NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

OPERATIONAL LOGISTICS WARGAME

by

Carolyn S. Fricke

December 2001

Thesis Advisor: Arnold H. Buss
Second Reader: Kevin J. Maher

Approved for public release; distribution is unlimited.

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
19 Dec 2001	N/A	-

Title and Subtitle Operational Logistics Wargame	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Fricke, Carolyn S.	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Naval Postgraduate School Monterey, California	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 119	

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Operational Logistics Wargame			**5. FUNDING NUMBERS**
6. AUTHOR(S) Carolyn S. Fricke			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

ABSTRACT (maximum 200 words)

This thesis provides an interactive wargame for use by students of Operational Logistics at the Naval Postgraduate School. The objective of the wargame is to show students how their decisions regarding resupply of combatant forces affect the ability of those forces to carry-out their wartime missions. The core programming of the Operational Logistics Wargame, as presented by this thesis, deals with a Carrier Battle Group and its missions of command of the sea and power projection ashore. Written in a modular fashion, the wargame can be expanded in scope at a later date to include other combatant missions and components such as submarines, amphibious forces, or ground forces. The modular design allows the wargame to have modifications made to it without alterations to components not directly involved. The wargame also draws data from an outside database by using Structured Query Language (SQL) and a Java Database Connectivity - Open Database Connectivity (JDBC-ODBC) Bridge. The wargame can be installed on most major operation systems. Other major design features of the wargame are Discrete Event Simulation and extensive use of Graphical User Interfaces (GUIs) for providing information to the player and obtaining information from the player.

14. SUBJECT TERMS Operational Logistics, Operations Research, Discrete Event Simulation, Wargame, Simulation, JDBC, JDBC-ODBC, GUI, Graphical User Interface, Java Swing, Java,	**15. NUMBER OF PAGES** 119
	16. PRICE CODE

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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

OPERATIONAL LOGISTICS WARGAME

Carolyn S. Fricke
Lieutenant Commander, United States Navy
B.S., University of Southwestern Louisiana, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 2001

Author: _____
Carolyn S. Fricke

Approved by: _____
Arnold H. Buss, Thesis Advisor

_____
Kevin J. Maher, Second Reader

_____
Professor James N. Eagle, Chairman
Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis provides an interactive wargame for use by students of Operational Logistics at the Naval Postgraduate School. The objective of the wargame is to show students how their decisions regarding resupply of combatant forces affect the ability of those forces to carry-out their wartime missions. The core programming of the Operational Logistics Wargame, as presented by this thesis, deals with a Carrier Battle Group and its missions of command of the sea and power projection ashore. Written in a modular fashion, the wargame can be expanded in scope at a later date to include other combatant missions and components such as submarines, amphibious forces, or ground forces. The modular design allows the wargame to have modifications made to it without alterations to components not directly involved. The wargame also draws data from an outside database by using Structured Query Language (SQL) and a JDBC - Open Database Connectivity (JDBC-ODBC) Bridge. The wargame can be installed on most major operating systems. Other major design features of the wargame are Discrete Event Simulation and extensive use of Graphical User Interfaces (GUIs) for providing information to the player and obtaining information from the player.

THIS PAGE INTENTIONALLY LEFT BLANK

DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the player.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS, ACRONYMS AND/OR ABBREVIATIONS

BDA	Battle Damage Assessment
CIWS	Close In Weapons System
CLF	Combat Logistics Force
CVBG	Carrier Battle Group
DSN	Data Source Name
FIFO	First-In, First-Out
GPF	General Planning Factors
GUI	Graphical User Interface
IDE	Integrated Development Environment
JDBC	Acronym has no meaning, JDBC is stand-alone phrase
JDBC-ODBC	JDBC-Open Database Connectivity
ODBC	Open Database Connectivity
Oplog	Operational Logistics
Pro-Log	A Fortran-based forerunner of the Operational Logistics Wargame
RAS	Replenishment at Sea
SQL	Structured Query Language
TACLOGS	Tactical Logistics Support System
Unrep	Underway Replenishment

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Military personnel at all levels play wargames to 'experience the realities of decision making.' (Perla, 302) Students of Operational Logistics at the Naval Postgraduate School play wargames for the same reason. A major issue in the development of a military wargame is the balance between realism and playability. Another issue is whether a particular game meets the training needs of specific users. Since a single wargame is not capable of being realistic and playable while still meeting the needs of all possible users, wargames are often developed to meet the needs of a specific group of end users. Operational Logistics students are one such specific group of end users. While there are a multitude of modern wargames and combat simulations, the vast majority do not include logistics in the game play. The few models that do include a logistics component (such as JWARS) require users to have a considerable amount of training to obtain a basic working knowledge of the game's operation. They also have an extremely detailed level of resolution and place the main emphasis on combat rather than logistics. Thus, these existing models are not suitable for basic Operational Logistics training.

Students of Operational Logistics at the Naval Postgraduate School take a required course on the fundamentals of the Naval Logistics system. Part of the course is spent using interactive computer simulations as training aids to better understand the material being taught. A forerunner of the Operational Logistics Wargame, called PRO-LOG, was developed in the 1980's by NPS students and instructors to acquaint Operational Logistics students with logistical concepts. PRO-LOG is a deterministic combat model with a single scripted scenario. It has virtually no graphic capabilities and has user interfaces that are cumbersome and difficult to use. The wargame was written in Fortran, an archaic language, and would require extensive program code changes to modify the wargame structure at the most basic level. While PRO-LOG met the needs of students at the time, advances in modeling combat techniques and computer technology have made PRO-LOG obsolete; however, the reasons for PRO-LOG's development still exist: logisticians, like warfighters, enhance their wartime capabilities by practicing with true-to-life simulations. Although many basic combat models available today seem

similar to this simulation, the similarity is on the surface only and extends only to the combat between the forces. Most basic models do not include logistics in the game play. Including logistics in a model requires that logistical considerations be incorporated from the very beginning. Retrofit of an existing combat model to include logistics simply isn't viable.

In order to meet the continuing goal of training Logistics Officers to make effective decisions in a combat situation, the introductory Operational Logistics course needed a modern replacement for PRO-LOG. To be a worthwhile replacement, the new wargame needed to be written in modern code using the latest Operations Research simulation techniques. It needed to be configurable, expandable, and stochastic. User interfaces needed to be user-friendly. And, the amount of time needed to train players needed to be equitable with the length of time that the Operational Logistics course devotes to the wargame. The Operational Logistics Wargame exceeds all qualifications desired in a replacement for PRO-LOG. It will assist Operational Logistics students in understanding their roles.

The core programming of the Operational Logistics Wargame, as presented by this thesis, deals with a Carrier Battle Group and its missions of command of the sea and power projection ashore. Written in the Java programming language and in a modular fashion, the wargame can be expanded in scope at a later date to include other combatant missions and components such as submarines, amphibious forces, or ground forces. The modular design also allows the wargame to have modifications made to it without alterations to components not directly involved. Component modifications and additions can be made in future versions that make the wargame more complex and robust.

The wargame draws data from an outside database using Structured Query Language (SQL) and a "JDBC" driver. The portability of the Java program language allows the wargame to be run on most major operating systems. Other major design features of the wargame are Discrete Event Simulation and extensive use of Graphical User Interfaces (GUIs) for providing information to the player and obtaining information from the player.

The Operational Logistics Wargame is intended as an introduction to Operational Logistics only. It is a precursor to more complex and challenging wargames encountered in Joint and Combined Logistics and Logistics Modeling coursework.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENT

I would like to thank Professor Arnold H. Buss for his outstanding guidance and his belief in my programming skills. I also wish to thank Professor David A. Schrady for his superior assistance and support throughout the thesis process. I would also like to express my appreciation to Commander Kevin J. Maher, Supply Corp, U. S. Navy for recommending this topic. Furthermore, I want to thank Captain Thomas Erlenbruch, German Army for his great troubleshooting assistance in the Geek Lab. I still don't know whether to thank or condemn all my other classmates for encouraging me to be a geek but it sure was interesting getting there.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Historians aren't sure when or why the first wargame was invented; but, they do know that wargames have existed throughout the recorded history of mankind. Contemporaries of the Chinese general and military philosopher Sun Tzu played his wargame *Wei Hai* about 3000 B.C. Christopher Weikhmann's peers played his wargame Koenigspiel at the end of the 17th century. Modern wargamers play wargames as (seemingly) simple as chess, as complex as JWARS, and wargames of any level of complexity in between. Although vastly different in appearance, entertainment value, rules, and complexity most of these wargames have a common thread running through them: their utility as training devices.

Military personnel at all levels play wargames to 'experience the realities of decision making.' (Perla, 302) Students of Operational Logistics at the Naval Postgraduate School play wargames for the same reason. A major issue in the development of a military wargame is the balance between realism and playability. Another issue is whether a particular game meets the training needs of specific users. Since a single wargame is not capable of being realistic and playable while still meeting the needs of all possible users, wargames are often developed to meet the needs of a specific group of end users. Operational Logistics students are one such specific group of end users. While there are a multitude of modern wargames and combat simulations, the vast majority do not include logistics in the game play. The few models that do include a logistics component (such as JWARS) require users to have a considerable amount of training to obtain a basic working knowledge of the game's operation, have an extremely detailed level of resolution, and place the main emphasis on combat rather than logistics. Thus, they are not suitable for basic Operational Logistics training.

Students of Operational Logistics at the Naval Postgraduate School take a required course on the fundamentals of the Naval Logistics system. Part of the course is spent using interactive computer simulations as training aids to better understand the material being taught. A forerunner of the Operational Logistics Wargame called PRO-LOG was developed in the 1980's by NPS students and instructors to acquaint students

with logistical concepts. PRO-LOG is a deterministic combat model with a single scripted scenario. It has virtually no graphic capabilities and has user interfaces that are cumbersome and difficult to use. The wargame was written in Fortran, an archaic language, and would require extensive program code changes to modify the wargame structure at the most basic level. While PRO-LOG met the needs of students at the time, advances in modeling combat techniques and computer technology have made PRO-LOG obsolete; however, the reasons for PRO-LOG's development still exist: Logisticians, like warfighters , enhance their wartime capabilities by practicing with true-to-life simulations. Although many basic combat models available today seem similar to this simulation, the similarity is on the surface only and extends only to the combat between the forces. Most basic combat models do not include logistics in the game play. Including logistics in a model requires that logistical considerations be incorporated from the very beginning. Retrofit of an existing combat model to add logistics simply isn't viable.

In order to meet the continuing goal of training Logistics Officers to make effective decisions in a combat situation, the introductory Operational Logistics course needed a modern replacement for PRO-LOG. To be a worthwhile replacement, the new wargame needed to be written in modern code using the latest Operations Research simulation techniques. It needed to be configurable, expandable, and stochastic. User interfaces needed to be user-friendly. And, the amount of time needed to train players needed to be equitable with the length of time that the Operational Logistics course devotes to the wargame. The Operational Logistics Wargame exceeds all qualifications desired in a replacement for PRO-LOG. It will assist Operational Logistics students in understanding their roles.

The core programming of the Operational Logistics Wargame, as presented by this thesis, deals with a Carrier Battle Group and its missions of command of the sea and power projection ashore. Written in the Java programming language and in a modular fashion, the wargame can be expanded in scope at a later date to include other combatant missions and components such as submarines, amphibious forces, or ground forces. The modular design allows the wargame to have modifications made to it without alterations

to components not directly involved. Component modifications and additions can be made in future versions that make the wargame more complex and robust.

The wargame draws data from an outside database by using Structured Query Language (SQL) and a JDBC driver. Due to the portability of the Java programming, the wargame can be installed on most major operating systems. Other major design features of the wargame are Discrete Event Simulation and extensive use of Graphical User Interfaces (GUIs) for providing information to the player and obtaining information from the player.

The Operational Logistics Wargame is intended as an introduction to Operational Logistics only. It is a precursor to more complex and challenging wargames encountered in Joint & Combined Logistics and Logistics Modeling coursework.

THIS PAGE INTENTIONALLY LEFT BLANK

II. METHODOLOGY

The Operational Logistics Wargame is a combat model designed with the best Operations Research modeling and simulation techniques throughout. This chapter highlights key design factors of the wargame and is divided into two sections. The first section discusses the architecture of the combat model. The second section discusses the design of the external database, which is the main source of data on the Carrier Battle Group force structure.

A. MODELING AND JAVA PROGRAMMING

The core of the Operational Logistics Wargame is a discrete event simulation. This core programming is wrapped in a layer of Graphical User Interfaces (GUIs) to interact with the player and JDBC interfaces to obtain its data. The wargame provides output through the use of data GUIs, output screens, and an animated map display. Where appropriate, this simulation incorporates randomness for more realism than deterministic, scripted events like its predecessor PRO-LOG.

This section describes the overall structure of the Operational Logistics Wargame, and is intended to aid future Operational Logistics developers in modifying this wargame. Figure 1 is a view of the program flow among the classes and shows the general relationships. Each box in Figure 1 shows one or more Java classes that are related in functionality. The uppermost class shown in each box is the central class for each of the loosely knit groupings shown. The arrows in Figure 1 represent the flow of logic as the game unfolds. After the actions headed by the *CourseAndSpeed* class are completed, the *Deployment* class is instantiated. The *Deployment* class and its related classes comprise the discrete event simulation portion of the wargame. The discrete event simulation has significant interactions with the *DataRepository*, *ArrivalProcess*, and *Animate* groups during game play.

In addition to the broad groupings shown, the classes are divided into Java packages by functional area. Appendix A provides a more detailed description of

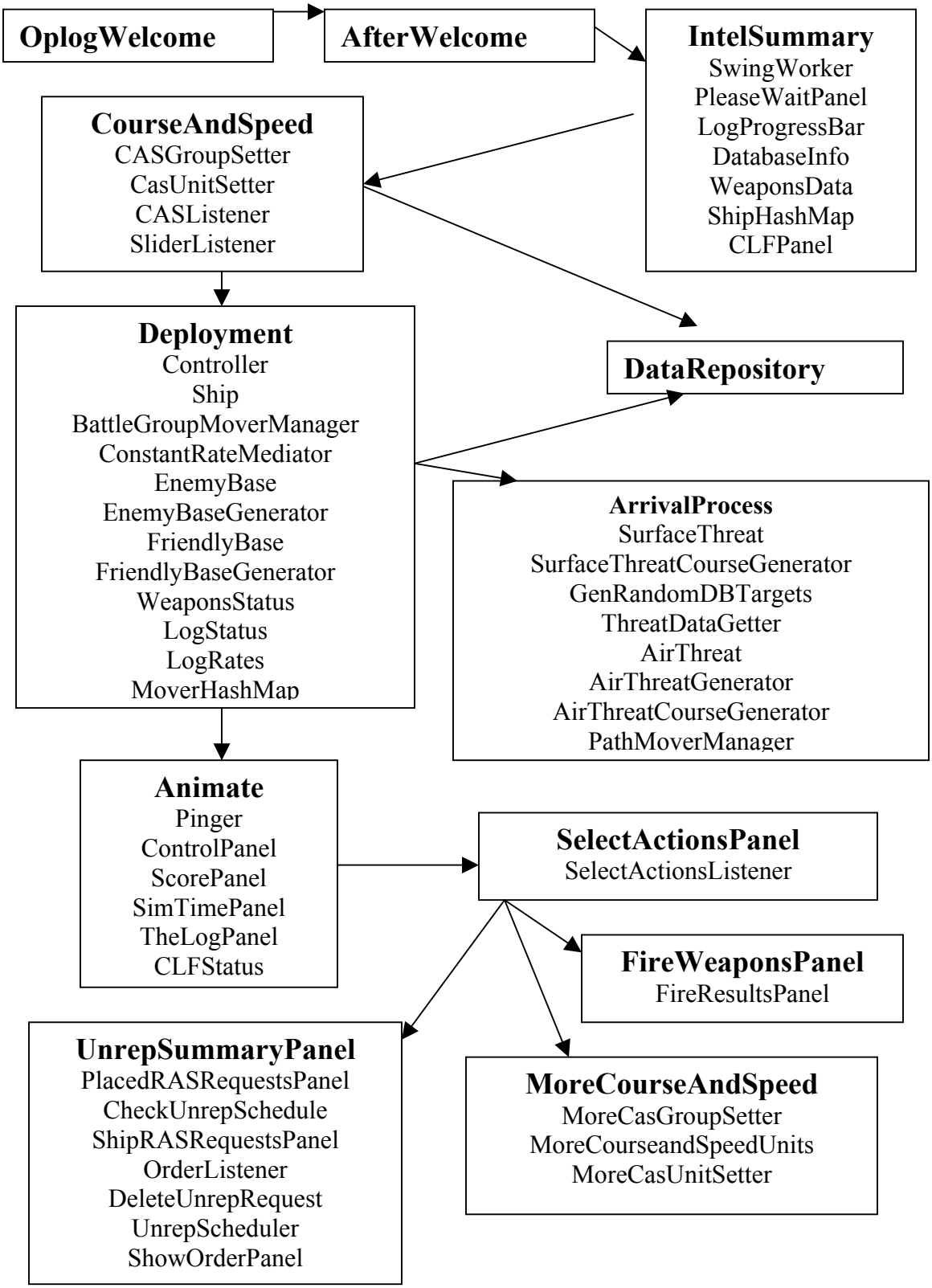

Figure 1. Wargame Programming Flow

individual classes. All classes are contained in the package *oplog* or in subpackages such as *oplog.database* for databases and related classes, *oplog.gui* for classes used to make graphical user interfaces, and *oplog.smd* for classes used by the discrete event simulation. Classes not discussed in the appendix are drawn from *java*, *javax*, and *simkit* packages. The details noted here about each class are considerably more in depth than those found in the Javadocs.

All program code and applicable Javadocs are included in some electronic versions of this thesis. Some classes that are not used in the current version of the Operational Logistics Wargame have also been included in those versions as a basis for future development.

B. DATABASE DEVELOPMENT

The databases used in the Operational Logistics Wargame were implemented in Microsoft Access but could have been in any SQL-compliant database with a JDBC driver. The databases were designed specifically for this wargame and contain information about forces assigned to both the Carrier Battle Group and enemy fleet. CVBG data is actual real-world ship characteristics.

1. CVBG Database

The Carrier Battle Group database, named *oplog.mdb*, contains current information on all U.S. Navy surface ships (both USS and USNS fleet assets) as found in *Jane's Online* and other unclassified sources. Information about each ship, such as its maximum speed, available weapons, sensors, and logistical capacities are based strictly on the ship's class and cross-referenced throughout the database by ship class. Minor differences between ships within a class are disregarded. Consumption of logistical items (except fuel) is based on typical Naval Logistics Planning Factors which, like consumption rates of other military branches, are calculated per person onboard per day. Propulsion fuel is calculated using unclassified fuel consumption equations while aviation fuel is calculated using a basic daily rate. The CVBG in the Operational Logistics Wargame does not yet have the ability to launch aircraft. Therefore, aviation fuel consumption is a basic daily rate rather than a basic daily rate plus a rate for number of sorties. The current version of the database contains several tables marked

"(original)". Those tables are intended for use when the wargame can be run in a larger Java Virtual Machine. Tables by the same name but not marked "(original)" are scaled down versions.

The organization of the database centers around the ships listed on the CVBG table (Table1). The "Ship" names, "Class" types and coordinates are all essential to the Operational Logistics Wargame. "Ship Hull Number" is not currently used, and "Number" is used to organize the data in the order desired for reports.

Ship	Ship Hull Number	Class	XCoord	YCoord	Number
George Washington	CVN 73	Nimitz	40	40	1
Bunker Hill	CG 52	Ticonderoga	60	60	2
Gettysburg	CG 64	Ticonderoga	0	40	3
Nicholson	DD 982	Spruance	20	0	4
Cole	DDG 67	Arleigh Burke	60	20	5
Rueben James	FFG 57	Perry	0	20	6
Halyburton	FFG 40	Perry	20	60	7
Blue Ridge	LCC 19	Blue Ridge	40	80	8
Supply	AOE 6	Supply	0	0	9

Table 1. CVBG Table

Nearly all data throughout the database is sourced using a ship's type. Using the class type, information about any ship in the CVBG can be accessed from the other tables. A table containing all U.S. Navy combatant ships and CLF ships, not including Amphibious ships, is included in the *oplog* database for reference (not shown). The TypeData Table (Table 2) provides a variety of basic data concerning ship class types. F76 fuel curve factors are included in the TypeData Table. The TypeSensors Table (Table 3) contains the various air and surface sensors found on each type of ship. The Sensors Table (Table 4) contains information specific to each sensor.

