right mouse button while the cursor is in the panel. The extents of the display will increase (so more area is visible) and become centered about the location corresponding to the mouse cursor position when the button was pressed. This is useful for locating players that may be beyond the current extents. There is a minimum size for the player graphic, which guarantees that they will be visible at any magnification; however, normally the size reflects the zoom (magnification) level. To zoom in, click the left mouse button. The extents will decrease, and the display will center about the location corresponding to the mouse cursor position. To zoom in to a particular area of the display, press and hold down the middle mouse button at a corner of the area, then drag the cursor diagonally to sweep out a rectangular region. Release the middle mouse button to complete the operation. The display extents will then zoom to fit that area as closely as possible to the frame of the Entities panel while preserving a one-to-one aspect ratio in the easting and northing (horizontal and vertical display) directions. Note that optimum results are obtained by describing a rectangle that is congruent to the shape of the panel.

To identify a player, position the mouse cursor over the player's graphic. This will display a unique name and/or call sign and coordinates of that entity.

Figure B-4 shows what this panel looks like. Note that currently the panel uses DIS protocol, so DIS must be enabled for the panel to display players. Also, it will not only display players but any DIS entities visible on the specified port and multicast address.

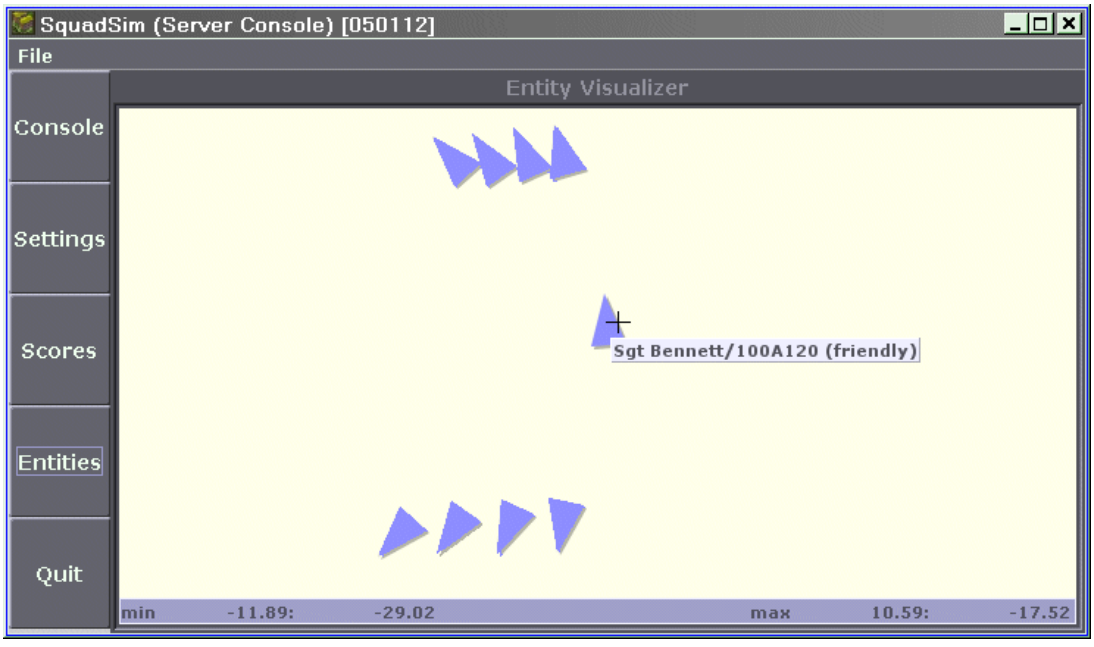

Figure B-4. Entities panel (cursor over entity pops up identifying information).

B.5 Exiting SS-SC

All clients consoles should be shut down prior to exiting the server, otherwise client instances will not operate properly and will have to be restarted. To shut down the server, click on the Quit button, or select Exit from the File menu.

B.6 Example serverconfig.cfg File

```
// Filename: serverconfig.cfg - generated by SquadSim (Server) [050112].


// Communications Settings
set "server_name" "ARL Dismounted Infantry Simulation Server"
set "server_host" "gracie.arl.army.mil"
set "server_ip" "128.63.39.51"
set "server_port" "12080"
set "simulation_protocol" "HLA Interface"
set "entity_timeout" "30000"

// DIS Configuration
set "DIS_port" "3313"
set "DIS_multicast_addr" "224.0.0.1"

// HLA Configuration
set "hla_federation" "MATREX"

// Terrain Map Settings
set "terrain_map" "McKenna MOUT"
set "map_offset_x" "488200.0"
set "map_offset_y" "-5370150.0"
```

Appendix C. SquadSim Network Packet Protocol Specification

The message traffic between the SquadSim Client Consoles (SS-CC) and the SquadSim Server Console (SS-SC), and also between the SS-CC and SquadSim, is composed of text strings (sequences of 1-byte ASCII characters). Each packet is terminated by a new line character. A packet is composed of the following concatenated parts:

1. The packet type, which is a 1-byte value converted to a string representation of the decimal value (e.g., 0x0A becomes "10").
2. A double-colon delimiter (i.e., "::").
3. A payload, which is a string representing one or more fields separated by double semicolons (i.e., ";;"). The payload is empty (zero-length) for some packet types.

