2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

AN OBJECT-ORIENTED SHIP-TO-SHORE MOVEMENT ANALYSIS MODEL (CUTTER)

by

Scott E. Shaw

September 1992

Thesis Advisor: Michael P. Bailey

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL *(If applicable)* OR	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000	7b. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS Program Element No. / Project No. / Task No. / Work Unit Accession Number

11. TITLE *(Include Security Classification)*
AN OBJECT-ORIENTED SHIP-TO-SHORE MOVEMENT ANALYSIS MODEL (CUTTER)

12. PERSONAL AUTHOR(S) SHAW, Scott E.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(year, month, day)*	15. PAGE COUNT
Master's Thesis	From To	1992 September	162

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

17. COSATI CODES

FIELD	GROUP	SUBGROUP

18. SUBJECT TERMS *(continue on reverse if necessary and identify by block number)*
Object-Oriented, Amphibious Assault, Simulation, Medium lift replacement, MLR, MODSIM, Simulation

19. ABSTRACT *(continue on reverse if necessary and identify by block number)*
This thesis documents the design and implementation of a simulation of the Ship-To-Shore movement phase of the amphibious assault in a modern, object-oriented, process-based simulation language called MODSIM II by CACI Corporation of La Jolla, CA. The main intent of the simulation is to build a model that will allow the Requirements, Plans and Programs Branch (RP&P), Headquarters, United States Marine Corps (HQMC) to quantitatively compare proposed replacements for the assault aircraft and amphibians currently used in the conduct of the ship-to-shore phase of the amphibious assault. Candidates from the Medium Lift Requirement (MLR) program are compared to identify that mix of aircraft which provides the most rapid build-up of combat power ashore.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS REPORT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael P. Bailey	22b. TELEPHONE *(Include Area code)* (408)646-2085	22c. OFFICE SYMBOL OR/Ba

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

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

An Object Oriented Ship-To-Shore Movement Analysis Model (CUTTER)

by

Scott E. Shaw
Captain, United States Marine Corps
B.S., University of South Carolina, 1980

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1992

Author: [illegible]
Scott E. Shaw

Approved by: [illegible]
Michael P. Bailey, Thesis Advisor

[illegible]
William G. Kemple, Second Reader

[illegible]
Peter Purdue, Chairman
Department of Operations Analysis

ABSTRACT

This thesis documents the design and implementation of a simulation of the Ship-To-Shore movement phase of an amphibious assault in a modern, object-oriented, process-based simulation language called MODSIM II by CACI Corporation of La Jolla, CA. The main intent of the simulation is to build a model that will allow the Requirements, Plans and Programs Branch (RP&P), Headquarters, United States Marine Corps (HQMC) to quantitatively compare proposed replacements for the assault aircraft and amphibians currently used in the conduct of the ship-to-shore phase of the amphibious assault. Candidates from the Medium Lift Requirement (MLR) program are compared to identify that mix of aircraft which provides the most rapid build-up of combat power ashore.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my wife Carla, without whose love, encouragement and assistance I could not have completed this program.

I must also thank my father, for without the first four years of college, the last two would not have been possible.

I. INTRODUCTION

A. BACKGROUND

According to doctrine, the United States Marine Corps (USMC) conducts an amphibious assault in five phases:

- Planning
- Embarkation
- Rehearsal
- Movement to the Objective Area
- Assault (Ship-to-Shore Movement).

The vehicles currently used to conduct the ship-to-shore movement phase of the amphibious assault are rapidly becoming obsolete. There are two reasons for this. First, in the case of the aviation vehicles, the primary medium lift assets (CH-46 and CH-53A/D helicopters) are well past their originally projected service life. Service Life Extension (SLEP) and Special Depot Level Maintenance (SDLM) programs somewhat extended each aircraft type's service life, but a tremendous maintenance effort is required on a daily basis to keep these aircraft flying.

Second, in the case of the surface vehicles, the recent change in amphibious doctrine to the Over-The-Horizon (OTH) [Ref. 1] concept envisions launching amphibious assault vehicles from distances that were never imagined when the

the current class of assault amphibians was designed and procured. The slow speeds at which these amphibians swim ashore render them tactically obsolete in view of the increased exposure times that result from the OTH concept.

In view of this growing obsolescence of the ship-to-shore vehicles, replacements must be procured which are able to accomplish the ship-to-shore movement as envisioned by the current OTH concepts. In an effort to meet these OTH requirements, two programs are currently underway to determine the replacement vehicles.

The Medium Lift Replacement (MLR) program has been tasked with finding a replacement aircraft for the aging CH-46 and CH-53A/D aircraft. Among the alternatives that are being studied are the MV-22, CH-60, an improved model of the CH-46 and the S-92. The aircraft lift capability, fuel endurance and cruise speed are among the prime considerations in evaluating the alternatives available. The identification and procurement of the MLR aircraft is the highest priority item in the Marine Corps today [Ref. 2:p. 73].

The Advanced Assault Amphibian Vehicle (AAAV) program seeks to find a suitable replacement for the current LVTP-7 family of assault amphibians. The speed at which the vehicle transits to shore is the major difference among the models currently under consideration.

The USMC budget for the next several years will dedicate a growing share of funds to developing and procuring these replacement vehicles. In view of this immense investment it is imperative that the vehicles chosen perform the OTH mission in the most efficient and effective manner possible.

A realistic simulation of the ship-to-shore movement would allow budget planners to evaluate the performance of the different vehicles competing for a share of the constantly shrinking defense budget. Such a simulation could be used to perform other evaluations as well. For example, a proposed landing plan could be run in order to identify potential choke points. The same simulation could be used to identify desired characteristics of future vehicles, such as fuel loads, cruise speeds, and cargo capacities.

This thesis documents the construction and use of an object-oriented ship-to-shore movement simulation in an effort to identify the superior of aircraft for the MLR program.

B. THE USERS

This model is an outgrowth of six weeks of research at the Requirements, Plans, and Programs (RP&P) Branch, Headquarters, United States Marine Corps (HQMC). RP&P is responsible for developing the Program Objective Memorandum (POM) which is the USMC input to the DoD budget process.

In addition to identifying the preferred MLR aircraft, this model will assist RP&P in the evaluation of proposed changes in the equipment and doctrine currently employed in the ship-to-shore movement phase of an amphibious assault.

II. THE AMPHIBIOUS ASSAULT

The goal of the amphibious assault is the rapid build-up of combat power ashore. This model provides a means to quantitatively compare the build-up of combat power ashore achieved by differing mixes of ships, assault craft and aircraft. Assault craft and the aircraft that perform the ship-to-shore movement are identical objects in the eyes of this model.

This thesis will examine the aircraft currently under consideration by the MLR program. It should be noted that these same procedures could be applied to the vehicles under consideration by the AAAV program.

A. SHIP-TO-SHORE MOVEMENT

The ship-to-shore movement constitutes the fifth phase of an amphibious assault. The proper execution of the first four phases has little impact on the ship-to-shore phase. As a result, the conduct of this simulation disregards the first four phases.

There are several key factors that affect the rate at which combat power is built-up ashore during the ship-to-shore movement. Ignoring the numbers of amphibious ships, assault craft and aircraft available to conduct the ship-to-shore movement, the following capabilities must be modeled.

1. Number of Landing Spots Available

All of the amphibious ships in use are capable of conducting air operations. The number of landing spots available for use on each ship will dictate the number and type of aircraft that may be employed. Figure 1 shows the landing spot configuration for the LHA class of amphibious ship.

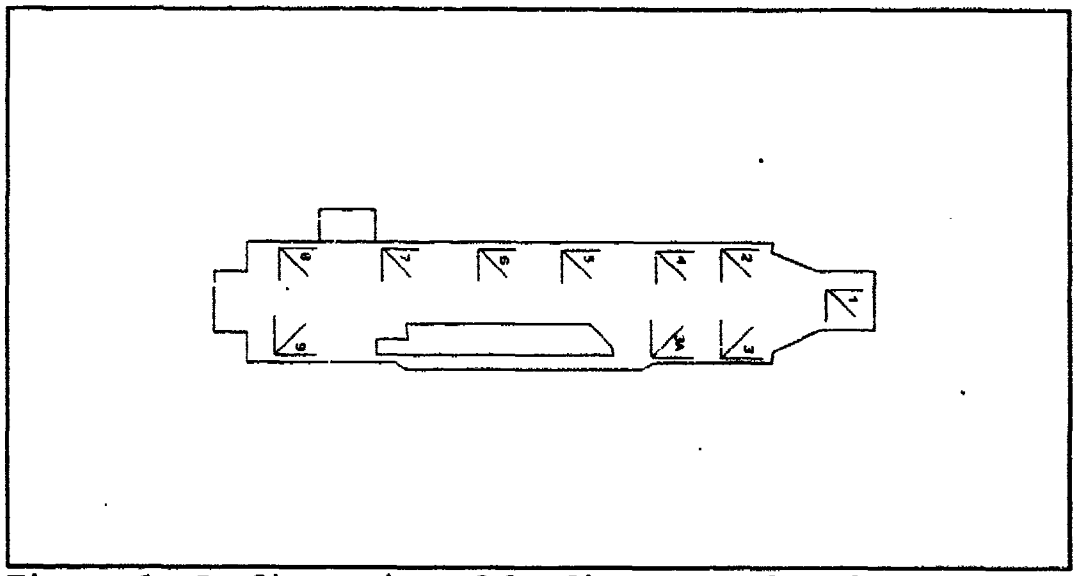
Figure 1: Configuration of landing spots for the LHA class of amphibious ship.

On the other hand, the types of aircraft employed, due to their size, could limit the number of landing spots available. It is apparent that the number of landing spots available aboard the amphibious ship has a significant impact on the rate at which combat power is built-up ashore.

The model must account for the proper use of these landing spots. Aboard an amphibious ship, different aircraft types are restricted to the use of specific landing spots. For example, on an LHA the CH-53 can not land on spot one. The model must allot aircraft a spot that can be utilized by that aircraft. Chapter III contains a more detailed description of the allocation and use of these landing spots.

The amount of time that a particular aircraft type spends in a holding pattern along with landing spot utilization rates will indicate the need for different aircraft mixes or ship designs.

2. Aircraft Capabilities and Limitations

The model will be used to compare the rate of build-up using different aircraft. In order to perform this comparison, the key differences in the aircraft must be modeled. These differences include, but are not limited to:

- Airspeed
- Cargo capacity
- Deck Spot restrictions
- Fuel usage and capacity (the "range" of the aircraft).

3. Ship Movement

It will be assumed that after the commencement of the amphibious assault, the amphibious ships will continue movement towards the beach until they reach their preset holding points. At these points the ships will maintain holding patterns for the duration of the operation. The initial starting point, the speed at which a ship presses for its holding point and the holding point location will be entered by the user. In this way the distance from ship to shore may be varied in order to test aircraft capabilities under different scenarios.

B. DESIGN DOCUMENT

Chapter III serves as a design document, describing in detail the amphibious assault and the actions of the model as it simulates the ship-to-shore movement. Chapter III is not meant as a detailed user's guide, but rather as an overview of model execution. Appendix A contains a complete listing of the *Cutter* model source code. Appendix B lists the minimum hardware requirements for using the model.

III. THE MODEL

The *Cutter* model is an object-oriented simulation [Ref. 3] developed to analyze the ship-to-shore movement phase of the amphibious assault. In the *Cutter* model key players within the ship-to-shore movement evolution are created as objects. These objects possess the attributes and actions required to model the ship-to-shore phase of the amphibious assault. As an example, a CH-46 would be modeled as an aircraft object, possessing attributes of airspeed and cargo capacity, among others. The objects and their attributes may then be altered in order to explore the effect of these changes on the ship-to-shore movement.

In this thesis, attributes of aircraft objects are altered to reflect those possessed by aircraft under consideration by the MLR program. In this way the most effective MLR aircraft may be identified. The primary objects of the *Cutter* model, along with their key attributes and actions are described in the following sections.

A. AMPHIBIOUS SHIPS

The ships which are used to transport the landing force to the Amphibious Operations Area (AOA) are referred to as amphibious shipping. Examples of amphibious ships are the Landing Ship, Tank (LST), Landing Ship, Dock (LSD) and the

Assault Landing Ship, Helicopter (LHA). Amphibious shipping constitutes the ship portion of the ship-to-shore movement. A group of amphibious ships steaming towards the AOA is referred to as an Amphibious Ready Group (ARG).

As an object within the *Cutter* model, the amphibious ship is known as an ***ARGObj*** (Amphibious Ready Group Object). The primary role of the amphibious ship in the ship-to-shore movement is to sail from an initial starting location to a selected holding area. During the ship-to-shore movement the amphibious ship transfers cargo to the embarked aircraft for movement to shore.

The number of landing spots on an amphibious ship has a dominant effect on its capability to move personnel and equipment from the ship to the shore. Each amphibious ship has deck space for one (an LST) to as many as ten (an LHA) landing spots. The actual number of landing spots available for use aboard a ship will vary depending on the number and type of aircraft embarked. If a large number of aircraft are embarked, landing spots are required for the storage of those aircraft not currently employed. As mentioned in the previous chapter, larger aircraft may be restricted to conducting operations on certain spots due to a lack of clearance from ship structures or other safety considerations.

B. THE AIRBOSS

The Airboss is responsible for allocating landing spots to those aircraft preparing for their initial launch and those requesting permission to land. Aircraft that request permission to land when no landing spots are available are sent to a holding queue known as "starboard delta". The Airboss allocates open landing spots using the following priorities:

- Aircraft awaiting launch
- Aircraft holding in starboard delta.

The aircraft holding in starboard delta are prioritized based on the amount of fuel remaining aboard the aircraft.

Each of the amphibious ships has its own Airboss and starboard delta queue. In addition to controlling flow in to and out of his starboard delta queue, the Airboss ensures that aircraft only operate from those landing spots that have no size or safety restrictions for that type of aircraft.

Within the ***Cutter*** model, the Airboss is created as a ***SpotManObj*** (Spot Manager Object).

C. SERIALS

A serial is a set of passengers and/or cargo scheduled for movement ashore aboard a single aircraft. Each of the serials is assigned a source (amphibious ship), a destination (Landing Zone), a priority, and a mode of transportation. The priority of the serial determines the order in which the serials are transported ashore. The lower the priority of the serial, the sooner it is scheduled to be sent ashore.

The mode of transportation, either as internal cargo or cargo to be carried externally, determines the airspeed of the aircraft which transports the serial ashore. Depending on the cargo to be moved externally, the airspeed of the aircraft could be reduced by as much as 60% of that which the aircraft would normally fly.

The weight of the serial will drive the decision as to which type of helicopter transports it to shore. In practice, passengers and lightweight cargo are usually broken into serials sized to fit onboard the smallest transport aircraft embarked. Larger transport aircraft combine two serials in order to exploit their payload advantage over the smaller transports. Some serials, due to their weight or size, require larger transports for movement ashore.

In the *Cutter* model, a serial to be moved ashore is created as a ***SerialObj***.

D. HELICOPTER DIRECTION CENTER

The Helicopter Direction Center (HDC) maintains a prioritized list of all serials to be moved ashore. This serial list is used to direct the flow of aircraft between the amphibious ships and the landing zones.

As an aircraft departs the landing zone, it checks in with HDC to determine the location of the next serial to be moved.

In the case of larger transport aircraft, HDC combines two serials whenever possible to fully utilize these larger aircraft.

There is only one HDC within the ARG. As an object in the *Cutter* model, the Helicopter Direction Center is known as an ***HDCObj***.

E. LANDING ZONES

The Landing Zone (LZ), modeled as an ***LZBeachObj***, represents the shore portion of the ship-to-shore movement. The LZ is the destination for the serials loaded aboard the transport aircraft. The only attributes that an LZ possess are location and a preset number of landing spots. The landing spots in the LZ are identical in nature to those aboard the amphibious ships.

Each of the landing zones contain a Forward Air Controller (FAC) who performs the same function for the LZ that the Airboss performs for the amphibious ship. The FAC, also modeled as a ***SpotManObj***, operates in the same manner as the Airboss with one exception. Since all aircraft start the simulation from the amphibious ships, the FAC does not have to contend with aircraft preparing for their initial launch. Therefore, the FAC allocates open landing spots solely to aircraft within his starboard delta queue.

F. AIRCRAFT

There are two types of aircraft utilized in the amphibious assault; transport aircraft and attack aircraft. The transport aircraft are used to move the serials from the amphibious ships to the landing zones. Attack aircraft provide covering fire for the transports as they perform their assigned tasks. Each of these aircraft and their roles in the amphibious assault will be described in separate sections.

1. Transport Aircraft

Transport aircraft are the vehicles which move the serials from the amphibious ships to the landing zones. Created as a *TransObj* within the *Cutter* model, the transport aircraft are the focus of this thesis. There are many actions that transport aircraft perform in the execution of their mission. These actions and some simplifying assumptions are described in the following sections.

a. *Initial Launch*

The transport aircraft are embarked aboard amphibious ships for movement to the AOA. At a pre-determined launch time, the aircraft obtains a landing spot from the Airboss in order to commence flight operations. Once the landing spot is obtained, the aircraft is positioned on the spot, loaded and launched.

The amount of time that an aircraft requires to load is determined by the serial that is assigned to the aircraft. Serials that consist entirely of passengers or that are carried externally usually require the least amount of time to load. Serials that consist entirely of internally carried cargo require the greatest amount of time to load. The *Cutter* model looks at the contents of the serial assigned to the aircraft to determine the appropriate loading time.

b. Transit to Shore

An aircraft must meet certain criteria prior to departing the amphibious ship. In the *Cutter* model, the aircraft ensures that there is enough fuel aboard for a round trip from the ship to the shore, plus a pre-determined reserve. If there is not enough fuel aboard the aircraft, the aircraft is refueled aboard the amphibious ship from which it is departing. In the *Cutter* model there are no refueling facilities ashore; aircraft are required to refuel aboard one of the amphibious ships. It is assumed that the refueling system aboard every amphibious ship is functional.

In addition to checking for sufficient fuel, the aircraft checks that the pilots have sufficient crew day for the round trip. While crew day is an administrative safety limitation placed on the aircrew, it could have a very negative effect on the ship-to-shore movement. If an aircrew does not have sufficient crew day to complete the round trip, the aircraft returns to its mother ship and commences the shut-down procedures.

The speed at which the aircraft transits to the shore is determined by several factors. The loaded cruise speed of the aircraft is the maximum speed at which the loaded aircraft would normally transit. If the aircraft is assigned a serial to be carried externally, the transit speed is reduced considerably.

c. Arrival at the LZ

Upon arrival at the destination LZ, the aircraft requests a landing spot from the FAC. If there are no landing spots available, the FAC will place the aircraft in his starboard delta queue. The aircraft will hold in this queue until cleared for landing by the FAC.

The time it takes to unload the aircraft is determined in the same manner as the loading time. Those serials composed entirely of passengers and those that are carried externally are the fastest to unload, with serials consisting of internal cargo being the slowest.

At the completion of the unloading evolution, the aircraft departs the LZ and transits to the amphibious ships.

d. Transit to Amphibious Shipping

As an aircraft departs the LZ, it will check in with HDC in order to determine the pick-up location of the next serial. The aircraft then transits to the pick-up location at its normal empty cruise speed.

As the aircraft approaches the pick-up ship, it calls that ships Airboss. The Airboss will either clear the aircraft to land if a landing spot is available, or direct the aircraft to enter the starboard delta queue. Upon landing on the amphibious ship, the load, transit to shore, unload, transit to amphibious shipping cycle is repeated until the aircraft is directed by HDC to shut-down.

e. *Shut-down*

Aircraft will repeat the cycle described above until directed by HDC to shut-down. In the *Cutter* model there are two reasons that HDC would direct an aircraft to shut-down. The first would be if the aircrew should exceed crew day limitations. Secondly, an aircraft would be directed to shut-down if there were no additional serials requiring movement to shore.

When directed to shut-down, the aircraft proceeds to the ship which transported it to the AOA. Upon arrival at the ship, the aircraft requests a landing spot from the Airboss and proceeds as directed. When all of the aircraft have returned to their mother ship and shut-down, the simulation is terminated.

f. *Fuel Usage*

One of the attributes that determines the effectiveness of an aircraft in the ship-to-shore arena is the range of that aircraft. The range of the aircraft is a combination of the amount of fuel an aircraft carries, and the rate at which the aircraft consumes that fuel (the fuel-flow). The model uses three fuel-flow rates; one when the aircraft is airborne loaded with cargo, one when the aircraft is airborne with no cargo, and one for ground operations. There are three periods during which an aircraft consumes fuel:

- In transit
- Holding in a starboard delta queue
- While loading and unloading cargo.

The ***Cutter*** model computes the amount of fuel consumed whenever an aircraft reaches an LZ or an amphibious ship. For example, an aircraft may load, transit to shore, hold in starboard delta awaiting an LZ landing spot and finally land. At this time, ***Cutter*** will compute the amount of fuel consumed based on the loading time, transit time and the amount of time spent holding. The fuel consumed is then subtracted from the amount of fuel currently aboard the aircraft.

2. Attack Aircraft

Attack aircraft are similar to transport aircraft, both possessing the same attributes and actions. In the ***Cutter*** model, an attack aircraft is modeled as an ***AttackObj*** which inherits [Ref. 3] the attributes and actions from the transport aircraft. There is one primary difference between an attack aircraft and a transport aircraft.

Unlike the transports, attack aircraft are not required to interact with the landing zones. An attack aircraft is assumed to fly for a given period of time and then return to the amphibious ship. The attack aircraft competes for ship landing spots, refuels, and shuts down in the same manner as the transport aircraft.

G. MODEL EXECUTION

There are several phases to the execution of the ship-to-shore movement simulation. A brief description of the methods that the *Cutter* model uses to complete these phases is presented in the following sections.

1. Input

The information required to create the desired scenario is input through the use of six data files. Each of the data files contains a description of each piece of required data, as well as an example of data used for this thesis. The ***FileForm.mod*** file consolidates this information for all of the input files. Appendix C contains a description and examples of the required input files.

2. Scenario Initialization

The *Cutter* model contains several procedures which create the desired scenario. At the completion of this phase all of the amphibious ships, their embarked aircraft, landing zones and serials have been created.

3. Replication

The simulation is replicated the requested number of times, resetting all of the starting values at the end of each replication. At the end of the final replication, all of the objects are disposed of, required statistics are computed, and the final output is created.

4. Output

The model creates several output files for each scenario. The user inputs the desired name, limited to five characters, of these output files using the OPplan.dat data file (described in Appendix C). Appendix D contains a description and examples of the output files.

IV. ANALYTICAL PROCEDURES

A. BACKGROUND

The *Cutter* model uses recursively generated random number streams to control the passage of time throughout its operation. As a result, the output from the model is a random process, with the parameters of the parent distribution unknown. In order to compare the results of two different aircraft mixes run under the same scenario, a means of comparing the unknown parent distributions must be found which identifies the superior distribution. The random variable which measures the build-up of combat power ashore will be examined in order to identify the superior mix of aircraft.

This build-up of combat power ashore may be thought of as a pure birth process, as there are no departures from the system [Ref. 4:p. 251]. In the ship-to-shore movement, the rate at which the mix of aircraft (the population) delivers the serials ashore (births) may be approximated. The population of serials ashore at any time t, as a function of the combat power possessed by each serial, may be written as

$$X(t) : t \in (0,T), \tag{1}$$

where X(t) is the amount of combat power ashore at time t (t ranging from time 0 to T, the time that the last serial arrived ashore). The amount of combat power ashore is measured as the percentage of total combat power to be moved ashore. For purposes of this thesis, these functions will be referred to as Combat Power Ashore functions.

In order to compare the distributions from two different random processes, it is typical to compare the measures of location and spread from the two parent distributions. Calculating interval and point estimators of a distribution's location and spread, using multiple runs of the simulation, requires several simplifying assumptions [Ref. 5:p. 278]:

- independence between replications
- normality of the output distribution
- constant variance.

It is felt that while these assumptions may serve to simplify the comparison of two distributions which occur naturally, they may not be appropriate when dealing with the output from a computer simulation.

Additionally, these methods are usually applied to "end" measures such as the completion time of a task or the amount of time spent in a queue. In the case of the build-up of combat power ashore, the rate of build-up is more important than the actual completion time. The methods commonly employed to analyze simulation output are unable to capture a measure of this rate.

B. THE COMBAT POWER ASHORE FUNCTION

Output from the *Cutter* model may be used to obtain estimates of the Combat Power Ashore (CPA) function for each aircraft mix. The CPA function for a specific aircraft defines the percentage of combat power ashore at every time *t*, using that aircraft. In this thesis it is desired to compare the CPA functions for each aircraft in order to identify the preferred replacement aircraft for the MLR program.

The CPA function presents two vital pieces of information concerning combat power ashore. First, the CPA function shows the amount of combat power ashore ($X(t)$) at every time *t*. Second, the amount of time that the combat power has been ashore is shown as the difference between the time that a serial arrives and the time *t* of interest. Clearly, the greater the time a serial is ashore the greater its worth to the Commander of the Landing Force (CLF).

Since the goal of the ship-to-shore movement is the rapid build-up of combat power ashore, the ideal CPA function is easily visualized. This ideal CPA function delivers all combat power ashore at time 0. In Figure 2a, it is clear that under the ideal CPA function, all combat power is available to the CLF at every time *t*. It is also clear that this CPA function will never be observed in practice.

Figure 2b presents a more probable CPA function. In this CPA function all serials are considered to possess equal combat power. The first serial arrives ashore at time *t1*, the second serial arrives at time *t2* and so forth for all serials. Each of the serials is represented by a rectangle, with the height (*h*) equal to the combat power of the serial and the length (*l*) equal to the amount of time the serial is ashore.