Class	Mission	Max Speed	Staying Power	Pers	F76p2	F76p1	F76p0	Notes
Kitty Hawk	carrier	32	5	5480	32.6666	-8937.6	10865.9	Fuel rate in KGal/hr
Kennedy	carrier	32	5	5480	32.6666	-8937.7	10865.9	capacity is bbls
Enterprise	carrier	33	5	5765	0	0	1	convert in code
Nimitz	carrier	30	5	5930	0	0	1	
Ticonderoga	combat	30	4	358	37.4831	-1429.04	2215.39	
Spruance	combat	33	4	339	27.0667	-1812.92	3097.97	
Arleigh Burke	combat	32	4	346	51.5925	-764.433	1379.62	
Arleigh Burke IIA	combat	31	4	344	51.5925	-764.433	1379.62	
Perry	combat	29	4	200	51.8843	-545.716	951.117	
Blue Ridge	combat	23	4	1095	112.9410	92.0583	699.553	
Austin	combat	21	3	666	95.4647	-1124.43	1566.79	
Sacramento	station	26	2	601	12.2579	-27553.4	27821.2	
Supply	station	26	2	531	-25.7866	12117.20	-12232.3	
Kilauea	shuttle	20	1	149	-8.86595	16343.7	-16150.3	
Mars	shuttle	20	1	176	55.5118	-1471.66	1727.46	
Sirius	shuttle	18	1	175	55.5118	-1471.66	1727.46	Sources: Schrady 1996
Kaiser	shuttle	20	1	104	-44.9642	4834.54	-4614.81	Janes Online 2001

Table 2. TypeData Table

Class	Air	Surface
Kitty Hawk	SPS48E	SPS67
Kennedy	SPS48E	SPS67
Enterprise	SPS48E	SPS67
Nimitz	SPS48E	SPS67V1
Ticonderoga	SPY1B	SPS55
Spruance	SPS40B	SPS55
Arleigh Burke	SPY1D	SPS67V3
Arleigh Burke IIA	SPY1D	SPS67V3
Perry	SPS49V5	SPS55
Blue Ridge	SPS48C	SPS65V1
Austin	SPS48C	SPS10F
Sacramento	SPS40E	SPS10F
Supply	MK23	SPS67
Kilauea	GenAir	GenNav
Mars	GenAir	GenNav
Sirius	GenAir	GenNav
Kaiser	GenAir	GenNav

Table 3. TypeSensors Table

Radar	Range
SPS48E	220
SPY1B	200
SPS40B	175
SPY1D	200
SPS49V5	250
SPS48C	220
SPS40E	175
MK23	25
SPS67	56
SPS67V1	56
SPS55	50
SPS67V3	56
SPS10F	54
SPS65V1	55
GenAir	25
GenNav	50

Table 4. Sensors Table

The TypeWeapons Table (Table 5) contains information about the weapons on each type of ship. Note the use of "none" rather than "0" for ships that do not carry certain weapons. To simplify inventory coding and calculations throughout the Operational Logistics Wargame, weapons that typically fire multiple projectiles in a single round are considered to have fired only one unit per firing event. The rates used to convert multiple projectile rounds are noted. Information about each weapon is found in the Weapons Table (Table 6). In the current version of the Operational Logistics Wargame, the use of published unclassified weapon ranges by the CVBG would give the advantage to the enemy since enemy damage to CVBG assets is not based on weapons range. If a Surface Threat or Air Threat is within detection range of a CVBG asset and that asset is detected, a weapon with unlimited range is fired. To ensure the enemy does not have an unfair advantage, some ranges of CVBG weapons are increased as compensation. Both real and adjusted ranges are provided in the Weapons Table.

The *oplog* database also contains extensive information about the logistical aspects of the CVBG. Where necessary, the data is cross-referenced by class type. The GPF Table (Table 7) provides Navy General Planning Factors for consumption of Logistical Items. The TypeLogistics Table (Table 8) is similar to the TypeWeapons table and provides information about each type of ship's capacity for certain logistical items. Fuel is referenced in barrels and other logistical items are referenced in pounds.

The final table of the *oplog* database is the CLFcapacity Table (Table 9). As the name implies, this table contains information about the storage capacity on board the different classes of CLF ships.

Class	Harpoon	Tomahawk	SeaSparrow	Sea Sparrow firing rate	Sea Sparrow actual capacity
Kitty Hawk	none	none	6	4	24
Kennedy	none	none	6	4	24
Enterprise	none	none	6	4	24
Nimitz	none	none	6	4	24
Ticonderoga	24	32	6		
Spruance	24	32	2	4	8
Arleigh Burke	24	32	None		
Arleigh Burke IIA	24	32	None		
Perry	24	none	None		
Blue Ridge	none	none	4	4	16
Austin	none	none	4	4	16
Sacramento	none	none	6	4	24
Supply	none	none	6	4	24
Kilauea	none	none	None		
Mars	none	none	None		
Kaiser	none	none	None		Source: TACLOG 1996

Table 5. TypeWeapons Table, partial data only

Item	Range	RealRange	Purpose	Weight	Launch weight	Notes
Harpoon	72	72	ASUW	2250	1500	All include 50% pack weight
SeaSparrow	7	7	PD	3036	506	SS weight per 4 rounds
Tomahawk	500	722	Strike	4791	3194	range to allow for screen size
CIWS	5	5	PD	990	0.22	CIWS: weight per 3000 rounds
G5in	50	13	Strike	10500	70	5in weight per 100 rounds
G3in	6	6	PD	4050	27	3 in weight per 100 rounds
SM2MR	200	42	AAW	2337	1558	range to allow attack on air when detected
SM1MR	200	22	AAW	2035	1357	SRBOC weight per 10 rounds
SRBOC	5	5	PD	75	50	Sources: TACLOG 1996, Jane's Online 2001

Table 6. Weapons Table

Item	Rate	Notes
Basic_Class_II	11.37	
Routine_Spares	.64	Does not incude CASREP major items
Fresh_Produce	1.0	subdivided from 2.42 original frozen rate
Medical_Consumables	.05	
F44	5.66	-6.11 + 2.31*numSorties in Kgal
Frozen_Goods	1.42	F44 rate false since no sorties in game
Soda	.63	
Unit_Issue_Clothing	.09	
Dry_Provisions	3.2	Source: Long 1992

Table 7. General Planning Factors

Class	F76	F44	Dry_Provisions	Fresh_Produce	Frozen_Goods
Kitty Hawk	47619	36000	360000	115000	160000
Kennedy	0	57000	360000	115000	160000
Enterprise	11904	57000	380000	121000	172000
Nimitz	0	57000	390000	124500	177000
Ticonderoga	12000	500	24000	7500	10700
Spruance	12000	500	22200	7100	10100
Arleigh Burke	12000	500	22700	7300	10300
Arleigh Burke IIA	12000	500	22900	7300	10400
Perry	4800	500	13200	4200	6000
Blue Ridge	12000	1190	74000	23000	33000
Austin	12000	500	45000	14000	20000
Sacramento	12000	500	40000	13000	18000
Supply	12000	500	34800	11000	15800
Kilauea	16000	500	97800	3100	4443
Mars	12000	500	11700	3700	5300
Sirius	12000	500	11400	3700	5200
Kaiser	12000	500	6800	2200	3100

Table 8. TypeLogistics Table, partial data only

Class	Fuel	Weapons	Stores	Weaps (tons)	Stores (tons)	Notes
Sacramento	177000	4300000	1500000	2150	750	fuel in bbls
Supply	156000	3600000	1300000	1800	650	weapons in lbs
Kilauea	0	3400000	0	1700	0	stores in lbs
Mars	0	0	7850000	0	3925	42 gal per bbl
Sirius	0	0	5786000	0	2893	
Kaiser	180000	0	0	0	0	Source: CNO

Table 9. CLFcapacity Table

2. Threat Database

The threat database contains only one table (Table 10). This table includes only basic surface threat names and speeds. Other ship characteristics, such as surface search radar range are written directly into the JAVA code. Currently, threat air assets all have the same characteristics. Threat air characteristics are also written directly into the JAVA code.

Type	Number	Speed
Cruiser1	1	20
Cruiser2	2	18
Destroyer1	3	17
Destroyer2	4	25
Carrier1	5	16
Carrier2	6	32
Patrol1	7	8
Patrol2	8	10
Patrol3	9	12
Patrol4	10	15
Frigate1	11	22
Frigate2	12	21
Frigate3	13	14
Cruiser3	14	23

Table 10. Threat Ships Table

C. METHODOLOGY SUMMARY

The Operational Logistics Wargame is a combat model with a solid Operations Research foundation and is designed for Operational Logistics students. It is expandable and configurable. Transparent to the user, the wargame uses the most up-to-date modeling and simulation techniques with its discrete event simulation and stochastic processes. The external database provides great flexibility in tailoring the wargame for specific classroom environments. The remaining chapters of this thesis give detailed guidance for players and referees of the wargame as well as recommendations for future enhancements.

THIS PAGE INTENTIONALLY LEFT BLANK

Operational Logistics Wargame

Player Manual

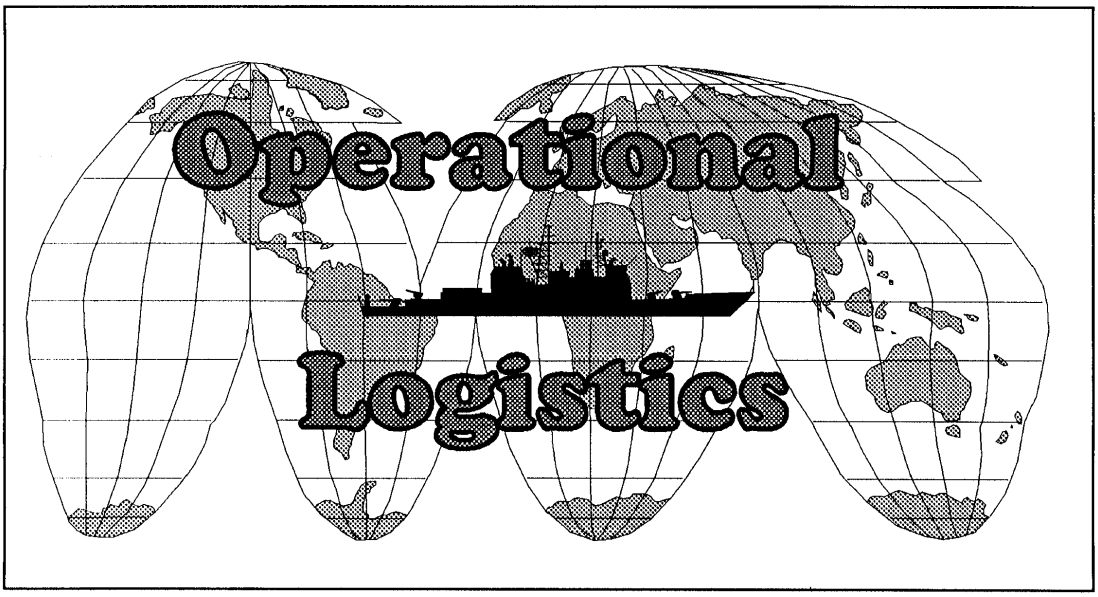

Figure 2. Courtesy of Professor D. A. Schrady

Amateurs discuss strategy,
Professionals study logistics

A. OPERATIONAL LOGISTICS WARGAME SUMMARY

The main goal of the player is to accumulate as many points as possible in either a given time or until one side's fleet is destroyed, as specified by the referee.
After the game begins, the player may perform one or more of several actions. These actions include: firing weapons, ordering logistical items, adding new coordinates to a ship's track, changing a ship's scheduled track, and quitting the game. Points (and penalties) are accumulated through offensive actions, defensive capabilities and logistics events.

1. Offense

To simplify game play, any weapon not designated Point Defense may fire at any threat target. No distinction is made among ASUW, AAW, or Strike weapons as to usage. Destructive power of specific weapons is not taken into account. From the beginning of game play, players are aware of the location of threat ports and bases. The player may fire weapons at threat ports and bases during any pause. To discourage the player from indiscriminately firing weapon without regard to range, if a weapon is fired at a target that is out of range, a penalty will be accrued (and the weapon will still be expended). If in range, a random number will determine whether the base or port is damaged, destroyed, or missed based on the respective probabilities. In the event of damage without destruction, Staying Power is decremented. Points are given when the enemy is damaged or destroyed. Penalties accrue when distance to target is outside a fired weapon's range.

2. Enemy Detection

Air Threats and Surface Threats are only shown to the player if detected by the respective air or surface radar. Radar ranges of the CVBG assets are those found in Jane's Online. Detection of a threat contact that is in range of the radar is determined stochastically. Upon detection, the player is notified. Ships that have no offensive power can report detection of threats but have no means of eliminating the threat.

3. Defense

Firing of defensive weapons is not under the control of the player in this game. However, it is the player's responsibility to ensure that an adequate supply of Point Defense weapons are onboard the ships at all times. It is also the player's responsibility

to ensure that Battle Group formation decisions are made with due regard to both offensive and defensive actions. Point Defense weapons are automatically consumed whenever that ship is attacked and ships with self-defense capabilities have higher assigned Staying Power than those without. The player is notified when any attack is made by a threat asset. Attack by a threat asset may result in damage, destruction, or no damage at all to the Battle Group unit.

Detection of CVBG assets by enemy air and surface threats is determined in the same stochastic manner as that of CVBG detection of threats, when within radar range. If the enemy detects and fires upon a CVBG asset, there is no guarantee that the threat will subsequently be detected by CVBG assets. Threat bases have no offensive capability. Penalties are accrued when ships are damaged or destroyed. Ships are repaired to full Staying Power during port visits.

4. Logistics

a. Logistics Items

For realism, at game start each ship has between 75 and 100 percent of its capacity of each logistical item including weapons as determined by a random number draw. Logistical items not including fuel or weapons are consumed on a per-man per-ship per-day basis. Fuel and weapons consumption are based on actual usage. All logistical items and weapons (except Tomahawk and Harpoon) can be replenished through underway replenishment with the Combat Logistics Force (CLF) ship and port visits. Tomahawk and Harpoon may only be replenished during port visits. In the Operational Logistics Wargame, all other weapons can be replenished at sea.

b. Logistics Units

Fuel capacity throughout the game is in barrels. Stores capacity is in pounds. Weapons are referenced using both weight and number of units. Each ship can only carry a specific number of each type of weapon but the Combat Logistics Force capacity is limited by total weapons weight not total number of weapon types or units. Therefore, weapons are ordered by weight with both number of units weight shown. When a weapon is fired in rounds, 1 unit is the equivalent of 1 round. For example, 1 unit of 5 in gun ammunition is equivalent to 100 projectiles of 5 in gun ammunition.

c. Replenishment Requests

To conduct underway replenishment with the station ship, logistical items must first be ordered from the CLF ship. In the basic game, the only CLF ship used is a "station" ship. The station ship capacity is based on published capacities for fuel, weapons, and stores. As with each ship's supply status at game start, the station ship starts the game with between 75 and 100 percent of its capacity as determined by a random number draw. Items may be ordered during any game pause and are assigned a priority (either routine or urgent) as the player desires. Ordered items assigned a routine priority are filled in due course. Routine unrep orders may or may not arrive during the next unrep, depending upon previously submitted orders for all ships. Orders placed with an urgent priority will normally be received during the next unrep. Penalties accrue for use of urgent priority. Penalties also accrue when inventory of a specific item falls below 40 percent. More severe penalties accrue if inventory falls to zero. Points are given for ships maintaining inventory above 50 percent between replenishments. Items 'received' during unrep in excess of the ship's capacity are not added to the inventory but do not accrue a penalty. At this time, there is no provision for canceling an order manually. Port visits result in cancellation of all outstanding unrep requests for that ship. Unreps occur automatically when the CLF ship is within range of a ship for which it has orders. There is no game delay during underway replenishment. Detection range for underway replenishment is currently set at 30 miles. Once the CLF ship's capacity is exhausted, it must be ordered back to a port for re-supply. Upon reaching port, the CLF ship's capacity is filled based on existing orders in a first-in first-out manner (with regard to priority). There is no delay of game for the CLF ship while it is in port.

d. Port Visits

Port visits occur whenever a ship is within range of a friendly port. Detection range for port visits is currently set at 30 miles. Since real-life port visits detract from a ship's ability to conduct its combat mission, in the Operational Logistics Wargame, penalties accrue for any port visit by a ship that is not a CLF ship. During a port visit, all logistical items and weapons including Tomahawk and Harpoon are replenished. Any ship that has been damaged is repaired to full operating status. There is no delay of game for a ship during the port visit.

B. GAME PLAY START-UP

1. Installation

The current version of Simkit and Java that are being run in the Operations Research Department at the Naval Postgraduate School must be installed on the computer where the game is to be played. A JDBC driver must also be installed for both the Oplog database and the threat database. In a Microsoft Windows environment, the driver can be established by installing a Microsoft Access driver in the ODBC Data Source panel and using the JDBC-ODBC bridge that ships with the Java Development Kit. The Player DSN name that refers to the Oplog database must be "wargame". The Player DSN name that refers to the ThreatDB database must be "wargameThreat".

2. Initialization

The Operational Logistics Wargame begins when the player inputs the applicable code on a Dos command line or otherwise executes the main method of the program. Running the wargame in a JAVA IDE is not necessary and is not recommended. Command line code to start the game will typically be:

"java oplog.gui.OplogWelcome"

C. WELCOME SCREEN

Upon game initialization, the player can select to start a new game or abort the game as shown in Figure 3. Throughout the game, whenever possible, buttons have keyboard mnemonics assigned. Buttons with shortcut keys use the underlined letter as the assigned mnemonic.

Figure 3. Operational Logistics Wargame Panel

D. PLEASE WAIT PANEL

While the game is loading, when short delays are expected to occur, a window noting that a delay will occur appears (Figure 4):

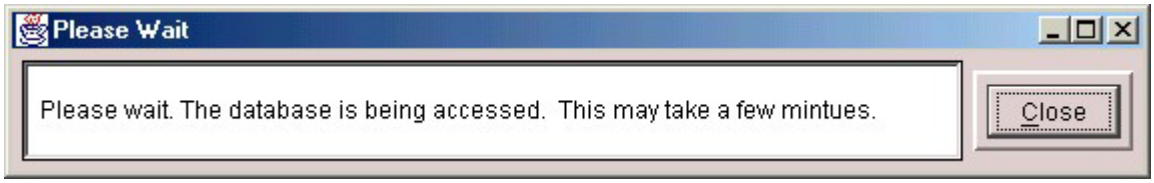

Figure 4. Please Wait Panel

E. INTELLIGENCE SUMMARY

After selecting a new game, the player is provided information concerning the game. This information includes an intelligence summary, composition of the Battle Group and CLF ship status, logistics status, and weapons status. The player may review the displayed data at leisure. When ready to proceed, the player must decide whether or

not to remain in port before deployment to replenish all supplies and weapons or to deploy immediately with the amounts onboard as indicated in the Logistics Status and Weapons Status tabs. Should the player decide to remain in port for replenishment, a penalty will be assessed. After the presail decision is made, the player may click the *Click this button when ready to proceed* button at any time. The Intelligence Summary data are presented in Tabbed Pane format.

1. Intelligence Summary Tab

The first tab of the Intelligence Summary Panel is the Intelligence Summary Tab (Figure 5). This pane is a textual description of the intelligence scenario and does not require or allow any input from the player.

Figure 5. Intelligence Summary Tab

2. Game Rules Tab

The second tab of the Intelligence Summary Pane is Game Rules Tab (Figure 6). This tab is a textual description of the most important rules of the Operational Logistics Wargame. This tab does not need or allow input from the player.