Table C-1 describes the packet protocol for illustrative purposes. It is not intended to be detailed enough to fully describe the formatting of the payload data and is subject to change as the software matures.

Table C-1. Packet protocol.

Packet Name	Packet Type	Payload	Routing
ConnectionRequest	0x01	Hostname	Client to server
ConnectionComplete	0x02	ServerName	Server to client
ChatMessage	0x03	ClientMessage ServerMessage or (Replicated payload)	Client to server Server to all clients
DisconnectRequest	0x04	(*empty*) Hostname	SquadSim to client Client to server
InfoTeam	0x07	{Name, UnitType, CallSign, MemberCount}	Client to server
InfoLeader	0x08	{Role, LeaderName, CallSign}	Client to server
InfoMember	0x09	{Role, MemberName, CallSign}	Client to server
PingRequest	0x0A	Time of Request	Client to server
PingResponse	0x0B	(Replicated payload from PingRequest)	Server to client
TeamStatus	0x0C	Name, UnitType, CallSign, LeaderName, Latency	Server to all clients

Table C-1. Packet protocol (continued).

Packet Name	Packet Type	Payload	Routing
PlayerStatus	0x0D	{TeamName, {MemberName, CallSign, RoundsLeft, AmmoType }(0 .. N)}	Server to all clients
ViewerInit	0x0E		SquadSim to client
AmmoStatus	0x10	{MemberName, AmmoType, RoundsLeft} (Replicated payload)	SquadSim to client Client to server
TeamFormation	0x11	{TeamIdent, MovingFormationDescriptor, MovingFormationSeparation, StaticFormationDescriptor, StaticFormationSeparation}	Client to SquadSim
FogSettings	0x12	{FogMode, StartVal, EndVal, Density, Brightness}	Client to SquadSim
TimeOfDay	0x13	Hour	Client to SquadSim
VolumeSettings	0x14	{AmbientVolume, MusicVolume, PlayerVolume}	Client to SquadSim
Activate	0x15	{X, Y, Z, Radius} (Replicated payload) (Replicated payload) (Replicated payload)	SquadSim to client Client to server Server to all clients Client to SquadSim
ServerConfRequest	0x16		Client to server
ServerConfResponse	0x17	ServerConfigFile	Server to client
Appearance	0x18	{CallSign,EnumerationVal} (Replicated payload) (Replicated payload) (Replicated payload)	SquadSim to client Client to server Server to all clients Client to SquadSim

ConnectionRequest – This packet is sent by the client to the server. It establishes a connection with the server. This is referred to as "joining" the game.

ConnectionComplete – This packet is sent from the server to the client. It indicates that the client has successfully connected and can commence network communications.

ChatMessage – This packet is either initiated by a user of a SquadSim client console or by an administrator at the server console. If initiated from the client, it goes to the server, and the server copies it to all clients (including the originating client). When initiated by a server administrator, it is sent to all clients. In either case, the message is prefixed by an identifier indicating the originator, i.e., by squad leader name or server name. Currently, all messages appear in the console panel of all clients and the server.

DisconnectRequest – This packet is sent from SquadSim to the client console and also from the client to the server. When sent by the client, it contains the hostname where the client is running. The purpose is to notify the recipient that the sender is shutting down, to allow for the disposition of associated resources and information.

InfoTeam – This packet is sent from client to server after joining to provide information about the squad. It is immediately followed by the InfoLeader packet and then InfoMember packets for each automated squad member.

InfoLeader – This packet is sent from client to server to provide identifying information regarding the squad leader. It must be sent immediately after the InfoTeam packet.

InfoMember – This packet is similar to the InfoLeader packet but identifies an automated squad member. One is sent for each member immediately following the InfoLeader packet.

PingRequest – This packet is sent from client to server to compute network latency.

PingResponse – This packet is returned by the server in response to a PingRequest.

TeamStatus – This packet is sent from server to all clients. This takes place when any client joins and every time a PingRequest is processed.

PlayerStatus – This packet is sent from server to all clients. This takes place when any client joins and every time an AmmoStatus packet is received.

ViewerInit – This packet is sent from SquadSim to the client console. Its purpose is to notify the console that SquadSim has finished initializing its graphics engine and it is safe to send configuration packets that depend upon this resource being available.