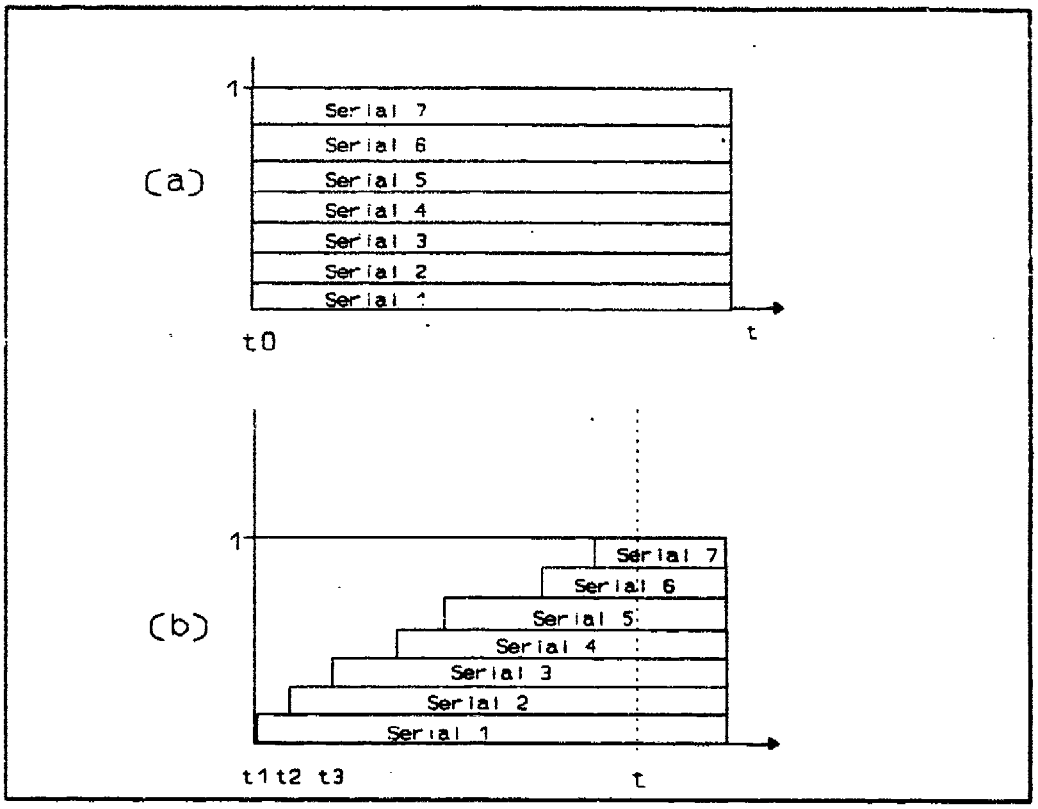

Figure 2 (a) The ideal CPA function. (b) A simplistic CPA function.

For any time *t*, the first serial is available for a length of time equal to *t* - *t1*, the second serial is available for a time of t - $t2$, and so on for each serial. The area of each rectangle (combat power * time ashore) represents the worth of that serial to the CLF at any time t. The sum of the area within all rectangles (and thus the area under the CPA curve) presents a measure of the combat power available to the CLF, at any time t.

In order to choose one CPA function over another, the area under each function must be compared. Furthermore, the method chosen must account for the area under the function for **every** time t.

C. COMPARING CPA FUNCTIONS

In this thesis, CPA functions produced by different aircraft are compared in a pairwise manner to identify the preferred aircraft for the MLR program. The goal of each pairwise comparison is to identify that CPA function which provides the CLF with the most rapid build-up of combat power ashore.

There are three cases that occur when comparing CPA functions. Each of these, along with examples, is discussed in the following sections.

1. Case 1

Suppose that the amount of combat power ashore at time t, using aircraft X, is defined by the CPA function X(t). Further suppose that the amount ashore using aircraft Y is defined by the CPA function Y(t). If the value of X(t) is greater than the value of Y(t) for every *t*, then aircraft X is obviously preferred over aircraft Y. This may be written as

$$X(t) \geq Y(t) \qquad (\forall\ t \in [0,T]). \qquad (2)$$

If the conditions of Equation (2) hold, then the area under X(t) will exceed that under Y(t) for all time *t*. Therefore, the CPA function X(t) provides a greater amount of combat power to the CLF than CPA function Y(t) at every time *t*.

Figure 3 contains a comparison of the CPA functions produced by two different aircraft. In this case X(t) is greater then Y(t) for every time *t*. Under the CPA function X(t) the CLF has more combat power available, at every time *t*, then would be available under the CPA function Y(t). Under the first case, the CLF clearly prefers that CPA function which has more combat power ashore at every time *t*.

2. Case 2

As shown in Figure 4, when two CPA functions are compared, it is likely that the conditions in Equation (2) will not hold for all time *t*. In this case, a different criteria is used in order to choose one CPA function over another.

For any time *t*, the longer a given serial is ashore, the greater utility (combat power * time ashore) that serial has to the CLF. As mentioned earlier, at any time *t*, the area under the CPA function presents a measure of the combat power

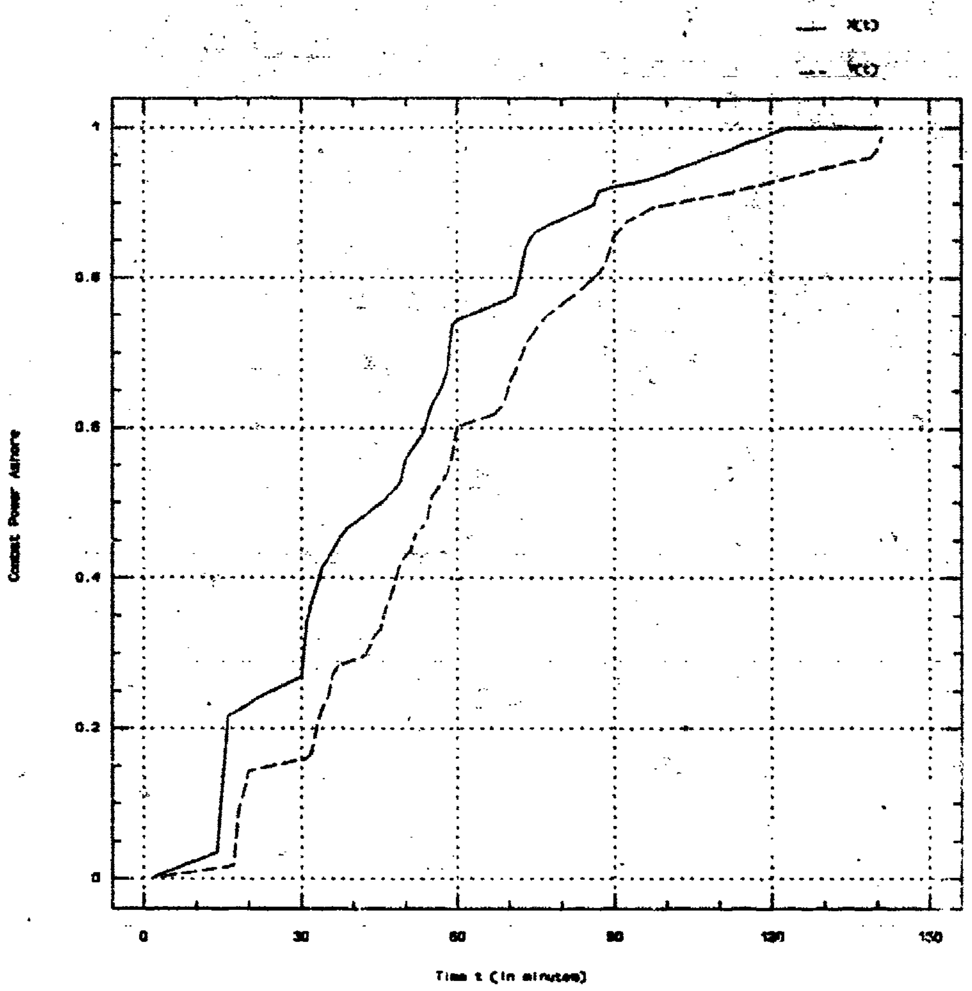

Figure 3 Case 1. The value of CPA function X(t) is greater than that of CPA function Y(t) for every t. The aircraft which produced the CPA function X(t) is preferred.

available to the CLF. If the area under X(t) is greater than the area under Y(t) for all t, then X(t) is preferred over Y(t). In this case

$$\int_0^t X(t)\,dt \geq \int_0^t Y(t)\,dt \qquad (\forall\ t \in [0,T]). \tag{3}$$

Equation (3) implies that CPA function X(t) offers a greater utility (in terms of combat power) to the CLF then the CPA function Y(t) for every *t*.

In the ship-to-shore environment, it is always preferable to have more combat power ashore now, than to have a promise of more later. Figure 4 shows a case where Equation (2) fails to hold, but where the conditions of Equation (3) are met. In this case CPA function X(t) is preferred over CPA function Y(t). Note, at some time *t*, the total amount of combat power ashore for Y(t) could exceed that of X(t) (as shown at time 70) with X(t) being the preferred CPA function.

3. Case 3

During the conduct of some pairwise comparisons, it may not be possible to pick one CPA function over another. Figure 5 shows an example where two CPA functions cross each other several times. In this case, neither Equation (2) nor Equation (3) holds at every time *t*. Therefore, it is not possible to pick one CPA function over the other. In this case, it will be assumed that the two CPA functions are equal, and that neither one is preferred over the other.

4. Dominance

When two CPA functions are compared, the **dominant** function is that which provides the most rapid build-up of combat power ashore. Throughout the rest of this thesis, during a pairwise comparison, the preferred CPA function will

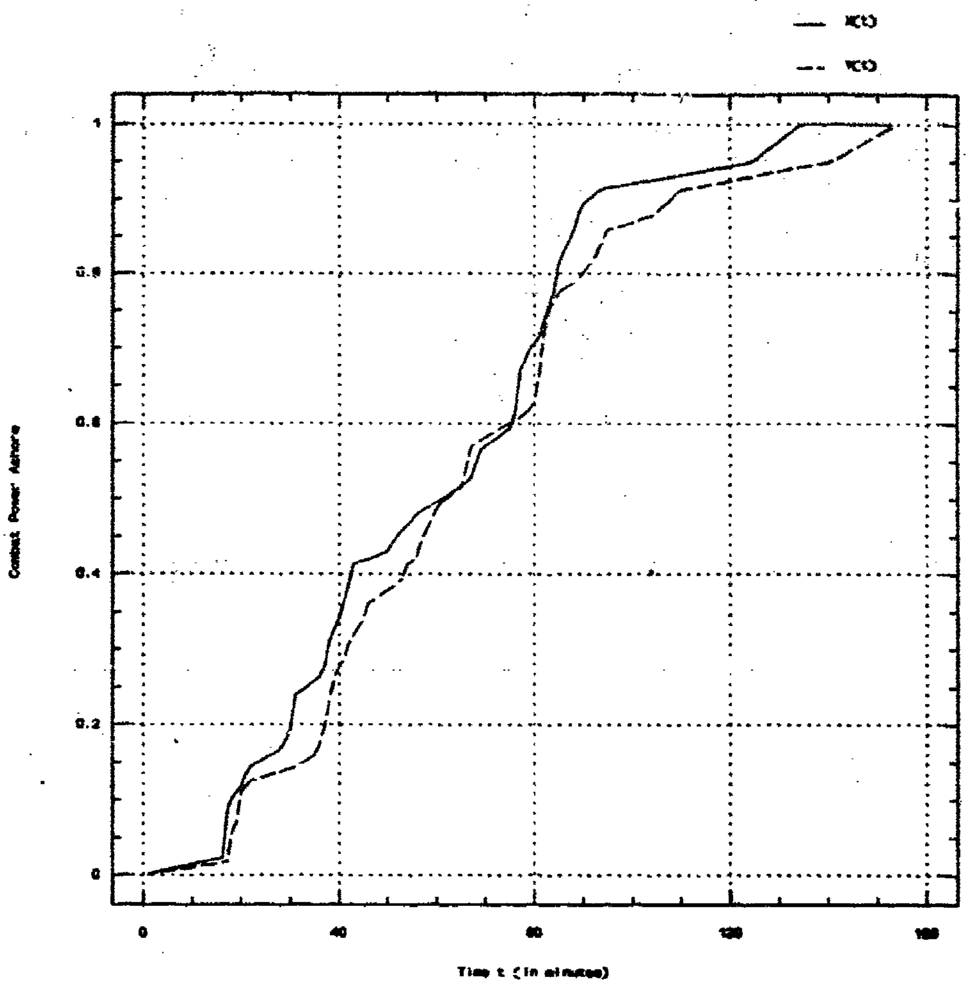

Figure 4 Case 2. The area under the CPA function X(t) is greater than the area under the CPA function Y(t) at every t. X(t) is the preferred CPA function X(t).

be referred to as the dominant of the two CPA functions. In order for CPA function $X(t)$ to dominate the function $Y(t)$, one of the following conditions must be met:

- $X(t) \geq Y(t) \quad (\forall\ t \in [0,T])$
- $\int_0^t X(t) \geq \int_0^t Y(t) \quad (\forall\ t \in [0,T])$.

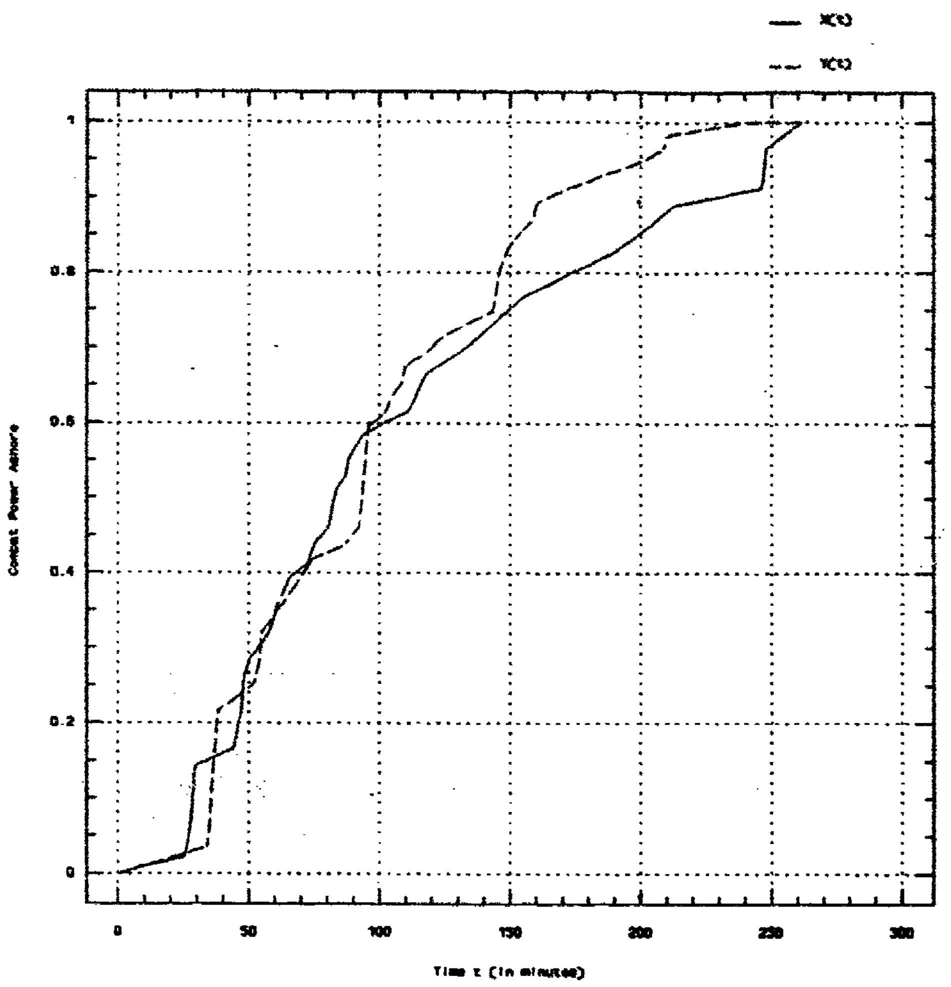

Figure 5 Neither function, X(t) nor Y(t), dominates the other.

If neither CPA function dominates the other, it will be assumed that either CPA function is acceptable to the CLF.

5. Assumptions

In order for the criteria outlined above to be applied in the comparison of two CPA functions, two assumptions must be accepted.

a. *Diminishing Marginal Returns*

The total utility of a serial to the CLF is a product of the combat power and the time ashore of the serial. As a result, two serials may contain the same amount of combat power, yet be of different value to the CLF due to their respective time ashore.

For example, consider two howitzers, A and B. Initially, both howitzers possess the same combat power. However, if A arrives ashore 30 minutes prior to B, then A has a greater utility to the CLF then B, due to the longer time spent ashore by howitzer A. Therefore, a serials worth to the CLF diminishes the longer it takes to deliver that serial ashore.

b. *Cutter Output*

Realizing that *Cutter* output is itself a random process, it is assumed that the output from each replication of the simulation is representative of the CPA function for the particular aircraft used. Under this assumption, the estimate of the CPA function resulting from the first replication of Aircraft X may be compared to the estimate of the CPA function resulting from the first replication of Aircraft Y, and so on for multiple replications of the simulation for each aircraft.

D. ANALYSIS PROGRAM

The ***Analysis*** program is used to compare CPA functions from each aircraft in the different ***Cutter*** scenarios. In order to estimate the CPA functions, the ***Cutter*** model completes one hundred replications for each aircraft. In order to compare two aircraft, the one hundred CPA function estimates for each aircraft are compared using the ***Analysis*** program. The following sections describe the required input and general workings of the ***Analysis*** program.

1. Input

The ***Analysis*** program takes as input the ***Cutter*** output file ***<filename>LZ.out*** (described in Appendix D) from each of the two aircraft to be compared. The ***Analysis.dat*** file is used to input the number of comparisons to perform, the names of the two data files to compare, and the desired name for the ***Analysis*** output file.

2. Data Preparation

The contents of each of the ***<filename>LZ.out*** files is manipulated to create two data arrays for each aircraft. These arrays contain the data required for the ***Analysis*** program to perform the required calculations. The following sections list these arrays, along with a short description of their use.

a. *Timex Array*

The Timex Array contains the integers from 1 to the time T that the last serial arrives ashore within the current simulation run. This array provides the *t* axis values for the CPA function.

b. Yvalue Array

The Yvalue Array contains the total combat power ashore at the corresponding time in the ***Timex*** array. This array provides the Y axis values for the CPA function.

3. Case 1 Comparison

The comparison procedure for the first case takes the Yvalue array for each data set, comparing the values for each time *t*. If the elements of the ***Yvalue*** array for Aircraft X are greater than or equal to the elements of the ***Yvalue*** array for Aircraft Y, at every time *t* (Eq. 2), then the CPA function for Aircraft X dominates the CPA function for Aircraft Y. In this example the use of Aircraft X is preferred over the use of Aircraft Y.

4. Case 2 Comparison

If the conditions in Equation (2) do not hold, a Case 2 comparison is performed in an effort to identify the dominant CPA function. The comparison procedure for the second case computes the area under each CPA function for every time *t*. If the area under X(t) is larger than the area under Y(t) at every time *t*, then X(t) dominates Y(t).

If neither comparison is able to choose a dominant CPA function, then Case 3 applies, and neither CPA function dominates the other.

E. CONTINGENCY TEST

The results of the ***Analysis*** program are used to perform a contingency test. The null hypothesis, that the CPA function for Aircraft X is the same as the CPA function for Aircraft Y (in a given scenario), is tested against the alternative hypothesis that the two CPA functions are different. This hypothesis test may be written as

$$H_0 : X(t) = Y(t) \quad vs \quad H_a : X(t) \neq Y(t) . \qquad (4)$$

The results of the CPA function comparisons for any two aircraft are arranged as shown in Table I.

Table I.-- CONTINGENCY TABLE FORMAT

Acft	Acft X	Either	Acft Y
Dominant	(cell 1)	(cell 2)	(cell 3)

In Table I, the value entered into cell 1 refers to the number of times that Aircraft X dominates Aircraft Y. The value entered into cell 3 represents the number of times that Aircraft Y dominates Aircraft X. The value in cell 2 represents the number of times that neither aircraft dominates the other.

The contingency test computes the following value for each of the cells in Table I:

$$\frac{(\textit{observed value} - \textit{expected value})^2}{\textit{expected value}} \quad . \qquad (5)$$

As mentioned earlier, the output from the simulation is a random variable. As such, an estimate of the expected values for each cell of Table I must be found. Suppose that *a* is the observed value for cell 1, that *b* is the observed value for cell 2 and that *c* is the observed value for cell 3. It can be shown that the Maximum Likelihood Estimator (MLE) for the expected values, given that H_0 (Eq. 4) is true, are as follows:

- $\hat{a} = (a+c)/2$ for cells 1 and 3
- $\hat{b} = b$ for cell 2.

The value of Equation (5) for each cell in Table I are calculated and summed. This final sum is then compared to the Chi-Square distribution with one degree of freedom (X^2_1) to test the null hypothesis H_0.

V. SIMULATION ANALYSIS

A. MODEL VERIFICATION

The *Cutter* model has been verified to work while running a number of simple test scenarios. The output from these test scenarios has been examined and is believed to be correct. The model has not been validated, as this would require comparing its results to the results obtained from actual ship-to-shore exercises. Since it has not been validated, these results should only be viewed relative to each other, and not as absolute numbers. Future users are encouraged to review the input required and conduct trials to confirm the proper input parameters.

B. TEST SCENARIOS

A brief description of each scenario and the different aircraft capabilities and limitations appear in the following sections. Appendix C contains the input data for each scenario and aircraft used.

1. Scenarios

Each of the three scenarios consisted of 3 ships (an LHA, LPD, and LST) and 2 landing zones. The landing zone locations were fixed for each scenario, while the ship-to-shore distance varied from scenario to scenario. In the first scenario the ships were 5 miles from shore, in the second scenario the ships were 25 miles from shore, while the third scenario had the ships 50 miles from shore. While the *Cutter* model has the ability to model ship movement this feature was not used in the test scenarios.

There are a total of 419 passengers and 97,000 pounds of cargo to be transported ashore under each scenario. The serial lists for each of the aircraft mixes were arranged in order to fully exploit the cargo capacity of the aircraft. For example, a serial for the MV-22 contains a maximum of twenty passengers while a serial for the CH-60 contains a maximum of ten. As a result, the aircraft mix containing MV-22 aircraft had a total of 39 serials to move ashore while the CH-60 mix required that 56 serials be transported ashore.

2. Aircraft

There are seven different aircraft used in the test scenarios. Six of these aircraft are under consideration by the MLR program. The seventh aircraft, the CH-53E, is included for reasons to be discussed in a later section. The input parameters for each of the aircraft were obtained from the Naval Air Systems Command. The sections below will provide a brief description of the aircraft involved, with emphasis on the more important capabilities and limitations of each.

a. *CH-46*

This is the current medium lift aircraft and is used as a baseline for aircraft comparisons.

b. CH-60

The CH-60 has the smallest payload, equivalent to ten passengers. This aircraft does possess an airspeed and range advantage over the CH-46 aircraft.

c. S-92

At this time, the data for the S-92 is proprietary and is therefore omitted from the body of this thesis. This aircraft does possess greater cargo capacity, range and airspeed then the CH-46.

d. CH-53E

This is a much larger aircraft then the CH-46, S-92 and CH-60. The CH-53E will carry more then twice as much as the CH-46 with an airspeed of 150 kts and a much greater range. However, due to its size, the CH-53E operates from six deck spots aboard the LHA, whereas the CH-46, S-92 and CH-60 aircraft have eight deck spots from which to operate.

e. MV-22

This is a tilt-rotor aircraft, able to operate in either the helicopter or fixed-wing mode. The fixed-wing cruise speed (approximately 250 kts) and the helicopter cruise speed (180 kts) give this aircraft a tremendous advantage over the other five candidates. The MV-22 payload is somewhat larger then the CH-46, but considerably less then the CH-53E. This is a large aircraft, and therefore has the same LHA deck spot limitation as the CH-53E.

f. CH-47D

The internal cargo capacity of this aircraft is identical to that of the CH-53E. While the airspeed of the CH-47 is comparable to that of the CH-53E, the range is significantly less. Due to its large size, the CH-47 is limited to six operating spots aboard the LHA.

g. EH-101

This aircraft has the same cargo capacity as the MV-22, but possesses average airspeed and range when compared to the other aircraft.

The aircraft listed were arranged into the following mixes and run under each scenario:

- 12 CH-46 and 4 CH-53E (Mix 1)
- 12 CH-60 and 4 CH-53E (Mix 2)
- 12 EH-101 and 4 CH-53E (Mix 3)
- 12 MV-22 and 4 CH-53E (Mix 4)
- 12 CH-47D (Mix 5)
- 12 S-92 and 4 CH-53E (Mix 6).

Note that five of the aircraft mixes include the CH-53E aircraft. The real world ship-to-shore movement includes several serials which require heavy lift assets to move ashore. The CH-47D is capable of lifting all loads that require external transportation to shore, therefore the CH-53E is not included in the fifth mix.

C. SCENARIO RESULTS

In the following sections the results from each scenario are discussed, stressing the ability of the model to evaluate the interactions of the various input parameters. Note that when, for example, reference is made to the "MV-22", this refers to the 12 MV-22/4 CH-53E mix of aircraft, and not solely to the MV-22 aircraft.

1. Scenario 1

As shown in Table II, the CH-47D clearly dominated all aircraft in the first scenario. The CH-47D is able to overcome the landing spot restrictions aboard the LHA through its superior cargo capacity.

The MV-22 and the EH-101 were equally effective, a result that is interesting. While the two aircraft possess the same cargo capacity, there are two major differences in the capabilities of the aircraft. The MV-22, with a speed of 180 kts, has a 40 knot advantage over the EH-101.[1] On the other hand, the EH-101 has the use of eight landing spots aboard the LHA while the MV-22 is restricted to six spots.

As expected, the S-92, possessing average range and cargo capacities, fell in the middle of the six mixes. This

[1] The MV-22 was limited to 180 kts for this scenario. The short ship-to-shore distance of the scenario would prevent the aircraft from completing transition to the fixed-wing mode. The 180 kts is the maximum airspeed for the aircraft in the helicopter mode.

aircraft is significantly more effective than the baseline CH-46 mix.

The CH-60 is totally inadequate. The restricted cargo capacity of the aircraft allows it to be dominated by all aircraft mixes.