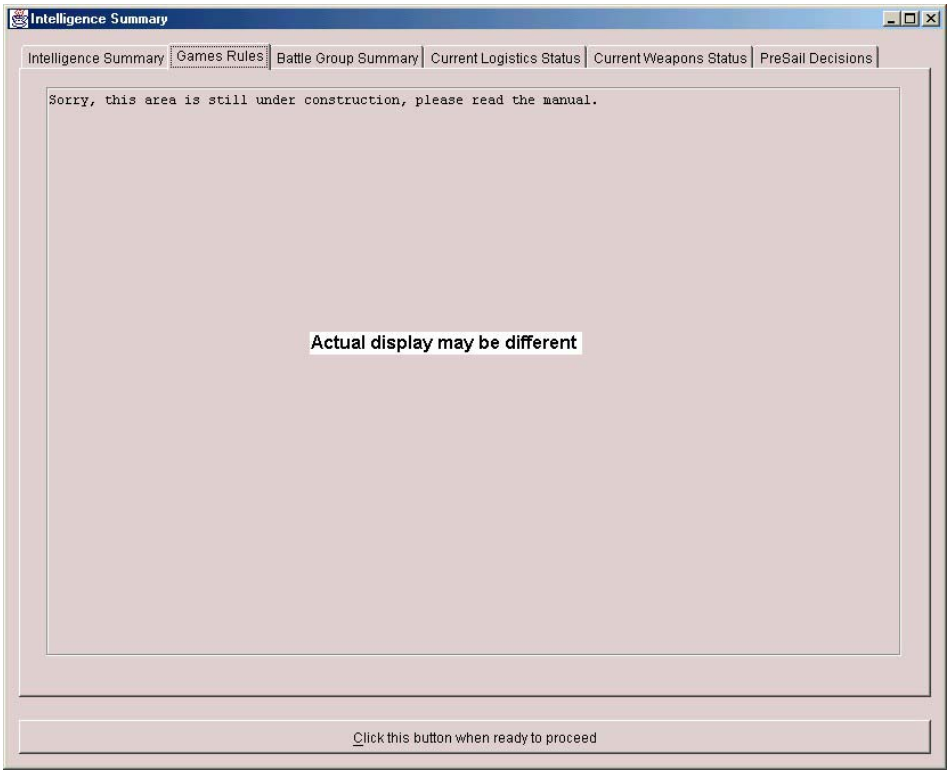

Figure 6. Game Rules Tab

3. Battle Group Summary Tab

The Battle Group Summary Tab (Figure 7) is the third tab of the Intelligence Summary Panel. This tab provides pertinent information to the player concerning the Battle Group's composition. All ships in the Battle Group are listed along with their main missions, maximum speeds, and number of personnel. This panel also displays the starting status of the Combat Logistics Force ship. The total amount of fuel, stores, and weapons that the CLF ship has onboard at game start is a random amount between 75% and 100% of its capacity for each broad type. This tab does not require or allow input from the player.

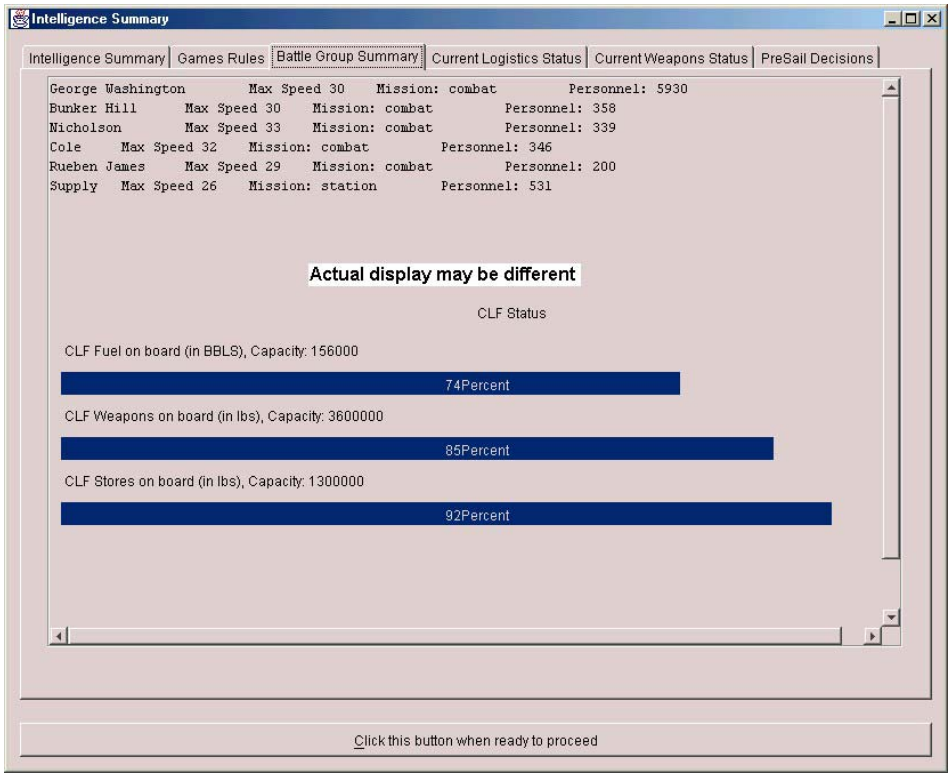

Figure 7. Battle Group Summary Tab

4. Current Logistics Status Tab

The fourth tab of the Intelligence Summary panel is the Current Logistics Tab (Figure 8). This tab has a separate tab panel for each ship in the CVBG. The individual ship tabs show the starting status of each logistical item that the ship has onboard. The starting status is between 75% and 100% of the ship's capacity. This tab does not require or allow input from the player.

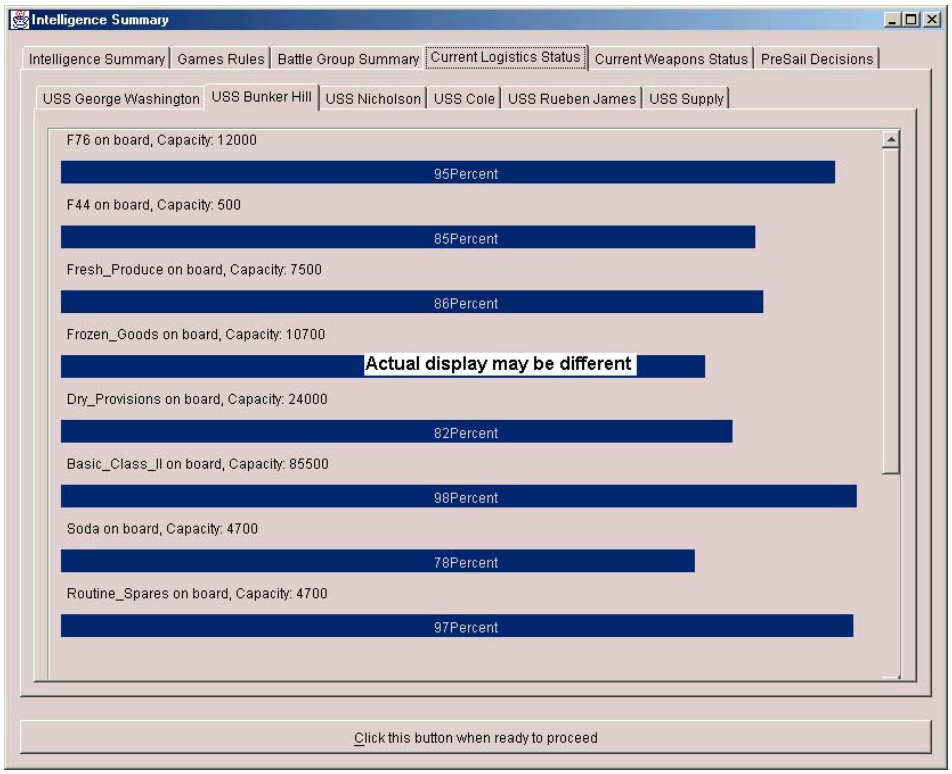

Figure 8. Current Logistics Status Tab

5. Current Weapons Status Tab

The last informational tab of the Intelligence Summary Panel is the Current Weapons Status Tab (Figure 9). This tab has a separate tab panel for each ship in the CVBG. The individual ship tabs show the starting status of each weapon that the ship has onboard. The starting status is between 75% and 100% of the ship's capacity. This tab does not require or allow input from the player.

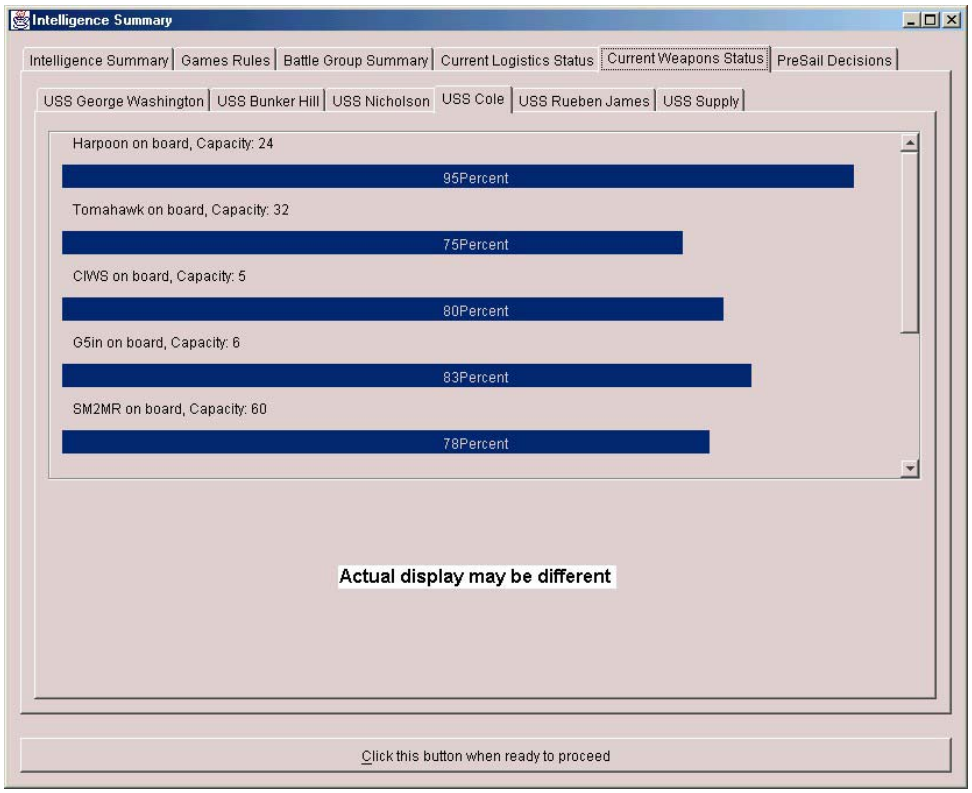

Figure 9. Current Weapons Status Tab

6. PreSail Decisions

Prior to deployment, the player must decide whether to delay sailing in order to top off all consumables or to deploy immediately. Should the player decide to remain in port, a penalty is accessed. (See the penalties section for more information on game penalties.) The Presail Decisions tab (Figure 10) allows the player to decide which option to pick. The player selects the button corresponding to the presail decision made. A specific decision must be made by the player or the game will not proceed. After making a decision, the player should click *Click this button when ready to proceed.*

Figure 10. PreSail Decisions Tab

F. SETTING COURSE AND SPEED

To begin the deployment, the player assigns the ships to sail together as a fleet or individually. Some ships may be selected to sail in formation while the remainder sail individually. Among other things, this option allows the CLF ship to sail to port, with or without escort, while the fleet sails toward the objective. Otherwise, the CLF ship will be unable to do underway replenishment once its initial stores run out. The player selects the ships' initial courses and speeds and the deployment begins.

1. Course and Speed

In the first graphical user interface of this section, entitled Course and Speed (Figure 11), the player should click on the check box of any ship that is to be assigned to sail in the Battle Group formation. In the Course and Speed selection panel, if the player wants to select only ships to sail individually, do not select and check boxes and click the *Start* button. After choosing the Battle Group ships and clicking the *Start* button, the next

panel opens. Note: ships may be added or removed from the main Battle Group during any pause of game play. Additionally, ships may be assigned to sail in multiple, smaller groups as desired during game play.

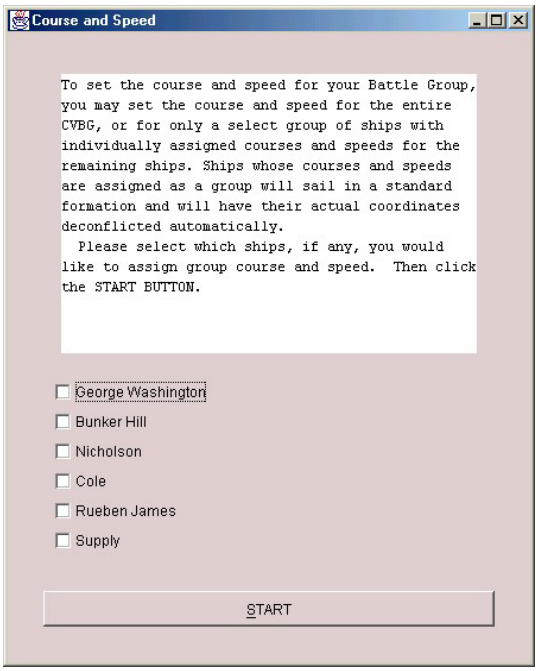

Figure 11. Course and Speed Panel

2. Course and Speeds for the CVBG (by group)

After proceeding from the previous panel, the Course and Speeds for the CVBG (by group) panel opens. (Figure 12) In this panel the player selects a speed for the Battle Group by moving the speed slider to the desired speed. The maximum possible speed is the lowest maximum speed of all ships in the intended group. The player also selects the Battle Group's initial course from Homeport to any desired location. The course is selected by clicking the *Start/reset coordinates button* then clicking significant waypoints along the desired course. The selected track is NOT shown on the screen. However, once a course has been chosen, clicking on the *Print coordinates* button prints the selected waypoints on a separate output screen. (Figure 13) Clicking the *Print Coordinates* button also enables the *PROCEED* button. The coordinate system used is the standard JAVA coordinate system where the origin is the top left corner of the designated window and x increases to the right with y increasing down. If the coordinates displayed are satisfactory, the player should click *PROCEED*.

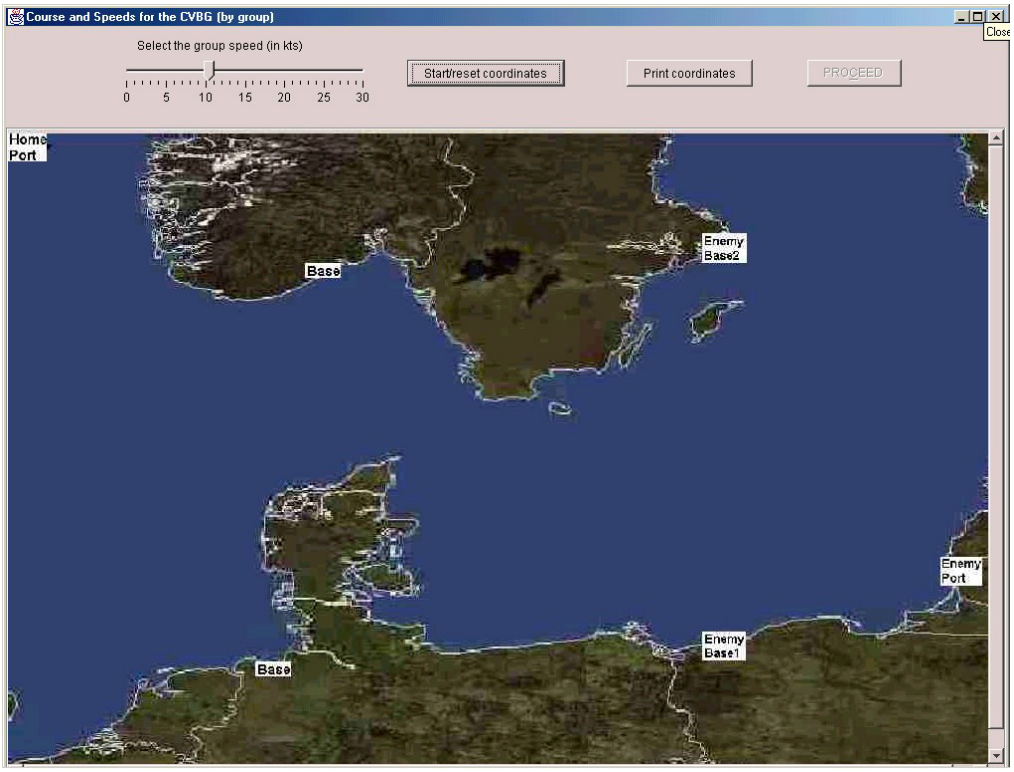

Figure 12. Course and Speeds for the CVBG (by group) Panel. After Ref. National Geographic.

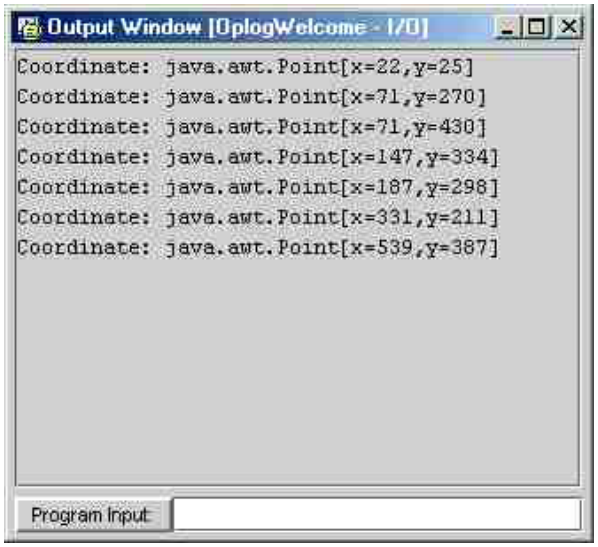

Figure 13. Sample CAS Group Output

Otherwise, the player may click the *Start/reset coordinates* button to try again. The basic version of this game does not differentiate between land and water. So, the player must take care to ensure that the ships do not sail across land. A future enhancement of will include a land-avoidance algorithm.

Note: All map backgrounds in the Operational Logistics Wargame and depicted in this document were obtained from the National Geographic Website (www.nationalgeographic.com) and have been altered by the author. All icons used in the Operational Logistics Wargame and depicted in this document were obtained from a variety of sources including theses by Lt J.R. Sterba and Lt A.W. Troxel as well as Web Clip Art Website (www.webclipart.about.com) and Air War College Website (www.au.af.mil/au/cpd/cpdgate/clip_af.htm) . All icons have been altered.

3. Course and Speeds for Individual Ships

Ships shown on individual tabs in the *Course and Speeds for individual ships* panel (Figure 14)) are all of the remaining ships of the Battle Group that were not selected to sail in formation. This panel is set up in much the same fashion as the *Course and Speeds for the CVBG (by group)* panel. However, the *PROCEED* button is not enabled until an individual course has been set for each ship shown. The course or speed for any ship listed may be changed at any time prior to clicking *PROCEED*. The selected coordinates are also printed to the output screen. (Figure 15) When the player clicks PROCEED, additional database information is drawn during a short wait and game play begins.

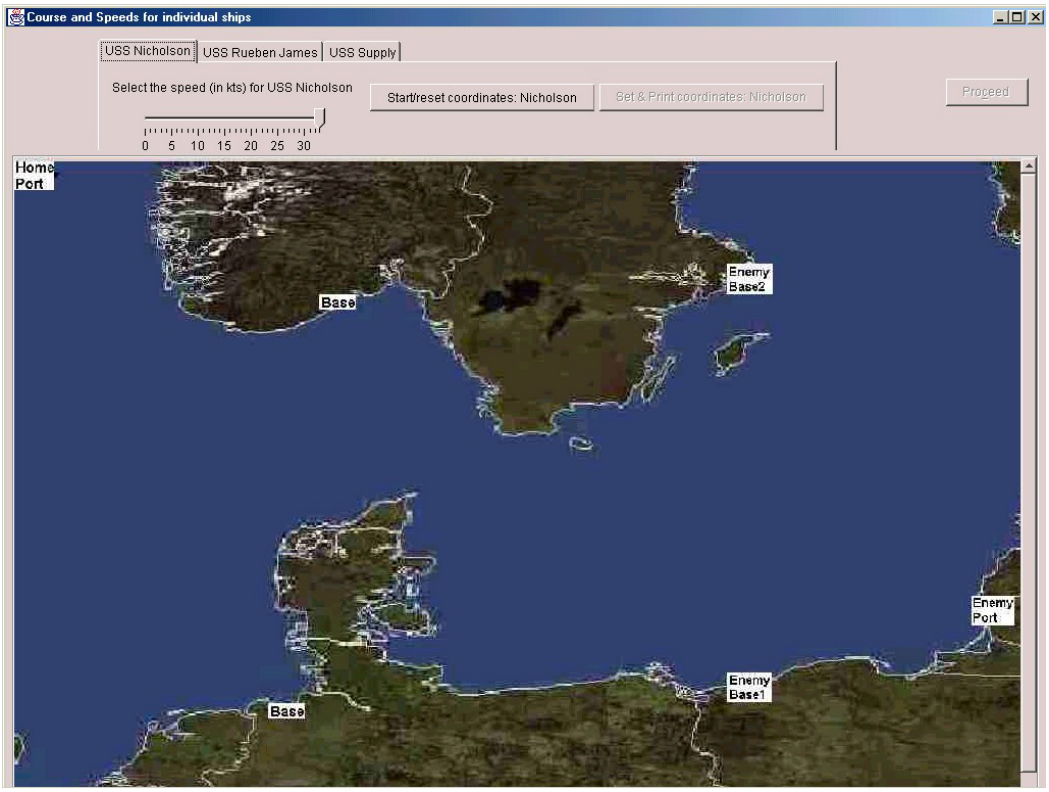

Figure 14. Course and Speeds for individual ships Panel. After Ref. National Geographic.