AmmoStatus – This packet is sent from SquadSim to the client console which reroutes the packet to the server (which generates PlayerStatus packets down to all clients). It updates a squad member's ammo count.

TeamFormation – This packet is sent from the client console to SquadSim. It conveys the preferred formation of a specific fire team. Each team has a static and a moving formation type and separation distance. This packet is sent after the initial ViewerInit packet is received and then whenever the user of the client console submits a change.

FogSettings – This packet is sent from the client console to SquadSim. Its purpose is to specify fog parameters. It is sent after the initial ViewerInit packet is received and subsequently whenever the user of the client console changes a setting. If the user moves either the "Density" or "Brightness" slider, streams of these packets are sent so that the affect on the virtual environment is seen in real time.

TimeOfDay – This packet is sent from the client console to SquadSim. Its purpose is to specify the hour of the day (using a 24-hour clock) so that daylight can be simulated. When the user moves the "Hour" slider, streams of these packets are sent so the effect is seen in real time.

VolumeSettings – This packet is sent from the client console to SquadSim. Its purpose is to adjust sound volume for ambient, music, and player sounds. As a slider is adjusted by the user,

streams of packets are sent so that the affect is heard in real time. (Note: SquadSim does not currently support this packet.)

Activate – This packet is originated by a SquadSim instance and travels to the client console which routes it to the server, which, in turn, propagates it to all clients. The clients reroute it down to their respective SquadSim instances. The purpose of this packet is to notify all clients of a change in state of an *activation node* of the terrain database. These nodes are used by the graphics engine to represent actuated objects such as a door or window.

ServerConfRequest – This packet is sent from the client console to the server to notify the server that it is ready to receive configuration information. This packet is sent once, soon after the client has successfully joined.

ServerConfResponse – This packet is sent by the server in response to a ServerConfRequest. The purpose of this packet is to send the server configuration down to the requesting client so that certain settings can be centrally managed by the server administrator. These settings are only conveyed at client initialization time.

Appearance – Like the Activate packet, this packet is originated by a SquadSim instance and travels to the client console, to the server, down to all client consoles, and finally down to their respective SquadSim instances. The purpose is to notify all clients of changes to a player's stance code (i.e., standing, kneeling, prone, etc.).

NO. OF COPIES ORGANIZATION

1 (PDF ONLY) DEFENSE TECHNICAL INFORMATION CTR
DTIC OCA
8725 JOHN J KINGMAN RD
STE 0944
FORT BELVOIR VA 22060-6218

1 US ARMY RSRCH DEV & ENGRG CMD
SYSTEMS OF SYSTEMS INTEGRATION
AMSRD SS T
6000 6TH ST STE 100
FORT BELVOIR VA 22060-5608

1 INST FOR ADVNCD TCHNLGY
THE UNIV OF TEXAS
AT AUSTIN
3925 W BRAKER LN STE 400
AUSTIN TX 78759-5316

1 US MILITARY ACADEMY
MATH SCI CTR EXCELLENCE
MADN MATH
THAYER HALL
WEST POINT NY 10996-1786

1 DIRECTOR
US ARMY RESEARCH LAB
IMNE ALC IMS
2800 POWDER MILL RD
ADELPHI MD 20783-1197

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRD ARL CI OK TL
2800 POWDER MILL RD
ADELPHI MD 20783-1197

3 DIRECTOR
US ARMY RESEARCH LAB
AMSRD ARL CS IS T
2800 POWDER MILL RD
ADELPHI MD 20783-1197

NO. OF COPIES ORGANIZATION

ABERDEEN PROVING GROUND

1 DIR USARL
AMSRD ARL CI OK TP (BLDG 4600)

NO. OF
COPIES ORGANIZATION

1 USA SBCCOM
NATICK SOLDIER CTR
AMSSB RSS MA (N)
D TUCKER
KANSAS ST
NATICK MA 01760-5020

1 US ARMY RSCH INST
B KNERR
12350 RSCH PKWY
ORLANDO FL 32826-3236

1 RSCH DEV & ENGR COMMAND
J GROSSE
12423 RSCH PKWY
ORLANDO FL 32828

1 US ARMY RSCH LAB
AMSRD ARL CI C
L TOKARCIK
ADELPHI MD 20783

1 US ARMY RSCH LAB
AMSRD ARL CI CB
R WINKLER
ADELPHI MD 20783

ABERDEEN PROVING GROUND

10 DIR USARL
AMSRD ARL CI CT
M THOMAS
F BRUNDICK
M LOPEZ
G MOSS
P JONES
AMSRD ARL SL
R SANDMEYER
P TANENBAUM
AMSRD ARL SL BE
L BUTLER
C KENNEDY
AMSRD ARL WM BF
G SAUERBORN