Table II.-- AIRCRAFT COMPARISONS FOR SCENARIO 1

Scenario1	1 vs 2		1 vs 3		1 vs 4		1 vs 5		1 vs 6	
Dominates	54	0	0	44	0	30	0	78	0	40
Does Not	46		56		70		22		60	

Scenario1	2 vs 3		2 vs 4		2 vs 5		2 vs 6		3 vs 4	
Dominates	0	73	0	50	0	92	0	74	12	8
Does Not	27		50		8		26		80	

Scenario1	3 vs 5		3 vs 6		4 vs 5		4 vs 6		5 vs 6	
Dominates	0	55	25	1	0	62	14	1	60	0
Does Not	45		74		38		85		40	

Note: Mix (1) ... CH-46
Mix (2) ... CH-60
Mix (3) ... EH-101
Mix (4) ... MV-22
Mix (5) ... CH-47
Mix (6) ... S-92

2. Scenario 2

As shown in Table III, the CH-47 dominance over the MV-22, while still significant (p-value = .0082), is dramatically reduced in Scenario 2. The MV-22 top speed of 250 knots, combined with the ship-to-shore distance of 25 miles is able to offset much of the cargo capacity advantage of the CH-47.

At 25 miles, the EH-101 is no longer as effective as the MV-22. The increased airspeed of the MV-22 is fully able to offset the two additional operating spots from which the EH-101 is able to operate.

Once again, the S-92 is the third most effective aircraft. The CH-60 is still dominated by the CH-46, in spite of the range and airspeed advantages of the CH-60.

Table III.-- AIRCRAFT COMPARISONS FOR SCENARIO 2

Scenario2	1 vs 2		1 vs 3		1 vs 4		1 vs 5		1 vs 6	
Dominates	61	0	0	74	0	56	0	87	0	58
Does Not	39		26		44		13		42	

Scenario2	2 vs 3		2 vs 4		2 vs 5		2 vs 6		3 vs 4	
Dominates	0	94	0	85	0	97	0	87	0	34
Does Not	6		15		3		13		66	

Scenario2	3 vs 5		3 vs 6		4 vs 5		4 vs 6		5 vs 6	
Dominates	0	59	29	0	0	7	33	0	68	0
Does Not	41		71		93		67		32	

Note: Mix (1) ... CH-46
Mix (2) ... CH-60
Mix (3) ... EH-101
Mix (4) ... MV-22
Mix (5) ... CH-47
Mix (6) ... S-92

3. Scenario 3

In the third and final scenario, there is no difference between the MV-22 and the CH-47 aircraft. The fifty mile ship-to-shore distance allows the MV-22 airspeed advantage to fully compensate for the cargo capacity of the CH-47.

The EH-101 continues to dominate the other three aircraft. It is interesting that the dominance of the EH-101 over the CH-46 and the CH-60 decreased between Scenarios 1 and 2, but then increased between Scenarios 2 and 3. This dip can be explained by examining the range of the aircraft involved.

In Scenario 1 the CH-60 and CH-46 require refueling in order to complete the evolution. The EH-101, due to its range advantage, is able to complete the first scenario without conducting refueling operations. The ship-to-shore distance in Scenario 2 is such that the EH-101 requires a greater increase in the number of refueling operations from Scenaric 1 then the increase in refueling operations for the CH-46 and CH-60 aircraft. Therefore, the build-up of combat power with EH-101 decreases at a greater amount then the build-up using CH-46 or CH-60 aircraft.

In Scenario 3 the opposite occurred. Due to the greater range of the EH-101, the CH-60 and CH-46 required more additional refueling operations from Scenario 2 to complete the third scenario then the EH-101 required. These additional refueling operations slow down the rate at which the CH-60 and CH-46 build-up combat power ashore.

The S-92 showed the same dominance pattern, to a lesser degree, between scenarios as did the EH-101. The S-92 is still the dominate aircraft when compared with the CH-46 and CH-60 aircraft.

The CH-60 is almost completely dominated by every aircraft in the study. This would stress the importance of the aircraft cargo capacity. It takes a major airspeed and/or range advantage to compensate for a very limited cargo capacity. Table IV contains the complete results for the third scenario.

Table IV.-- AIRCRAFT COMPARISONS FOR SCENARIO 3

Scenario3	1 vs 2		1 vs 3		1 vs 4		1 vs 5		1 vs 6	
Dominates	70	0	0	71	0	63	0	83	0	55
Does Not	30		29		37		17		45	

Scenario3	2 vs 3		2 vs 4		2 vs 5		2 vs 6		3 vs 4	
Dominates	0	96	0	96	0	100	0	97	0	40
Does Not	4		4		0		3		60	

Scenario3	3 vs 5		3 vs 6		4 vs 5		4 vs 6		5 vs 6	
Dominates	0	45	42	0	0	0	53	0	58	0
Does Not	55		58		100		47		42	

Note:

Mix (1)	... CH-46	Mix (4)	... MV-22
Mix (2)	... CH-60	Mix (5)	... CH-47
Mix (3)	... EH-101	Mix (6)	... S-92

4. Summary of Test Results

The results from the three scenarios confirm the obvious. The aircraft with the largest payload will most likely be the most effective in terms of the rapid build-up of combat power ashore.

The analysis above also demonstrates the ability of the *Cutter* model to quantify the capability trade-offs among different aircraft. While one aircraft may possess a speed advantage and another a larger cargo capacity, both aircraft may be equally effective. In this case, the MV-22, with the 110 knot airspeed advantage proved as effective as the CH-47, with a 10 passenger capacity advantage, given a certain ship-to-shore distance.

Another trade-off comparison exists between the CH-46 and the CH-60. The CH-60 has a significantly longer range and a slightly greater airspeed then the CH-46. The CH-46, on the other hand, is able to carry five more passengers than the CH-60. It was shown that this trade-off between the two aircraft allowed the CH-46 to dominate the CH-60 under all three scenarios.

The ability to quantify these capability trade-offs proves that the *Cutter* model is a valuable tool when used to analyze the ship-to-shore movement.

APPENDIX A CUTTER MODEL SOURCE CODE

```
MAIN MODULE Cutter;
{-------------------------------------------------------------
   MODULE NAME:    Cutter          DATE WRITTEN:    18 Mar 92
   AUTHOR:         S. E. Shaw      LAST MODIFIED:
                   Capt  USMC
   DESCRIPTION  :  Ship-To-Shore  movement  analysis  model.
Simulates the build up of combat power ashore. The user is
able to change the scenario, to include the ship/LZ locations,
as well as the number and type of aircraft employed.
-------------------------------------------------------------}

FROM Debug IMPORT TraceStream;
FROM DebugRun IMPORT SetUpD;
FROM SimMod IMPORT StartSimulation, ResetSimTime, SimTime;
FROM CATFMod IMPORT CATFObj;
FROM Input IMPORT ReadEmAll;
FROM global IMPORT NewRandoms, repetition, showerrors;
FROM Statistics IMPORT StatisticsObj, lastdeliverytime;
FROM OutputDriver IMPORT OpenFiles, CloseFiles,
     EndTimerecorder;

VAR
    CATF            : CATFObj;
    Statistician    : StatisticsObj;
    totalruns       : INTEGER;

BEGIN

    OUTPUT("Enter Number of Runs to Complete");
    INPUT (totalruns);
    SetUpD (TRUE);
    showerrors := FALSE;

    ReadEmAll;
    NewRandoms;
    OpenFiles;
    NEW (Statistician);
    ASK Statistician TO StartStats;

    repetition := 1;
    WHILE ( repetition <= totalruns)
        ASK TraceStream TO WriteString ("Starting Cutter");
        ASK TraceStream TO WriteLn;
        NEW (CATF);
        StartSimulation;
        ASK Statistician TO CollectRepStats (CATF);
        ASK TraceStream TO WriteString ("Destroying CATF");
        ASK TraceStream TO WriteLn;
        ASK CATF TO DestroyCATF;

        OUTPUT ("Repetition := ",repetition," completed");
        ASK EndTimerecorder TO
           WriteString(REALTOSTR(lastdeliverytime));
        ASK EndTimerecorder TO WriteLn;
        ResetSimTime(0.0);
        ASK Statistician TO ResetStats;
    INC (repetition);
    END WHILE;
    ASK Statistician TO StopStats;
    CloseFiles;
    OUTPUT ("Ending Cutter");

END { MAIN } MODULE { Cutter }.

DEFINITION MODULE ARGMod;

{-----------------------------------------------------------
   MODULE NAME:   ARGMod          DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC

        DESCRIPTION : Defines the Amphibious Ready Group
  (ship) objects.
----------------------------------------------------------}

FROM ResMod IMPORT ResourceObj;
FROM GrpMod IMPORT QueueObj;
FROM global IMPORT LocationXY;
FROM SpotMan IMPORT SpotManObj;
FROM RGlobals IMPORT SHierRecType;

EXPORTTYPE
        ARGObj = OBJECT; FORWARD;

TYPE

ARGObj = OBJECT (QueueObj);
  name          : STRING;          steamspeed     : REAL;
  airboss       : SpotManObj;      holdingspeed   : REAL;
  location      : LocationXY;      course         : REAL;
  squadron      : QueueObj;        steaming       : BOOLEAN;
  type          : STRING;
  holdlocation  : LocationXY;      pumprate       : REAL;

        ASK METHOD ReadData (IN record : SHierRecType);
        ASK METHOD ObjInit;
        ASK METHOD CurrentPos (OUT xcoord : REAL;
                               OUT ycoord : REAL);
        ASK METHOD DestroyARG;
        TELL METHOD SetHoldingTime;
 END OBJECT { ARGObj };

END { DEFINITION } MODULE { ARGMod }.


IMPLEMENTATION MODULE ARGMod;

{------------------------------------------------------------
   MODULE NAME:   ARGMod          DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC

        DESCRIPTION : Defines the Amphibious Ready Group
(ship) objects.
------------------------------------------------------------}

FROM global IMPORT LocationXY, Distance, DeBug;
FROM RGlobals IMPORT SHierRecType;
FROM MathMod IMPORT SQRT, pi, ACOS, COS, SIN;
FROM SimMod IMPORT SimTime;
FROM Transport IMPORT TransObj;
FROM Debug IMPORT TraceStream;

OBJECT ARGObj;

{------------------------------------------------------------}
    ASK METHOD ReadData (IN record : SHierRecType);

    VAR
        placeholder : INTEGER;

        BEGIN
              name := record.TopString;
              type := record.OwnedString[1];
              location.x := STRTOREAL(record.OwnedString[2]);
              location.y := STRTOREAL(record.OwnedString[3]);
              steamspeed := STRTOREAL(record.OwnedString[4]);
              holdingspeed :=
                          STRTOREAL(record.OwnedString[5]);
              holdlocation.x :=
                          STRTOREAL(record.OwnedString[6]);
              holdlocation.y :=
                          STRTOREAL(record.OwnedString[7]);
              pumprate := STRTOREAL(record.OwnedString[8]);
```

```
            ASK airboss TO PaintSpots (type, name,
                                OBJTYPENAME(SELF), placeholder);
            ASK airboss TO SetName(name);

      END METHOD { ReadData };

{------------------------------------------------------------}
{------------------------------------------------------------}

      ASK METHOD ObjInit;
      BEGIN
      NEW (airboss);
              NEW (squadron);
              TELL SELF TO SetHoldingTime;

      END METHOD { ObjInit };

{------------------------------------------------------------}
{------------------------------------------------------------}

      TELL METHOD SetHoldingTime;
      VAR
              dist      : REAL;
              steamtime : REAL;
      BEGIN
              dist :=  Distance (location, holdlocation);
              steamtime := (dist / steamspeed) * 60.0;

      IF (dist >= 0.0)
         steaming := TRUE;

                 WAIT DURATION steamtime;
                 END WAIT;
      END IF;

              steaming := FALSE;

      END METHOD { SetHoldingTime };

{------------------------------------------------------------}
{------------------------------------------------------------}

   ASK METHOD CurrentPos (OUT xcoord : REAL;
                 OUT ycoord : REAL);
   VAR
      x0coord          : REAL;
      y0coord          : REAL;
      x1coord          : REAL;
      y1coord          : REAL;
      Angle            : REAL;
      hyp              : REAL;
```

```
        xvel, yvel          : REAL;

    BEGIN
        x0coord := location.x;
        y0coord := location.y;
        x1coord := holdlocation.x;
        y1coord := holdlocation.y;

        IF steaming
           hyp := SQRT((x1coord - x0coord)*(x1coord - x0coord)
                + (y1coord - y0coord)*(y1coord - y0coord));
           Angle := ACOS ((y1coord - y0coord)/hyp);

           IF ((x1coord >= x0coord) AND (y1coord <= y0coord))
               Angle := Angle;
               xvel := SIN(Angle)*steamspeed;
               yvel := COS(Angle)*steamspeed;

           ELSIF  ((x1coord >= x0coord) AND (y1coord >=
                   y0coord))
               Angle := Angle;
               xvel := SIN (Angle) * steamspeed;
               yvel := COS (Angle) * steamspeed;

           ELSIF ((x1coord <= x0coord) AND (y1coord >=
                   y0coord))
               Angle := (2.0*pi - Angle);
               xvel := SIN (Angle) * steamspeed;
               yvel := COS (Angle) * steamspeed;

           ELSE {((x1coord <= x0ccord) AND (y1coord <=
                   y0coord))}
               Angle := (2.0*pi - Angle);
               xvel := SIN (Angle) * steamspeed;
               yvel := COS (Angle) * steamspeed;

           END IF;

           xcoord := (SimTime()/60.0)*xvel + x0coord;
           ycoord := (SimTime()/60.0)*yvel + y0cocrd;

        ELSE

           xcoord := holdlocation.x;
           ycoord := holdlocation.y;
        END IF;

    END METHOD { CurrentPos };

{-----------------------------------------------------------}
{-----------------------------------------------------------}
```

```
    ASK METHOD DestroyARG;
    VAR
        acft : TransObj;
    BEGIN
        ASK airboss TO DestroySpotMan;
        IF (ASK squadron numberIn > 0)
        acft := ASK squadron First();
           WHILE (acft <> NILOBJ);
                ASK squadron TO RemoveThis(acft);
                ASK acft TO DestroyVehicle;
                acft := ASK squadron First();
           END WHILE;
        END IF;
        DISPOSE (squadron);

    END METHOD { DestroyArg };

{-------------------------------------------------------------}
{-------------------------------------------------------------}

END OBJECT { ARGMod };

END { IMPLEMENTATION } MODULE { ARGMod }.

DEFINITION MODULE Attack;

{--------------------------------------------------------------
     MODULE NAME:   AttackAcft      DATE WRITTEN:   7 Apr 92
     AUTHOR:        S. E. Shaw       LAST MODIFIED:
                    Capt  USMC       MODIFIED BY:

     DESCRIPTION : Contains the Attack Aircraft Object.
-----------------------------------------------------------}

FROM RandMod IMPORT RandomObj;
FROM ARGMod IMPORT ARGObj;
FROM LZBeach IMPORT LZBeachObj;
FROM SerialMod IMPORT SerialObj;
FROM HDCMod IMPORT HDCObj;
FROM global IMPORT LocationXY;
FROM RGlobals IMPORT SHierRecType;
FROM Transport IMPORT TransObj;

EXPORTTYPE
        AttackObj = OBJECT; FORWARD;

TYPE

        AttackObj = OBJECT (TransObj);

        returntime : RandomObj;
        OVERRIDE
        ASK METHOD ObjInit;
        TELL METHOD Operate;
        TELL METHOD Load;
        TELL METHOD GetClearance ;
        TELL METHOD TransitToBeach;
        TELL METHOD FlyToShip;
        TELL METHOD Spot;


        END OBJECT { AttackObj};

END { DEFINITION } MODULE { AttackAcft }.


IMPLEMENTATION MODULE Attack;

{-------------------------------------------------------------
    MODULE NAME:   AttackAcft      DATE WRITTEN:   7 Apr 92
    AUTHOR:        S. E. Shaw       LAST MODIFIED:
                   Capt  USMC       MODIFIED BY:

    DESCRIPTION : Contains the Attack Aircraft Object.
-----------------------------------------------------------}

FROM SimMod IMPORT SimTime;
FROM RandMod IMPORT RandomObj;
FROM ARGMod IMPORT ARGObj;
FROM HDCMod IMPORT HDCObj,BriefingRec;
FROM SerialMod IMPORT SerialObj;
FROM global IMPORT LocationXY, Distance, ReturnTime,
     moreserials;
FROM RGlobals IMPORT SHierRecType;
FROM SpotProcedures IMPORT GetShipSpot, GetLZSpot,
     GiveBackShipSpot, GiveBackLZSpot, InitialLaunch;
FROM LoadProcedures IMPORT LoadCargo, UnLoadCargo, ReArmAcft;
FROM Transport IMPORT TransObj;
FROM Debug IMPORT TraceStream;

OBJECT AttackObj;

{------------------------------------------------------------}
{------------------------------------------------------------}

        ASK METHOD ObjInit;
        BEGIN
        NEW (holdingtimestats);
        ADDMONITOR (holdingtime, holdingtimestats);

        NEW (holdingshipstats);
        ADDMONITOR (holdingship, holdingshipstats);
        NEW (holdingbeachstats);
        ADDMONITOR (holdingbeach, holdingbeachstats);

        END METHOD { ObjInit };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

        TELL METHOD Spot;
        VAR
                ship            : STRING;
                startpoint      : LocationXY;
                endpoint        : LocationXY;
                gonogo          : REAL;
                newserial       : SerialObj;
                available       : BOOLEAN;
                loadtime        : REAL;
        BEGIN
                destination := mymother;

                WAIT DURATION 5.0
                END WAIT; { spread time wait }

                airbornetime := SimTime();

                TELL SELF TO TransitToBeach;

        END METHOD { Spot };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

        TELL METHOD Load;
        VAR
                rearmtime : REAL;

        BEGIN
                holding := SimTime() - holding;
                ReArmAcft (SELF, rearmtime);


                WAIT DURATION rearmtime
                END WAIT; { load serial wait }

                TELL SELF TO TransitToBeach;

        END METHOD { Load };

{-----------------------------------------------------------}
{-----------------------------------------------------------}
```

```
        TELL METHOD Operate;
        VAR
                available    : BOOLEAN;

        BEGIN
                InitialLaunch (SELF, available);
                IF (available)
                        TELL SELF TO Spot;
                END IF;

         END METHOD { Operate };

    {-----------------------------------------------------------}
    {-----------------------------------------------------------}

        TELL METHOD TransitToBeach;
        VAR
                flighttime    : REAL;
                available     : BOOLEAN;
        BEGIN

                WAIT FOR SELF TO GetClearance;
                END WAIT;

                IF (cleared)

                        GiveBackShipSpot (SELF);

                        flighttime := ASK ReturnTime
                                UniformReal (60.0, 120.0);

                        WAIT DURATION flighttime;
                        END WAIT; { sortie time to shore }

                        TELL SELF TO FlyToShip;

                ELSE
                        TELL SELF TO ShutDown;

                END IF;

        END METHOD { TransitToBeach };

    {-----------------------------------------------------------}
    {-----------------------------------------------------------}

        TELL METHOD FlyToShip;
        VAR
                available     : BOOLEAN;

        BEGIN
```

```
                holding := SimTime();

                GetShipSpot (SELF, available);
                IF (available)
                        TELL SELF TO Load;
                END IF;

        END METHOD { FlyToShip };

    {---------------------------------------------------------}
    {---------------------------------------------------------}

        TELL METHOD GetClearance;

        BEGIN
                cleared := TRUE;
                INC (totalsorties);
                IF ((totalsorties > 3)
                    OR ((SimTime() - airbornetime) >crewday))
                   cleared := FALSE;
                END IF;

        END METHOD { GetClearance};

    {---------------------------------------------------------}
    {---------------------------------------------------------}


    END OBJECT { AttackAcftObj };

    END { IMPLEMENTATION } MODULE { AttackAcft }.


    DEFINITION MODULE CATFMod;

    {------------------------------------------------------------
       MODULE NAME:    CATFMod             DATE WRITTEN:    18 Mar 92
       AUTHOR:         S. E. Shaw          LAST MODIFIED:
                       Capt  USMC

       DESCRIPTION : Defines the Commander, Amphibious Task Force
    object. This object keeps track of all ships, LZs, and HDCobjs
    that are created. Primarily, these objects are tracked for
    disposal at the end of each replication.
     ---------------------------------------------------------}

    FROM GrpMod IMPORT QueueObj;
    FROM ARGMod IMPORT ARGObj;
    FROM LZBeach IMPORT LZBeachObj;
    FROM SerialMod IMPORT SerialObj;
    FROM HDCMod IMPORT HDCObj;
```

```
EXPORTTYPE
        CATFObj = OBJECT; FORWARD;

TYPE

        CATFObj = OBJECT;
                shiplist      : QueueObj;
                lzbeachlist   : QueueObj;
                hdclist       : QueueObj;

                ASK METHOD ObjInit;
                ASK METHOD AddShip (IN newship : ARGObj;
                                    IN HDC : HDCObj);
                ASK METHOD AddLZBeach (IN newlz : LZBeachObj);
                ASK METHOD DestroyCATF;


        END OBJECT { CATFObj };

END { DEFINITION } MODULE { CATFMod }.


IMPLEMENTATION MODULE CATFMod;

{-------------------------------------------------------------
   MODULE NAME:   CATFMod         DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC

   DESCRIPTION : Defines the Commander, Amphibious Task Force
object. This object keeps track of all ships, LZs, and HDCobjs
that are created. Primarily, these objects are tracked for
disposal at the end of each replication.

 -----------------------------------------------------------}

FROM ARGMod IMPORT ARGObj;
FROM LZBeach IMPORT LZBeachObj;
FROM SerialMod IMPORT SerialObj;
FROM Transport IMPORT TransObj;
FROM HDCMod IMPORT HDCObj;
FROM global IMPORT DeBug;
FROM Procedures IMPORT FindSource, FindDestination;
FROM CreateARG IMPORT Scenario;

OBJECT CATFObj;

{-----------------------------------------------------------}
{-----------------------------------------------------------}

        ASK METHOD ObjInit;
```

```
      VAR
              HDC : HDCObj;

      BEGIN
              NEW (shiplist);
              NEW (lzbeachlist);
              NEW (hdclist);
              NEW(HDC);

              ASK hdclist TO Add (HDC);
              Scenario(SELF, HDC);

              FindSource (HDC, SELF);
              FindDestination (HDC, SELF);

      END METHOD { ObjInit };

{----------------------------------------------------------}
{----------------------------------------------------------}

      ASK METHOD AddShip (IN newship : ARGObj;
                          IN HDC : HDCObj);
      VAR
              acft : TransObj;
      BEGIN
              ASK shiplist TO Add (newship);
              acft := ASK newship.squadron First();
              WHILE acft <> NILOBJ
                      ASK acft TO NewHDC (HDC);
                      acft := ASK newship.squadron
                              Next(acft);
              END WHILE;

      END METHOD { AddShip };

{----------------------------------------------------------}
{----------------------------------------------------------}

      ASK METHOD AddLZBeach (IN newlz : LZBeachObj);
      BEGIN

              ASK lzbeachlist TO Add (newlz);
      END METHOD { AddLZ };

{----------------------------------------------------------}
{----------------------------------------------------------}

   ASK METHOD DestroyCATF;
   VAR
      ship : ARGObj;
```

```
        lz    : LZBeachObj;
        hdc   : HDCObj;
    BEGIN

        IF ( ASK shiplist numberIn > 0 )
           ship := ASK shiplist First();
           WHILE ( ship <> NILOBJ)
                ASK shiplist TO RemoveThis(ship);
                ASK ship TO DestroyARG;
                ship := ASK shiplist First();
           END WHILE;
        END IF;

        DISPOSE (shiplist);

        IF (ASK lzbeachlist numberIn > 0)
           lz := ASK lzbeachlist First();
           WHILE ( lz <> NILOBJ)
                ASK lzbeachlist TO RemoveThis (lz);
                ASK lz TO DestroyLZ;
                lz := ASK lzbeachlist First();
           END WHILE;
        END IF;

        DISPOSE (lzbeachlist);

        IF (ASK hdclist numberIn > 0)
           hdc := ASK hdclist First();
           WHILE ( hdc <> NILOBJ)
                ASK hdclist TO RemoveThis(hdc);
                ASK hdc TO DestroyHDC;
                hdc := ASK hdclist First();
           END WHILE;
        END IF;

        DISPOSE (hdclist);

    END METHOD { DestroyCATF };

{------------------------------------------------------------}
{------------------------------------------------------------}

END OBJECT { CATFObj };
```

END { IMPLEMENTATION } MODULE { CATFMod }.