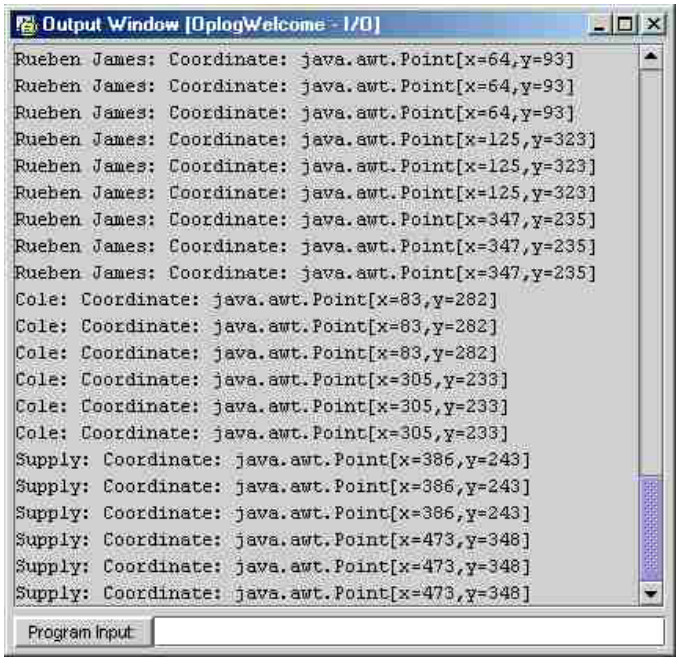

Figure 15. Sample Individual Ship CAS Output

G. ANIMATION

The Animation Frame has three main sections: The Animation Tab, the Actions Tab, and the Control Panel.

1. Animation Tab

The Animation Tab (Figure 16) is an animated graphical display of the game as it progresses. The current map is displayed with existing friendly and enemy bases. As the CVBG's ships sail at various speeds and courses, their progress can be seen on the screen. Ships sailing in formation will normally appear as one unit and as a single icon while the ships sailing individually will be distinguishable. If a surface threat or air threat is detected by CVBG sensors, the threat unit's icon is also displayed at the appropriate coordinates for the duration of its detection. If any friendly or enemy unit is destroyed, the icon for that unit is removed from the display. The Animation Tab does not require or allow any input from the player.

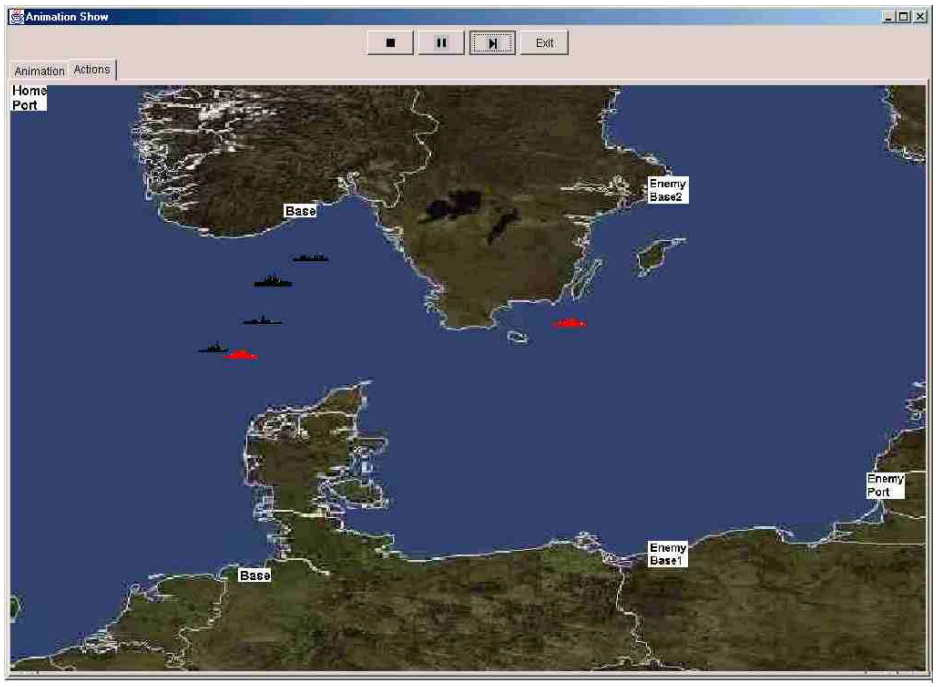

Figure 16. Animation Tab. After Ref. National Geographic.

2. Actions Tab

The primary GUI of the Operational Logistics Wargame is the Actions Tab. This tab provides a multitude of information to the player and allows the player to take action. The wargame is designed to pause its simulation and animation whenever certain events occur that warrant notice to the player. Events that cause the simulation to pause automatically include:

- First detection of enemy air or surface threat (multiple detections are simply logged).
- First detection of threat bases and ports within range of surface radar.
- Loss of contact of a threat by all CVBG assets.
- Damage or destruction of an enemy by the CVBG.
- Damage or destruction of a CVBG asset by the enemy.
- Port visits.
- Underway replenishments.
- Ships reaching the end of scheduled tracks.

a. Reason For Auto-Pause Panel

The uppermost panel seen on the Actions Tab is the Reason for auto-pause panel (Figure 17). Whenever the game is paused automatically, the event that caused the pause is noted in this panel. This panel does not require or allow player input.

b. Score Panel

The total accumulation of all penalty and bonus points is noted in the Score panel (Figure 17). This panel is updated at every pause. This panel does not require or allow player input.

c. Sim Time Panel

The simulation time at the instance of the pause is shown in the Sim Time panel (Figure 17). Simulation time is shown in hours. This panel does not require or allow player input.

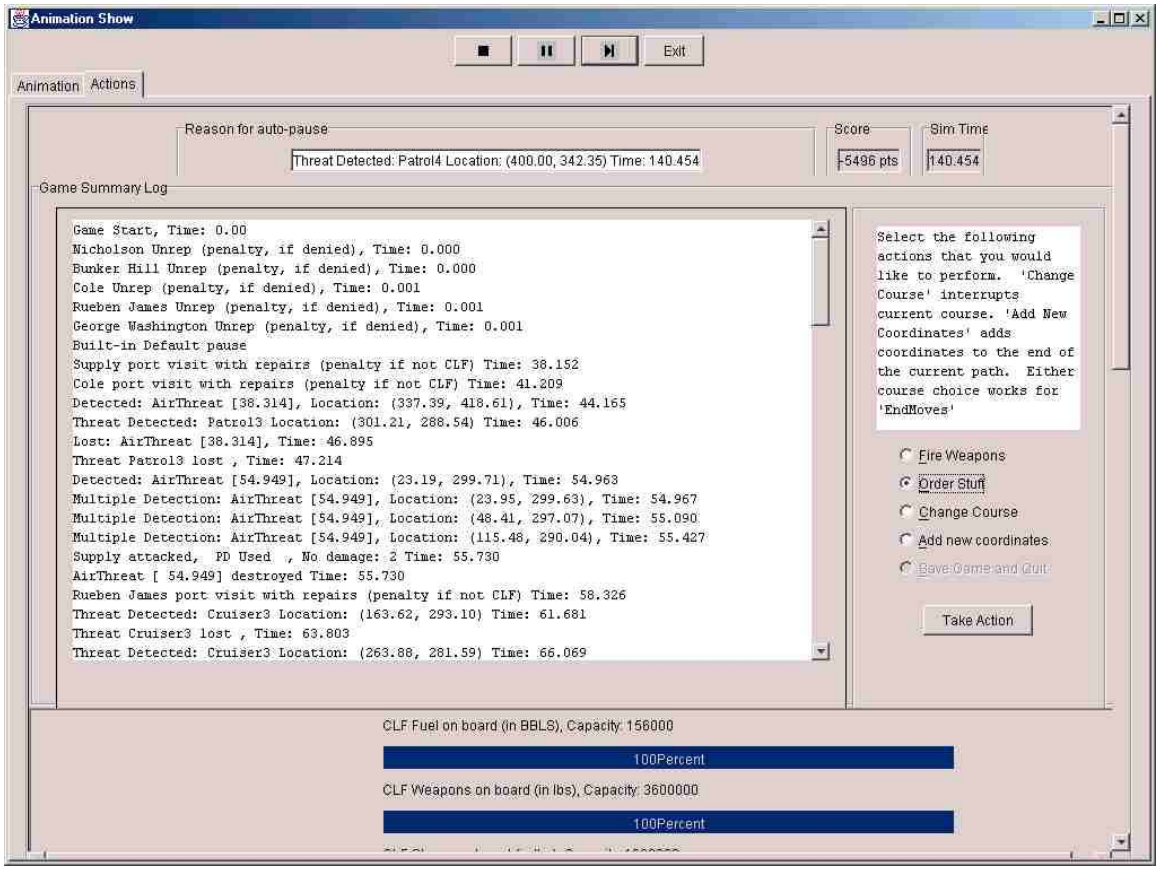

Figure 17. Actions Tab, View 1

d. Game Summary Log Panel

The Game Summary Log panel (Figure 17) contains a list of all pertinent events that have occurred during the Operational Logistics Wargame and the time the event occurred. In addition to all auto-pause events, this log contains a listing of all multiple detections and undetections by CVBG assets. This log does not contain any events, significant or not, that are not known to the player. For example, a surface threat generated by the wargame would not be noted on the log unless detected by the CVBG. This panel does not require or allow player input.

e. Select Actions Panel

The most important player input panel is the Select Actions panel (Figure 17). The Select Actions panel allows the player to control various aspects of the game. Actions that the player can select to perform include:

- Firing weapons,
- Review and place unrep requests,
- Change the course of a ship or ships,
- Add waypoints to the end of a course for a ship or ships,
- Save and Exit the game. (This feature is not functional.)

The player selects which action to take by clicking on the appropriate radio button for the selection then clicking the *Take Action* button. See individual sections of this manual for detailed descriptions of each of these selections.

f. CLF Status Panel

The CLF Status panel (Figure 18) is similar to the CLF status displayed before game start. It displays the current status of the Combat Logistics Force ship. The total amount of fuel, stores, and weapons that the CLF ship has onboard at the time of the simulation pause is shown here. This panel is updated at every game pause. This tab does not require or allow input from the player.

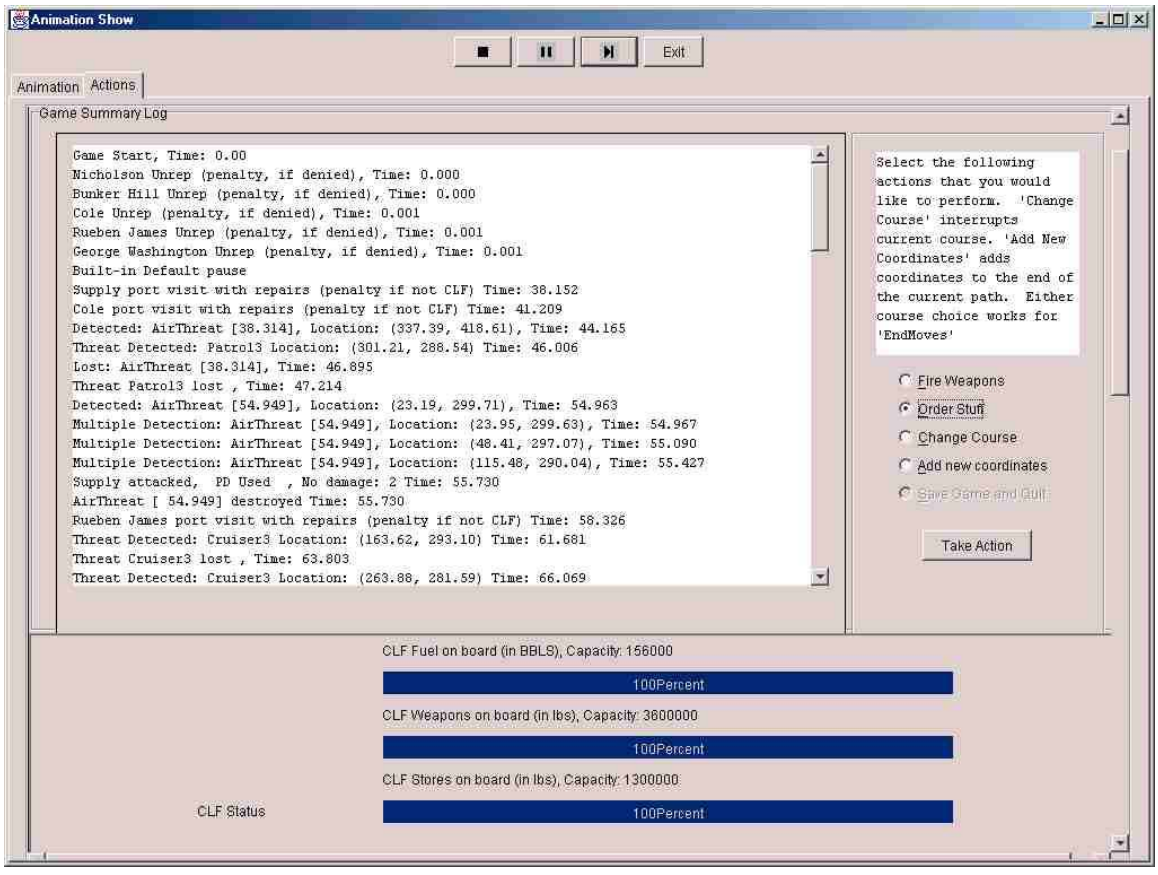

Figure 18. Actions Tab, View 2

g. Logistics Status Panel

The Logistics Status panel (Figure 19) is also similar to its counterpart that was provided at game start. In addition to previously described information, this panel shows the current Staying Power of the indicated vessel. This panel is updated at every game pause. This panel does not require or allow player input.

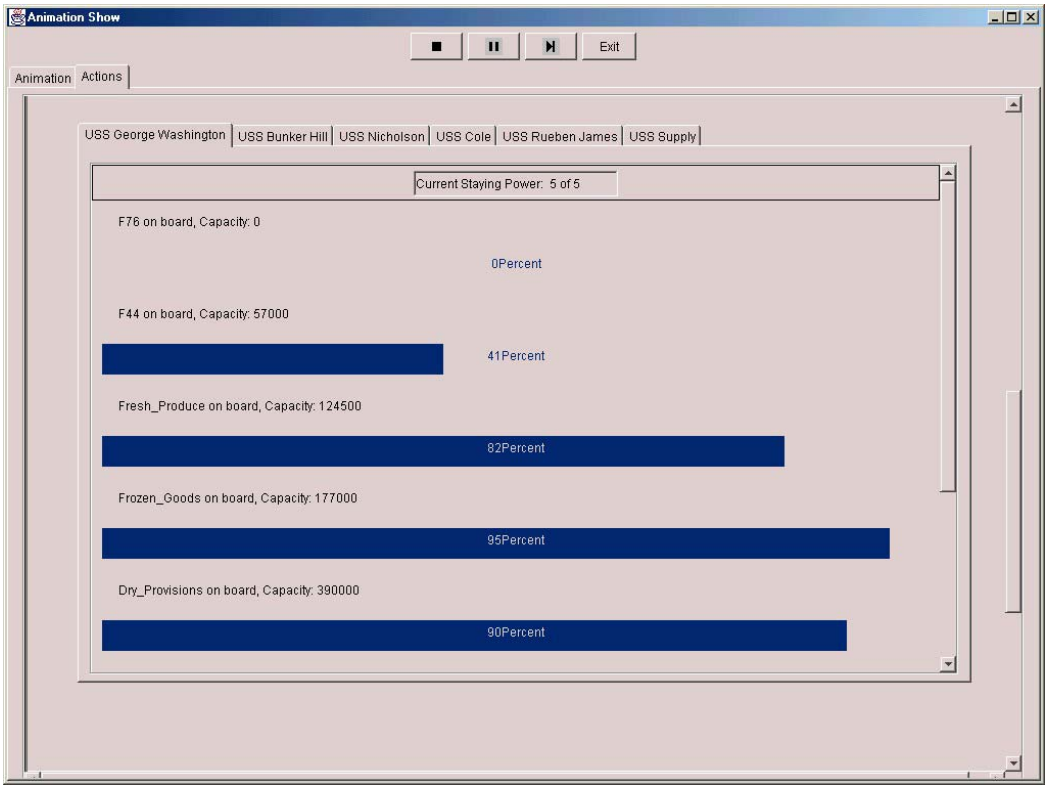

Figure 19. Actions Tab, View 3

h. Weapons Status Panel

The Weapons Status panel (Figure 20) is similar to its counterpart that was provided at game start. This panel does not require or allow player input and is updated at every game pause.

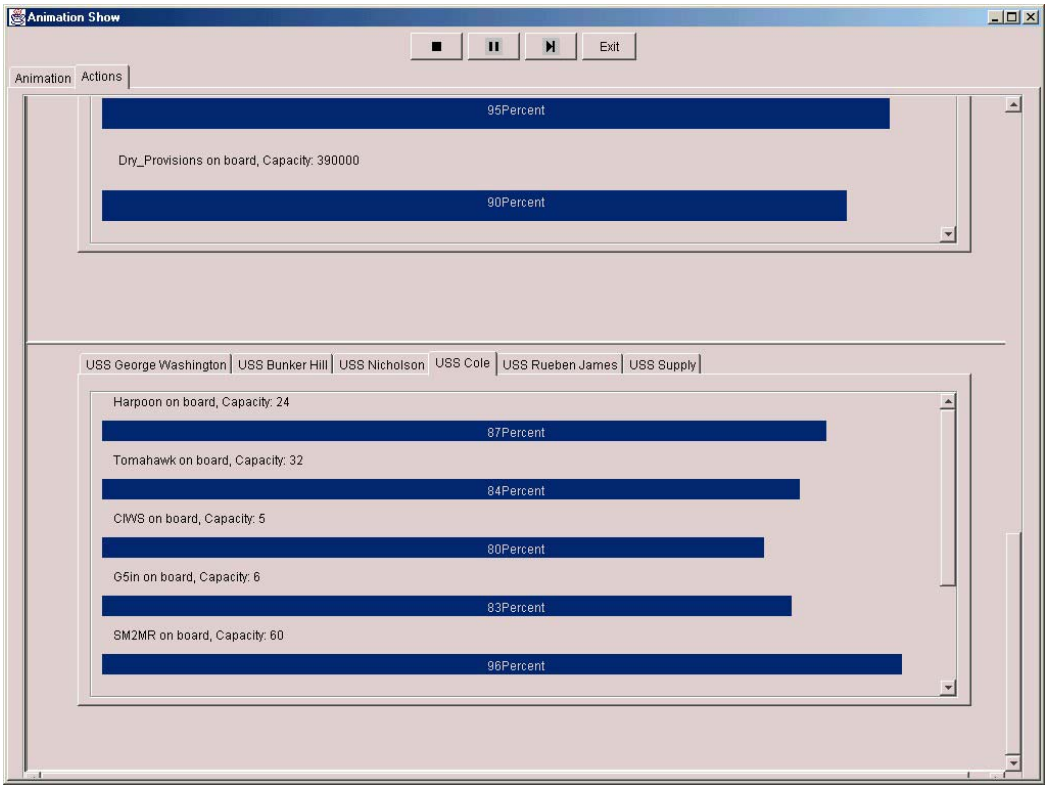

Figure 20. Actions Tab, View 4

3. Control Panel

The Control Panel (seen at the top of Figure 20) provides the player very basic control over the progress of the game and subsequent animation. Each button on the Control Panel has a tool tip that pops up when the cursor is moved over the face of the button. The player can choose to *Stop* the animation portion of wargame which stops the animation screen from refreshing. The player can *Pause* the animation and the underlying simulation. The player can *Resume* the animation and simulation. Or, the player can *Exit* the wargame altogether. The current version of the wargame does not have the capability to save a game in progress.

H. PANELS SPAWNED BY SELECT ACTIONS PANEL

As previously discussed, there are several actions that can be accomplished from the Select Actions panel on the Action tab of the Animation Window.

1. Fire Weapons Panel

The Fire Weapons panel (Figure 21) allows the player to take offensive action against surface threats, air threats, and enemy bases or ports. The left side of the panel lists all Battle Group assets with offensive capabilities and their current positions. A ship that normally has the capability to fire a specific weapon but does not have any onboard at that time shows a ZERO inventory level for that weapon. On the right side of the panel, all existing enemy bases or ports and all detected surface or air threats are displayed. The current position and staying power of each target are noted.

The current version of the Operational Logistics Wargame does not discriminate between offensive weapon types. Any weapon shown on the Fire Weapons panel may fire at any target. Due to a variety of reasons, the standard range of several of the weapons have been modified for this wargame. The range currently assigned to each offensive weapon is shown in Table 11. It is the player's responsibility to determine if specific weapons on a platform are in range of a specific target.

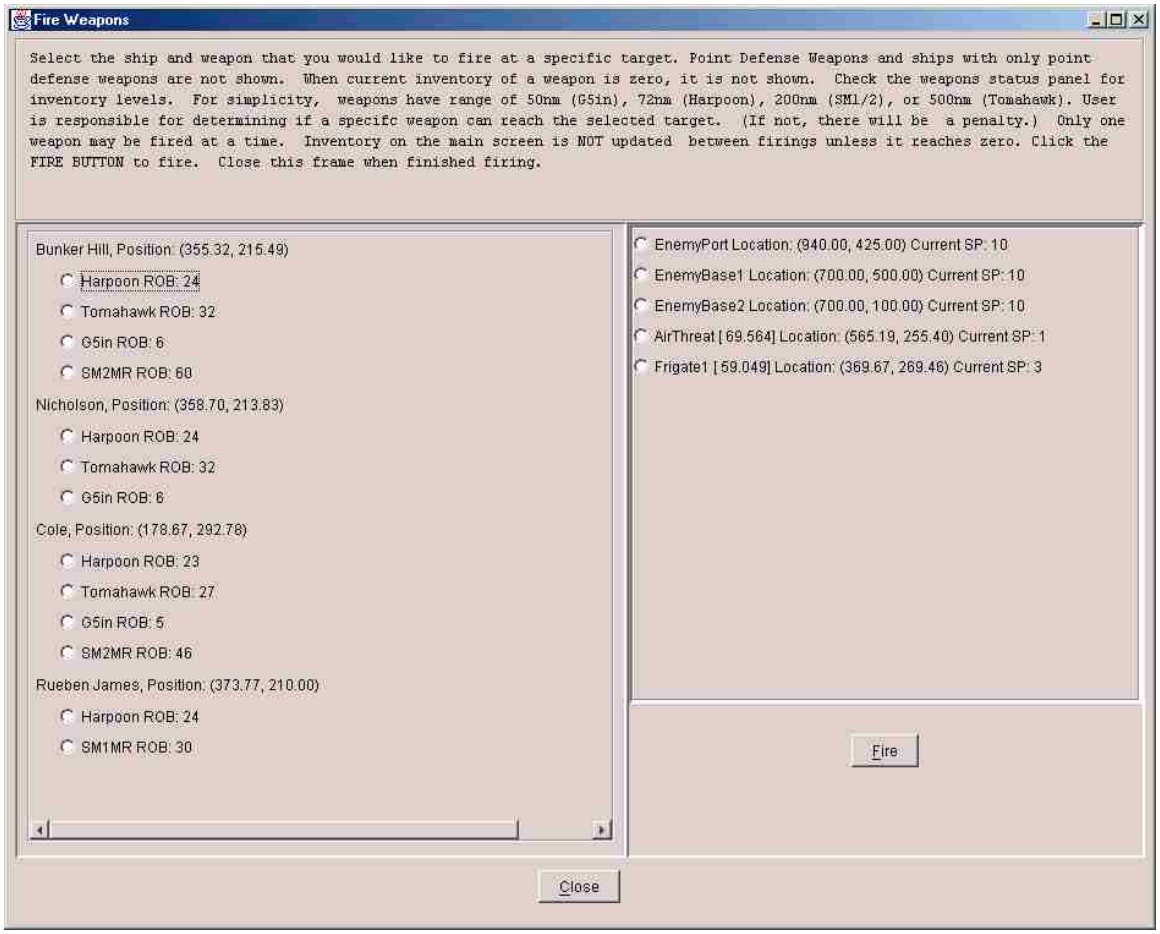

Figure 21. Fire Weapons Panel

Weapon	Range
Harpoon	72
Tomahawk	500
CIWS	5
G5in	50
SM2MR	200
SM1MR	200

Table 11. Oplog Weapon Ranges

Each firing action fires 1 unit of the indicated weapon at the chosen target. The player selects a weapon to fire by clicking on the radio button next to the desired weapon. The player selects the intended target in a similar manner. After both a weapon and a target are chosen, the player should click *Fire*. Through generation of a random number, a determination is made as to which of several results occur when a weapon is fired.

Results of weapons firing include: destruction of a target due to one well-placed shot or due to reduction of the Staying Power to zero, damage of a target and reduction of the target's staying power by 1, a missed shot or a shot that hit the target but did not damage it, a shot that was not in range of the target and subsequent penalty, or an attempt was made to fire a weapon that is out of stock and subsequent penalty. A Battle Damage Assessment panel appears to report the results of each weapon firing. Examples of each of the possible weapons firing results are shown as Figure 22 thru Figure 27. After each weapons fire, the Fire Weapons panel is updated to reflect the new target and weapons inventory status. The player may repeat the fire weapons process as many times as desired. When done, the player should click Done to terminate the panel and return to the main Action tab.

Figure 22. BDA Report: Lucky Shot

Figure 23. BDA Report: Destruction

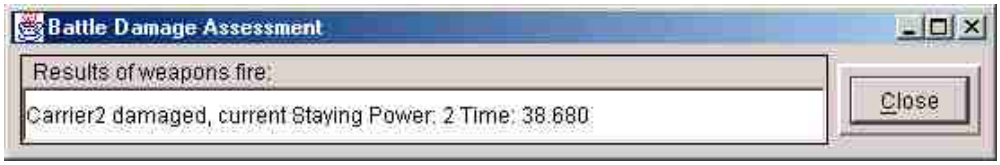

Figure 24. BDA Report: Damage

Figure 25. BDA Report: Undamaged

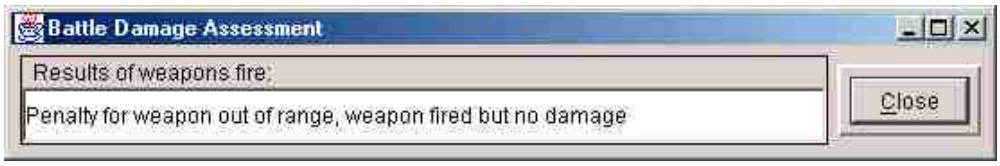

Figure 26. BDA Report: Weapon Out of Range Penalty

Figure 27. BDA Report: Zero Inventory Penalty

2. Unrep Orders Panel

The Unrep Orders Panel (Figure 28) enables the player to place unrep requests and to review details about pending underway replenishment orders. In this version of the Operational Logistics Wargame, destruction of the CLF ship does not disable the unrep request panel; but, the panel is useless. The right side of the panel contains a list of all existing ships in the Battle Group, less CLF ships. The player may place an unrep request for any of the ships shown by clicking on the appropriate radio button. The left side of the panel lists all pending underway replenishment requests. The requests are grouped by supply class: fuel, weapons, or stores. With each class, the requests are listed in First-in, First-out (FIFO) order by priority. The player may review the details of a specific order by clicking on the appropriate radio button. More details on these options are provided below.

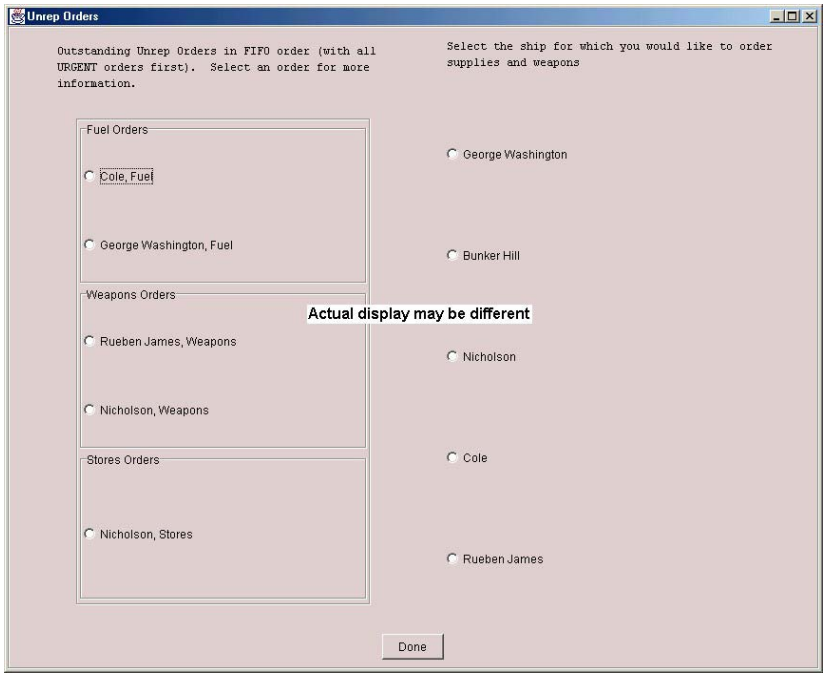

Figure 28. Unrep Orders Panel

a. Place an Order

After clicking a specific ship's button on the Unrep orders panel to order supplies and weapons, the Placing an Unrep request panel is generated (Figure 29). This panel contains a slider for all fuel, weapons, and stores onboard the specified ship provided the item is available via underway replenishment. Harpoon and Tomahawk are not shown because they are not replenished at sea. The maximum value of each item's slider is the maximum capacity on the ship. The sliders for weapons show both capacity by weight and number of units, as previously discussed. The player orders a specific item by moving the slider to the desired amount. After selecting all items to be ordered, the player should select a priority for the order (urgent or routine). Use of urgent priority incurs a penalty. Once all items are selected, click the *Place the Order* button. Any item with an amount of zero is not ordered. If all sliders are set to zero when the *Place the Order* button is clicked, no order will be recorded. The player may use the *Exit* icon to exit the screen without terminating the game.

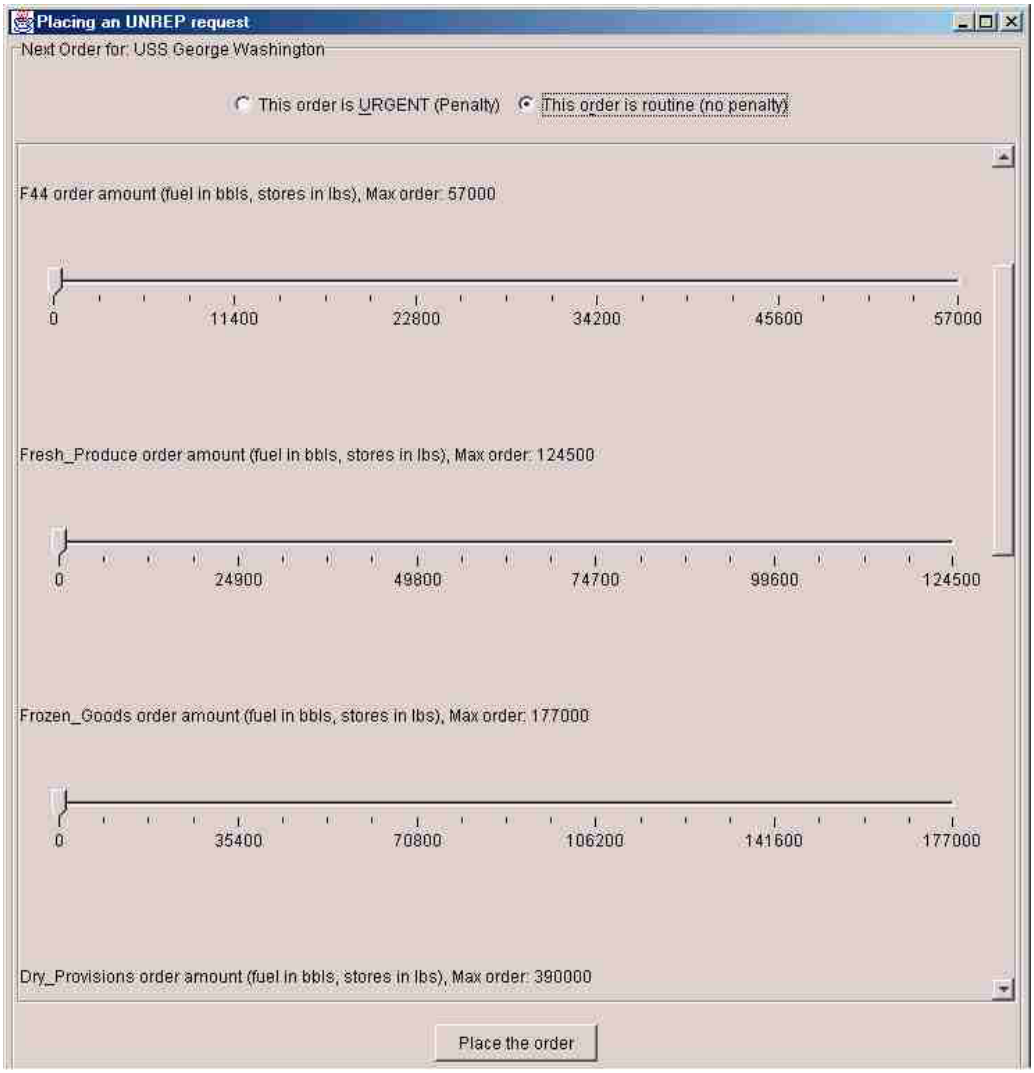

Figure 29. Placing an UNREP request Panel

b. The Unrep Schedule

At frequent intervals throughout the wargame, the unrep schedule is reviewed and modified as necessary. When under review, the request list is compared to a clone of the CLF ship's onboard inventory for that commodity. If the inventory clone has the inventory to fill the first order, that order is scheduled to be filled, and the clone is decremented by the first order's total amount. These steps are repeated until an order amount is greater than the remaining value of the clone. At that point the order under review is scheduled to remain unfilled and all the
following orders on the list are also scheduled to remain unfilled.

During an underway replenishment, the schedule is checked for all orders by that ship. Any orders scheduled to be filled are honored and the CLF inventory is reduced by the amount of the request. The CLF inventory is always reduced by the amount of the request, regardless of the actual amount onboard the ship. Additionally, if any orders are denied during an unrep because it is not scheduled to be filled, the player is accessed a penalty. So, the player should take care when placing orders since the amount ordered determines the scheduling of unrep requests and reduces the CLF inventory without noticeable advantage to the player.

Once a request has been honored, the request is removed from the summary of unrep requests. If a ship is destroyed, all of the ship's requests are removed from the summary. If a ship makes a port visit, all of the ship's requests are removed from the summary.

c. Check an Order

When the player selects a radio button on the outstanding unrep requests part of the Unrep Orders panel, a summary of the indicated order is displayed. Information displayed includes the total weight (or barrels, for fuel) of all desired items, the specific weight of each item, and the status of the request. The status of the request notes whether or not an unrep request will be honored during the next underway replenishment for that ship. Figure 30's order is scheduled to be filled during the ship's next underway replenishment while Figure 31's order is not.

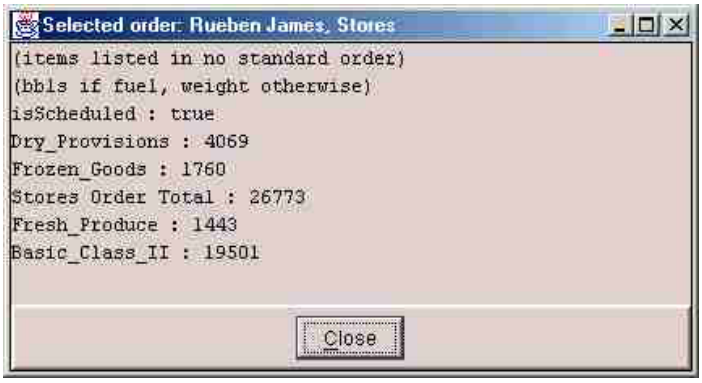

Figure 30. Sample Selected Order, Scheduled

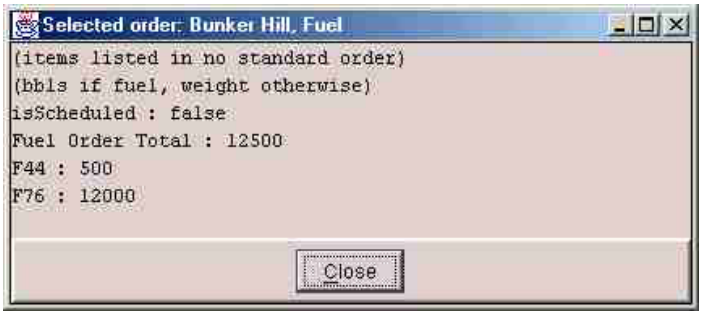

Figure 31. Sample Selected Order, Unscheduled

3. Change Coordinates Panels

The Change Coordinates panels are very similar to the Course and Speed panels seen at the beginning of the game; however, there are notable differences. The player may exit from any of these panels at any point without terminating the game. The first two panels of this section are similar as the original panels(Figure 32 and Figure 33). An output panel also displays the results of course selection (not shown).

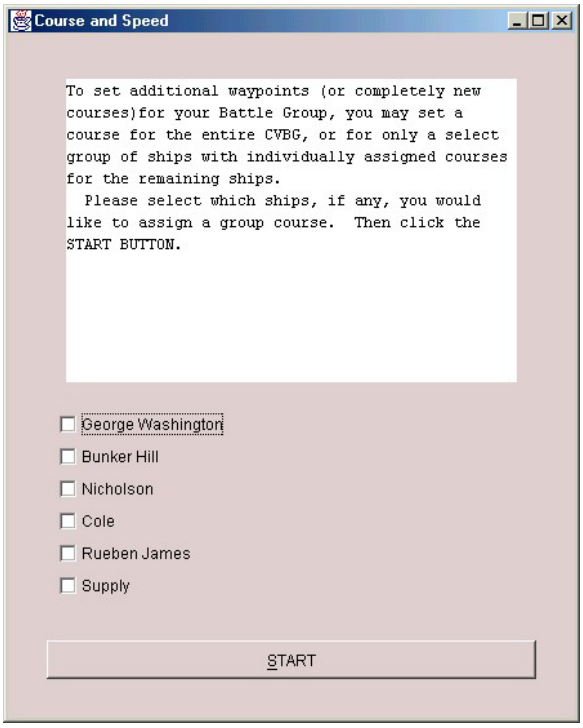

Figure 32. Change Coordinates Course and Speed Panel

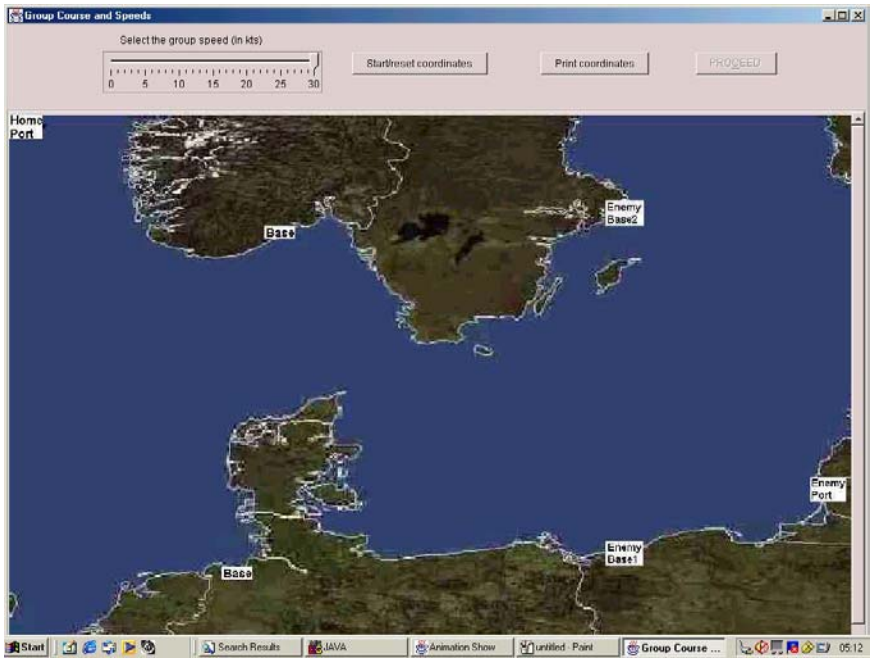

Figure 33. Change CoordinatesGroup Course and Speeds Panel. After Ref. National Geographic.