DEFINITION MODULE CreateARG;

```
{------------------------------------------------------------
   MODULE NAME:   CreateARG         DATE WRITTEN:   18 Mar 92
   AUTHOR:         S. E. Shaw       LAST MODIFIED:
                   Capt  USMC

   DESCRIPTION : The procedures here are used to create all of
the objects in the simulation and to initialize the current
scenario.

 ------------------------------------------------------------}

FROM CATFMod IMPORT CATFObj;
FROM HDCMod IMPORT HDCObj;
FROM RGlobals IMPORT SHierRecType;
FROM ARGMod IMPORT ARGObj;

TYPE

PROCEDURE Scenario(INOUT CATF : CATFObj;
                   INOUT HDC : HDCObj);

PROCEDURE MakeShips(INOUT CATF : CATFObj;
                    INOUT HDC : HDCObj);

PROCEDURE PlotLZ (INOUT CATF : CATFObj);

PROCEDURE BuildSerials (INOUT CATF : CATFObj;
                        INOUT HDC : HDCObj);

PROCEDURE MakeAcft ( INOUT newrec : SHierRecType;
                     INOUT ship : ARGObj;
                     INOUT HDC : HDCObj);


END { DEFINITION } MODULE { CreateArg }.


IMPLEMENTATION MODULE CreateARG;

{------------------------------------------------------------
   MODULE NAME:    CreateARG        DATE WRITTEN:   18 Mar 92
   AUTHOR:         S. E. Shaw       LAST MODIFIED:
                   Capt  USMC

   DESCRIPTION : The procedures here are used to create all of
the objects in the simulation and to initialize the current
scenario.

------------------------------------------------------------}

FROM RGlobals IMPORT SHierRecType,
                     ShipSHArray,
                     AcftSHArray,
                    LZSHArray,
                    SerialSHArray;
FROM CATFMod IMPORT CATFObj;
FROM Debug IMPORT TraceStream;
FROM ARGMod IMPORT ARGObj;
FROM HDCMod IMPORT HDCObj;
FROM LZBeach IMPORT LZBeachObj;
FROM SerialMod IMPORT SerialObj;
FROM Transport IMPORT TransObj;
FROM FindSHRec IMPORT FindSHRec;
FROM global IMPORT moreserials, totalserials, repetition,
     paxtolift, cargotolift;
FROM Attack IMPORT AttackObj;
FROM OutputDriver IMPORT Scenariorecorder;
FROM Statistics IMPORT vehiclestatrec, vehiclestatlist;

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE Scenario(INOUT CATF : CATFObj;
                   INOUT HDC : HDCObj);
BEGIN

        MakeShips (CATF, HDC);
        PlotLZ (CATF);
        BuildSerials (CATF, HDC);

END PROCEDURE { Scenario };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE MakeShips(INOUT CATF : CATFObj;
                    INOUT HDC : HDCObj);

VAR
        ship : ARGObj;
        newrec : SHierRecType;
        i :INTEGER;

BEGIN

        i := 1;

        IF (repetition = 1)
        ASK Scenariorecorder TO WriteString
        ("-----------------SHIP
           DATA-----------------------------");
        ASK Scenariorecorder TO WriteLn;
        ASK Scenariorecorder TO WriteLn;
        ASK Scenariorecorder TO WriteString
```

```
            ("SHIP NAME      SHIP TYPE      STARTx      STARTy
              HOLDx"+"      HOLDy");
            ASK Scenariorecorder TO WriteLn;
            ASK Scenariorecorder TO WriteLn;
        END IF;

            REPEAT
            newrec := ShipSHArray[i];

            IF (newrec = NILREC)
                   ASK TraceStream TO WriteString("NILREC in
                   MakeShips");
                   ASK TraceStream TO WriteLn;
            END IF;

            NEW(ship);
            ASK ship TO ReadData (newrec);

            IF (repetition = 1)
                   ASK Scenariorecorder TO WriteString (ship.name+
                   "          "+ship.type+"          "
                   +INTTOSTR(ROUND(ship.location.x))+
                   "         "+INTTOSTR(ROUND(ship.location.y))
                   +"         "
                   +INTTOSTR(ROUND(ship.holdlocation.x))
                   +"         "+INTTOSTR(ROUND(ship.location.y)));
                   ASK Scenariorecorder TO WriteLn;
            END IF;

                    ASK CATF TO AddShip (ship, HDC);
                    MakeAcft (newrec, ship, HDC);
                    INC(i);

            IF (repetition = 1)
               ASK Scenariorecorder TO WriteLn;
            END IF;

            UNTIL (i > HIGH(ShipSHArray));

    END PROCEDURE;

    {------------------------------------------------------------}
    {------------------------------------------------------------}


    PROCEDURE PlotLZ (INOUT CATF : CATFObj);

    VAR
            LZ     : LZBeachObj;
            newLZ  : SHierRecType;
            i      :INTEGER;
```

```
BEGIN
        i := 1;

        IF (repetition = 1)
           ASK Scenariorecorder TO WriteLn;
           ASK Scenariorecorder TO WriteLn;
           ASK Scenariorecorder TO WriteString
           ("  -----------------LZ
               DATA--------------------------------");
           ASK Scenariorecorder TO WriteLn;
           ASK Scenariorecorder TO WriteLn;
           ASK Scenariorecorder TO WriteString
           ("LZ NAME     LOCATIONx     LOCATIONy     SPOTS");
           ASK Scenariorecorder TO WriteLn;
           ASK Scenariorecorder TO WriteLn;
        END IF;

        REPEAT
           newLZ := LZSHArray[i];
           IF newLZ = NILREC
               ASK TraceStream TO WriteString("NILREC in
               PlotLZS");
               ASK TraceStream TO WriteLn;
           END IF;

           NEW(LZ);
           ASK LZ TO ReadData (newLZ);
           ASK CATF TO AddLZBeach (LZ);
           INC(i);

           IF (repetition = 1)
               ASK Scenariorecorder TO WriteString (LZ.name+
               "          "+INTTOSTR(ROUND(LZ.location.x))+
               "            "+INTTOSTR(ROUND(LZ.location.y))+
               "             "+INTTOSTR(LZ.numlandingspots));
               ASK Scenariorecorder TO WriteLn;
               ASK Scenariorecorder TO WriteLn;
           END IF;

        UNTIL (i > HIGH(LZSHArray));

END PROCEDURE { PlotLZ };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

PROCEDURE BuildSerials (INOUT CATF : CATFObj;
                        INOUT HDC : HDCObj);

VAR
        serial     : SerialObj;
        newserial   : SHierRecType;
        i           :INTEGER;

BEGIN

        i := 1;
        REPEAT
           newserial := SerialSHArray[i];
           IF newserial = NILREC
                   ASK TraceStream TO WriteString
                   ("NILREC in BuildSerials");
                   ASK TraceStream TO WriteLn;
           END IF;

           NEW(serial);
           ASK serial TO ReadData (newserial);
           ASK HDC.seriallist TO Add (serial);

           IF (repetition = 1)
               paxtolift := paxtolift + serial.pax;
               cargotolift := cargotolift + serial.cargo;
           END IF;

           INC(i);
           moreserials := TRUE;
        UNTIL (i > HIGH(SerialSHArray));

        totalserials := ASK HDC.seriallist numberIn;

END PROCEDURE { BuildSerials };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

PROCEDURE MakeAcft ( INOUT newrec : SHierRecType;
                     INOUT ship : ARGObj;
                     INOUT HDC : HDCObj);

VAR
        i              : INTEGER;
        n              : INTEGER;
        acft           : TransObj;
        acftdata       : SHierRecType;
        transports     : BOOLEAN;
        attack         : BOOLEAN;
        attackacft     : AttackObj;
        record         : vehiclestatrec;

BEGIN
        i := 1;
        n := 1;
```

```
transports := TRUE;
WHILE ( (newrec.OwnedString[i] <> "H")  AND
        (newrec.OwnedString[i] <> "\\") AND
        (i < HIGH(newrec.OwnedString))  );

    INC (i);

END WHILE;

INC(i);

IF ( (newrec.OwnedString[i-1] = "\\") OR
           (i >= HIGH(newrec.OwnedString)) )
       transports := FALSE;
END IF;

IF (transports)
   IF (repetition = 1)
       ASK Scenariorecorder TO WriteString
       ("        Transports Aboard:");
   END IF;

   REPEAT
       FindSHRec (AcftSHArray, newrec.OwnedString[i]
                 ,acftdata);

       IF (repetition = 1)
           ASK Scenariorecorder TO WriteString
           ("     "+acftdata.TopString+"    "
           +newrec.OwnedString[i+1]);
       END IF;

       n := 1;
      WHILE (n <=(STRTOINT(newrec.OwnedString[i+1])))

           NEW(acft);
            ASK acft TO ReadData (acftdata);
           ASK acft TO SetSide (n);
            ASK acft TO AssignMother(ship);
            ASK acft NewHDC (HDC);
           ASK acft TO SetLaunchTime (STRTOREAL
           (newrec.OwnedString[i+1+n]));
            TELL acft TO Operate;
            ASK ship.squadron TO Add(acft);

           IF (repetition = 1)
               NEW (record);
               record.name := acft.name;
               record.sidenum := acft.sidenumber;
               record.mother := acft.mymother.name;
               ASK vehiclestatlist TO Add (record);
```

```
                END IF;

                 INC(n);
            END WHILE;
            INC(i,n+1);
            n := 1;
            UNTIL ( (i > HIGH(newrec.OwnedString)) OR
                    (newrec.OwnedString[i] = "S") OR
                    (newrec.OwnedString[i] = "A") OR
                    (newrec.OwnedString[i] = "\\") );

            IF (repetition = 1)
                ASK Scenariorecorder TO WriteLn;
            END IF;
      END IF;

      i := 1;
      n := 1;
      attack := TRUE;
      WHILE ( (newrec.OwnedString[i] <> "A")  AND
              (newrec.OwnedString[i] <> "\\") AND
              (i < HIGH(newrec.OwnedString))  );
              INC (i);
      END WHILE;
      INC(i);

      IF ( (newrec.OwnedString[i-1] = "\\") OR
              (i >= HIGH(newrec.OwnedString)) )

         attack := FALSE;
      END IF;

      IF (attack)
         IF (repetition = 1)
            ASK Scenariorecorder TO WriteString
            ("        Attack Acft Aboard:");
         END IF;

         REPEAT
            FindSHRec (AcftSHArray, newrec.OwnedString[i]
                                   ,acftdata);
            IF (repetition = 1)
                ASK Scenariorecorder TO WriteString
                ("     "+acftdata.TopString+"     "
                +newrec.OwnedString[i+1]);
            END IF;

            n := 1;
           WHILE (n <=(STRTOINT(newrec.OwnedString[i+1])))
                NEW(attackacft);
                 ASK attackacft TO ReadData (acftdata);
```

```
                    ASK attackacft TO SetSide (n);
                     ASK attackacft TO AssignMother(ship);
                     ASK attackacft NewHDC (HDC);
                    ASK attackacft TO SetLaunchTime (STRTOREAL
                         (newrec.OwnedString[i+1+n]));
                     TELL attackacft TO Operate;
                     ASK ship.squadron TO Add(attackacft);

                    IF (repetition = 1)
                         NEW (record);
                         record.name := attackacft.name;
                         record.sidenum :=
                                  attackacft.sidenumber;
                         record.mother :=
                                 attackacft.mymother.name;
                         ASK vehiclestatlist TO Add (record);
                     END IF;

               INC(n);
               END WHILE;

               INC(i,n+1);
               n := 1;
               UNTIL ( (i > HIGH(newrec.OwnedString)) OR
                        (newrec.OwnedString[i] = "S") OR
                        (newrec.OwnedString[i] = "H") OR
                        (newrec.OwnedString[i] = "\\") );

                    IF (repetition = 1)
                         ASK Scenariorecorder TO WriteLn;
                    END IF;

        END IF;

END PROCEDURE { MakeAcft };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

END { IMPLEMENTATION } MODULE { CreateARG }.

DEFINITION MODULE DebugRun;

{-----------------------------------------------------------
    MODULE NAME:   FindSHRec        DATE WRITTEN:   01 Mar 92
    AUTHOR:        M. Bailey        LAST MODIFIED:  18 Mar 92
                   Prof NPGS        MODIFIED BY:    S. E. Shaw
```

```
                                                   Capt  USMC
    DESCRIPTION : Used to turn the trouble shooting messages
on and off.
------------------------------------------------------------}

PROCEDURE SetUpD(IN Trace : BOOLEAN);

END { DEFINITION } MODULE { DebugRun }.


IMPLEMENTATION MODULE DebugRun;

{------------------------------------------------------------
    MODULE NAME:    FindSHRec       DATE WRITTEN:   01 Mar 92
    AUTHOR:         M. Bailey        LAST MODIFIED:  18 Mar 92
                    Prof NPGS       MODIFIED BY:    S. E. Shaw
                                                   Capt USMC
    DESCRIPTION : Used to turn the trouble shooting messages
on and off.
------------------------------------------------------------}

FROM IOMod IMPORT FileUseType(Output);
FROM UtilMod IMPORT DateTime;
FROM Debug IMPORT TraceStream;

{-----------------------------------------------------------}
    PROCEDURE SetUpD(IN Trace : BOOLEAN);
{-----------------------------------------------------------}
    VAR DT : STRING;

    BEGIN
        NEW(TraceStream);
        ASK TraceStream TO Open("debug.out", Output);

        DateTime(DT);
        ASK TraceStream TO WriteString(DT);
        ASK TraceStream TO WriteLn;
        ASK TraceStream TO WriteLn;
        ASK TraceStream TO WriteLn;

        IF Trace
           ASK TraceStream TO TraceOff;
           ASK TraceStream TO WriteString("Initially, trace is
                                              on.");
           ASK TraceStream TO WriteLn;
        ELSE
           ASK TraceStream TO WriteString("Initially, trace is
                                              off.");
           ASK TraceStream TO WriteLn;
        END IF;
    END PROCEDURE;

END { IMPLEMENTATION } MODULE { DebugRun }.

DEFINITION MODULE FindSHRec;

{-----------------------------------------------------------
    MODULE NAME:   FindSHRec        DATE WRITTEN:    01 Mar 92
    AUTHOR:        M. Bailey         LAST MODIFIED:   18 Mar 92
                   Prof NPGS        MODIFIED BY:     S. E. Shaw
                                                     Capt  USMC

    DESCRIPTION : Searches for the requested data record from
the input data array.
-----------------------------------------------------------}

FROM RGlobals IMPORT SHierRecType,
             SHArrayType;

PROCEDURE FindSHRec(IN SHArray : SHArrayType;
                    IN TopString : STRING;
                    OUT SHRec : SHierRecType);

END { DEFINITION } MODULE { FindSHRec }.

IMPLEMENTATION MODULE FindSHRec;

{-----------------------------------------------------------
    MODULE NAME:   FindSHRec        DATE WRITTEN:    01 Mar 92
    AUTHOR:        M. Bailey         LAST MODIFIED:   18 Mar 92
                   Prof NPGS        MODIFIED BY:     S. E. Shaw
                                                     Capt  USMC

    DESCRIPTION :  Searches for the requested data record from
the input data array.
-----------------------------------------------------------}

FROM RGlobals IMPORT SHierRecType,
             SHArrayType;
FROM global IMPORT DeBug;

{----------------------------------------------------------}

    PROCEDURE FindSHRec(IN SHArray : SHArrayType;
                        IN TopString : STRING;
                        OUT SHRec : SHierRecType);
{---------------------------------------------------------}

    VAR
```

```
        ThisRec  : SHierRecType;
        i        : INTEGER;

    BEGIN
        i := 0;

        REPEAT
           INC(i);
           ThisRec := SHArray[i];
        UNTIL((i >= HIGH(SHArray)) OR (ThisRec.TopString =
               TopString));

        IF (ThisRec.TopString = TopString)
           SHRec := ThisRec;
        ELSE
           SHRec := NILREC;
        END IF;

    END PROCEDURE;

END { IMPLEMENTATION } MODULE { FindSHRec }.


DEFINITION MODULE FuelGuage;

{------------------------------------------------------------
   MODULE NAME:   FuelGuage         DATE WRITTEN:   01 Mar 92
   AUTHOR:        S. E. Shaw        LAST MODIFIED:
                  Capt  USMC        MODIFIED BY:

   DESCRIPTION : Procedures used to track the fuel usage of
the TransportCraft objects.

------------------------------------------------------------}

FROM Transport IMPORT TransObj;

TYPE
        PROCEDURE BurnFuel ( IN vehicle        : TransObj;
                             IN groundtime     : REAL;
                             IN flighttime     : REAL);

        PROCEDURE CheckGas ( IN vehicle        : TransObj;
                            OUT fuelrequired : BOOLEAN);

        PROCEDURE Getfuel ( IN vehicle   : TransObj;
                           OUT duration : REAL);


END { DEFINITION } MODULE { FuelGuage }.
```

```
IMPLEMENTATION MODULE FuelGuage;

{-------------------------------------------------------------
   MODULE NAME:    FuelGuage        DATE WRITTEN:    18 Mar 92
   AUTHOR:         S. E. Shaw       LAST MODIFIED:
                   Capt  USMC       MODIFIED BY:

   DESCRIPTION :  Procedures used to track the fuel usage of
the TransportCraft objects.

 ------------------------------------------------------------}

FROM SimMod IMPORT SimTime;
FROM global IMPORT LocationXY, Distance;
FROM Transport IMPORT  TransObj;
FROM Debug IMPORT TraceStream;

PROCEDURE BurnFuel (IN vehicle         : TransObj;
                    IN groundtime      : REAL;
                    IN emptytime       : REAL:
                    IN loadedtime      : REAL);

VAR
        groundburn    : REAL;
        loadedburn    : REAL;
        emptyburn     : REAL;
        startfuel     : REAL;
        endfuel       : REAL;

BEGIN
        groundburn := vehicle.groundburnrate;
        loadedburn := vehicle.loadedburnrate;
        emptyburnrate := vehicle.emptyburnrate;
        startfuel := vehicle.fuelonboard;
        endfuel := startfuel
                   - ( groundburn * (groundtime/60.0))
                   - ( emptyburn * (emptytime/60.0))
                   - ( loadedburn * (loadedtime/60.0));
        ASK vehicle TO UseFuel (endfuel);


END PROCEDURE { BurnFuel };

{------------------------------------------------------------}
{------------------------------------------------------------}
PROCEDURE Getfuel (IN vehicle : TransObj;
                   OUT duration : REAL);

VAR
         amount     : REAL;
         refuelrate : REAL;
```

```
          max          : REAL;
          totfuel      : REAL;

        BEGIN
                amount := vehicle.maxfuel -
                          vehicle.fuelonboard;

                IF ((vehicle.fuelonboard <= 0.0) AND
                    (showerrors))
                    ASK TraceStream TO WriteString("GetFuel
                        "+vehicle.name+"   "+
                        INTTOSTR(vehicle.sidenumber)+
                        " OUT OF FUEL");
                   ASK TraceStream TO WriteLn;
                END IF;

                refuelrate := vehicle.destination.pumprate;
                duration := amount / refuelrate;
                ASK vehicle TO TakeOnFuel;

END PROCEDURE { Getfuel };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE CheckGas ( IN vehicle : TransObj;
                     OUT fuelrequired : BOOLEAN);

VAR
       start          : LocationXY;
       end            : LocationXY;
       transitdist    : REAL;
       requiredfuel   : REAL;
BEGIN

    fuelrequired := FALSE;
    ASK vehicle.serialonboard.source TO CurrentPos (start.x,
        start.y);
    end := vehicle.serialonboard.destination.location;
    transitdist := Distance(start, end);
    requiredfuel := (((2.0 * transitdist)/vehicle.loadedspeed)
                   * vehicle.loadedburnrate) *
                   1.3+vehicle.minfuel;

    IF requiredfuel >= vehicle.fuelonboard
        fuelrequired := TRUE;
    END IF;

END PROCEDURE { CheckGas };

{------------------------------------------------------------}
```

```
{----------------------------------------------------------}

END { IMPLEMENTATION } MODULE { FuelGuage }.


DEFINITION MODULE global;

{----------------------------------------------------------
   MODULE NAME:     global          DATE WRITTEN:    18 Mar 92
   AUTHOR:          S. E. Shaw      LAST MODIFIED:
                    Capt  USMC      MODIFIED BY:

   DESCRIPTION : Contains all random number streams, as well
as several control variables. Procedures within create the
random variable streams, as well as empty queues prior to
disposing of them.

-----------------------------------------------------------}

FROM RandMod IMPORT RandomObj;
FROM Debug IMPORT DebugStream;
FROM GrpMod IMPORT QueueObj, RankedObj;
FROM StatMod IMPORT RStatObj;

VAR
    moreserials       : BOOLEAN;
    showerrors        : BOOLEAN;
    InternalStream    : RandomObj;
    ExternalStream    : RandomObj;
    FoldStream        : RandomObj;
    SpreadStream      : RandomObj;
    CargoStream       : RandomObj;
    ReturnTime        : RandomObj;
    RearmTime         : RandomObj;
    totalserials      : INTEGER;
    repetition        : INTEGER;
    paxtolift         : REAL;
    cargotolift       : REAL;

TYPE

    DeBug = OBJECT(DebugStream);
    END OBJECT;

    LocationXY = FIXED RECORD
        x : REAL;
        y : REAL;

    END RECORD { LocationXY };

    PROCEDURE Distance (IN location1 : LocationXY;
                         IN location2 : LocationXY) : REAL;

    PROCEDURE NewRandoms;
    PROCEDURE EmptyQ (IN queue : QueueObj);
    PROCEDURE EmptyRankedQ  (IN queue : RankedObj);

END { DEFINITION } MODULE { global }.


IMPLEMENTATION MODULE global;

{------------------------------------------------------------
   MODULE NAME:   global         DATE WRITTEN:    18 Mar 92
   AUTHOR:        S. E. Shaw     LAST MODIFIED:
                  Capt  USMC     MODIFIED BY:

   DESCRIPTION :  Contains all random number streams, as well
as several control variables. Procedures within create the
random variable streams, as well as empty queues prior to
disposing of them.
------------------------------------------------------------}

FROM RandMod IMPORT RandomObj;
FROM ReadSeed IMPORT ReadSeed;
FROM MathMod IMPORT SQRT;
FROM GrpMod IMPORT QueueObj, RankedObj;

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE Distance (IN location1 : LocationXY;
                    IN location2 : LocationXY) : REAL;
BEGIN
        RETURN SQRT ((location1.x - location2.x) *
                    (location1.x - location2.x) + (location1.y
                    - location2.y) * (location1.y -
                    location2.y));
END PROCEDURE;

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE NewRandoms;
BEGIN

    NEW (InternalStream);
    ASK InternalStream TO SetSeed (ReadSeed());

    NEW (ExternalStream);
    ASK ExternalStream TO SetSeed (ReadSeed());
    NEW (SpreadStream);
    ASK SpreadStream TO SetSeed (ReadSeed());

    NEW (FoldStream);
    ASK FoldStream TO SetSeed (ReadSeed());

    NEW (CargoStream);
    ASK CargoStream TO SetSeed (ReadSeed());

    NEW (ReturnTime);
    ASK ReturnTime TO SetSeed (ReadSeed());

    NEW (RearmTime);
    ASK RearmTime TO SetSeed (ReadSeed());

END PROCEDURE { NewRandoms };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE EmptyQ (IN queue : QueueObj);

VAR
    trash : ANYOBJ;

BEGIN

    IF (ASK queue numberIn > 0)
        trash := ASK queue First();
        WHILE (trash <> NILOBJ)
           ASK queue TO RemoveThis(trash);
           trash := ASK queue First();
        END WHILE;
    END IF;

END PROCEDURE { EmptyQ };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE EmptyRankedQ  (IN queue : RankedObj);

VAR
    trash : ANYOBJ;

BEGIN
    IF (ASK queue numberIn > 0)
        trash := ASK queue First();
        WHILE (trash <> NILOBJ)
           ASK queue TO RemoveThis(trash);
           trash := ASK queue First();
        END WHILE;
    END IF;

END PROCEDURE { EmptyRankedQ };

{------------------------------------------------------------}
{------------------------------------------------------------}

END { IMPLEMENTATION } MODULE { global }.

DEFINITION MODULE Input;

{------------------------------------------------------------
    MODULE NAME:   Input          DATE WRITTEN:    01 Mar 92
    AUTHOR:        M. Bailey       LAST MODIFIED:   18 Mar 92
                   Prof NPGS      MODIFIED BY:     S. E. Shaw
                                                   Capt USMC
    DESCRIPTION : Reads the file containing the names of all
data files used for the simulation.
------------------------------------------------------------}

    PROCEDURE ReadEmAll;

END { DEFINITION } MODULE { Input }.

IMPLEMENTATION MODULE Input;

{------------------------------------------------------------
    MODULE NAME:   Input          DATE WRITTEN:    01 Mar 92
    AUTHOR:        M. Bailey       LAST MODIFIED:   18 Mar 92
                   Prof NPGS      MODIFIED BY:     S. E. Shaw
                                                   Capt  USMC

    DESCRIPTION : Reads the file containing the names of all
data files used for the simulation.
------------------------------------------------------------}

FROM RGlobals IMPORT  FileNameType;
FROM IOMod IMPORT StreamObj, FileUseType(Input);
FROM RGlobals IMPORT MasterFileName,
     AcftSHArray, ShipSHArray, SpotSHArray, LZSHArray,
     OutputFileName, SerialSHArray;
FROM RGlobals IMPORT SeedArray;
FROM ReadLst IMPORT ReadLst;
FROM ReadSeed IMPORT ReadTheSeeds;
FROM global IMPORT DeBug;

    VAR
```

```
        AcftFileName,
        ShipFileName,
        SpotFileName,
        LZFileName,
        SerialFileName,
        SeedFileName : FileNameType;

{------------------------------------------------------------}
    PROCEDURE ReadAcft;
{------------------------------------------------------------}
    BEGIN
        ReadLst(AcftSHArray , AcftFileName);

    END PROCEDURE { ReadAcft };

{------------------------------------------------------------}
    PROCEDURE ReadShip;
{------------------------------------------------------------}
    BEGIN
        ReadLst(ShipSHArray , ShipFileName);

    END PROCEDURE { ReadShip };

{------------------------------------------------------------}
    PROCDURE ReadSpots;
{------------------------------------------------------------}
    BEGIN
        ReadLst(SpotSHArray , SpotFileName);

    END PROCEDURE { ReadSpots };

{------------------------------------------------------------}
    PROCEDURE ReadLZ;
{------------------------------------------------------------}
    BEGIN
        ReadLst(LZSHArray, LZFileName);

    END PROCEDURE { ReadLZ };

{------------------------------------------------------------}
    PROCEDURE ReadSerial;
{------------------------------------------------------------}
    BEGIN
        ReadLst(SerialSHArray, SerialFileName);

    END PROCEDURE { ReadSerial };

{------------------------------------------------------------}
    PROCEDURE ReadEmAll;
{------------------------------------------------------------}
```

```
VAR

File : StreamObj;
str  : STRING;

BEGIN
    NEW(File);
    ASK File TO Open(MasterFileName, Input);

    ASK File TO ReadString(AcftFileName);
    ASK File TO ReadLine(str);

    ASK File TO ReadString(ShipFileName);
    ASK File TO ReadLine(str);

    ASK File TO ReadString(SpotFileName);
    ASK File TO ReadLine(str);

    ASK File TO ReadString(LZFileName);
    ASK File TO ReadLine(str);

    ASK File TO ReadString(SerialFileName);
    ASK File TO ReadLine(str);

    ASK File TO ReadString(SeedFileName);
    ASK File TO ReadLine(str);

    ASK File TO ReadString(OutputFileName);
    ASK File TO ReadLine(str);

    ReadAcft;
    ReadShip;
    ReadSpots;
    ReadLZ;
    ReadSerial;
    ReadTheSeeds(SeedFileName);

END PROCEDURE { ReademAll };