After the group setting panels have been completed, another Course and Speed ship selection panel displays (Figure 34). This panel displays the names of the ships that were not selected to sail as a group. The player may select which ships to set course and speed for individually by clicking the appropriate check boxes. Click *Start* after ship selection. If the player does not desire to set any individual course and speeds, the player can click *Start* or the *Exit* icon. The last panel of this section is similar to the original panels (Figure 35). The speed slider for the individual ships is set to the ship's current speed. If the player does not desire to change the ship's speed, the slider bar should not be moved. The new coordinates are printed to an output panel (not shown).

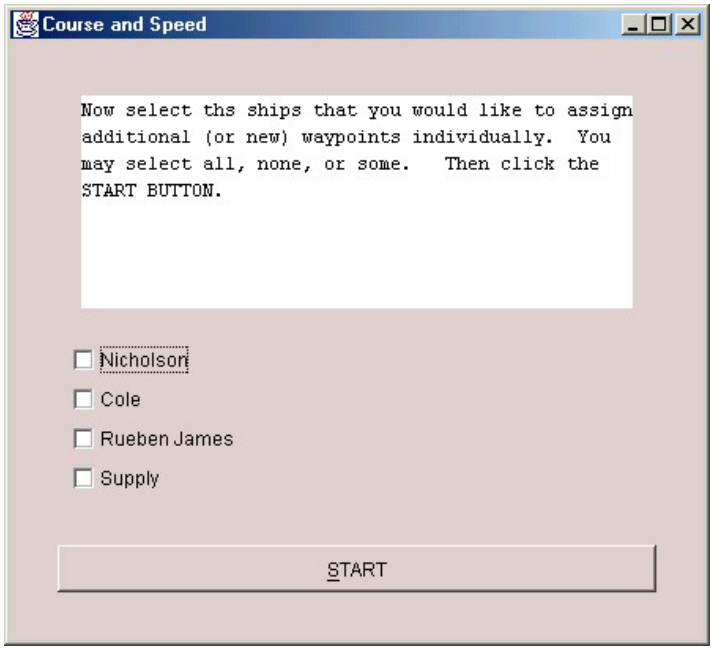

Figure 34. Change Coordinates Course and Speed Unit Selection Panel

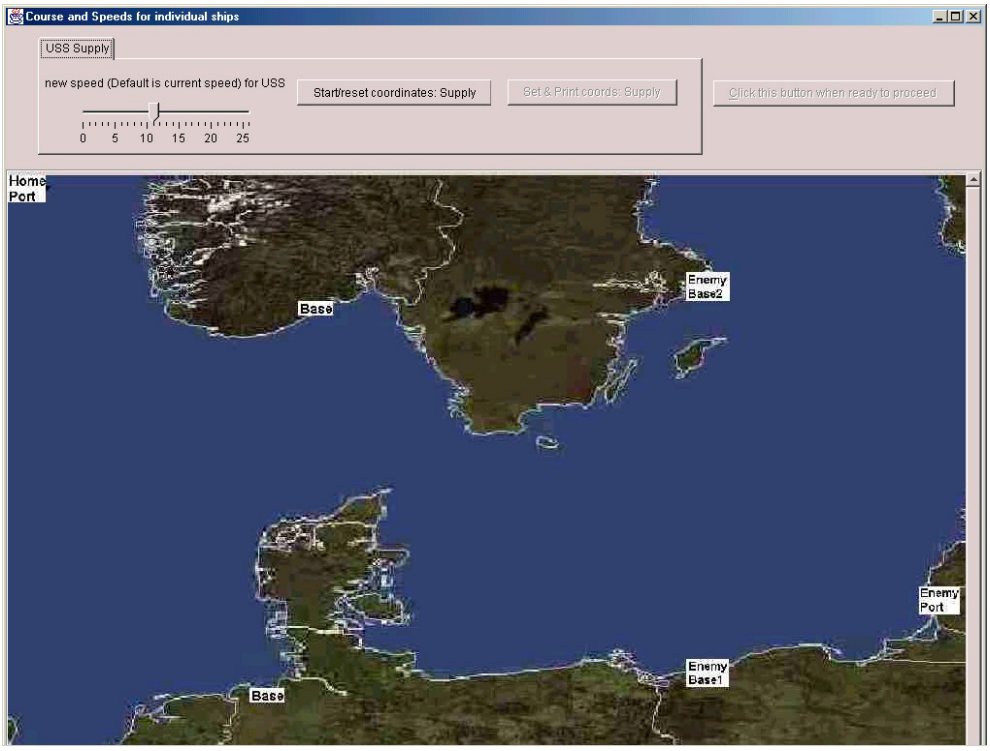

Figure 35. Change Coordinates Unit Course and Speed Panel. After Ref. National Geographic.

4. Add Coordinates Panel

The Add Coordinates selection of the Select Actions panel results in the same GUI's being produced that are produced for the Change Coordinates panel. (Figure 32 thru Figure 35) The only difference between the Change Coordinates action and the Add Coordinates action is that Change Coordinates replaces the indicated ship's existing scheduled Course and Speed track with the new list of waypoints. The Add Coordinates action adds the new list of waypoints to the end of the existing scheduled Course and Speed track.

5. Save and Exit

This option is not functional in the current version of the Operational Logistics Wargame.

I. OTHER TOPICS

1. Bonus and Penalty Points

Bonus or penalty points are accrued for a variety of reasons throughout the Operational Logistics Wargame. Since the goal of this wargame is to increase the student's understanding of Operational Logistics, logistics actions are weighted more heavily than combat actions. To calculate inventory bonus and penalty points, the wargame tracks the inventory on a daily basis and awards points accordingly for daily inventory status. Bonus points increase the total score and penalty points decrease the total score. The current point values assigned to relevant actions are shown in Table 12.

Action	When calculated	Bonus Value	Penalty Value
Damage to a ship during an attack	on event		1
Destruction of a ship	on event		5
Port visit by any non-CLF ship	on event		1
Zero inventory of specific logistic item	daily, per ship		5
Zero inventory of specific weapon	daily, per ship		5
> 0 & <= 40% onboard of specific logistic item	daily, per ship		1
> 0 & <= 40% onboard of specific weapon	daily, per ship		1
>=40% & < 50% onboard of specific logistic item	daily, per ship	0	0
>=40% & < 50% onboard of specific weapon	daily, per ship	0	0
>= 50% onboard of specific logistic item	daily, per ship	1	
>= 50% onboard of specific logistic item	daily, per ship	1	
Use of Urgent priority for an unrep request	on event		20
Attempting unrep that is denied	on event, per supply class		1
Delaying deployment to replenish inport	on event		1
Trying to fire a weapon with zero inventory	on event		25
Firing a weapon not in range of target	on event		5
Zero inventory of Point Defense weapon in attack	on event		1
Destruction of Surface Threat	on event	5	
Destruction of Air Threat	on event	1	
Destruction of Enemy Base	on event	10	
Damage to Surface Threat	on event	1	
Damage to Enemy Base	on event	1	

Table 12. Bonus and Penalty Point Values

2. Surface Threats and Air Threats

Surface Threats and Air Threats are randomly generated throughout the wargame. The threat generation processes use random number generators to determine the length of time between threat initializations. Each surface threat and air threat is assigned a unique identification tag when it is generated. The identification tag indicates what class of threat it is and the time that the asset was generated. Damage to a specific threat asset does not affect any other threat asset, including assets of the same class. The current

version of the Operational Logistics Wargame allows generation of surface threats and air threats for the duration of the simulation run, regardless of how many threat assets are damaged or destroyed. Threat assets always fire weapons when detecting Battle Group assets. Threat assets do not have a specific fire power but fire only once per Battle Group target per detection.

3. Scheduling Port Visits and Underway Replenishments

The current version of the Operational Logistics Wargame does not have a method to specifically schedule a port visit or underway replenishment. To make a port visit, the player should generate a waypoint near the desired port for the desired ship(s) in the *Change Coordinates* or *Add Coordinates* panel. To schedule an underway replenishment, the CLF ship and the desired recipients should be scheduled to sail along several waypoints as a group using the *Change Coordinates* panel.

IV. REFEREE MANUAL

By using an external database for configuration, the Operational Logistics Wargame structure allows the referee considerable control over the wargame. With basic knowledge of the Java programming language, the referee has control over virtually every aspect of game play.

A. DATABASE CONTROL

The referee may assign specific ships to the wargame's fleet by making changes as desired to the "CVBG" table. Such changes may include increasing or decreasing the size of the Battle Group or the addition of shuttle ships to the fleet. The default CVBG table has only a single CLF ship which performs the role of both a station ship and a shuttle ship. The CVBG table also includes the starting position of the assigned ships. Other tables in the database are also easily modified to change details such as ship logistic capacities, logistics items, weapon capacities, and sensor data. The referee should take care to ensure that any changes made to the database retain compliance with JAVA and SQL programming rules and that changes are made throughout the database. For example, changes to a specific weapon's name should be made in both the "TypeWeapons" table and the "Weapons" table. See the Methodology Chapter for more information on the database design. Each ship in the Battle Group is assigned a Staying Power value. For this wargame, this value depends not only upon a realistic staying power for that type of ship but also the defensive value (i.e. whether or not the ship has defensive weapons).

B. JAVA CODE CONTROL

With minimal knowledge of the JAVA programming language, the referee may also make changes to numerous other parameters of the game.

1. Maps, Bases, and Coordinate System

Any appropriate jpeg file may be used as the background map provided it is renamed to match that noted in specific JAVA classes. However, some coordinates are hardwired into the program and are based on the default jpeg file's geography. Therefore, if the background map is changed, those coordinates may require modification

too. Note that the coordinate system of the game is that used by JAVA. The top left corner of the map is the origin with horizontal positions (x) increasing to the right and vertical positions (y) increasing downward. (Use of the JAVA coordinate system prompted the drastic decrease in range of Tomahawk.)

If the map is changed, coordinates which may need modification include the Battle Group's starting position, which is found in the *oplog* database, friendly and threat port positions, threat base positions, threat surface ship tracks, and threat air tracks. Port and base coordinates, as well as the number and names of them, are specifically assigned within the Java code. The code used for port and base parameters can be easily modified with additions or deletions or can be modified to draw from an outside source such as another database table which would make it easier to modify the data. Both threat surface and threat air tracks are randomly selected from a group of predetermined tracks that are coded directly into specific JAVA classes. These classes can also be easily modified directly or modified to draw from an outside source.

2. Penalty and Bonus Points

Penalty and Bonus points are accrued on various occasions during game play. Point increments range from 1 to 25 points, depending upon the event for which the bonus or penalty is being awarded. The default amount for bonus and penalty awards is one point. A second method for each type of award allows the code writer to specify how many points are to be awarded. The referee can change, or eliminate these values wherever desired. See the Student Manual for a specific list of currently assigned bonus and penalty point values. Penalty and bonus points are changed by making the desired changes directly in the applicable Java class.

3. Random Variables

If desired, the referee can change the mean interarrival time of air and surface threats and the mean detection time of each sensor. The referee can also change the probability of hit and probability of kill rates for both Battle Group and threat attacks, as well as the logistics and weapons status determinations at the beginning of the wargame. The referee has control over the type of random number generated for any random number generation.

4. Other Variables

Interspersed throughout the JAVA code are numerous variables that are easily changed by the referee. These variables include:

- Threat sensor range,
- CLF sensor range,
- Friendly base sensor range,
- Air threat speed,
- Enemy base Staying Power,
- Air threat Staying Power,
- Surface threat Staying Power,
- Total number of enemy weapons,
- Maximum number of air threats at any given time,
- Maximum number of surface threats,
- Maximum number of days to run the wargame.

C. MISCELLANEOUS CONTROL

The speed at which the simulation runs can be controlled by the Referee. Additionally, the frequency of screen refresh updates can be controlled by the Referee. The wargame intelligence scenario and summary of game rules are simple Strings of text that can be modified at any point. In future versions of the Operational Logistics Wargame, the intelligence scenario and summary of game rules are designed to be simple text files that may be modified at the referee's discretion.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

The Operational Logistics Wargame is a combat model with a solid Operations Research foundation. It is specifically designed for Operational Logistics students. It will help the Naval Postgraduate School meet the continuing goal of training Logistics Officers to make effective decisions in combat situations. The wargame has numerous advantages over existing combat models, particularly PRO-LOG that make it ideally suited to meet the needs of its users. The major advantages are discussed below. Also, it is recognized that to best meets the needs of the Operational Logistics curriculum, building this wargame will span the work of several theses. To that end, key recommended enhancements are discussed in this chapter. Additional notes on enhancements can be found in Appendix B.

A. ADVANTAGES

1. Flexible

This wargame is expandable and configurable. The Carrier Battle Group along with its supply and weapons lists can be tailored in size by simple changes to the accompanying database. Other aspects of combat can be added to the wargame without changing the basic model. The scenario, including the geographic region can be modified at will. PRO-LOG was neither expandable nor configurable. PRO-LOG had a single scenario.

2. Modern

The Operational Logistics Wargame uses the most up-to-date modeling and simulation techniques. The graphical user interfaces provide a modern, intuitive, windows-based environment in which to run the wargame. The wargame's predecessor was a deterministic, text-based model.

3. User-Friendly

Detailed graphical displays throughout the game give players immediate feedback on their progress. Players have direct control over their units for both logistical and combat aspects of the game.

4. Quick Starter

Training time to use this wargame is minimal and is commensurate with the classroom time devoted to the subject.

5. Portable

The Operational Logistics Wargame can be installed on individual personal computers or laptops and can be run on all major operating systems.

B. RECOMMENDED ENHANCEMENTS

1. State Variable Statistics

The simulation model makes extensive use of *firePropertyChange()* methods to indicate that certain variables have changed. Examples of these variables include arrivals of enemy aircraft, detection of surface threats, underway replenishments, and port visits by Battle Group assets. Statistical data can be collected about these variables with accompanying statistical analysis performed. Code should be added to collect these simulation data with subsequent analysis done and reported to the player.

2. Logistics and Weapons Inventory Statistics

Data on instantaneous logistics and weapons inventory levels is calculated and reported throughout the simulation. Code should be added to monitor these levels for later analysis. One possible approach would be to use *firePropertyChange()* methods to report inventory levels whenever the inventory levels are calculated for a particular simulation time. Although consumption of logistics items is continuous, consumption is only calculated whenever the game is paused. Due to the Discrete Event Simulation nature of the Operational Logistics Wargame, the pauses are not periodic and thus the statistics reported on the inventory levels would not be periodic either. This complicates collection and reporting of simulation-wide statistics on logistics and weapons inventory levels. This area requires additional research.

3. Other Game Play Statistics

In other areas of game play such as combat, results of the wargame or score categories should be tracked. Relevant data such as number of enemy targets damaged or destroyed, number of weapons fire ‘misses’ due to zero inventory, and number of Battle

Group assets damaged, repaired, and destroyed should be available for analysis. Additionally, statistics on how the score of the game was reached should be gathered and reported.

4. Cookie Cutter Sensors

Underway replenishments are conducted when ships are within a certain range of the CLF ship. The sensor used to detect and subsequently schedule underway replenishment events is the same constant rate sensor used by all assets. Since the detection is based on a random number generator, this can result in the occasional non-detection of the combatant ship by the CLF ship and the resulting failure to conduct underway replenishment. The *CLF sensor* on the station ship should be a *Cookie Cutter Sensor* that always detects its target rather than the constant rate sensor. Addition of a *CookieCutterSensor* and its *Mediator* as well as modifications to the *ConstantRateMediator*, *ConstantRateSensor*, and *Referee* instantiations are among the changes that will be necessary to refine this feature.

5. Enhancing Realism

The Operational Logistics Wargame includes stochastic methods in certain areas of game play such as Surface Threat generation, Air Threat generation, damage and destruction calculations, and sensor detections. However, the game could be enhanced by adding more random events. For example, if a ship is damaged during an attack, a requirement for specific repair parts (by weight) could be randomly generated. Or, a storage tank/room might be destroyed which would reduce the ship's capacity until repaired inport. Or, unordered logistics items, such as seasonal foul-weather coats for the whole crew, might be added to a ship's unrep request at the time of delivery which would increase the amount of stores received. These events would serve to confound the unrep request process in the wargame and make it more closely reflect reality.

6. Hardwired Data

a. Instance Variables

In the program code, there are many instance variables that have their values hardwired into the code. These variables range from the number of days to run the simulation to the range of the CLF sensor. These variables, more properly, should refer

to JAVA properties files to get their values. Program code throughout should be modified accordingly.

b. Friendly and Enemy Bases

Both Friendly and Enemy Base data are written directly into applicable JAVA classes. As is other Friendly and Enemy force data, this data should be retrieved from an outside database. The classes should be modified to obtain the data from an outside source.

c. Threat Air and Threat Surface Movement

Threat Air and Threat Surface movements are based on preset coordinates. Although each path is randomly selected, there are only a set number of paths to choose from. These choices are "hardwired" into applicable classes. First, these paths should be obtained from an outside source with the code modified accordingly. Second, more paths should be added. Third, when methods have been implemented to differentiate between land and water, Threat Air and Threat Surface movement should become more random with some sort of random path generator.

C. CONCLUSION

The Operational Logistics Wargame as presented by this thesis is a functional tool for use in introductory coursework of the Operational Logistics Curriculum at the Naval Postgraduate School. It presents a combat logistics system complex enough to challenge the intended audience along with a player-friendly data presentation; this wargame is ready for immediate use. The flexible design provides a solid basis for future expansion or other modification. And, the use of Java, a modern program language, ensures that this tool can be easily maintained.

APPENDIX A . THE OPLOG PACKAGE

This appendix contains detailed information on each class written for this simulation. All classes are contained in the package oplog or in subpackages such as oplog.database for databases and related classes, oplog.gui for classes used to make graphical user interfaces, and oplog.smd for classes used by the discrete event simulation. Classes not discussed in the appendix are drawn from java, javax, and simkit packages. The details noted here about each class are considerably more in depth than those found in the Javadocs. All program code and applicable Javadocs are included in some electronic versions of this thesis. Some classes that are not used in the current version of the Operational Logistics Wargame have also been included in those versions as a basis for future development.

1. OPLOG.DATABASE CLASSES

Classes in the oplog.database package are those classes of the Operational Logistics Wargame whose main purpose is to interact with external databases. These classes generally retrieve database information and return the data to the instantiating class.

a. DataBaseInfo.java

The DataBaseInfo Class contains the database information methods that retrieve specific data from the Oplog.mdb database. Performs look up of ships and ship classes from the CVBG (Carrier Battle Group) Table. Uses that information to look up data in other tables.

b. DataRepository.java

The DataRepository Class has two primary functions. Its first function is as a go-between for the IntelSummary class and whatever follows. In the current version of the Operational Logistics Wargame, the DataRepository is created by IntelSummary and receives information created by IntelSummary. The Constructor requires information on the game option the player has chosen in the form of an integer. The Constructor also calls for 7 vectors of information gathered in the IntelSummary instance that instantiated the DataRepository. Based on the information received, the DataRepository then

implements one of two options for continuing the wargame. The second function of this class is to store information about weapons and logistics of the fleet during the entire wargame. This class also provides storage for information about the CLF ship and underway replenishment requests. It contains methods to update unrep requests and CLF inventories. The DataRepository provides methods for adding bonus or penalty points to the accumulated score and maintains the accumulated score.

c. LogRates.java

The LogRates Class is a database intermediary that gets the logistics consumption rates and storage requirements. The Constructor requires input of two Vectors. The first is a Vector of ShipHashMaps. Each ShipHashMap contains the ship names, ship classes, and ship capacity for all logistical items of the named ship. The elements of the second Vector are the fleet's logistic items as Strings. A Vector of basic logisitics capacities is used rather than the current logistics status of each ship since consumption rates are added to the input ShipHashMap and the updated version is returned upon request. This version adds a rate key and storage key to each ship for each logistical item.

d. ThreatDataGetter.java

The ThreatDataGetter is the interface between the Operational Logistics Wargame and the Threat.mdb database. It has functionality similar to that of the DataBaseInfo class and retrieves information for use by Surface Threats.

e. WeaponsData.java

The WeaponsData Class is an intermediary that gets additional information about weapons from the database. The Constructor requires input of two Vectors. The first Vector is a Vector of ShipHashMaps. Each ShipHashMap should contain the ship names, ship classes, and ship capacity for all weapons of the named ship. The elements of the second Vector are the fleet's weapons in String form. A Vector of basic weapons capacities is used rather than the current weapons status of each ship since static information is added to the input ShipHashMap and the updated version is returned upon request. This version adds a Weight key, Purpose key, and Range key for each weapon to each ShipHashMap.