END { Implementation } MODULE { Input }.


DEFINITION MODULE HDCMod;

{-----------------------------------------------------------
   MODULE NAME:    HDCMod          DATE WRITTEN:    18 Mar 92
   AUTHOR:         S. E. Shaw      LAST MODIFIED:
                   Capt  USMC      MODIFIED BY:

   DESCRIPTION : The Helicopter Direction Center obj is used
to control the TransportCraft movements. This object assigns
serials to the craft, as well as direct where the craft go to
pick up their next serial.
--------------------------------------------------------------}

FROM GrpMod IMPORT QueueObj,RankedObj;
FROM SerialMod IMPORT SerialObj;
FROM ARGMod IMPORT ARGObj;
FROM LZBeach IMPORT LZBeachObj;

EXPORTTYPE
        HDCObj = OBJECT; FORWARD;

TYPE

        BriefingRec = RECORD
           serial    : INTEGER;
           dest      : ARGObj;
           lz        : LZBeachObj;
           loadsize  : INTEGER;
        END RECORD;

        SerialListObj = OBJECT (RankedObj);
                OVERRIDE
                ASK METHOD Rank (IN a, b : ANYOBJ) : INTEGER;
        END OBJECT { SerialListObj };

        HDCObj = OBJECT;
                seriallist    : SerialListObj;

                ASK METHOD ObjInit;
                ASK METHOD GiveLoad (IN serialnum : INTEGER;
                                     OUT newload : SerialObj);
                ASK METHOD GiveFirstLoad (IN ship : STRING;
                                          IN acftsize  : INTEGER;
                                          OUT newload : SerialObj;
                                          OUT othership : BOOLEAN);
                ASK METHOD NewDestination (OUT briefing :
                                                  BriefingRec;
                                           IN acftsize :
                                                     INTEGER;
                                           OUT assignedaload
                                                   : BOOLEAN);
        ASK METHOD DestroyHDC;
        ASK METHOD CombineLoads (IN briefing : BriefingRec;
                                 OUT combined : BOOLEAN);

        END OBJECT { HDCObj };

END {DEFINITION } MODULE { HDCMod }.
```

```
IMPLEMENTATION MODULE HDCMod;

{------------------------------------------------------------------
   MODULE NAME:    HDCMod          DATE WRITTEN:    18 Mar 92
   AUTHOR:         S. E. Shaw      LAST MODIFIED:
                   Capt  USMC      MODIFIED BY:

   DESCRIPTION : The Helicopter Direction Center obj is used
to control the TransportCraft movements. This object assigns
serials to the craft, as well as direct where the craft go to
pick up their next serial.
-----------------------------------------------------------------}

FROM SerialMod IMPORT SerialObj;
FROM SimMod IMPORT SimTime;
FROM global IMPORT moreserials, EmptyQ, EmptyRankedQ;
FROM RGlobals IMPORT SerialSHArray;
FROM Debug IMPORT TraceStream;

OBJECT SerialListObj;

{-----------------------------------------------------------------}
{-----------------------------------------------------------------}

    ASK METHOD Rank (IN a, b : ANYOBJ) : INTEGER;
    VAR
        seriala, serialb : SerialObj;

    BEGIN
        seriala := a;
        serialb := b;

        IF seriala.priority < serialb.priority
           RETURN -1;
        ELSIF seriala.priority > serialb.priority
           RETURN 1;
        ELSE
           RETURN 0;
        END IF;

     END METHOD { Rank };

{-----------------------------------------------------------------}
{-----------------------------------------------------------------}

END OBJECT { SerialListObj };

{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

OBJECT HDCObj;
```

```
{----------------------------------------------------------------}
{----------------------------------------------------------------}

    ASK METHOD ObjInit;

    BEGIN

        NEW(seriallist);
        END METHOD { ObjInit };

{----------------------------------------------------------------}
{----------------------------------------------------------------}

    ASK METHOD GiveLoad (IN serialnum : INTEGER;
                         OUT newload : SerialObj);
    VAR
        thisload  : BOOLEAN;
        checkthis : SerialObj;

    BEGIN
        thisload := FALSE;
        checkthis := ASK seriallist First();

        IF (checkthis <> NILOBJ)
           WHILE NOT thisload
               IF checkthis.serialnum = serialnum
                   ASK seriallist TO RemoveThis
                         (checkthis);
                    thisload := TRUE;
               ELSE
                   checkthis := ASK seriallist Next
                         (checkthis);
               END IF;
           END WHILE;

        END IF;

        newload := checkthis;

        END METHOD {GiveLoad };

{----------------------------------------------------------------}
{----------------------------------------------------------------}

    ASK METHOD GiveFirstLoad (IN ship : STRING;
                              IN acftsize : INTEGER;
                              OUT newload : SerialObj;
                             OUT othership  : BOOLEAN);
    VAR
        goodload       : BOOLEAN;
        repositionload : BOOLEAN;
        checkthis          : SerialObj;
        i                  : INTEGER;

    BEGIN
        i := 1;
        goodload := FALSE;
        othership := FALSE;
        checkthis := ASK seriallist First();
        REPEAT
           IF ((checkthis.source.name = ship) AND
                (NOT checkthis.allocated) AND
                (checkthis.minliftsize <= acftsize));

                goodload := TRUE;
                newload := checkthis;

           END IF;
           checkthis := ASK seriallist Next (checkthis);
           INC (i);
        UNTIL ((goodload) OR (i > ASK seriallist numberIn));

        IF (NOT goodload)
           i := 1;
           checkthis := ASK seriallist First();
           REPEAT
                IF ((NOT checkthis.allocated) AND
                (checkthis.minliftsize <= acftsize))
                     repositionload := TRUE;
                     newload := checkthis;
                END IF;
                checkthis := ASK seriallist Next (checkthis);
                INC (i);
           UNTIL ((repositionload) OR (i > ASK seriallist
                     numberIn));
           othership := TRUE;

        END IF;

        ASK newload TO AllocateSelf;

        END METHOD { GiveFirstLoad };

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD NewDestination ( OUT briefing : BriefingRec;
                                IN acftsize : INTEGER;
                                OUT assignedaload : BOOLEAN);
    VAR
        cleared            : BOOLEAN;
        checkthis          : SerialObj;
        serialsleft    : INTEGER;
        assignthisload : SerialObj;

    BEGIN

        serialsleft := ASK seriallist numberIn;
        assignthisload := NILOBJ;
        assignedaload := FALSE;
        NEW(briefing);
        cleared := FALSE;
        checkthis := ASK seriallist First ();

        IF serialsleft > 0
           REPEAT
               IF ((NOT checkthis.allocated) AND
                    (checkthis.minliftsize <= acftsize))

                    ASK checkthis TO AllocateSelf;
                     briefing.dest := checkthis.source;
                     briefing.serial :=
                     checkthis.serialnum;
                    briefing.loadsize :=
                    checkthis.minliftsize;
                     cleared := TRUE;
                    assignthisload := checkthis;
                    assignedaload := TRUE;

               END IF;

               checkthis := ASK seriallist Next (checkthis);

           UNTIL ((cleared) OR (checkthis = NILOBJ));

        ELSE

           moreserials := FALSE;

        END IF;

        IF (assignthisload = NILOBJ)
           briefing.dest := NILOBJ;
           briefing.serial := -100;
           cleared := TRUE;
        END IF;

    END METHOD { NewDestination };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD DestroyHDC;
```

```
BEGIN

    EmptyRankedQ (seriallist);
    DISPOSE (seriallist);
    DISPOSE (SELF);

END METHOD { DestroyHDC };

{---------------------------------------------------------------}
{---------------------------------------------------------------}

ASK METHOD CombineLoads (IN briefing : BriefingRec;
           OUT combined : BOOLEAN);
VAR
    found       : BOOLEAN;
    checkthis   : SerialObj;
    firstload   : SerialObj;
    secondload  : SerialObj;
    match       : BOOLEAN;
    dest        : STRING;

BEGIN
    found := FALSE;
    checkthis := ASK seriallist First();

    IF (checkthis <> NILOBJ)
       REPEAT
           IF (checkthis.serialnum = briefing.serial)
                found := TRUE;
                firstload := checkthis;
           ELSE
                checkthis := ASK seriallist
                 Next(checkthis)
           END IF;
       UNTIL((checkthis = NILOBJ) OR (found));
    END IF;

    IF (NOT found)
       combined := FALSE;

    ELSE

       IF (briefing.dest = NILOBJ);
           dest := briefing.lz.name;
       ELSE
           dest := briefing.dest.name;
       END IF;
       checkthis := NILOBJ;
       match := FALSE;
       checkthis := ASK seriallist First();
```

```
            IF (checkthis <> NILOBJ)
                REPEAT
                     IF((checkthis.destination.name = dest)
                          AND (checkthis.source.name =
                          firstload.source.name)
                          AND (checkthis.minliftsize = 1)
                          AND (NOT checkthis.allocated))

                          secondload := checkthis;
                          match := TRUE;
                     ELSE
                          checkthis := ASK seriallist
                                            Next(checkthis);
                     END IF;

                UNTIL ((match) OR (checkthis = NILOBJ))
            END IF;

            IF (match)
                combined := TRUE;
                ASK firstload TO AddPax (secondload.pax);
                ASK firstload TO AddCargo (secondload.cargo);
                ASK seriallist TO RemoveThis(secondload);

            END IF;

         END IF;

     END METHOD { CombineLoads };

END OBJECT { HDCObj };

{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

END { IMPLEMENTATION } MODULE { HDCMod }.


DEFINITION MODULE LZBeach;

{-----------------------------------------------------------
   MODULE NAME:   LZBeach          DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw       LAST MODIFIED:
                  Capt  USMC       MODIFIED BY:

   DESCRIPTION  :  Defines  the  landing  zone/beach  objects.
Tracks the amount of cargo and pax delivered to each LZ. Uses
the OutputDriver to output this data to the output file.
----------------------------------------------------------}

FROM global IMPORT LocationXY;
FROM ResMod IMPORT ResourceObj;
FROM SpotMan IMPORT SpotManObj;
FROM RGlobals IMPORT SHierRecType;

EXPORTTYPE

    LZBeachObj = OBJECT; FORWARD;

TYPE
    LZBeachObj = OBJECT;

        name            : STRING;
        location        : LocationXY;
        paxinzone       : REAL;
        totalsorties    : INTEGER;
        cargoinzone     : REAL;
        fac             : SpotManObj;
        numlandingspots : INTEGER;
        priorcargo      : REAL;
        priorpax        : REAL;

        ASK METHOD ReadData (IN newlz : SHierRecType);
        ASK METHOD ObjInit;
        ASK METHOD ReceiveLoad (IN pax        : REAL;
                                IN cargo      : REAL;
                                IN numserials : INTEGER);
        ASK METHOD DestroyLZ;

    END OBJECT { LZBeachObj };

END { DEFINITION } MODULE { LZBeachObj }.


IMPLEMENTATION MODULE LZBeach;

{------------------------------------------------------------
   MODULE NAME:   LZBeach          DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw       LAST MODIFIED:
                  Capt  USMC       MODIFIED BY:

   DESCRIPTION : Defines the landing zone/beach objects.
Tracks the amount of cargo and pax delivered to each LZ. Uses
the OutputDriver to output this data to the output file.
------------------------------------------------------------}

FROM global IMPORT LocationXY, totalserials;
FROM Statistics IMPORT lastdeliverytime;
FROM ResMod IMPORT ResourceObj;
FROM RGlobals IMPORT SHierRecType;
FROM SimMod IMPORT SimTime;
FROM Debug IMPORT TraceStream;
FROM OutputDriver IMPORT WriteLZData;

OBJECT LZBeachObj;

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD ObjInit;

    BEGIN
        NEW (fac);

    END METHOD { ObjInit };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD ReadData (IN newlz : SHierRecType);

    BEGIN
        name := newlz.TopString;
        location.x := STRTOREAL(newlz.OwnedString[1]);
        location.y := STRTOREAL(newlz.OwnedString[2]);

        ASK fac TO PaintSpots (name, name, OBJTYPENAME(SELF),
                  numlandingspots);
        ASK fac TO SetName(name);

    END METHOD { ReadData };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD ReceiveLoad (IN pax        : REAL;
                            IN cargo      : REAL;
                            IN numserials : INTEGER);
    BEGIN

        cargoinzone := cargoinzone + cargo;
        paxinzone := paxinzone + pax;
        WriteLZData (pax, cargo);

        INC (totalsorties);

        totalserials := totalserials - numserials;

        IF (totalserials = 0)
           lastdeliverytime := SimTime();
        END IF;

    END METHOD { ReceiveLoad };
```

```
{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD DestroyLZ;

    BEGIN
        ASK fac TO DestroySpotMan;
        DISPOSE (SELF);

    END METHOD { DestroyLZ };

{-------------------------------------------------------------}
{-------------------------------------------------------------}

END OBJECT { LZBeachObj };

END { IMPLEMENTATION } MODULE { LZBeachObj }.


DEFINITION MODULE LoadProcedures;

{--------------------------------------------------------------
   MODULE NAME:   LoadProcedures  DATE WRITTEN:   3 Apr 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC      MODIFIED BY:

   DESCRIPTION : These procedures determine the time required
to load and unload each serial.
-------------------------------------------------------------}

FROM Transport IMPORT TransObj;
FROM Attack IMPORT AttackObj;

TYPE

    PROCEDURE LoadCargo (IN vehicle : TransObj;
                         OUT loadtime : REAL);

    PROCEDURE UnLoadCargo (IN vehicle : TransObj;
                           OUT loadtime : REAL);

    PROCEDURE ReArmAcft (IN vehicle : AttackObj;
                         OUT rearmtime : REAL);

END { DEFINITION } MODULE { LoadProcedures }.


IMPLEMENTATION MODULE LoadProcedures;

{--------------------------------------------------------------
   MODULE NAME:   LoadProcedures  DATE WRITTEN:   3 Apr 92
   AUTHOR:          S. E. Shaw      LAST MODIFIED:
                    Capt  USMC      MODIFIED BY:

   DESCRIPTION :  These procedures determine the time required
to load and unload  each serial.
----------------------------------------------------------}

FROM Transport IMPORT TransObj;
FROM RandMod IMPORT RandomObj;
FROM Attack IMPORT AttackObj;
FROM SimMod IMPORT SimTime;
FROM global IMPORT InternalStream, ExternalStream,
     CargoStream, RearmTime;

TYPE

    PROCEDURE LoadCargo (IN vehicle : TransObj;
                         OUT loadtime : REAL);
    VAR
        loadtype      : STRING;
        pax           : REAL;
        cargo         : REAL;
        ptime         : REAL;
        ctime         : REAL;

    BEGIN
        loadtype := vehicle.serialonboard.lift;
        pax := vehicle.serialonboard.pax;
        cargo := vehicle.serialonboard.cargo;
        ptime := ASK InternalStream UniformReal(1.0,5.0)+
                 (pax / 100.0) * ASK InternalStream
                 UniformReal(15.0, 25.0);
        ctime := ASK CargoStream Exponential
                 (vehicle.cargotime);

        IF (loadtype = "INTERNAL")

           loadtime := ((ptime) + (cargo / 1000.0) * ctime);

        ELSIF (loadtype = "EXTERNAL")

           loadtime := ASK ExternalStream Exponential
                       (vehicle.externaltime);
        END IF;

    END PROCEDURE { LoadCargo };

{---------------------------------------------------------}
{---------------------------------------------------------}

    PROCEDURE UnLoadCargo (IN vehicle    : TransObj;
                              OUT loadtime : REAL);
  VAR
      loadtype      : STRING;
      pax           : REAL;
      cargo         : REAL;
      ptime         : REAL;
      ctime         : REAL;

  BEGIN
      loadtype := vehicle.serialonboard.lift;
      pax := vehicle.serialonboard.pax;
      cargo := vehicle.serialonboard.cargo;
      ptime := ASK InternalStream UniformReal (1.0, 5.0);
      ctime := ASK CargoStream Exponential
                (vehicle.cargotime);;

      IF (loadtype = "INTERNAL")

         loadtime := (ptime) + ((cargo / 1000.0) * ctime);

      ELSIF (loadtype = "EXTERNAL")

          loadtime := ASK ExternalStream Exponential
                       (vehicle.externaltime);
      END IF;

  END PROCEDURE { UnLOadCargo };

{------------------------------------------------------------}
{------------------------------------------------------------}

  PROCEDURE ReArmAcft (IN vehicle : AttackObj;
                        OUT rearmtime : REAL);
  BEGIN
      rearmtime := ASK RearmTime UniformReal (25.0, 60.0);

  END PROCEDURE { ReArmAcft };

{------------------------------------------------------------}
{------------------------------------------------------------}

END { IMPLEMENTATION } MODULE { LoadProcedures }.

DEFINITION MODULE OutputDriver;
{------------------------------------------------------------
   MODULE NAME:   OutputDriver     DATE WRITTEN: 4 May 92
   AUTHOR:        S. E. Shaw       LAST MODIFIED:
                  Capt  USMC       MODIFIED BY:
```

```
   DESCRIPTION: Opens all output files, combines files as
necessary to form the final scenario file, then closes all
files.
-------------------------------------------------------------}

FROM LZBeach IMPORT LZBeachObj;
FROM IOMod IMPORT StreamObj, FileUseType (Output);
FROM CATFMod IMPORT CATFObj;


VAR
    LZrecorder        : StreamObj;
    Scenariorecorder  : StreamObj;
    StatsRecorder     : StreamObj;
    FinalOutputFile   : StreamObj;
    EndTimerecorder   : StreamObj;
    datestamp         : STRING;

TYPE

    PROCEDURE OpenFiles;
    PROCEDURE WriteLZData (IN pax   : REAL;
                          IN cargo : REAL);
    PROCEDURE CombineFiles;
    PROCEDURE CloseFiles;

END { Definition } MODULE { OutputDriver }.


IMPLEMENTATION MODULE OutputDriver;
{-------------------------------------------------------------
   MODULE NAME:   OutputDriver    DATE WRITTEN:    18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:    5 Jun 92
                  Capt  USMC

   DESCRIPTION :  Opens all output files, combines files as
necessary to form the final scenario file, then closes all
files.

-------------------------------------------------------------}

FROM IOMod IMPORT StreamObj, FileUseType (Output, Input);
FROM UtilMod IMPORT DateTime;
FROM SimMod IMPORT SimTime;
FROM CATFMod IMPORT CATFObj;
FROM RGlobals IMPORT OutputFileName;
FROM Statistics IMPORT beforejump;
FROM global IMPORT paxtolift, cargotolift;

{-------------------------------------------------------------}
{-------------------------------------------------------------}
```

```
PROCEDURE OpenFiles;

BEGIN

    DateTime (datestamp);
    NEW (LZrecorder);
    ASK LZrecorder TO Open ( OutputFileName +"LZ.out",Output);
    ASK LZrecorder TO WriteString (datestamp);
    ASK LZrecorder TO WriteString ("        "+OutputFileName);
    ASK LZrecorder TO WriteLn;
    ASK LZrecorder TO WriteString("  TIME        BEFOREJUMP
        JUMP  "+"   AFTERJUMP");
    ASK LZrecorder TO WriteLn;

    NEW (Scenariorecorder);
    ASK Scenariorecorder TO Open ("Scenario.out", Output);
    ASK Scenariorecorder TO WriteLn;
    ASK Scenariorecorder TO WriteLn;

    NEW (EndTimerecorder);
    ASK EndTimerecorder TO Open (OutputFileName + "End.out",
        Output);
    ASK EndTimerecorder TO WriteString (datestamp);
    ASK EndTimerecorder TO WriteString ("   "+OutputFileName);
    ASK EndTimerecorder TO WriteLn;
    ASK EndTimerecorder TO WriteLn;

    NEW (StatsRecorder);
    ASK StatsRecorder TO Open ("Stats.out", Output);
    ASK StatsRecorder TO WriteString("-----------------
        ---------Stats-----------------------");
    ASK StatsRecorder TO WriteLn;
    ASK StatsRecorder TO WriteLn;

END PROCEDURE { OpenFiles };

{-------------------------------------------------------------}
{-------------------------------------------------------------}

PROCEDURE WriteLZData (IN pax        : REAL;
                       IN cargo      : REAL);
VAR
    x               : REAL;
    paxjump         : REAL;
    cargojump       : REAL;
    totaljump       : REAL;

BEGIN
    x := SimTime();
    paxjump := ((pax / paxtolift)/2.0);
    cargojump := ((cargo / cargotolift)/2.0);
    totaljump := paxjump + cargojump;

    ASK LZrecorder TO WriteString(REALTOSTR(x)+"      "+
    REALTOSTR(beforejump)+"      "+REALTOSTR(totaljump));

    beforejump := beforejump + totaljump;

    ASK LZrecorder TO WriteString("  "+REALTOSTR(beforejump));
    ASK LZrecorder TO WriteLn;


END PROCEDURE { WriteLZData };
{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE CloseFiles;

BEGIN
    ASK LZrecorder TO Close;
    ASK EndTimerecorder TO Close;
    ASK Scenariorecorder TO Close;
    ASK StatsRecorder TO Close;

    CombineFiles;

    DISPOSE (LZrecorder);
    DISPOSE (Scenariorecorder);
    DISPOSE (StatsRecorder);
    DISPOSE (FinalOutputFile);

END PROCEDURE { CloseFiles };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE CombineFiles;

VAR
    placeholder : STRING;

BEGIN
    ASK Scenariorecorder TO Open ("Scenario.out", Input);
    ASK StatsRecorder TO Open ("Stats.out", Input);

    NEW (FinalOutputFile);
    ASK FinalOutputFile TO Open (OutputFileName+".txt",
         Output);
    ASK FinalOutputFile TO WriteString (datestamp);
    ASK FinalOutputFile TO WriteString ("  "+OutputFileName);
    ASK FinalOutputFile TO WriteLn;
    ASK FinalOutputFile TO WriteLn;
    WHILE ( NOT Scenariorecorder.eof)
        ASK Scenariorecorder TO ReadLine (placeholder);
        ASK FinalOutputFile TO WriteString (placeholder);
        ASK FinalOutputFile TO WriteLn;
    END WHILE;

    ASK FinalOutputFile TO WriteLn;
    ASK FinalOutputFile TO WriteLn;

    ASK Scenariorecorder TO Close;

    ASK StatsRecorder TO Open ("Stats.out", Input);

    WHILE ( NOT StatsRecorder.eof)
           ASK StatsRecorder TO ReadLine (placeholder);
           ASK FinalOutputFile TO WriteString (placeholder);
           ASK FinalOutputFile TO WriteLn;
    END WHILE;

    ASK StatsRecorder TO Close;

    ASK Scenariorecorder TO Delete;
    ASK StatsRecorder TO Delete;
    ASK FinalOutputFile TO Close;

END PROCEDURE { CombineFiles };
{-----------------------------------------------------------}
{-----------------------------------------------------------}

END { IMPLEMENTATION } MODULE { OutputDriver }.


DEFINITION MODULE Procedures;

{------------------------------------------------------------
   MODULE NAME:   Procedures      DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC      MODIFIED BY:

   DESCRIPTION : These 2 procedures assign the source and
destination to each serial as it is read in. This information
is entered as a STRING and must be converted to the
appropriate object.
-----------------------------------------------------------}

FROM HDCMod IMPORT HDCObj;
FROM CATFMod IMPORT CATFObj;

TYPE

    PROCEDURE FindSource (INOUT HDC : HDCObj;
                                    IN CATF : CATFObj);

    PROCEDURE FindDestination (INOUT HDC : HDCObj;
                               IN CATF : CATFObj);

END { DEFINITION } MODULE { Procedures }.


IMPLEMENTATION MODULE Procedures;

{-----------------------------------------------------------

   MODULE NAME:   Procedures      DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC      MODIFIED BY:

   DESCRIPTION : These 2 procedures assign the source and
destination to each serial as it is read in. This information
is entered as a STRING and must be converted to the
appropriate object.
-----------------------------------------------------------}

FROM ARGMod IMPORT ARGObj;
FROM LZBeach IMPORT LZBeachObj;
FROM SerialMod IMPORT SerialObj;
FROM HDCMod IMPORT HDCObj;
FROM global IMPORT DeBug;
FROM CATFMod IMPORT CATFObj;

{----------------------------------------------------------}
{----------------------------------------------------------}

    PROCEDURE FindDestination (INOUT HDC : HDCObj;
                               IN CATF : CATFObj);

    VAR
        gohere       : STRING;
        checkzone    : LZBeachObj;
        dropoff      : LZBeachObj;
        load         : SerialObj;
        gooddest     : BOOLEAN;
        i            : INTEGER;

    BEGIN
        i := 0;
        load := ASK HDC.seriallist First();

        WHILE load <> NILOBJ
           INC(i);
           gohere := load.goto;
           gooddest := FALSE;
```

```
            checkzone := ASK CATF.lzbeachlist First();

            REPEAT
                IF gohere = checkzone.name
                    gooddest := TRUE;
                    dropoff := checkzone;
                END IF;
                checkzone := ASK CATF.lzbeachlist Next
                                 (checkzone);
            UNTIL (gooddest);

            IF gooddest
                ASK load TO SetDestination (dropoff);
            ELSE
                OUTPUT ("HDC ERROR IN FINDESTINATION ");
                OUTPUT (load.goto," ",load.gofrom," ",
                        load.serialnum," ",load.cargo," ",
                        load.pax," ",load.priority,"
                        ",load.lift);
            END IF;

            load := ASK HDC.seriallist Next (load);
        END WHILE;

    END PROCEDURE { FindDestination };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

    PROCEDURE FindSource (INOUT HDC : HDCObj;
                          IN CATF : CATFObj);

    VAR
        fromhere        : STRING;
        checkship       : ARGObj;
        origin          : ARGObj;
        goodsource      : BOOLEAN;
        load            : SerialObj;
        i               : INTEGER;

    BEGIN
        load := ASK HDC.seriallist First();
        i := 0;
        WHILE load <> NILOBJ
            INC(i);
            fromhere := load.gofrom;
            goodsource := FALSE;
            checkship := ASK CATF.shiplist First();

            REPEAT
                IF fromhere = checkship.name
                goodsource := TRUE;
                 origin := checkship;
            END IF;
            checkship := ASK CATF.shiplist
                            Next (checkship);
        UNTIL (goodsource);

        IF goodsource
            ASK load TO SetSource (origin);
        ELSE
            OUTPUT ("HDC ERROR IN FINDSOURCE ");
            OUTPUT (load.goto," ",load.gofrom," ",
                    load.serialnum," ",load.cargo," ",
                    load.pax," ",load.priority,"
                   ",load.lift);
        END IF;

        origin := NILOBJ;
        load := ASK HDC.seriallist Next (load);
    END WHILE;

    END PROCEDURE { FindSource };

{-----------------------------------------------------------}
{-----------------------------------------------------------}
```