2. OPLOG.GRAPHICS CLASSES

Classes in the oplog.graphics package provide generic graphics services to the Operational Logistics Wargame. These classes do not provide any information to the player concerning the logistical or combat status of the wargame.

a. Animate.java

Animate is the animation class of the Operational Logistics Wargame Simulation. This Class extends JFrame and implements SimEventListener. The Constructor requires a Vector whose elements are the Battle Group's ship names as Strings and reference to the current instantiations of LogStatus, WeaponsStatus & DataRepository. The Animation methods are borrowed from the AnimationTest Class written by Professor A. Buss. The Animation Frame has three main areas. The first is an animation screen display of the current Ships, Surface Threats, Air Threats, Friendly Bases and Enemy Bases superimposed on a backgound map. Only Air and Surface Threats that are currently detected by Battle Group assets are displayed. The second feature is a panel displaying various simulation data. The third feature is a control panel for pausing, resuming, and exiting the simulation.

b. ControlPanel.java

The ControlPanel class extends JPanel. It is a modified version of Professor Arnold Buss's PingerPanel class. The Control Panel generated by ControlPanel contains the buttons to restart, stop, and exit the wargame. The Constructor requires reference to the current instantiation of the Object that it will be acting upon. In the current version of the Operational Logistics Wargame, this panel is one of the three main elements of the animation screen and is instantiated in the Animate constructor.

c. Pinger.java

The Pinger class extends SimEntityBase and implements Runnable. It is borrowed directly from PingThread2.java by Professor Arnold Buss. The Constructor requires doubles representing the time between ping events and number of realtime milliseconds per Simulation Time. The Constructor also requires a Boolean representing the desired status of the pinger.

d. SimTimePanel.java

The SimTimePanel class extends JPanel. It is a component of the actions tab of the animation window. It displays the simulation time and is updated to reflect the current simulation time whenever the game is automatically paused.

3. OPLOG.GUI CLASSES

The oplog.gui package contains classes that provide information to the player in the form of Graphical User Interface displays. All information concerning both the logistical and combat progress of the game is presented by these classes.

a. AfterWelcome.java

The AfterWelcome Class is a go-between for the OplogWelcome Class and the IntelSummary Class. It is instantiated by OplogWelcome and reacts to the selection of the specific buttons from the OplogWelcome Class. The Constructor requires an integer representing which button was chosen. This class enables the wargame structure to be easily modified once the Resume feature is available by adding more specific button actions without modification to the OplogWelcome or IntelSummary Classes. This Class instantiates a PleaseWaitPanel and the IntelSummary Classes. The IntelSummary Class is instantiated within a SwingWorker Class to allow the PleaseWaitPanel to display properly.

b. CasGroupSetter.java

The CasGroupSetter Class extends JPanel and implements MouseListener. It is a GUI for the Operational Logistics Wargame. The Constructor requires Vectors of Strings representing ships in the Battle Group and ships sailing individually as well as reference to the current instantiation of the DataRepository. This GUI allows the player to set the course and speed for the Battle Group. It generates a frame containing a map background, a speed slider, and three option buttons.

c. CASListener.java

The CASListener Class implements ItemListener. The Constructor requires the name of the ship that it is representing in String form. It provides Listeners for the CheckBoxes in the Course and Speed, the MoreCourseAndSpeed and MoreCourseAndSpeedUnits Classes.

d. CasUnitSetter.java

The CasUnitSetter Class extends JPanel and implements MouseListener. It is a GUI for the Operational Logistics Wargame. The Constructor requires a Vector of Strings representing the ships that will sail individually and reference to the current instantiation of DataRepository. The CasUnitSetter Class allows the player to set course and speed for individual ships of the CVBG. It displays a frame containing a map background with a separate tab for each ship containing a speed slider, and three option buttons.

e. CourseAndSpeed.java

The CourseAndSpeed Class extends JPanel. It is a GUI for the Operational Logistics Wargame that allows the player to decide how to set the Course and Speed for the CVBG. The Constructor requires reference to the current DataRepository. This class is instantiated by the DataRepository Class.

f. FireResultsPanel.java

The FireResultsPanel extends JPanel. It is a GUI for the Operational Logistics Wargame. This GUI provides Battle Damage Assessment of targets in text format. This class is instantiated by the FireWeaponsPanel Class and its Constructor requires a String representing the results of weapons firing.

g. FireWeaponsPanel.java

The FireWeaponsPanel extends JPanel. It generates a GUI for the Operational Logistics Wargame. This GUI allows the player to fire weapons at known threats. It uses a uniform random number generator to determine if the threat is damaged or destroyed. The probability of hit and probability of kill rates are slightly higher against threats than threat attacks against Battle Group assets due to technology differences and weapons limitations for Battle Group assets. The Constructor requires a Vector of existing wargame Ships, a Vector of current wargame Threat assets and EnemyBases,

and reference to the current DataRepository. This Class is instantiated by the SelectActionsPanel.

h. IntelSummary.java

The IntelSummary Class extends JFrame. It generates a GUI for the Operational Logistics Wargame. The GUI provides opening scenario information. This information includes an intelligence summary as well as the Carrier Battle Group composition and logistical status. This class uses a uniform random variate to determine fuel, weapons, and stores onboard each ship at the start of the wargame. Each ship has 75-100% of its capacity for each item onboard with the inventory level of each determined independently.

i. LogProgressBar.java

The LogProgressBar Class extends JProgressBar. It is a ProgressBar specialized for the Operational Logistics Wargame Intelligence Summary. It is used to display percentage on board versus total capacity of the indicated logistics item or weapon. The Constructor requires a String representation of the maximum capacity and an integer representing the current inventory value.

j. MoreCasGroupSetter.java

The MoreCasGroupSetter Class extends JPanel and implements MouseListener. Based on the CasGroupSetter Class, it also allows the player to set the Course and Speed for the CVBG. The Constructor calls for input of two Vectors and one Boolean. The first Vector contains a list of ships, presented as Ship vice ShipHashMaps, that will have course and speed set by the MoreCasGroupSetter instantiation. The second Vector contains a list of ships to be sent on to the MoreCourseAndSpeedUnits Class. This class instantiates the MoreCourseAndSpeedUnits Class if the indicated Vector is not empty. Otherwise, this class adds new coordinates to a Ship's path or makes a new path for the Ship as indicated by the input Boolean.

k. MoreCasUnitSetter.java

The MoreCasUnitSetter Class extends JPanel and implements MouseListener. It allows the player to set additional Coordinates for individual ships in the CVBG. The Constructor calls for input of a Vector and a Boolean. The Vector contains a list of ships, presented as Ship vice ShipHashMaps, that will have course and speed set by this Class.

This class adds new coordinates to a Ship's path or makes a new path for the Ship as indicated by the input Boolean.

l. MoreCourseAndSpeed.java

The MoreCourseAndSpeed Class extends JPanel. This GUI allows the player to select which ships to set a group course and speed for. The Constructor requires a Vector of Ships and a Boolean. The Vector of Ships is used to construct a list of ships that can have course and speed set. Ships chosen to be in the group added to the group Vector and others are added to the units Vector. If any Ships are in the group Vector, the MoreCasGroupSetter Class is instantiated and is passed both Vectors and the Boolean. If the group Vector is empty, this Class instantiates a MoreCasUnitSetter Class and passes the MoreCasUnitSetter the new unit Vector of Ships and the Boolean received by the Constructor.

m. MoreCourseAndSpeedUnits.java

The MoreCourseAndSpeedUnits Class extends JPanel. It allows the player to select which ships to set individual course and speed for. The Constructor requires a Vector of Ships and a Boolean. The Vector of Ships is used to construct a list of ships that did not have course and speed set by the instantiating MoreCasGroupSetter Class. If any ships are chosen for individual course and speed additions, a new Vector of ships is created and this Class instantiates a MoreCasUnitSetter Class This class passes the MoreCasUnitSetter the new Vector of Ships and the Boolean received by the Constructor.

n. OplogWelcome.java

The OplogWelcome Class is the opening GUI for the Operational Logistics Wargame. It has no real function other than to start the wargame and contains the main method of the wargame. The code used in this Class to generate panels is organized differently, and is perhaps, more clumsy than code used to generate panels later in the wargame. This class intantiates the AfterWelcome Class.

o. PleaseWaitPanel.java

The PleaseWaitPanel extends JPanel and generates a simple text frame to provide information to the player. This GUI tells the player that the wargame is running correctly and loading data from the database. This Class is instantiated whenever necessary. The

current version of the Operational Logistics Wargame instantiates this Class in the AfterWelcome, CasGroupSetter and CasUnitSetter classes.

p. SelectActionsListener.java

The SelectActionsListener Class implements ItemListener. It provides Listeners for the CheckBoxes in the Select Actions GUI of the Operational Logistics Wargame.

q. SelectActionsPanel.java

The SelectActionsPanel extends JPanel. This GUI generates a Radio Button Group that offers the player several choices of actions that may be taken when the game is paused. The Constructor calls for a Vector of the existing Ships.

r. SliderListener

The SliderListener Class is an implementation of ChangeListener. This listener tracks changes to a speed slider from various Course and Speed setting GUIs. Constructor requires input of the ship's name and a default speed. When used by the MoreCASUnitSetter Class, the default speed is the ship's current speed. Otherwise, the speed is usually the maximum speed.

s. SwingWorker.java

The SwingWorker Class is a Class for the Operational Logistics Wargame. It provides simple thread capabilities for various sections of the wargame. In the current version of the Operational Logistics Wargame, SwingWorker is instantiated when needed. It is copied directly from the JFC Swing Tutorial: A Guide to Constructing GUIs. Other information about this class is provided from the original document:

> This is the 3rd version of SwingWorker (also known as SwingWorker 3), an abstract class that you subclass to perform GUI-related work in a dedicated thread … Note that the API changed slightly in the 3rd version: You must now invoke start() on the SwingWorker after creating it.

t. TheLogPanel.java

The LogPanel Class extends JPanel. It generates a panel for the animation frame of the Operational Logistics Wargame Simulation. This frame is used to display a running log of textual information concerning the game status. This panel is updated whenever the game is paused automatically. The Animate Class instantiates this Class.

4. OPLOG.LOG CLASSES

The oplog.log classes contain the methods for consuming, replenishing, and tracking inventory of logistical items and weapons in the Operational Logistics Wargame. These classes use information obtained by oplog.database Classes from external sources or use information stored as instance variables in oplog.database Classes.

a. CheckUnrepSchedule.java

The CheckUnrepSchedule Class is a class that provides additional functionality for the DataRepository class. The Constructor needs reference to the current DataRepository and a Ship representation of the ship whose underway replenishment schedule should be checked. This Class is used by various Classes throughout the Operational Logistics Wargame to check whether a ship's unrep requests should be filled. Normally this class is used when the CLF ship detects another ship. If an unrep is scheduled, the appropriate amount is added to the ship's inventory of that item. If the new total could be above maximum capacity, new total is changed to maximum capacity. When a previously scheduled underway replenishment is accomplished, the CLF ship's inventory of that item is reduced by the amount ordered. If the unrep was not a scheduled unrep, no action is taken.

b. CLFPanel.java

The CLFPanel Class extends JPanel. A Vector representation of the CLF ship's capacity is required by the Constructor. It calculates and displays inventory levels of the CLF ship for each broad class of supply at the beginning of game play. This class uses a uniform random variate to determine fuel, weapons, and stores inventory onboard the CLF ship at the start of the wargame. The starting inventory is 75-100% of the maximum capacity onboard with the inventory level of each determined independently.

c. CLFStatus.java

The CLFStatus Class extends JPanel. It retrieves and displays inventory levels of the CLF ship for each broad class of supply throughout game play. This Class uses the same basic code as that of the CLFPanel Class but reports current inventory levels rather than generating random amounts. The Constructor requires reference to the current DataRepository.

d. DeleteUnrepRequest.java

The DeleteUnrepRequest Class is a class that provides additional functionality for the DataRepository Class. It is used to delete all of a specific ships's orders from all unrep request lists. This Class is normally used only when a ship has been destroyed or when a ship makes a port visit. The Constructor requires reference to the current DataRepository and the indicated ship.

e. LogStatus.java

The LogStatus Class is the workhorse for logistics in the Operational Logistics Wargame. It calculates consumption of logistical items. Also calculates penalties or bonus points for daily inventory status. Penalties and bonuses are only calculated and charged every 24 hours. This Class provides a method to replenish all logistical items for a ship when a port visit is made by that ship. In calculating consumption, it uses consumption rates obtained from the Oplog database. The Constructor requires reference to the current DataRepository.

f. OrderListener.java

The OrderListener Class implements ChangeListener. It is a listener for the Operational Logistics Wargame. This listener tracks changes to its order slider from the Ship Order Panel. Constructor requires input of a default order amount.

g. PlacedRASRequestsPanel.java

The PlacedRASRequestsPanel Class extends JPanel. A GUI for the Operational Logistics Wargame, it displays all current Unrep requests. Each is displayed with a radio button. If the player selects a specific Unrep request, a ShowOrderPanel for that order is instantiated. The Constructor requires reference to the current DataRepository.

h. ScorePanel.java

The ScorePanel Class extends JPanel. Drawing information from the current instantiation of the DataRepository Class, it displays the current score of wargame.

i. ShipRASRequestsPanel.java

The ShipRASRequestsPanel Class extends JPanel. It generates a Panel used to request specifc items during the next Unrep between the indicated ship and the CLF ship. All fuel, stores, and weapons available via Unrep are displayed on individual sliders. (Tomahawk and Harpoon are not available via Unrep.) Maximum value of each slider is

the ship's capacity for that item. The player may set routine or urgent priority for the order. The default priority is routine. If urgent priority is selected, a penalty is accessed. The Constructor requires the name of the specified ship in String form and reference to the current DataRepository.

j. ShowOrderPanel.java

The ShowOrderPanel Class extends JPanel. It generates a panel that displays all pertinent information about the selected Unrep request such as amount and types of stores, fuel, or weapons ordered; whether the Unrep is currently scheduled to be filled; and the total amount of the order. Due to the use of the HashTable Class, the information does not appear in any particular order. The Constructor requires reference to the indicated order in ShipHashMap form.

k. UnrepScheduler.java

The UnrepScheduler Class provides a method to review all existing unrep requests and determine which orders should be filled based on priority of each order, FIFO status, and CLF inventory. Once the requirements to be filled can no longer be met, no other requests of that category are considered for filling, reqardless of priority. Fuel, Weapons, and Stores requests are considered independently from each other. This class should be used to update the Unrep schedule, whenever necessary, including after any port visit, any new request, or when a ship is destroyed. No partial order filling occurs. The Constructor requires reference to the current DataRepository.

l. UnrepSummaryPanel.Java

The UnrepSummaryPanel Class extends JPanel. It generates a GUI for the Operational Logistics Wargame. This GUI allows the player to review previously placed unrep orders and place additional orders. The Constructor requires a Vector of the existing Ships and reference to the current DataRepository.

m. WeaponsStatus.java

The WeaponsStatus Class is the workhorse for Weapons Inventory. Similar in methodology to the LogStatus Class, it tracks consumption of weapon items. The Constructor requires reference to the current DataRepository.

5. OPLOG.SMD CLASSES

The oplog.smd package contains all classes that form the Discrete Event Simulation portion of the Operational Logistics Wargame.

a. AirBG.java

The AirBG Class extends OplogMover. It is an OplogMover that is specialized to be a Battle Group Air Asset. It generates a unique tag for each unit that is created so, no matter how many are created, no two are alike. This Class is not used in the current version of the Operational Logistics Wargame. The Constructor requires the origin as a Coordinate and the maximum speed as a double.

b. AirThreat.java

The AirThreat Class extends OplogMover. It is specialized for the Arrival Process. It generates a unique identifier tag for each AirThreat generated. AirThreats are instantiated by the AirThreatGenerator Class. The Constructor requires the origin as a Coordinate and the maximum speed as a double.

c. AirThreatCourseGenerator.java

The AirThreatCourseGenerator extends SimEntityBase. This class contains a group of predetermined courses for Air Threats. The courses are Vectors of Coordinates. This Class uses a uniform random number generator to determine which of those courses will be assigned to a specific AirThreat.

d. AirThreatGenerator.java

The AirThreatGenerator Class extends SimEntityBase. Constructor requires input of two doubles to use as X and Y coordinates for the unit's homeport, a double as maximum speed, a long as the seed for a random number generator, and a reference to the current wargame's instance of the Controller. This Class Listens for the Arrival Process designated for AirThreat Arrivals. Upon notification that an arrival has occurred, the AirThreatGenerator randomly generates a new AirThreat with a starting position generated for the unit based on the input homeport parameters and a speed as passed in the Constructor. The AirThreat's course is generated by a call to the AirThreatCourseGenerator. This Class is instantiated in the Deployment Class. This Class instantiates an AirThreat, an AirThreatCourseGenerator and a PathMoverManager.

e. ArrivalProcess.java

The ArrivalProcess Class extends SimEntityBase. This class implements an arrival process. (See *Simulation Modeling and Analysis* by Law & Kelton for additional information on the arrival process.) This Class has a variety of Constructors. The current version of the Operational Logistics Wargame uses the Constructor that requires a RandomVariate class name and appropriate parameters the RandomVariate Class. The Arrival Process is instantiated in the Deployment Class

f. BattleGroupMoverManager.java

The BattleGroupMoverManager extends SimEntityBase. It is a variation of the PathMoverManager Class and is specialized for use with some OplogMovers of the Operational Logistics Wargame. It can be used with OplogMovers that have a ConstantRateSensor. This function of this class is to control movement from one waypoint to another as provided by the player. This Class listens for EndMoves of the assigned OplogMover and schedules movement to the next waypoint, as available. If no additional waypoints are present, provides notification of same. Provides methods to add additional waypoints to a Vector of Coordinates or to establish a totally new Vector of Coordinates. When the OplogMover is an instance of the Ship Class, it checks the current speed of the Ship; otherwise, it uses the default maximum speed. The Constructor requires the applicable Mover and a Vector of Coordinates. This Class is instantiated in the Deployment, FriendlyBaseGenerator, and EnemyBaseGenerator Classes.

g. ConstantRateMediator.java

The ConstantRateMediator extends SimEntityBase and implements Mediator. It is a Mediator specialized for use with a Constant Rate Sensor. This version does not contain methods for use with AirBG assets. This class uses a Exponential Variate generator to determine if and when a ConstantRateSensor detects and undetects a target. The mean detection time used to generate the Exponential Variate is obtained by a call to the specific Constant Rate Sensor's parameters. This Class uses the broad types of sensors, targets, and mover defined in the current MediatorFactory to refine specific detection parameters. The Constructor needs the current Sensor and the Mover it is targeting.

h. ConstantRateSensor.java

The ConstantRateSensor Class extends BasicSensor. When used in conjunction with a ConstantRateMediator, makes detection of targets a stochastic process rather than a deterministic process. The Constructor requires input of the Sensor's Mover, the Sensor's range, and the mean detection time.

i. Controller.java

The Controller Class extends SimEntityBase and is a centralized control center for the Operational Logistics Wargame simulation. It provides internal controls over the animation panel. It listens for arrivals of new SurfaceThreats and AirThreat. Upon arrival of a new threat asset, the threat is registered with the referee. Additionally the Controller determines at which point, if any, a threat asset is added to the animation panel or removed from the animation panel. This Class unregisters obsolete targets, handles destruction of all friendly and threat assets, and listens for final End Moves of Battle Group assets. It prompts reporting of specific events to the player and orders automatic simulation pauses. The Controller Class is instantiated in the Deployment Class. The Controller Class instantiates the Animate Class. Its Constructor requires input of the current Referee, a Vector of the current Ships, and references to the current LogStatus, WeaponsStatus and DataRepository instantiations.

j. Deployment.java

The Deployment Class extends SimEntityBase. It is the central class for the Discrete Event Simulation portion of the Operational Loogistics Wargame. Without the lead-in GUIs and database retrieval Classes, this would be the Main Method for the simulation. This class instantiates the ArrivalProcess Classes for SurfaceThreat and AirThreat Mover generation. It instantiates the EnemyBaseGenerator and FriendlyBaseGenerator Classes. The CVBG ships are instantiates as Ship Movers in this Class. The Ships' Air Sensors and Surface Sensors are instantiated as ConstantRateSensors. A special CLF sensor, used by the CLF ship to conduct Unreps when alongside a CVBG ship, is instantiated in the Deployment Class as a ConstantRateSensor. The Surface Sensors used by FriendlyBases to detect ship port visits are instantiated in this class. All Classes required to run a Simkit Discrete Event Simulation are instantiated in this Class including the Referee and

ConstantRateMediators (using Mediator Factory). The LogStatus and WeaponsStatus Classes are instantiated by the Deployment Class. This Class contains code to track certain statistics gathered during the wargame but the current version of the Operational Logistics Wargame does not report the final statistical results. The Constructor requires reference to the current DataRepository.

k. EnemyBase.java

The EnemyBase Class extends OplogMover. It is a Basic Mover specialized for enemy bases. Current design uses hard-wired data in the EnemyBaseGenerator Class to assign instance variables. Can be modified to use information drawn from an outside source. Although this Class is a child of the Mover Class, it doesn't actually move since it has speed of zero hard-wired. Otherwise, it fulfills all requirements to be a Simkit Mover. Extending the Basic Mover and Mover Class to this class enables a broad range of possibilities for interaction with other components of Simkit. The Constructor requires the origin as a Coordinate and the maximum speed.

l. EnemyBaseGenerator.java

The EnemyBaseGenerator Class extends SimEntityBase. The Constructor requires input of references to the current Operational Logistics Wargame instances of the Referee and the Controller. This Class is a very basic Mover Generator. It uses hardwired data to set parameters for its Movers and is only able to generate a deterministic number of EnemyBases. This Class generates a new EnemyBase for each set of hardwired data it contains. All EnemyBases have a speed of zero and an origin as assigned. The current version of the wargame does not assigned sensors to EnemyBases. This Class is instantiated in the Deployment Class. This Class instantiates a EnemyBases and a BattleGroupMoverManager for each EnemyBase. It registers the new EnemyBases as targets.

m. FriendlyBase.java

The FriendlyBase Class extends OplogMover. It is a Basic Mover specialized for friendly bases. Current design uses hard-wired data in the FriendlyBaseGenerator Class to assign instance variables. Can be modified to use information drawn from an outside source. Although this Class is a child of the Mover Class, it doesn't actually move since it has speed of zero hard-wired. Otherwise, it fulfills all requirements to be a Simkit

Mover. Extending the Basic Mover and Mover Class to this class enables a broad range of possibilities for interaction with other components of Simkit. The Constructor requires the origin as a Coordinate and the maximum speed.

n. FriendlyBaseGenerator.java

The FriendlyBaseGenerator Class extends SimEntityBase. The Constructor requires input of references to the current Operational Logistics Wargame instances of the Referee and the Controller. This Class is a very basic Mover Generator. It uses hardwired data to set parameters for its Movers and is only able to generate a deterministic number of FriendlyBases. This Class generates a new FriendlyBase for each set of hardwired data it contains. All FriendlyBases have a speed of zero and an origin as assigned. Each FriendlyBase is assigned a ConstantRateSensor. This Class is instantiated in the Deployment Class. This Class instantiates FriendlyBases, a BattleGroupMoverManager for each FriendlyBase, and a ConstantRateSensor for each FriendlyBase. It registers the new FriendlyBases' ConstantRateSensors as sensors.

o. GenRandomDBTargets.java

The GenRandomDBTargets Class extends SimEntityBase. This Class uses a stochastic process to determine which data set found in the Threat database should be assigned to a SurfaceThreat when it is generated. The GenRandomDBTargets Constructor requires input of two doubles to use as X and Y coordinates for the unit's homeport, a long as the seed for a random number generator, and a reference to the current wargame's instance of the Controller. This Class Listens for the Arrival Process designated for SurfaceThreat Arrivals. Upon notification that an arrival has occurred, the SurfaceThreatGenerator randomly generates a new SurfaceAirThreat with a starting position generated for the unit based on the input homeport parameters, a ship type as determined by the database random draw and the ship type's associated speed. The SurfaceThreat's course is generated by a call to the SurfaceThreatCourseGenerator. This Class is instantiated in the Deployment Class. This Class instantiates a SurfaceThreat, a SurfaceThreatCourseGenerator and a PathMoverManager.

p. OplogMover.java

The OplogMover Class extends BasicMover. It contains methods specialized for the Operational Logistics Wargame. Methods in this Class are common to all Movers in the wargame. The Constructor requires the origin as a Coordinate and the maximum speed.

q. PathMoverManager.java

The PathMoverManager Class extends SimEntityBase. It is specialized for use with some OplogMovers of the Operational Logistics Wargame. It can be used with OplogMovers that have a ConstantRateSensor. This function of this class is to control movement from one waypoint to another as provided by the player. This Class listens for EndMoves of the assigned OplogMover and schedules movement to the next waypoint, as available. If no additional waypoints are present, provides notification of same. The Constructor needs the applicable Mover and a Vector of Coordinates.

r. Ship.java

The Ship Class extends OplogMover. It is a Basic Mover specialized for ships. Tags each mover with the unique ship's name as found in a database. This class has methods that are unique to Mover objects representing Battle Group ships. All Movers of the Class Ship are instantiated in the Deployment Class. The Constructor requires the ships name and hull type as Strings, its origin as a Coordinate, the maximum and current speeds as doubles, and the ship's Staying Power & mission as Strings. Ships are instantiated in the Deployment Class.

s. SurfaceThreat.java

The Surface Threat Class extends OplogMover. It is specialized for database surface threats. Tags each SurfaceThreat Mover with a unique tag based on the SurfaceThreat type and creation time. Can be used with the Arrival Process. The Constructor requires the ship type as a String, the origin as a Coordinate, and the maximum speed as a double.

t. SurfaceThreatCourseGenerator.java

The SurfaceThreatCourseGenerator extends SimEntityBase. It uses a stochastic process to determine which of several courses will be assigned to a specific SurfaceThreat. Randomly selects a Vector of coordinates from a group of predetermined courses.

6. OPLOG.UTIL CLASSES

Classes in the oplog.util packages are specialized versions of some classes found in the java.util package.

a. MoverHashMap.java

The MoverHashMap Class extends HashMap. It allows HashMaps to be created for each Ship or other Mover though an iterative process without hardwiring ship names into variable names. Thus a new HashMap variable is created for each Ship. Can be used for any mover including Ships. The Constructor requires reference to the applicable OplogMover.

b. ShipHashMap.java

The ShipHashMap Class extends extends HashMap. It allows HashMaps to be created for each ship though an iterative process without hardwiring ship names into variable names. Thus, a new HashMap variable is created for each ship. This method should not be used for movers of Class Ship. See MoverHashMap. The Constructor requires the ship's name in String form.

APPENDIX B. RECOMMENDED ENHANCEMENTS

The Operational Logistics Wargame as presented by this thesis is fully functional; however, there are areas where the design should be enhanced to improve playability, functionality, realism, computer resource usage, and to make the wargame more robust. The modular nature of the JAVA programming language allows for the wargame to be modified in a piecemeal fashion so upgrades to the program can be either major or minor in nature. There are virtually an unlimited number of enhancements that can be made to the Operational Logistics Wargame. This appendix is not intended to be an all-inclusive list of potential upgrades, but rather a recommended starting point for future versions of the wargame. This chapter includes known recommendations organized by design area. Within each design area, the recommendations viewed as most important are noted first.

1. PLAYABILITY AND FUNCTIONALITY ENHANCEMENTS

a. Save and Resume

The single most important upgrade that should be added to the Operational Logistics Wargame is the ability to save a game session and subsequently resume the game at a later time. Considered to be a major enhancement, changes would be necessary to nearly every class in the Operational Logistics Wargame package. One possible way to implement this feature would be through the use of the JAVA Serialization API. In JAVA, objects exist only in the JAVA Virtual Machine and current memory when the program is running. But, by implementing the *Serializable* Interface where needed, objects could become persistent and exist outside the Java Virtual Machine. (Greanier, 2000) Caution is necessary when using the Serializable Interface due to its fragility. Another alternative for adding a Save and Resume function is the through the use of XML. XML offers a much more robust method for the desired Save and Resume features.

b. Course and Speed Visual Aids

The ships' scheduled courses and speeds are not shown during game play, during pauses, or when the courses and speeds are being set. Currently, the course is printed on an output screen whenever a new course is chosen or an existing course has waypoints

added to it; but this is not a player-friendly method of passing the information. Adding a visual reference showing the courses and speeds would enhance the interface between the player and the wargame.

c. Identification of Movers

Methods should be added that allow the player to *point and click* on any Mover displayed during a pause in game play. When a Mover is chosen, pertinent information should be displayed on a pop-up screen such as it's name, speed, and scheduled course. Enemy Movers should only display name, speed, and current direction of movement. It would also be helpful for the player to be able to find out the distance between a specific Mover and any other point on the map.

d. Manual Deletion Of Underway Replenishment Requests

Methods should be added that allow the player to selectively delete Underway Replenishment requests.

e. Underway Replenishment Rendezvous

There is no method that schedules a specific underway replenishment rendezvous between Combat Logistics Force Ships and combatant ships. The current method does not guarantee that the CLF ship and the combatant ship will rendezvous, it only schedules the specified ships to sail toward a specific waypoints. This alternative could replace the "Add new coordinates" selection on the "Select Actions Panel." The "Add new coordinates" does not seem to be as useful as the "Change Course" selection so can probably be eliminated without notice.

2. IMPROVING COMPUTER RESOURCE USAGE AND ROBUSTNESS

a. JDBC Interface

Classes that use the JDBC driver to access the external database are not as efficient as possible. Subsequently, the wargame program runs considerably slow whenever information is needed from the database. These classes could be improved through the use of *joins* wherever possible and other more efficient methods of calling the data. In some cases, data is drawn that is never used. For example, Tomahawk and Harpoon weights are retrieved along with the other weapon weights but since Tomahawk and Harpoon cannot be replenished at sea, their weights are irrelevant. Another example

of irrelevant data being retrieved is that of weapons Range for Point Defense weapons. Modifying the "Select … Where …" code in appropriate classes can eliminate unnecessary look-ups.

b. Logical Class Structure

During development of the Operational Logistics Wargame, some classes were written with more regard to game flow than logical Java class groupings. Although significant revisions have been made, there are still some classes that require modification to remove methods that more logically belong elsewhere or classes that are large and awkward and should be sub-divided. For example, the Animation class was originally intended only to show the player a visual display of the simulation. The animation portion of this class is actually only a small part of the current version. The methods of the Animation class that deal directly with the visual display of the various Movers on the map should be made into a separate class of their own. The Animation class should be renamed to reflect this change and the remaining elements of the class should be further subdivided in other smaller logical classes. Other classes that are excellent candidates for sub-division are the IntelSummary and Deployment classes.

c. Reduce Game Delays

The delay between the opening GUI (OplogWelcome) and the Intelligence Summary GUI (IntelSummary) can be made less noticeable by subdividing IntelSummary. Several of the tabbed panes in IntelSummary do not depend upon data from the database. Those panels can be shown in a separate frame in advance of the remaining tabs. This will minimize the wait time since the player will have something to do while waiting.

3. INCREASING REALISM AND COMPLEXITY

a. Scenario Development

There is no fully developed intelligence scenario. One should be developed. Once other modifications noted herein are made, multiple scenarios, with correspondingly different maps and enemy assets.

b. Friendly Air Assets

Friendly air assets should be added to the model. The Battle Group Air class is included in the Operational Logistics Wargame package but is not currently used. The BGAir class was designed with Fixed Wing aircraft sorties in mind. Using the BGAir class would also require the addition of Air sensors for Surface Threat and Air Threat assets as well as subsequent modifications to classes involving the BGAir movers, their sensors, and sensors to detect BGAir movers such as the Constant Rate Mediator, Constant Rate Sensor, Controller, and Deployment, among others. Allowing the player to set the course and speed for BGAir would be useful. The issue of weaponry aboard each BGAir mover needs to be addressed as does logistics inventory of same. Possibly, the *mission* parameter of each ship could be refined to differentiate aircraft carriers from other combatants which would enable the simulation to determine whether or not a specific ship should have the ability to launch fixed wing aircraft. Rotor wing aircraft are not particularly needed at this stage of the wargame's development.

c. F44 Consumption

Consumption of F44 should be based on the number of sorties per day rather than a fixed "per day" rate. When the BGAir class is implemented for the aircraft carriers, consumption of F44 for the aircraft carriers can be directly linked to the number of sorties flown per day. For other ships, F44 consumption can be based on an average number of sorties per day per ship type. The F44 consumption rate might also take into account whether a ship has been in combat within a 24 hour period. The assumption being that a combat versus non-combat rate per day per ship type could be used.

d. Link Inventory Levels and Ship Capabilities

In addition to the severe penalties encountered when a ship reaches unsatisfactory (or zero) inventory levels, ship capabilities should also be affected by low inventory levels. Methods need to be added to the program code to do this. For example, if a ship runs out of propulsion fuel, its speed should be reduced to zero. If a ship runs out of Point Defense weapons, its probability of being damaged during an attack should be increased. After implementation of BGAir assets, if an aircraft carrier has no F44, it will not be able to launch aircraft.

e. Weapons Use

Weapons ranges are not based on real weapons ranges and all offensive weapons can fire at any target. Defensive weapons do not affect the probability of hit that a Surface Threat or Air Threat has when it attacks a Battle Group asset. (See the Methodology Chapter for more information.) The correct unclassified ranges for all weapons are noted in the database and can be implemented whenever appropriate with only slight code modifications. Modifications to allow weapons to fire only at certain types of targets based on their type will require significant changes to the *FireWeapons* classes. Corresponding changes should also be made to better simulate weapons fire by threat assets. Changes to threat asset weaponry might include adding weapons counter methods to Surface Threat and Air Threat Movers when the Mover is generated.

f. Weapons Inventory

Weapons Inventory is also calculated on rounds fired per event with a single unit used representing the total amount fired per round. For example aircraft carriers typically have the capacity to carry 12000 rounds of CIWS ammunition and fire 3,000 rounds per raid by enemy assets. CLF ships with CIWS (Sacramento and Supply Class ships) also consume 3,000 rounds per raid. However, other ships only use 1000 rounds per raid. In the wargame, consumption has been simplified so that all ships expend the same amount per raid, with capacities adjusted accordingly to reflect an accurate number of possible rounds fired. For transportation purposes, each round weighs approximately .5 pounds with the casing and packing materials. In the Operational Logistics Wargame database, the capacity of CIWS aboard aircraft carriers is 4 units with 1 round expended per enemy raid, and the weight per unit is 3000 rounds x .5 = 1,500 pounds. While technically accurate for some of the ships, this method sets artificial capacities for other ships, and the numbers displayed may be confusing to the player. The oplog database contains rounds expended per raid as well as the simplified versions. Wargame code should be modified to use the actual capacities, weights, and expenditure values. This modification would require changes to both inventory reporting and consumption methods.

g. Multiple Combat Logistics Force Ships

In some classes, the model is not designed to handle more than one Combat Logistics Force Ship. The database contains all classes of CLF ships, and any single CLF ship can be used with its actual capacity; however, if more than one CLF ship is desired in the wargame for both station ship and shuttle ship roles, numerous classes will need to be modified. Less modification is required if all CLF ships are used only as station ships.

h. Additional Combatant and Combat Types

Additional types of combatants can be added to the simulation. Among the possibilities are Subsurface Battle Group assets and Subsurface Threats. Additional types of combat and appropriate forces can also be added such as Amphibious landings by Amphibious forces or land warfare.

i. Simulation times

In the wargame, many events occur instantly when in reality the events are not instantaneous. In some cases this makes no difference to the outcome. For example, a real ship must accelerate or decelerate to change its speed. In the simulation, speed changes occur instantly at the scheduled time. When a simulation is run covering several days, the small amount of time involved in a real speed change is not relevant. However, other events, such as the length of a Port Visit or Underway Replenishment could affect game play and should be modeled to include a delay factor. Creative use of Simkit's *waitDelay()* method where needed may enable the model to realistically include delay times for significant events that are not currently modeled as such.

j. Land versus Water

The current version of the Operational Logistics Wargame does not differentiate between land and water for the movement of ships and aircraft. Methods should be developed that establish boundaries between the two types of geography. Once these methods exist, the scenario, including background map will be easier to change.

k. Refine Stores and Weapons RAS.

The method used to determine if either weapons or stores are replenished at sea is based on a simple aggregate total weight. Additionally, the Java code is hardwired to determine if specific weapons can be replenished at sea. These methods should be

refined to have the determination based on whether or not the CLF ship has a particular item onboard and if specific weapons can be replenished at sea.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Greanier, T., "Discover the Secrets of the JAVA Serialization API", http://developer.java.sun.com/developer/technicalArticles Programming/serialization, (reprinted from *JavaWorld*, July 2000), September 2001.

Mitchell, M. L., *Pro-Log 4.0: An Interactive War Game for teaching the importance of LOGISTICS Planning and Execution*, NPS, July 1988.

Perla, P.P., *The Art of Wargaming*, Naval Institute Press, 1990.

Schrady, D. A., *User's Guide for TACLOGS: Battle Group Tactical Logistics Support System*, NPS, December 1996.

Schrady, D. A., G.K.Smyth, and R. B. Vassian, Predicting Ship Fuel Consumption: Update, NPS, July 1996.

Sterba, J. R., *Operational Maneuver from the Sea Logistics Training Aid*, NPS, September 1999,

Troxell, A. W., *Naval Logistics Simulator*, NPS, September 1999.

Walrath,K. and M. Campione, *The JFC Swing Tutorial: A Guide to Constructing GUIs*, Addison Wesley, November 2000.

www.au.af.mil/au/cpd/cpdgate/clip_af.htm, Air War College Website, November 2001.

www.janesonline.com, Jane's Online 2001 website, November 2001.

www.nationalgeographic.com, National Geographic website, November 2001.

www.webclipart.about.com, Web Clip Art web site, November 2001.

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

Aydin, E., *Screen Dispositions of Naval Task Forces Against Anti-Ship Missiles*, NPS, March 2000.

Blanchette, B.J., Modeling Surface ASW and ASUW Engagements for the Naval Postgraduate School Logistics Wargame, NPS, September 1988.

Bracken, J., M. Kress, and R. E. Rosenthal, eds, *Warfare Modeling*, MORS, 1995.

Buss, A., "Discrete Event Programming with Simkit", *Simulation News Europe*, Issue 32, August 2001.

Campione, M., K. Walrath, and A. Huml, *The Java Tutorial Continued: The Rest of JDK*, Addison Wesley, January 2000.

Greanier, T. , "Discover the Secrets of the JAVA Serialization API", http://developer.java.sun.com/developer/technicalArticles/Programming/serialization, (reprinted from *JavaWorld*, July 2000), September 2001.

Hodges, J. S. and J. A. Dewar, *Is it You or Your Model Talking? A Framework for Model Validation*, Rand, 1992.

Law, A. M. and W. D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 2000.

Le, H. B., *Advanced Naval Surface Fire Support Weapon Employment Against Mobile Targets*, NPS, December 1999.

Mitchell, M. L., *Pro-Log 4.0: An Interactive War Game for teaching the importance of LOGISTICS Planning and Execution*, NPS, July 1988.

Perla, P.P., *The Art of Wargaming*, Naval Institute Press, 1990.

Schrady, D. A., *User's Guide for TACLOGS: Battle Group Tactical Logistics Support System*, NPS, December 1996.

Schrady, D. A., G.K.Smyth, and R. B. Vassian, Predicting Ship Fuel Consumption: Update, NPS, July 1996.

Sterba, J. R., *Operational Maneuver from the Sea Logistics Training Aid*, NPS, September 1999,

Thesis Preparation Manual, NPS, August 1999.

Troxell, A. W., *Naval Logistics Simulator*, NPS, September 1999.

Walrath,K. and M. Campione, *The JFC Swing Tutorial: A Guide to Constructing GUIs*, Addison Wesley, November 2000.

White, S., Fisher, R. Cattell, G. Hamilton, and M. Hapner, *JDBC API Tutorial and Reference, Second Edition: Universal Data Access for the Java 2 Platform*, Addison Wesley, July 2000.

www.au.af.mil/au/cpd/cpdgate/clip_af.htm, Air War College Website, November 2001.

www.janesonline.com, Jane's Online 2001 website, November 2001.

www.nationalgeographic.com, National Geographic website, November 2001.

www.webclipart.about.com, Web Clip Art web site, November 2001.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Dan Boger
   Naval Postgraduate School
   Monterey, California

4. Defense Logistic Studies Information Exchange
   U.S. Army Logistics Management College
   Fort Lee, Virginia

5. Professor David Schrady, Code OR/SO
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

6. Professor Arnold Buss, Code OR/BU
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

7. CDR Kevin J. Maher, Code OR/MK
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California

8. LCDR Carolyn S. Fricke
   5322 Chieftain Circle
   Alexandria, Virginia 22312