END { IMPLEMENTATION } MODULE { Procedures }.

DEFINITION MODULE ReadLst;

```
{------------------------------------------------------------
    MODULE NAME:    ReadLst          DATE WRITTEN:    01 Mar 92
    AUTHOR:         M. Bailey         LAST MODIFIED:   18 Mar 92
                    Prof NPGS        MODIFIED BY:     S. E. Shaw
                                                       Capt USMC

    DESCRIPTION : Reads the input data files. Puts all data
into arrays for later use.
-----------------------------------------------------------}

FROM RGlobals IMPORT SHArrayType,
                     FileNameType;

    PROCEDURE ReadLst(INOUT SHArray : SHArrayType;
                      IN FileName   : FileNameType);
```

END { DEFINITION } MODULE { ReadLst }.

IMPLEMENTATION MODULE ReadLst;

```
{-------------------------------------------------------------
    MODULE NAME:    ReadLst         DATE WRITTEN:    01 Mar 92
    AUTHOR:         M. Bailey        LAST MODIFIED:  18 Mar 92
                    Prof NPGS       MODIFIED BY:     S. E. Shaw
                                                     Capt USMC
    DESCRIPTION : Reads the input data files. Puts all data
into arrays for later use.
------------------------------------------------------------}

FROM IOMod IMPORT StreamObj, FileUseType(Input);
FROM RGlobals IMPORT SHArrayType,
     FileNameType;
FROM ReadSH IMPORT ReadSH;
FROM global IMPORT DeBug;

{-----------------------------------------------------------}
    PROCEDURE ReadLst(INOUT SHArray : SHArrayType;
                      IN FileName   : FileNameType);
{-----------------------------------------------------------}
    VAR
        File        : StreamObj;
        numberOfSH  : INTEGER;
        i           : INTEGER;
        error       : BOOLEAN;
        string      : STRING;

    BEGIN
        NEW(File);
        ASK File TO Open(FileName, Input);

        ASK File TO ReadInt(numberOfSH);
        ASK File TO ReadLine(string);

        NEW(SHArray, 1..numberOfSH);

        FOR i := 1 TO numberOfSH
           ReadSH(File, SHArray[i], error);

        END FOR;

    END PROCEDURE { ReadLst };

END { IMPLEMENTATION } MODULE { ReadLst }.


DEFINITION MODULE ReadSeed;

{-------------------------------------------------------------
    MODULE NAME:    ReadSeed        DATE WRITTEN:    01 Mar 92
    AUTHOR:         M. Bailey        LAST MODIFIED:  18 Mar 92
                    Prof NPGS       MODIFIED BY:     S. E. Shaw
```

```
                                                         Capt USMC

     DESCRIPTION : Used to read the initial seeds for the
random variable streams.
-----------------------------------------------------------}

FROM RGlobals IMPORT FileNameType;

    PROCEDURE ReadSeed() : INTEGER;
    PROCEDURE ReadTheSeeds(IN FileName : FileNameType);

END { DEFINITION } MODULE { ReadSeed }.


IMPLEMENTATION MODULE ReadSeed;

{-----------------------------------------------------------
    MODULE NAME:    ReadSeed         DATE WRITTEN:   01 Mar 92
    AUTHOR:         M. Bailey         LAST MODIFIED:  18 Mar 92
                    Prof NPGS        MODIFIED BY:    S. E. Shaw
                                                     Capt USMC

    DESCRIPTION : Used to read the initial seeds for the
random variable streams.
-----------------------------------------------------------}

FROM global IMPORT DeBug;
FROM IOMod IMPORT FileUseType(Input),
     StreamObj;
FROM RGlobals IMPORT FileNameType,
     SeedCount, SeedArray;

{-----------------------------------------------------------}
    PROCEDURE ReadSeed() : INTEGER;
{-----------------------------------------------------------}
    BEGIN
        IF (SeedCount > HIGH(SeedArray))
           OUTPUT("Ran out of seeds with count = " +
               INTTOSTR(SeedCount));
           OUTPUT("Ran out of seeds, make more ");
           HALT;
           RETURN(0);
        ELSE
           IF (SeedCount <= 0)
               SeedCount := 1;
           END IF;
           INC(SeedCount);
           RETURN(SeedArray[SeedCount - 1]);
        END IF;

    END PROCEDURE;
```

```
{-------------------------------------------------------------}
    PROCEDURE ReadTheSeeds(IN FileName : FileNameType);
{-------------------------------------------------------------}
    VAR
        file          : StreamObj;
        str           : STRING;
        i             : INTEGER;
        NumberOfSeeds : INTEGER;

    BEGIN
        NEW(file);
        ASK file TO Open(FileName, Input);
        ASK file TO ReadInt(NumberOfSeeds);
        NEW(SeedArray, 1..NumberOfSeeds);

        FOR i := 1 TO NumberOfSeeds
           ASK file TO ReadInt(SeedArray[i]);
           ASK file TO ReadLine(str);
        END FOR;

    END PROCEDURE;

END { IMPLEMENTATION } MODULE { ReadSeed }.

DEFINITION MODULE ReadSH;

{--------------------------------------------------------------
    MODULE NAME:    ReadSH            DATE WRITTEN:   01 Mar 92
    AUTHOR:         M. Bailey          LAST MODIFIED:  18 Mar 92
                    Prof NPGS         MODIFIED BY:    S. E. Shaw
                                                       Capt USMC

    DESCRIPTION : Reads the data arrays from the input files.

--------------------------------------------------------------}

FROM RGlobals IMPORT SHierRecType;
FROM IOMod IMPORT StreamObj;

    PROCEDURE ReadSH( IN File : StreamObj;
                      OUT SHeirRec : SHierRecType;
                      OUT error : BOOLEAN);

END { DEFINITION } MODULE { ReadSH }.

IMPLEMENTATION MODULE ReadSH;

{--------------------------------------------------------------
    MODULE NAME:    ReadSH            DATE WRITTEN:   01 Mar 92
    AUTHOR:          M. Bailey       LAST MODIFIED:  18 Mar 92
                     Prof NPGS      MODIFIED BY:     S. E. Shaw
                                                     Capt  USMC
    DESCRIPTION : Reads the data arrays from the input files.
-------------------------------------------------------------}

FROM IOMod IMPORT StreamObj, FileUseType(Input);
FROM RGlobals IMPORT SHierRecType;
FROM global IMPORT DeBug;
FROM IOMod IMPORT ReadKey;

{--------------------------------------------------------------}
    PROCEDURE ReadSH( IN File : StreamObj;
                      OUT SHierRec : SHierRecType;
                      OUT error : BOOLEAN);
{--------------------------------------------------------------}

    TYPE
        StringRecType = RECORD
           String : STRING;
           Next : StringRecType;
        END RECORD;

    VAR
        string           : STRING;
        numberOfStrings : INTEGER;
        StringRec, OldStringRec : StringRecType;
        first            : StringRecType;
        arrow            : STRING;
        stringRec        : StringRecType;
        i                : INTEGER;
        z                : CHAR;

    BEGIN
        NEW(SHierRec);
        ASK File TO ReadString(SHierRec.TopString);
        NEW(StringRec);
        numberOfStrings := 1;
        first := StringRec;

        ASK File TO ReadString(arrow);

        IF arrow <> "->"
           OUTPUT("file not formatted correctly");
           error := TRUE;
           RETURN;
        ELSE
           error := FALSE;
        END IF;

        WHILE string <> "\\"
```

```
            ASK File TO ReadString(string);
            IF string = ".."
                ASK File TO ReadLine(string);
            ELSE
                OldStringRec := StringRec;
                StringRec.String := string;
                NEW(StringRec);
                OldStringRec.Next := StringRec;
                numberOfStrings := numberOfStrings + 1;
            END IF;
        END WHILE;

        ASK File TO ReadLine(string);

        IF (numberOfStrings > 0) AND NOT error
            NEW(SHierRec.OwnedString, 1..numberOfStrings - 2);
            stringRec := first;

            FOR i := 1 TO numberOfStrings - 2
                SHierRec.OwnedString[i] := stringRec.String;
                stringRec := stringRec.Next;
            END FOR;
        END IF;

END PROCEDURE { ReadSH };
```

END { IMPLEMENTATION } MODULE { ReadSH }.

DEFINITION MODULE RGlobals;

```
{------------------------------------------------------------

    MODULE NAME:   RGlobals        DATE WRITTEN:   01 Mar 92
    AUTHOR:        M. Bailey        LAST MODIFIED:  18 Mar 92
                   Prof NPGS       MODIFIED BY:    S. E. Shaw
                                                   Capt  USMC

    DESCRIPTION : Contains global variables primarily used for
the input of data.

-----------------------------------------------------------}

    CONST
        MasterFileName = "OPplan.dat";

    TYPE
        FileNameType = STRING;
        SArrayType = ARRAY INTEGER OF STRING;

        SHierRecType = RECORD
```

```
           TopString : STRING;
           OwnedString : SArrayType;
        END RECORD;

        SHArrayType = ARRAY INTEGER OF SHierRecType;
        SeedArrayType = ARRAY INTEGER OF INTEGER;

    VAR
        ShipSHArray     : SHArrayType;
        SpotSHArray     : SHArrayType;
        AcftSHArray     : SHArrayType;
        LZSHArray       : SHArrayType;
        SerialSHArray   : SHArrayType;
        SeedArray       : SeedArrayType;
        OutputFileName  : FileNameType;
        SeedCount       : INTEGER;

END { DEFINITION } MODULE { RGlobals }.


DEFINITION MODULE SerialMod;
{-----------------------------------------------------------------
    MODULE NAME:   SerialMod        DATE WRITTEN:   18 Mar 92
    AUTHOR:        S. E. Shaw        LAST MODIFIED:
                   Capt  USMC        MODIFIED BY:

    DESCRIPTION : Defines the serial objects to be transported
in the simulation.
----------------------------------------------------------------}

FROM LZBeach IMPORT LZBeachObj;
FROM ARGMod IMPORT ARGObj;
FROM RGlobals IMPORT SHierRecType;

EXPORTTYPE
    SerialObj = OBJECT; FORWARD;

TYPE
    CargoLiftType = (internal, external);

    SerialObj = OBJECT;
        destination     : LZBeachObj;
        source          : ARGObj;
        serialnum       : INTEGER;
        cargo           : REAL;
        pax             : REAL;
        priority        : INTEGER;
        lifttype        : STRING;
        goto            : STRING;
        gofrom          : STRING;
        lift            : STRING;
        allocated       : BOOLEAN;
        minliftsize     : INTEGER;
        externalspeed   : REAL;

        ASK METHOD ReadData (IN newserial : SHierRecType);
        ASK METHOD SetDestination (INOUT to : LZBeachObj);
        ASK METHOD SetSource (INOUT from : ARGObj);
        ASK METHOD AllocateSelf;
        ASK METHOD DeAllocateSelf;
        ASK METHOD DestroySerial;
        ASK METHOD AddPax (IN newpax : REAL);
        ASK METHOD AddCargo (IN newcargo : REAL);

    END OBJECT { SerialObj };

END { DEFINITION } MODULE { SerialMod }.


IMPLEMENTATION MODULE SerialMod;
{--------------------------------------------------------------
    MODULE NAME:   SerialMod       DATE WRITTEN:   18 Mar 92
    AUTHOR:        S. E. Shaw       LAST MODIFIED:
                   Capt  USMC       MODIFIED BY:

    DESCRIPTION : Defines the serial objects to be transported
in the simulation.
--------------------------------------------------------------}

FROM ARGMod IMPORT ARGObj;
FROM LZBeach IMPORT LZBeachObj;
FROM RGlobals IMPORT SHierRecType;
FROM Debug IMPORT TraceStream;

VAR
    placeholder : STRING;

OBJECT  SerialObj;
{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD ReadData (IN newserial : SHierRecType);
    BEGIN
        serialnum := STRTOINT(newserial.OwnedString[1]);
        goto := newserial.OwnedString[2];
        gofrom := newserial.OwnedString[3];
        cargo := STRTOREAL(newserial.OwnedString[4]);
        pax := STRTOREAL(newserial.OwnedString[5]);
        priority := STRTOINT(newserial.OwnedString[6]);
        lift := newserial.OwnedString[7];
        minliftsize := STRTOINT(newserial.OwnedString[8]);
        externalspeed := STRTOREAL(newserial.OwnedString[9]);
    END METHOD { ReadData };
{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD SetSource (INOUT from : ARGObj);
    BEGIN
        source := from;

    END METHOD { SetSource };

{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD SetDestination (INOUT to : LZBeachObj);
    BEGIN
        destination := to;

    END METHOD { SetDestination };

{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD DeAllocateSelf;
    BEGIN
        allocated := FALSE;

    END METHOD;

{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD AllocateSelf;
    BEGIN
        allocated := TRUE;

    END METHOD;

{-------------------------------------------------------------}
{-------------------------------------------------------------}

    ASK METHOD DestroySerial;
    BEGIN
        destination := NILOBJ;
        source := NILOBJ;
        DISPOSE (SELF);

    END METHOD { DestroySerial };

{-------------------------------------------------------------}
{-------------------------------------------------------------}
```

```
    ASK METHOD AddPax (IN newpax : REAL);
    BEGIN
        pax := pax + newpax;

    END METHOD { AddPax};

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD AddCargo (IN newcargo : REAL);
    BEGIN
        cargo := cargo + newcargo;

    END METHOD { AddCargo};

{------------------------------------------------------------}
{------------------------------------------------------------}

END OBJECT { SerialObj };

END { IMPLEMENTATION } MODULE { SerialMod }.


DEFINITION MODULE SpotMan;

{-------------------------------------------------------------
   MODULE NAME:    SpotMan         DATE WRITTEN:   18 Mar 92
   AUTHOR:         S. E. Shaw      LAST MODIFIED:
                   Capt  USMC      MODIFIED BY:

   DESCRIPTION  :  This  object  controls  the  landing  and
launching of the TransportCrfat. It tracks the allocation and
use of each landing spot.
------------------------------------------------------------}

FROM GrpMod IMPORT QueueObj, RankedObj;
FROM SpotObject IMPORT SpotObj;

TYPE

    StarboardDObj = OBJECT (RankedObj);
        OVERRIDE
        ASK METHOD Rank (IN a, b : ANYOBJ) : INTEGER;

    END OBJECT { StarboardDObj };

    SpotManObj = OBJECT
        name             : STRING;
        starboardD       : StarboardDObj;
        spotsavail       : QueueObj;
        awaitinglaunch  : QueueObj;

        ASK METHOD ObjInit;
        ASK METHOD SetName(IN newname : STRING);
        ASK METHOD PaintSpots (IN shiptype : STRING;
                               IN shipname : STRING;
                               IN spottype : STRING;
                               OUT numspots : INTEGER);
        ASK METHOD DestroySpotMan;

    END OBJECT { SpotManagerObj };

END { DEFINITION } MODULE { SpotManager }.


IMPLEMENTATION MODULE SpotMan;

{------------------------------------------------------------
   MODULE NAME:   SpotMan         DATE WRITTEN:   18 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC      MODIFIED BY:

   DESCRIPTION  :  This  object  controls  the  landing  and
launching of the TransportCraft. It tracks the allocation and
use of each landing spot.
-----------------------------------------------------------}

FROM ResMod IMPORT ResourceObj;
FROM GrpMod IMPORT QueueObj;
FROM FindSHRec IMPORT FindSHRec;
FROM RGlobals IMPORT SHierRecType, SpotSHArray,
FROM SpotObject IMPORT SpotObj;
FROM global IMPORT EmptyRankedQ, EmptyQ, repetition;
FROM Transport IMPORT TransObj;
FROM Statistics IMPORT lzspotstatlist, shipspotstatlist,
     spotstatrec;

OBJECT StarboardDObj;
{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD Rank (IN a, b : ANYOBJ) : INTEGER;
    VAR
        acfta, acftb : TransObj;

    BEGIN
        acfta := a;
        acftb := b;

        IF acfta.fuelonboard < acftb.fuelonboard
           RETURN -1;
        ELSIF acfta.fuelonboard > acftb.fuelonboard
           RETURN 1;
        ELSE
           RETURN 0;
        END IF;

    END METHOD { Rank };

{----------------------------------------------------------}
{----------------------------------------------------------}

END OBJECT { StarboardDObj };

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

OBJECT SpotManObj;

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD ObjInit;
    BEGIN
        NEW (awaitinglaunch);
        NEW (starboardD);
        NEW (spotsavail);

    END METHOD { ObjInit };

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD PaintSpots (IN shiptype : STRING;
                           IN shipname : STRING;
                           IN spottype : STRING;
                           OUT numspots : INTEGER);

    VAR
        spotdata  : SHierRecType;
        spot      : SpotObj;
        i         : INTEGER;
        record    : spotstatrec;

    BEGIN
        FindSHRec (SpotSHArray, shiptype, spotdata);
        numspots := STRTOINT (spotdata.OwnedString[1]);

        FOR i := 1 TO numspots
           NEW(spot);
           ASK spot TO SetNumber (i);
           ASK spot TO Size (STRTOINT(spotdata.
                                OwnedString[i+1]));
```

```
              ASK spotsavail TO Add (spot);

              IF (repetition = 1)
                  NEW (record);
                  record.name := shipname;
                  record.number := i;
                  IF (spottype = "ARGObj")
                      ASK shipspotstatlist TO Add (record);
                  ELSE
                      ASK lzspotstatlist TO Add (record);
                  END IF;
              END IF;

          END FOR;

END METHOD { PaintSpots };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD SetName(IN newname : STRING);
    BEGIN
        name := newname;

    END METHOD { SetName };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD DestroySpotMan;
    VAR
        spot : SpotObj;
    BEGIN
        EmptyRankedQ (starboardD);
        DISPOSE (starboardD);
        EmptyQ (awaitinglaunch);
        DISPOSE ( awaitinglaunch);

        spot := ASK spotsavail First();

        WHILE ( spot <> NILOBJ)
           ASK spotsavail TO RemoveThis(spot);
           ASK spot TO DestroySpot;
           spot := ASK spotsavail First();
        END WHILE;

        DISPOSE (spotsavail);
        DISPOSE (SELF);

    END METHOD { DestroySpotMan };
```

```
{-------------------------------------------------------------}
{-------------------------------------------------------------}
END OBJECT { SpotManagerObj };

END { IMPLEMENTATION } MODULE { SpotMan }.


DEFINITION MODULE SpotObject;

{-------------------------------------------------------------
   MODULE NAME:    SpotObj         DATE WRITTEN:    23 Mar 92
   AUTHOR:         S. E. Shaw      LAST MODIFIED:
                   Capt  USMC      MODIFIED BY:

   DESCRIPTION : Defines the landing spots for the ARGObj and
for the LZBeachObj.
-------------------------------------------------------------}

FROM StatMod IMPORT RStatObj;

TYPE

    SpotObj = OBJECT
        spotnumber    : INTEGER;
        spotsize      : INTEGER;
        open          : BOOLEAN;
        acftcnspot    : STRING;
        acftsidenum   : INTEGER;
        inuseat       : REAL;
        landings      : INTEGER;

        allocatedtime      : LMONITORED REAL BY RStatObj;
        allocatedtimestats : RStatObj;


        ASK METHOD ObjInit;
        ASK METHOD SetNumber (IN i : INTEGER);
        ASK METHOD Size (IN i : INTEGER);
        ASK METHOD Allocate (IN acft : STRING; IN side :
                   INTEGER);
        ASK METHOD DeAllocate;
        ASK METHOD DestroySpot;

    END OBJECT { SpotObj };

END { DEFINITION } MODULE { SpotObj }.


IMPLEMENTATION MODULE SpotObject;

{-------------------------------------------------------------
```

```
   MODULE NAME:    SpotObj        DATE WRITTEN:   24 Mar 92
   AUTHOR:         S. E. Shaw     LAST MODIFIED:
                   Capt  USMC     MODIFIED BY:

   DESCRIPTION : Defines the landing spots for the ARGObj and
for the LZBeachObj.
----------------------------------------------------------}

FROM SimMod IMPORT SimTime;
FROM Debug IMPORT TraceStream;

OBJECT SpotObj;

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD ObjInit;
    BEGIN
        NEW (allocatedtimestats);
        ADDMONITOR (allocatedtime, allocatedtimestats);

    END METHOD { ObjInit };

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD SetNumber (IN i : INTEGER);
    BEGIN
        spotnumber := i;

    END METHOD { SetNumber };

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD Size (IN i : INTEGER);
    BEGIN
        spotsize := i;
        ASK SELF TO DeAllocate;

    END METHOD { Size };

{----------------------------------------------------------}
{----------------------------------------------------------}

    ASK METHOD Allocate (IN acft : STRING; IN side : INTEGER);
    BEGIN
        INC (landings);
        inuseat := SimTime();
        open := FALSE;
        acftonspot := acft;
        acftsidenum := side;

    END METHOD { Allocate };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD DeAllocate;
    BEGIN
        allocatedtime := SimTime() - inuseat;
        open := TRUE;

    END METHOD { DeAllocate };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD DestroySpot;
    BEGIN
        DISPOSE (SELF);

    END METHOD { DestroySpot };

{------------------------------------------------------------}
{------------------------------------------------------------}

END OBJECT { SpotObj };

END { IMPLEMENTATION } MODULE { SpotObj }.


DEFINITION MODULE SpotProcedures;

{------------------------------------------------------------
   MODULE NAME:   SpotProcedures  DATE WRITTEN:    24 Mar 92
   AUTHOR:        S. E. Shaw      LAST MODIFIED:
                  Capt  USMC

   DESCRIPTION : The procedures used to allocate, deallocate
which SpotObjects are in use, which are available for use.
------------------------------------------------------------}

FROM Transport IMPORT TransObj;
FROM SpotObject IMPORT SpotObj;
FROM HDCMod IMPORT HDCObj;
FROM SpotMan IMPORT SpotManObj;

TYPE
    PROCEDURE InitialLaunch (IN requestor  : TransObj;
                             OUT available : BOOLEAN);
    PROCEDURE GetShipSpot (IN requestor : TransObj;
                           OUT available : BOOLEAN);

    PROCEDURE GetLZSpot (IN requestor : TransObj;
                         OUT available : BOOLEAN);

    PROCEDURE GetSpot (IN requestor  : TransObj;
                       IN controller : SpotManObj;
                       OUT available : BOOLEAN);

    PROCEDURE GiveBackShipSpot (IN requestor  : TransObj);

    PROCEDURE GiveBackLZSpot (IN requestor  : TransObj);

    PROCEDURE FindSpot (IN requestor : TransObj;
                        IN controller : SpotManObj;
                        OUT spot : SpotObj);

END { DEFINITION } MODULE { SpotProcedures }.


IMPLEMENTATION MODULE SpotProcedures;

{------------------------------------------------------------
    MODULE NAME:   SpotProcedures  DATE WRITTEN:   24 Mar 92
    AUTHOR:        S. E. Shaw       LAST MODIFIED:
                   Capt  USMC

    DESCRIPTION :  The procedures used to allocate, deallocate
which SpotObjects are in use, which are available for use.
------------------------------------------------------------}

FROM Transport IMPORT TransObj;
FROM SpotObject IMPORT SpotObj;
FROM HDCMod IMPORT HDCObj;
FROM SpotMan IMPORT SpotManObj;
FROM SimMod IMPORT SimTime;
FROM Debug IMPORT TraceStream;

TYPE

{------------------------------------------------------------}
{------------------------------------------------------------}
    PROCEDURE InitialLaunch (IN requestor  : TransObj;
                             OUT available : BOOLEAN);
    VAR
        checkspot  : SpotObj;
        i          : INTEGER;
        controller : SpotManObj;

    BEGIN
```

```
        controller := requestor.mymother.airboss;
        i := 1;
        available := FALSE;
        checkspot := ASK controller.spotsavail First();
        REPEAT
           IF ((checkspot.open) AND ( checkspot.spotsize >=
                    requestor.spotsizereqd))
               available := TRUE;
               ASK checkspot TO Allocate(requestor.name,
               requestor.sidenumber);

           ELSE
               checkspot := ASK controller.spotsavail
                                            Next(checkspot);
           END IF;

           INC (i);
        UNTIL ((i > ASK controller.spotsavail numberIn) OR
                                (available));
        IF NOT available
           ASK controller.awaitinglaunch TO Add(requestor);
        END IF;

    END PROCEDURE { InitialLaunch };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

    PROCEDURE GetShipSpot (IN requestor : TransObj;
                           OUT available : BOOLEAN);
    VAR
        controller : SpotManObj;

    BEGIN
        controller := requestor.destination.airboss;
        GetSpot (requestor, controller, available);

    END PROCEDURE { GetLZSpot };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

    PROCEDURE GetLZSpot (IN requestor : TransObj;
                         OUT available : BOOLEAN);
    VAR
        controller : SpotManObj;

    BEGIN
        controller := requestor.serialonboard.destination.fac;
        GetSpot (requestor, controller, available);
```

```
END PROCEDURE { GetLZSpot };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE GetSpot (IN requestor   : TransObj;
                   IN controller  : SpotManObj;
                   OUT available  : BOOLEAN);
VAR
    checkspot : SpotObj;
    i         : INTEGER;

BEGIN
    i := 1;
    available := FALSE;
    checkspot := ASK controller.spotsavail First();

    REPEAT
       IF ((checkspot.open) AND ( checkspot.spotsize >=
                requestor.spotsizereqd))
          available := TRUE;
          ASK checkspot TO Allocate(requestor.name,
                requestor.sidenumber);
       ELSE
          checkspot := ASK controller.spotsavail
                 Next(checkspot);
       END IF;
       INC (i);
    UNTIL ((i > ASK controller.spotsavail numberIn) OR
                      (available));
    IF NOT available
       ASK controller.starboardD TO Add (requestor);
    END IF;

END PROCEDURE { GetSpot };

{------------------------------------------------------------}
{------------------------------------------------------------}

PROCEDURE GiveBackShipSpot (IN requestor  : TransObj);

VAR
    spot        : SpotObj;
    landacft    : TransObj;
    launchacft  : TransObj;
    waiting     : INTEGER;
    controller  : SpotManObj;
    i, j        : INTEGER;
    launchone   : BOOLEAN;
    landone     : BOOLEAN;
```

```
BEGIN
    controller := requestor.destination.airboss;
    FindSpot(requestor, controller, spot);
    launchone := FALSE;
    landone := FALSE;

    ASK spot TO DeAllocate;

    IF (ASK controller.awaitinglaunch numberIn >= 1)
       launchacft := ASK controller.awaitinglaunch
                First();
       i := 1;

       REPEAT
           IF (spot.spotsize >= launchacft.spotsizereqd)
               launchone := TRUE;
                TELL launchacft TO Spot;
               ASK spot TO Allocate (launchacft.name,
               launchft.sidenumber);
               EXIT;
           END IF;
           INC (i);
           launchacft := ASK controller.awaitinglaunch
               Next(launchacft);
       UNTIL ((i > ASK controller.awaitinglaunch numberIn)
               OR (launchone));
    END IF;

    IF launchone
       ASK controller.awaitinglaunch TO
           RemoveThis(launchacft);
    END IF;

    IF ((ASK controller.starboardD numberIn >= 1) AND
           ( NOT launchone))
       landacft := ASK controller.starboardD First();
       j := 1;

       REPEAT
           IF(spot.spotsize >= landacft.spotsizereqd)
               landone := TRUE;
               IF (NOT landacft.shutdown)
                    TELL landacft TO Load;
               END IF;
               ASK spot TO Allocate (landacft.name,
                    landacft.sidenumber);
               EXIT:
           END IF;
           INC (j);
           landacft := ASK controller.starboardD
                Next(landacft);
```

```
         UNTIL ((j > ASK controller.starboardD numberIn) OR
             (landone));
      END IF;

      IF landone
         ASK controller.starboardD TO
         RemoveThis(landacft);
      END IF;

      IF ((NOT launchone) AND (NOT landone))
         ASK spot TO DeAllocate;
      END IF;

   END PROCEDURE { GiveBackSpot };

{------------------------------------------------------------}
{------------------------------------------------------------}

   PROCEDURE GiveBackLZSpot (IN requestor  : TransObj);

   VAR
      spot        : SpotObj;
      landacft    : TransObj;
      launchacft  : TransObj;
      waiting     : INTEGER;
      controller  : SpotManObj;

   BEGIN
      controller := requestor.serialonboard.destination.fac;
      FindSpot(requestor, controller, spot);

      ASK spot TO DeAllocate;

      IF (ASK controller.awaitinglaunch numberIn >= 1)
         launchacft := ASK controller.awaitinglaunch
             Remove();
         TELL launchacft TO Spot;
         ASK spot TO Allocate (launchacft.name,
             launchacft.sidenumber);

      ELSIF (ASK controller.starboardD numberIn >= 1)
         landacft := ASK controller.starboardD Remove();
         TELL landacft TO Unload;
         ASK spot TO Allocate (landacft.name,
             landacft.sidenumber);

      ELSE
         ASK spot TO DeAllocate;

      END IF;
```

```
    END PROCEDURE { GiveBackSpot };

{------------------------------------------------------------}
{------------------------------------------------------------}

    PROCEDURE FindSpot (IN requestor : TransObj;
                        IN controller : SpotManObj;
                        OUT spot : SpotObj);
    VAR
        thisspot        : BOOLEAN;
        checkspot       : SpotObj;
        i               : INTEGER;

    BEGIN
        i := 1;
        thisspot := FALSE;
        checkspot := ASK controller.spotsavail First();

        REPEAT
           IF ((checkspot.acftonspot = requestor.name) AND
               (checkspot.acftsidenum = requestor.sidenumber))
               thisspot := TRUE;
               spot := checkspot;
           ELSE
               checkspot := ASK controller.spotsavail
                    Next(checkspot);
           END IF;
           INC (i);
        UNTIL ((i > ASK controller.spotsavail numberIn) OR
           (thisspot));

        IF ((NOT thisspot) AND (showerrors))
           ASK TraceStream TO WriteString
           ("NO MATCH FOUND IN FindSpot*****");
        END IF;

    END PROCEDURE { FindSpot };

{------------------------------------------------------------}
{------------------------------------------------------------}

END { IMPLEMENTATION } MODULE { SpotProcedures }.


DEFINITION MODULE Statistics;

{------------------------------------------------------------
    MODULE NAME:   Statistics       DATE WRITTEN:   18 Mar 92
    AUTHOR:        S. E. Shaw        LAST MODIFIED:   5 Jun 92
                   Capt  USMC
    DESCRIPTION : The procedures used to initialize, reset and
collect the final data.
-----------------------------------------------------------}

FROM StatMod IMPORT RStatObj;
FROM ListMod IMPORT QueueList;
FROM CATFMod IMPORT CATFObj;

VAR
    lastdeliverytime        : LMONITORED REAL BY RStatObj;
    lastdeliverytimestats   : RStatObj;
    vehiclestatlist         : QueueList;
    shipspotstatlist        : QueueList;
    lzspotstatlist          : QueueList;
    beforejump              : REAL;

TYPE

    vehiclestatrec =  RECORD
        name      : STRING;      sorties    : INTEGER;
        sidenum   : INTEGER;     cargo      : REAL;
        mother    : STRING;      pax        : REAL;
        totaltime : REAL;        reps       : INTEGER;
        holding   : REAL;
        shiphold  : REAL;
        beachhold : REAL;
    END RECORD;

    spotstatrec = RECORD
        name       : STRING;     landings   : INTEGER;
        number     : INTEGER;    reps       : INTEGER;
        totaltime  : REAL;
        inuse      : REAL;
    END RECORD;

    StatisticsObj = OBJECT;

        ASK METHOD StartStats;
        ASK METHOD ResetStats;
        ASK METHOD StopStats;
        ASK METHOD CollectRepStats (IN CATF : CATFObj);

    END OBJECT { StatisticsObj };

    PROCEDURE CollectVehicleStats(IN CATF : CATFObj);
    PROCEDURE CollectShipSpotStats (IN CATF : CATFObj);
    PROCEDURE CollectLZSpotStats (IN CATF : CATFObj);
    PROCEDURE FindVehicleRec (IN name : STRING;
                              IN side : INTEGER;
                              IN mother : STRING;
                              OUT record : vehiclestatrec);
    PROCEDURE FindSpotRec (IN name : STRING;
                           IN side : INTEGER;
                           IN list : QueueList;
                           OUT record : spotstatrec);

    PROCEDURE CompileStats;

END { DEFINITION } MODULE { Statistics }.


IMPLEMENTATION MODULE Statistics;
{------------------------------------------------------------
    MODULE NAME:    Statistics      DATE WRITTEN:   18 Mar 92
    AUTHOR:         S. E. Shaw       LAST MODIFIED:
                    Capt  USMC
    DESCRIPTION : The procedures used to initialize, reset and
collect the final data.
------------------------------------------------------------}

FROM StatMod IMPORT RStatObj;
FROM OutputDriver IMPORT StatsRecorder, LZrecorder;
FROM CATFMod IMPORT CATFObj;
FROM ARGMod IMPORT ARGObj;
FROM Transport IMPORT TransObj;
FROM SpotObject IMPORT SpotObj;
FROM GrpMod IMPORT QueueObj;
FROM SimMod IMPORT SimTime;
FROM LZBeach IMPORT LZBeachObj;
FROM ListMod IMPORT QueueList;

OBJECT StatisticsObj;
{-----------------------------------------------------------}
{-----------------------------------------------------------}

    ASK METHOD StartStats;
    BEGIN
        NEW (lastdeliverytimestats);
        ADDMONITOR (lastdeliverytime, lastdeliverytimestats);
        NEW (vehiclestatlist);
        NEW (shipspotstatlist);
        NEW (lzspotstatlist);

    END METHOD { StartStats };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

    ASK METHOD ResetStats;
    BEGIN
        ASK LZrecorder TO WriteString ("-1 -1 -1 -1 -1 -1);
        ASK LZrecorder TO WriteLn;
        beforejump := 0.0;

    END METHOD { ResetStats };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD StopStats;
    BEGIN
        ASK StatsRecorder TO WriteString
           ("lastdeliverytime.count := "+INTTOSTR
           (lastdeliverytimestats.Count));
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteString
           ("lastdeliverytime.mean := "+REALTOSTR
           (lastdeliverytimestats.Mean()));
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteString
           ("lastdeliverytime.maximum :=  "+REALTOSTR
           (lastdeliverytimestats.Maximum));
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteString
           ("lastdeliverytime.minimun :=  "+REALTOSTR
           (lastdeliverytimestats.Minimum));
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteString
           ("lastdeliverytime.variance :=  "+REALTOSTR
           (lastdeliverytimestats.Variance()));
        ASK StatsRecorder TO WriteLn;

        CompileStats;

    END METHOD { StopStats };

{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD CollectRepStats (IN CATF : CATFObj);
    BEGIN
        CollectVehicleStats (CATF);
        CollectShipSpotStats (CATF);
        CollectLZSpotStats (CATF);
        OUTPUT("Collected rep stats");

    END METHOD { CollectRepStats };

{------------------------------------------------------------}
{------------------------------------------------------------}

END OBJECT { StatisticsObj };
```

```
{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

    PROCEDURE CollectVehicleStats(IN CATF : CATFObj);
    VAR
        acft   : TransObj;
        ship   : ARGObj;
        record : vehiclestatrec;

    BEGIN
        ship := ASK CATF.shiplist First();

        IF (ship <> NILOBJ)

           REPEAT
               acft := ASK ship.squadron First();

               IF (acft <> NILOBJ)

                   REPEAT
                        FindVehicleRec(acft.name,
                             acft.sidenumber,
                             acft.mymother.name, record);
                        INC (record.reps);
                        record.totaltime := record.totaltime
                             + (acft.shutdowntime -
                             acft.airbornetime);
                        record.holding := record.holding +
                             acft.holdingtimestats.Sum;
                        record.shiphold := record.shiphold +
                             acft.holdingshipstats.Sum;
                        record.beachhold := record.beachhold
                             +acft.holdingbeachstats.Sum;
                        record.cargo := record.cargo +
                             acft.totalcargo;
                        record.pax := record.pax +
                             acft.totalpax;
                        record.sorties := record.sorties +
                             acft.totalsorties;
                        acft := ASK ship.squadron Next(acft);
                   UNTIL ( acft = NILOBJ);
               END IF;
               ship := ASK CATF.shiplist Next(ship);

           UNTIL (ship = NILOBJ);
        END IF;

    END PROCEDURE { CollectVehicleStats };
{---------------------------------------------------------}
{---------------------------------------------------------}

    PROCEDURE FindVehicleRec (IN name : STRING;
                                   IN side : INTEGER;
                                   IN mother : STRING;
                                   OUT record : vehiclestatrec);
VAR
    checkthis : vehiclestatrec;
    found     : BOOLEAN;

BEGIN
    found := FALSE;
    checkthis := ASK vehiclestatlist First();

    IF (checkthis <> NILREC);

       REPEAT

           IF ((checkthis.name = name) AND
               (checkthis.sidenum = side) AND
               (checkthis.mother = mother))

               record := checkthis;
               found := TRUE;
           ELSE
               checkthis := ASK vehiclestatlist
               Next(checkthis);
           END IF;

       UNTIL ((checkthis = NILREC) OR (found));

    ELSE

       OUTPUT ("Error in FindVehicleRec");
       HALT;

    END IF;

END PROCEDURE { FindVehicleRec };
{----------------------------------------------------------}
{----------------------------------------------------------}

PROCEDURE CompileStats;
VAR
    record     : vehiclestatrec;
    spotrecord : spotstatrec;
    stat1, stat2, stat3      : REAL;

BEGIN
    ASK StatsRecorder TO WriteLn;
    ASK StatsRecorder TO WriteString
       ("-----Vehicle  Holding Stats-------");
    ASK StatsRecorder TO WriteLn;
    ASK StatsRecorder TO WriteLn;
ASK StatsRecorder TO WriteString
   ("  Ship     Vehicle    TotalHold%    Shiphold%
   BeachHold%");
ASK StatsRecorder TO WriteLn;
ASK StatsRecorder TO WriteLn;

record := ASK vehiclestatlist First();

IF (record <> NILREC)

   REPEAT
       ASK StatsRecorder TO WriteString
            (record.mother+"    "+record.name+" "+
            INTTOSTR(record.sidenum)
            +"        "
            +INTTOSTR(TRUNC(100.0*
            (record.holding/record.totaltime)))
            +"           "
            +INTTOSTR(TRUNC(100.0*
            (record.shiphold/record.totaltime)))
            +"           "
            +INTTOSTR(TRUNC(100.0*
            (record.beachhold/record.totaltime))));
       ASK StatsRecorder TO WriteLn;
       record := ASK vehiclestatlist Next(record);
   UNTIL (record = NILREC);

ELSE

   OUTPUT("NO VEHICLE RECORDS TO COMPILE");

END IF;

ASK StatsRecorder TO WriteLn;
ASK StatsRecorder TO WriteString
   ("----------Vehicle Sortie Stats-----------");
ASK StatsRecorder TO WriteLn;
ASK StatsRecorder TO WriteLn;
ASK StatsRecorder TO WriteString
   ("  Ship   Vehicle    Cargo    Pax    Sorties ");
ASK StatsRecorder TO WriteLn;
ASK StatsRecorder TO WriteLn;

record := ASK vehiclestatlist First();

IF (record <> NILREC)

   REPEAT
       ASK StatsRecorder TO WriteString
            (record.mother+"    "+record.name+" "+
            INTTOSTR(record.sidenum)
```

```
                  +"      "+INTTOSTR(TRUNC
                  (record.cargo/FLOAT(record.reps)))
                  +"        "+INTTOSTR(TRUNC
                  (record.pax/FLOAT(record.reps)))
                  +"          "
                  +INTTOSTR(record.sorties DIV
                   record.reps));
              ASK StatsRecorder TO WriteLn;
              record := ASK vehiclestatlist Next(record);
          UNTIL (record = NILREC);

       ELSE

          OUTPUT("NO VEHICLE RECORDS TO COMPILE");

       END IF;

       ASK StatsRecorder TO WriteLn;
       ASK StatsRecorder TO WriteString
          ("-------Ship Spot Stats-----------");
       ASK StatsRecorder TO WriteLn;
       ASK StatsRecorder TO WriteLn;
       ASK StatsRecorder TO WriteString
       ("  Ship         Spot      Utilized%    Landings ");
       ASK StatsRecorder TO WriteLn;
       ASK StatsRecorder TO WriteLn;

       spotrecord := ASK shipspotstatlist First();

       IF (spotrecord <> NILREC)

          REPEAT
              ASK StatsRecorder TO WriteString
                  (spotrecord.name+"          "+INTTOSTR
                  (spotrecord.number)
                  +"            "
                  +INTTOSTR(TRUNC(100.0*(spotrecord.inuse
                  /spotrecord.totaltime)))
                  +"         "
                  +INTTOSTR(spotrecord.landings
                  DIV spotrecord.reps));
              ASK StatsRecorder TO WriteLn;
              spotrecord := ASK shipspotstatlist
                  Next(spotrecord);
          UNTIL (spotrecord = NILREC);

       ELSE

          OUTPUT("NO SHIPSPOT RECORDS TO COMPILE");

       END IF;
```

```
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteString
           ("---------LZBeach Spot Stats------");
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteString
           ("LZBeach     Spot      Utilized      Landings  ");
        ASK StatsRecorder TO WriteLn;
        ASK StatsRecorder TO WriteLn;

        spotrecord := ASK lzspotstatlist First();

        IF (spotrecord <> NILREC)

           REPEAT
               ASK StatsRecorder TO WriteString
                   (spotrecord.name+"          "+INTTOSTR
                   (spotrecord.number)
                   +"              "+INTTOSTR(TRUNC
                   (100.0*(spotrecord.inuse
                   /spotrecord.totaltime)))
                   +"              "+INTTOSTR
                   (spotrecord.landings   DIV
                   spotrecord.reps));
               ASK StatsRecorder TO WriteLn;
               spotrecord    :=    ASK    lzspotstatlist
                   Next(spotrecord);
           UNTIL (spotrecord = NILREC);

        ELSE

           OUTPUT("NO LZBEACH SPOT RECORDS TO COMPILE");

        END IF;

    END PROCEDURE { CompileStats };
{------------------------------------------------------------}
{------------------------------------------------------------}

    PROCEDURE CollectShipSpotStats (IN CATF : CATFObj);
    VAR
        spot        : SpotObj;
        ship        : ARGObj;
        record      : spotstatrec;
        spotsavail  : QueueObj;

    BEGIN
        ship := ASK CATF.shiplist First();
        IF (ship <> NILOBJ)
        REPEAT

            spotsavail := ship.airboss.spotsavail;
            spot := ASK spotsavail First();

            IF (spot <> NILOBJ)

                REPEAT

                    FindSpotRec(ship.name,
                         spot.spotnumber,
                         shipspotstatlist, record);

                    INC (record.reps);
                    record.totaltime := record.totaltime
                    + lastdeliverytime;
                    record.inuse := record.inuse +
                    spot.allocatedtimestats.Sum;
                    record.landings := record.landings +
                    spot.landings;

                    spot := ASK spotsavail Next(spot);

                UNTIL ( spot = NILOBJ);
            END IF;

            ship := ASK CATF.shiplist Next(ship);

        UNTIL (ship = NILOBJ);

    END IF;

END PROCEDURE { CollectShipSpotStats };

{----------------------------------------------------------}
{----------------------------------------------------------}

PROCEDURE FindSpotRec (IN name : STRING;
                       IN side : INTEGER;
                       IN list : QueueList;
                       OUT record : spotstatrec);
VAR
    checkthis : spotstatrec;
    found     : BOOLEAN;

BEGIN
    found := FALSE;
    checkthis := ASK list First();

    IF (checkthis <> NILREC);
```

```
          REPEAT
              IF ((checkthis.name = name) AND
                  (checkthis.number = side))

                  record := checkthis;
                  found := TRUE;
              ELSE
                  checkthis := ASK list Next(checkthis);
              END IF;

          UNTIL ((checkthis = NILREC) OR (found));

       ELSE

          OUTPUT ("Error in FindShipSpotRec");
          HALT;

       END IF;

    END PROCEDURE { FindSpotRec };
{-------------------------------------------------------------}
{-------------------------------------------------------------}

    PROCEDURE CollectLZSpotStats (IN CATF : CATFObj);
    VAR
        spot          : SpotObj;
        lz            : LZBeachObj;
        record        : spotstatrec;
        spotsavail    : QueueObj;
    BEGIN
        lz := ASK CATF.lzbeachlist First();

        IF (lz <> NILOBJ)

               REPEAT
                    spotsavail := lz.fac.spotsavail;
                    spot := ASK spotsavail First();

                    IF (spot <> NILOBJ)

                         REPEAT
                              FindSpotRec
                              (lz.name,spot.spotnumber,
                               lzspotstatlist, record);

                              INC (record.reps);
                              record.totaltime :=
                                   record.totaltime
                                   + lastdeliverytime;
                               record.inuse := record.inuse +
                                   spot.allocatedtimestats.Sum;
```

```
                                    record.landings :=
                                        record.landings +
                                       spot.landings;
                                    spot := ASK spotsavail
                                        Next(spot);
                               UNTIL ( spot = NILOBJ);

                          END IF;

                          lz := ASK CATF.lzbeachlist Next(lz);
                      UNTIL (lz = NILOBJ);
             END IF;

         END PROCEDURE { CollectLZSpotStats };

     END { IMPLEMENTATION } MODULE { Statistics }.


     DEFINITION MODULE Transport;

     {-----------------------------------------------------------
         MODULE NAME:   TransportCraft  DATE WRITTEN:   18 Mar 92
         AUTHOR:        S. E. Shaw      LAST MODIFIED:
                        Capt  USMC      MODIFIED BY:

         DESCRIPTION : Defines the objects used to move pax and
     cargo ashore. Can be used for either surface craft (LCAC, LCU,
     etc) or aircraft.
     -----------------------------------------------------------}

     FROM ARGMod IMPORT ARGObj;
     FROM LZBeach IMPORT LZBeachObj;
     FROM SerialMod IMPORT SerialObj;
     FROM HDCMod IMPORT HDCObj;
     FROM global IMPORT LocationXY;
     FROM RGlobals IMPORT SHierRecType;
     FROM StatMod IMPORT RStatObj;

     EXPORTTYPE
         TransObj = OBJECT; FORWARD;

     TYPE
         TransObj = OBJECT

             name         : STRING;      location     : LocationXY;
             sidenumber     : INTEGER;    launchtime     : REAL;
             totalpax       : REAL;       totalcargo     : REAL;
             totalsorties   : INTEGER;    spreadupper    : REAL;
             mymother       : ARGObj;     spreadlower    : REAL;
             maxfuel        : REAL;       totalfuel      : REAL;
             emptyspeed     : REAL;       myhdc          : HDCObj;
```

```
        loadedspeed    : REAL;          emptyburnrate : REAL;
        loadedburnrate : REAL;          destination    : ARGObj;
        serialonboard  : SerialObj;      foldlower      : REAL;
        fuelonboard    : REAL;           foldupper      : REAL;
        acftrange      : REAL;           groundburnrate : REAL;
        crewday        : REAL;           shutdowntime   : REAL;
        serialnum      : INTEGER;       cleared        : BOOLEAN;
        minfuel        : REAL;           holding        : REAL;
        externaltime   : REAL;           paxtime        : REAL;
        cargotime      : REAL;          spotsreqd      : INTEGER;
        spotsizereqd   : INTEGER;       shutdown       : BOOLEAN;
        maxloadsize    : INTEGER;       numserials     : INTEGER;
        airbornetime   : REAL;           totalholding   : REAL;

        holdingtime : LMONITORED REAL BY RStatObj;
        holdingtimestats : RStatObj;

        holdingship : LMONITORED REAL BY RStatObj;
        holdingshipstats : RStatObj;

        holdingbeach : LMONITORED REAL BY RStatObj;
        holdingbeachstats : RStatObj;

        ASK METHOD ObjInit;
        ASK METHOD ReadData (IN record : SHierRecType);
        ASK METHOD NewHDC (IN newHDC : HDCObj);
        ASK METHOD SetSide (IN side : INTEGER);
        ASK METHOD AssignMother (IN mother : ARGObj);
        ASK METHOD UseFuel (IN amount : REAL);
        ASK METHOD TakeOnFuel;
        ASK METHOD SetLaunchTime (IN time : REAL);
        ASK METHOD DestroyVehicle;
        TELL METHOD Load;
        TELL METHOD Operate;
        TELL METHOD GetClearance ;
        TELL METHOD TransitToBeach;
        TELL METHOD FlyToShip;
        TELL METHOD Reposition (IN newserial : SerialObj);
        TELL METHOD ReturnToBase;
        TELL METHOD Unload;
        TELL METHOD Spot;
        TELL METHOD ShutDown;

    END OBJECT { TransPortObj};

END {DEFINITION} MODULE {TransportCraftMod}.

IMPLEMENTATION MODULE Transport;
{-----------------------------------------------------------
```

```
    MODULE NAME:   TransportCraft  DATE WRITTEN:   18 Mar 92
    AUTHOR:        S. E. Shaw       LAST MODIFIED:
                   Capt  USMC       MODIFIED BY:

    DESCRIPTION : Defines the objects used to move pax and
cargo ashore. Can be used for either surface craft (LCAC, LCU,
etc) or aircraft.
------------------------------------------------------------}

FROM SimMod IMPORT SimTime;
FROM ARGMod IMPORT ARGObj;
FROM HDCMod IMPORT HDCObj,BriefingRec;
FROM SerialMod IMPORT SerialObj;
FROM global IMPORT LocationXY, Distance, moreserials,
     SpreadStream, FoldStream;
FROM FuelGuage IMPORT BurnFuel, CheckGas, Getfuel;
FROM RGlobals IMPORT SHierRecType;
FROM SpotProcedures IMPORT GetShipSpot, GetLZSpot,
     GiveBackShipSpot, GiveBackLZSpot, InitialLaunch;
FROM LoadProcedures IMPORT LoadCargo, UnLoadCargo;
FROM Debug IMPORT TraceStream;

OBJECT TransObj;
{-----------------------------------------------------------}
{-----------------------------------------------------------}

    ASK METHOD ObjInit;
    BEGIN
        NEW (holdingtimestats);
        ADDMONITOR (holdingtime, holdingtimestats);
        NEW (holdingshipstats);
        ADDMONITOR (holdingship, holdingshipstats);

        NEW (holdingbeachstats);
        ADDMONITOR (holdingbeach, holdingbeachstats);

    END METHOD { ObjInit };

{-----------------------------------------------------------}
{-----------------------------------------------------------}

    ASK METHOD ReadData (IN record : SHierRecType);

    BEGIN
        name := record.TopString;
        maxfuel := STRTOREAL(record.OwnedString[1]);
        minfuel := STRTOREAL(record.OwnedString[2]);
        emptyspeed := STRTOREAL(record.OwnedString[3]);
        loadedspeed := STRTOREAL(record.OwnedString[4]);
        loadedburnrate := STRTOREAL(record.OwnedString[5]);
        emptyburnrate := STRTOREAL(record.OwnedString[6]);
        groundburnrate := STRTOREAL(record.OwnedString[7]);
        foldlower := STRTOREAL(record.OwnedString[8]);
        foldupper := STRTOREAL(record.OwnedString[9]);
        spreadlower := STRTOREAL(record.OwnedString[10]);
        spreadupper := STRTOREAL(record.OwnedString[11]);
        acftrange := STRTOREAL(record.OwnedString[12]);
        crewday := STRTOREAL(record.OwnedString[13]);
        externaltime := STRTOREAL(record.OwnedString[14]);
        paxtime := STRTOREAL(record.OwnedString[15]);
        cargotime := STRTOREAL(record.OwnedString[16]);
        spotsreqd := STRTOINT(record.OwnedString[17]);
        spotsizereqd := STRTOINT(record.OwnedString[18]);
        maxloadsize := STRTOINT(record.OwnedString[19]);

        fuelonboard := maxfuel;
        serialnum := 0;
        cleared := TRUE;
        crewday := crewday * 60.0;

    END METHOD { ReadData };

{------------------------------------------------------------}
{------------------------------------------------------------}

    TELL METHOD Spot;
    VAR
        ship            : STRING;
        startpoint      : LocationXY;
        endpoint        : LocationXY;
        gonogo          : REAL;
        newserial       : SerialObj;
        available       : BOOLEAN;
        othership       : BOOLEAN;
        loadtime        : REAL;
        briefing        : BriefingRec;
        combined        : BOOLEAN;

    BEGIN
        destination := mymother;
        ship := mymother.name;
        ASK myhdc TO GiveFirstLoad (ship, maxloadsize,
                                    newserial,othership);
        serialonboard := newserial;
        numserials := 1;

        IF ((serialonboard.minliftsize = 1) AND
           (maxloadsize > 1))
               NEW(briefing);
               briefing.serial := serialonboard.serialnum;
               briefing.dest := NILOBJ;
               briefing.lz := serialonboard.destination;
```

```
                briefing.loadsize := serialonboard.minliftsize;
                ASK myhdc TO CombineLoads (briefing, combined);

                IF (combined)
                     numserials := 2;
                END IF;
         END IF;

         WAIT DURATION (ASK  SpreadStream UniformReal
             (spreadlower, spreadupper));
         END  WAIT;  {spread acft wait}

         IF othership
            TELL SELF TO Reposition (newserial);
            TERMINATE;
         ELSE
            ASK myhdc.seriallist TO RemoveThis (serialonboard);
         END IF;

         startpoint := mymother.location;
         endpoint := newserial.destination.location;
         gonogo := Distance (startpoint, endpoint);

         WAIT DURATION loadtime
         END WAIT; { load serial wait }

         BurnFuel (SELF, loadtime, 0.0, 0.0);
         airbornetime := SimTime();

         TELL SELF TO TransitToBeach;

     END METHOD { Spot };
{-------------------------------------------------------------}
{-------------------------------------------------------------}

     TELL METHOD Reposition (IN newserial : SerialObj);
     BEGIN
         GiveBackShipSpot (SELF);
         destination := newserial.source;
         serialnum := newserial.serialnum;
         airbornetime := SimTime();
         TELL SELF TO FlyToShip;

     END METHOD { Reposition };
{-------------------------------------------------------------}
{-------------------------------------------------------------}

     TELL METHOD ReturnToBase;
     BEGIN
         GiveBackShipSpot (SELF);
         destination := mymother;
```

```
        WAIT FOR SELF TO FlyToShip;
        END WAIT;

    END METHOD { ReturnToBase };
{------------------------------------------------------------}
{------------------------------------------------------------}

    TELL METHOD Load;
    VAR
        loadtime   : REAL;
        newload    : SerialObj;

    BEGIN
        holdingtime := SimTime() - holding;
        holdingship := SimTime() - holding;
        BurnFuel (SELF, 0.0, holdingtime, 0.0); {fuel used
awaiting
                                                    deckspot}
        IF (serialnum > -1)

           ASK myhdc TO GiveLoad (serialnum, newload);
           serialonboard := newload;
           IF (serialonboard <> NILOBJ)
               LoadCargo (SELF,  loadtime);
               WAIT DURATION loadtime
               END WAIT; { load serial wait }
               totalpax := serialonboard.pax + totalpax;
               totalcargo := serialonboard.cargo
                             + totalcargo;
               BurnFuel (SELF,loadtime, 0.0, 0.0);
            END IF;

        END IF;

         TELL SELF TO TransitToBeach;

    END METHOD { Load };
{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD SetSide (IN side : INTEGER);
    BEGIN
        sidenumber := side;

    END METHOD { SetSide };
{------------------------------------------------------------}
{------------------------------------------------------------}

    ASK METHOD AssignMother (IN mother : ARGObj);
    BEGIN
        mymother := mother;
```

```
   END METHOD { AssignMother };
{------------------------------------------------------------}
{------------------------------------------------------------}

   TELL METHOD Operate;
   VAR
      available    : BOOLEAN;

   BEGIN
      WAIT DURATION launchtime;
      END WAIT;
      InitialLaunch (SELF, available);

      IF (available)
         TELL SELF TO Spot;
      END IF;

   END METHOD { Operate };
{------------------------------------------------------------}
{------------------------------------------------------------}

   ASK METHOD NewHDC (IN HDC : HDCObj);
   BEGIN
      myhdc := HDC;

   END METHOD { SetHDC };
{------------------------------------------------------------}
{------------------------------------------------------------}

   ASK METHOD SetLaunchTime (IN time : REAL);
   BEGIN
      launchtime := time;

   END METHOD { SetLaunchTime };
{------------------------------------------------------------}
{------------------------------------------------------------}

   TELL METHOD TransitToBeach;
   VAR
      start         : LocationXY;
      end           : LocationXY;
      leg           : REAL;
      flighttime    : REAL;
      available     : BOOLEAN;
      airspeed     : REAL;

   BEGIN
      WAIT FOR SELF TO GetClearance;
      END WAIT;

      IF (cleared)
```

```
            ASK serialonboard.source TO CurrentPos(start.x,
                                                   start.y);
            end := serialonboard.destination.location;
            leg := Distance (start, end);

            IF (serialonboard.lift = "EXTERNAL")
                airspeed := serialonboard.externalspeed;
            ELSE
                airspeed := cruisespeed;
            END IF;

            flighttime := (leg / airspeed) * 60.0;
            GiveBackShipSpot (SELF);

            WAIT DURATION flighttime;
            END WAIT; { transit time to shore }
            BurnFuel (SELF, 0.0, 0.0, flighttime);

            holding := SimTime();
            GetLZSpot (SELF, available);

            IF (available)
                TELL SELF TO Unload;
            END IF;

         ELSE

            TELL SELF TO ShutDown;

         END IF;

      END METHOD { TransitToBeach };
{------------------------------------------------------------}
{------------------------------------------------------------}

      TELL METHOD Unload;
      VAR
         briefing       : BriefingRec;
         unloadtime     : REAL;
         available      : BOOLEAN;
         combined       : BOOLEAN;
         assignedaload  : BOOLEAN;

      BEGIN
         location := serialonboard.destination.location;
         holdingtime := SimTime() - holding;
         holdingbeach := SimTime() - holding;

         UnLoadCargo (SELF, unloadtime);
         INC (totalsorties);
```

```
        WAIT DURATION unloadtime;
        END WAIT; { unload serial wait }

        ASK serialonboard.destination TO ReceiveLoad
                (serialonboard.pax,serialonboard.cargo,
                numserials);
        BurnFuel ( SELF, unloadtime, 0.0, holdingtime);

        ASK myhdc NewDestination (briefing, maxloadsize,
                    assignedaload);

        IF assignedaload

           destination := briefing.dest;
           serialnum := briefing.serial;
           numserials := 1;

           IF ((briefing.loadsize = 1) AND (maxloadsize > 1))
               ASK myhdc TO CombineLoads(briefing, combined);

               IF combined
                   numserials := 2;
               END IF;

           END IF;

        ELSE

           destination := mymother;
           serialnum := -100;

        END IF;

        DISPOSE (briefing);
        GiveBackLZSpot(SELF);
        TELL SELF TO FlyToShip;
        ASK serialonboard TO DestroySerial;
        serialonboard := NILOBJ;

    END METHOD { Unload };
{------------------------------------------------------------}
{------------------------------------------------------------}

    TELL METHOD FlyToShip;
    VAR
        start          : LocationXY;
        end            : LocationXY;
        leg            : REAL;
        flighttime     : REAL;
        available      : BOOLEAN;
    BEGIN
        start := location;
        ASK destination CurrentPos (end.x, end.y);
        leg := Distance (start, end);
        flighttime := (leg / cruisespeed) *60.0;

        WAIT DURATION flighttime;
        END WAIT; { transit time to ship }

        holding := SimTime();
        GetShipSpot (SELF, available);

        IF (available)

           IF (NOT shutdown)
               TELL SELF TO Load;
           END IF;

        END IF;

        BurnFuel (SELF, 0.0, flighttime, 0.0);

    END METHOD { FlyToShip };
{--------------------------------------------------------------}
{--------------------------------------------------------------}

    TELL METHOD ShutDown;
    BEGIN
        IF (destination.name <> mymother.name)

           WAIT FOR SELF TO ReturnToBase;
           END WAIT;

        END IF;

        WAIT DURATION ASK FoldStream UniformReal
                                  (foldlower, foldupper)
        END WAIT;

        shutdowntime := SimTime();
        GiveBackShipSpot (SELF);

    END METHOD { ShutDown };
{--------------------------------------------------------------}
{--------------------------------------------------------------}

    ASK METHOD UseFuel (IN amount : REAL);
    BEGIN
        fuelonboard := amount;
    END METHOD;
{---------------------------------------------------------------}
{---------------------------------------------------------------}

    ASK METHOD TakeOnFuel;
    BEGIN
        fuelonboard := maxfuel;

    END METHOD;
{---------------------------------------------------------------}
{---------------------------------------------------------------}

    TELL METHOD GetClearance;
    VAR
        needfuel    : BOOLEAN;
        duration    : REAL;
        Transitdist : REAL;
        start       : LocationXY;
        end         : LocationXY;
    BEGIN
        cleared := FALSE;

        IF (serialnum > -1)
           CheckGas (SELF, needfuel);

           IF needfuel
               Getfuel (SELF, duration);
               WAIT DURATION duration
               END WAIT;
           END IF;

           cleared := TRUE;
           ASK serialonboard.source TO CurrentPos (start.x,
                                                   start.y);
           end := serialonboard.destination.location;
           transitdist := Distance(start, end);

           IF ((launchtime + crewday) <= (SimTime()
                    + 2.0*transitdist/60.0));
               cleared := FALSE;
               shutdown := TRUE;
           END IF;

        END IF;

    END METHOD { GetClearance };
{---------------------------------------------------------------}
{---------------------------------------------------------------}

    ASK METHOD DestroyVehicle;
    BEGIN
```

```
            serialonboard := NILOBJ;
            mymother := NILOBJ;
            myhdc := NILOBJ;
            destination := NILOBJ;
            DISPOSE (SELF);

        END METHOD { DestroyVehicle };
    {------------------------------------------------------------}
    {------------------------------------------------------------}

    END OBJECT { TransPortObj };

    END { IMPLEMENTATION } MODULE { TransportCraft }.
```

APPENDIX B HARDWARE REQUIREMENTS

The simulation presented will run on an IBM compatible personal computer under OS/2 with both a hard disk drive and one floppy drive. For speed considerations, a fast (20+) 80386 or higher computer is recommended. The presence of a math coprocessor is not required, although the use of a coprocessor would reduce the model run time. The simulation itself is written in MODSIM II, an advanced, object-oriented simulation language.

In the OS/2 (ver. 1.21) environment, the modification and compilation of the program requires the MODSIM II language (ver. 1.6) and at least an IBM AT compatible (80386+) computer with 4 megabytes of memory and a hard disk drive. In addition, the MODSIM II compiler requires the Microsoft C compiler (ver. 6.0). MODSIM compiles to the computer language 'C' which is then compiled by the 'C' compiler to the native format of the personal computer. This represents the minimum configuration required to modify and compile the model.

APPENDIX C SAMPLE SHIP-TO-SHORE PROGRAM INPUT FILES

```
------------------- OPplan.dat ---------------------------
Acft.dat
Ship.dat
Spot.dat
LZBeach.dat
Sermv22.dat
Seeds.dat
R34A
```

This data file contains the names of the other input files as well as the desired name for the output files. These input files may have any name that the user desires, but must appear in the order given below:

1) The Aircraft data file name.
2) The Ship data file name.
3) The Spot data file name.
4) The Serial file name.
5) The Seeds file name. Always use 'Seeds.dat', the user has no need to charge this file.
6) The name desired for all of the output files. Allows output from different scenarios to be saved.

```
-------------------- Ship.dat ----------------------------
3
PELELIEU -> LHA 15 50 20 17 15 50 500.0 H MV22 8 0 0 0 0 0 0
20 20  CH53AD 4 20 20 20 20    \\
NEWPORT  -> LST  10 50  15 10 10 50 300 0  \\
RALIEGH -> LPD 23 50 18 10 23 50 200.0  \\
```

This file contains input regarding the operating characteristics of the amphibious ships. There must be one record for each ship within the scenario to be run.

numberOfShipsInAll ... The number of ship records to be read.

ShipName -> ... Name of the first ship.
type ... The ship type.

location.x ... The x and y coordinates of the ships start
location.y point

steamspeed ... The ships steaming speed.

holdingspeed ... The ships holding speed.

holdlocation.x ... The x and y coordinates for the ships
holdlocation.y holding position.

pumprate ... The rate (in pounds per minute) that the ship can pump fuel into the aircraft.

<H> ... Signifies the beginning of the transport vehicles aboard this ship.

TransportAcftType1 ... The type of the first transport aircraft. Must match the ones contained in the Acft.dat file.

#TransportAcftType1 ... Number of type 1 Transportcraft.

launchtimes ... Launch times for each of the type 1 Transportacft listed in minutes from time zero. MUST BE ONE LAUNCH TIME FOR EACH TYPE 1 TransportAcft.

TransportAcftType2 ... Type 2 TransportAcft.

launchtimes ... Launch times for each of the type 1 Transportacft listed in minutes from time zero. MUST BE ONE LAUNCH TIME FOR EACH TYPE 2 TransportAcft.

<A> ... Signifies the start of the attack aircraft aboard this ship.

AttackAcftType1 ...The type of the first attack aircraft. Must match the ones contained in the Acft.dat file.

#AttackAcftType1 ... Number of type 1 Attackacft.

launchtimes ... Launch times for each of the type 1 AttackAcft Listed in minutes from time zero. MUST BE ONE LAUNCH TIME FOR EACH TYPE 1 AttackAcft.

AttackAcftType2 ... Type 2 AttackAcft.

#AttackAcftType2 ... Number of type 2 Attackacft.

launchtimes ... Launch times for each of the type 1 AttackAcft Listed in minutes from time zero. MUST BE ONE LAUNCH TIME FOR EACH TYPE 2 AttackAcft.

```
< \\ > ... Signifies the end of this ships record.

The above are repeated for each ship in the scenario.


-------------------- Acft.dat -------------------------
7
CH53E -> 11000 1200   147 139   2539   2218 1109   10   15   10   15
         200 8   2.0   2.0   3.0   1 3 2 \\

CH46    -> 2400    400   132 129   1237   1146    575   10   15   10
         15   100   8   1.5   2.0   3.0   1 2 1 \\

CH60    -> 2340    400   131 128    938    856    425   10   15   10
         15    100 8   1.5   2.0   3.0   1 2 1 \\

CH47 ->   6700   1100   145 144   2230    1869    935   10   15   10 15
         100 8   1.5    2.0   3.0   1 3 2 \\
```

(The S-92, MV-22 and EH-101 data is omitted here. At the time that this thesis was submitted the information was proprietary in nature.)

```
1  numberOfAircraftListsInAll
2   AcftName -> maxfuel minfuel   emptyspeed loadedspeed
    loadedburnrate  emptyburnrate  groundburnrate  foldlower
    foldupper  spreadlower spreadupper   acftrange   crewday
    externaltime paxtime cargotime spotsreqd    spotsizereqd
    maxloadsize < \\ >

---------------------------------------
```

AcftName -> The name of the aircraft. Must match the name used in the Ship.dat file.

maxfuel ... The maximum amount of fuel (in pounds) that the aircraft can carry. This is used when aircraft refuels.

minfuel ... The NATOPS minimum for fuel (in pounds). Used to determine when an aircraft requires refueling.

emptyspeed ... The speed at which the empty aircraft transits from the beach to the ship.

loadedspeed ... The speed at which the loaded aircraft transits from the ship to the beach.

loadedburnrate ... The rate (in pounds per hour) at which the loaded aircraft burns fuel inflight.

emptyburnrate ... The rate (in pounds per hour) at which the empty aircraft burns fuel inflight.

groundburnrate ... The rate (in pounds per hour) at which the aircraft burns fuel while on the deck.

foldlower ... Used to determine how long it takes to fold the Foldupper aircraft during shutdown.

spreadlower ... Used to determine how long it takes to spread spreadlower aircraft prior to launch.

acftrange ... The round trip range of the aircraft. Any number will do for ncw, not used by this version of the model.

crewday ... The crewday for the pilots. One of the factors used to determine the time for shutting down the aircraft.

externaltime ... A parameter to determine the amount of time to hook up an external load.

paxtime ... A parameter for determining the amount of time to load passengers.

cargotime ... A parameter for determining the amount of time to load cargo.

spotsreqd ... The number of landings spots an aircraft requires. Not used here, should be set to 1.

spotsizereqd ... The spot size an aircraft requires for landing. These match with the size of spots contained in the Spot.dat file. Preverts aircraft from landing on spots that they are not allowed on.

maxloadsize ... The maximum size load an aircraft can carry. For example, a CH46 would be set to 1 while a CH53 would get a 2. This allows for the combining of serials to take advantage of the payload capacity if larger aircraft.

-------------------- **Spot.dat** --------------------------

```
7
LHA -> 8 2 2 3 4 5 6 6 6 \\
LST -> 1 6 \\
LSD -> 1 6 \\
LPD -> 2 6 6 \\
LPH -> 6 1 1 4 4 6 6 \\
LZOWL -> 4 6 6 6 6 6 6  \\
LZSPARROW -> 3 6 6 6 6 6 6  \\
```

The number of spots and the size of each spot for all amphibious ships and each LZ may be altered with this file.

numberofShips/LZsinAll ... Number of spot records in the file.

ShipLZType1 -> ... The type of ship, or the LZ name of the first record. For ships, this must match the type field in the Ship.dat file.

numspots ... The number of spots available on the ship or in the LZ.

SizeSpot1 ... The size of the first spot. This should match up with the spotsizereqd field in the Acft.dat file. For example, if a CH53 is given a spotsizereqd of 3, then it would only be allowed to land on spots with a SizeSpot value of 3 or greater.

SizeSpot2 .... Same as for spot 1.

.

.

.

SizeSpotN ... Size of the last spot.

<\\> signifies the end of the current spot record.

There must be one spot record for each Ship type, as well as one spot record for each LZ. The total number of spot records must equal the numberofShips/LZsinAll value given.

---------------------- **LZBeach.dat** ------------------------

```
2
LZSPARROW -> 23 3  \\
LZOWL  -> 8 3  \\
```

This file contains the attributes for each landing zone.

numberOfLZBeachesInAll ... The total number of LZBeach records to be read in.

LZBeachName -> ... The name of the LZ or Beach. Must match the destinations given to the serials in the Serial.dat file.

location.x ... The x and y coordinates of the LZ or Beach.
location.y

< \\ > ... Signifies the end of the LZ or Beach record.

Repeat the above for every LZ or Beach. The number of records must match the numberofLZBeachesInAll value given.

```
--------------------- Serial.dat ----------------------

44
1 ->  101 LZOWL     PELELIEU    0    15    1    INTERNAL  1 0 \\
2 ->  102 LZOWL     PELELIEU    0    15    1    INTERNAL  1 0 \\
3 ->  103 LZOWL     PELELIEU    0    15    1    INTERNAL  1 0 \\
4 ->  104 LZOWL     PELELIEU    0    15    1    INTERNAL  1 0 \\
5 ->  105 LZOWL     PELELIEU    0    15    1    INTERNAL  1 0 \\
.
.
.
44 ->  144 LZSPARROW  PELELIEU  4000  0  22  EXTERNAL  2 80 \\
```

The data required to create the serials within the current scenario is contained within this file.

number of serials ... The total number of serial records to be read in.

record number ... The record number for this serial.

serial number ... The number assigned to the serial. May be different from the record number.

destination ... The destination LZ for the serial. Must match one of the landing zone names in the LZBeach.dat file.

source ... The location of the serial when the simulation commences. Must match one of the ship names in the Ship.dat file.

external cargo ... The amount of external cargo (in pounds) in the serial.

passengers ... The number of passengers in the serial.

priority ... The priority of the serial. Determines when the serial is moved ashore.

mode ... The mode in which the serial is transported ashore. Must be either INTERNAL or EXTERNAL.

minlift ... Minimum sized aircraft required to move the serial ashore. Must correspond to one of the maxloadsize fields of the Acft.dat file.

externalspeed ... The airspeed limitation on the cargo to be carried externally.

APPENDIX D CUTTER SAMPLE OUTPUT

This Appendix contains examples of the various output files for one run of the *Cutter* model. The <filename> used below refers to the user name input through the ***OPplan.dat*** file.

A. <filename>.txt

This is the summary file for the current scenario. The file will list the ships used, their locations, the number and types of aircraft aboard each ship and various summary statistics.

```
------------------SHIP DATA-------------------------------
SHIP NAME    SHIP TYPE    STARTx    STARTy    HOLDx    HOLDy
PELELIEU        LHA         15         5        15        5
        Transports Aboard:      CH46     12      CH53AD     4
NEWPORT         LST         10         5        10        5
RALEIGH         LPD         23         5        23        5
-----------------LZ DATA----------------------------------
LZ NAME     LOCATIONx      LOCATIONy      SPOTS
LZSPARROW     23               3            3
LZOWL          8               3            4
----------------------------Stats------------------------
lastdeliverytime.count := 120
lastdeliverytime.mean := 163.945180
lastdeliverytime.maximum :=  336.245950
lastdeliverytime.minimum :=  100.208837
lastdeliverytime.variance :=  1435.075719
```

B. <filename>LZ.out

This file contains the data recording the build-up of combat power ashore. The data within this file is arranged in four columns. The first, *Timex*, records the time *t* at which a serial arrives ashore, either to an LZ or to a beach. *BeforeJump*, the second column, records the total combat power ashore prior to the arrival of the current serial. The *Jump* column contains the combat power value of the arriving serial. The last column, *AfterJump*, contains the total combat power ashore including the new arrival. The data from consecutive replications of the scenario are separated by a row of -1's which are added by the model.

These four columns of data are manipulated by the ***Analysis*** program, described in Chapter IV, to compare two mixes of aircraft.

```
  TIME        BEFOREJUMP       JUMP       AFTERJUMP
17.452585      0.000000      0.035800     0.035800
18.713340      0.035800      0.035800     0.071599
18.971841      0.071599      0.035800     0.107399
19.280203      0.107399      0.035800     0.143198
20.231016      0.143198      0.035800     0.178998
20.625452      0.178998      0.035800     0.214797
29.301536      0.214797      0.032967     0.247764
30.099510      0.247764      0.024725     0.272489
33.884638      0.272489      0.032967     0.305456
35.876570      0.305456      0.035800     0.341256
37.766903      0.341256      0.043956     0.385212
38.137090      0.385212      0.035800     0.421012
38.150543      0.421012      0.021978     0.442990
39.310552      0.442990      0.035800     0.478789
39.464394      0.478789      0.035800     0.514589
```

```
39.592847     0.514589     0.024725     0.539314
39.739906     0.539314     0.035800     0.575113
45.160249     0.575113     0.027473     0.602586
45.911192     0.602586     0.038462     0.641047
48.103371     0.641047     0.032967     0.674015
56.408952     0.674015     0.049923     0.723937
59.120837     0.723937     0.032967     0.756904
62.442687     0.756904     0.016956     0.773860
66.652223     0.773860     0.017900     0.791760
74.186491     0.791760     0.016956     0.808715
81.357447     0.808715     0.032967     0.841682
81.588816     0.841682     0.037740     0.879422
84.377447     0.879422     0.017900     0.897322
85.622038     0.897322     0.017900     0.915222
88.182672     0.915222     0.032967     0.948189
88.332213     0.948189     0.016956     0.965145
93.830519     0.965145     0.017900     0.983044
114.928264     0.983044     0.016956     1.000000
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

C. <filename>End.out

This file has one column which contains the completion time for each replication of the current scenario. Like the *<filename>LZ.out* file, this file contains data for analytical purposes.

```
133.654668
136.116764
109.300731
128.518234
127.457097
147.941331
139.407063
138.621305
126.488858
184.891796
164.029736
121.885986
203.290792
162.114241
162.144698
118.164527
.
.
.
```

```
191.904999
118.680501
138.275285
139.032046
318.220818
140.441441
159.673867
```

LIST OF REFERENCES

1. U. S. Marine Corps FMFRP 14-7, *Over-the-Horizon (OTH) Amphibious Operations Operational Concept*, 15 Mar 91.
2. United States Marine Corps, *Concepts and Issues*, 1992.
3. LeClair, Brian, "Object Oriented, An Overview of Key Concepts," *OR/MS TODAY*, volume 18, number 1, pp.20-24, Feb 91.
4. Ross, Sheldon M., *Introduction to Probability Models*, Academic Press, Inc., Fourth Edition, 1989.
5. Welch, Peter, D., *Computer Performance Modeling Handbook*, Academic Press, Inc., 1983.

BIBLIOGRAPHY

Barlow, Richard E., and Frank Proschan, *Statistical Theory of Reliability and Life Testing*, McArdle Press, Inc., 1981.

Bratley, Paul, Bennet L. Fox, and Linus E. Schrage, *A Guide to Simulation*, Second Edition, Springer-Verlag New York, Inc, 1987.

CACI Products Company, *MODSIMII, The Language for Object-Oriented Programming, Reference Manual*, 1990 de., 3344 North Torrey Pines Court, La Jolla, CA 92037.

Koopmans, Lambert H., *Introduction to Contemporary Statistical Methods*, Duxbury Press, 1991.

Law, Averill M. and David W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 1982.

Salt, John, "Tunnel Vision," *OR/MS TODAY*, volume 18, number 1, pp.42-48, Feb 91.

U. S. Marine Corps FMFRP 14-7, "Over-the-horizon (OTH) Amphibious Operations Operational Concept", 15 Mar 91.

U. S. Navy, NATOPS Flight Manual, *Navy Model MV-22A Aircraft*, 14 Feb 92.

U. S. Navy, NATOPS Flight Manual, *Navy Model CH-53E Aircraft*, 1 May 90.

U. S. Navy, NATOPS Flight Manual, *Navy Model VH-60N Aircraft*, 15 Sept 91.

U. S. Navy, NATOPS Flight Manual, *Navy Model CH-46A/E Aircraft*, 15 Aug 92.

INITIAL DISTRIBUTION LIST

		copies
1)	Commandant of the Marine Corps Code TE06 Headquarters, U.S. Marine Corps Washington, D.C. 20380-0001	1
2)	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
3)	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
4)	Professor Michael P. Bailey Code OR/BA Naval Postgraduate School Monterey, CA 93943-5000	4
5)	Professor William G. Kemple Code OR/Ke Naval Postgraduate School Monterey, CA 93943-5000	1
6)	Capt. Scott E. Shaw HMX-1 Marine Corps Air Facility Marine Corps Combat Development Command Quantico, VA 22134-5061	1
7)	LtCol. J. V. Orlando (Code RPR-4) Headquarters Marine Corps Washington, D.C. 20380	1
8)	Vincent M. Balderrama Sikorsky Aircraft Division S-437A 6900 Main St. Stratford CT 06601-1381	1
9)	Anthony Jareb CNA Representative MCCDC Quantico VA 22134-5001	1
10)	Jeffrey Schneider Sikorsky Aircraft Division S-322A4 6900 Main St. Stratford CT 06601-1381	1