1 Design

The most important question to answer is 'How do you actually simulate something?' There are two steps. First you must determine the attributes (or characteristics) which sufficiently define the object or phenomenon which you are simulating. This means describing it to its fullest extent. The second step is to define the relationships of these attributes to themselves and to the environment.

For example, we can choose a star, our Sun, and describe it. It is located at a particular point in space, it is 2.7 million kilometers across, it has a mass of 2*10133 grams, it is travelling at 32 kilometers-per-second towards Antares, and it gives off radiation. We now choose attributes which delimit these observations. We call its particular point in space its location, its speed and direction is its velocity vector, its weight we call mass, and so on. Then we can accurately describe any particular star simply by defining the values of these attributes.

The second step is to define the relationships of these attributes with themselves and with the attributes of the other objects and phenomena in the simulation. For the star these would include physical relationships such as F = M*A, the inverse square law of radiation, etc., and simulation relationships such as what happens when the star is hit by another star or when the radiated energy is depleted. I define *simulation relationships* to mean those relationships which are unknown but which can be determined to some degree of accuracy by the running of the simulation.

This can be a difficult part of the simulation process. In some cases where we do not know the relationships we will have to make (hopefully educated) guesses. In a task such as the STAR

SHIP simulation (to be described fully in subsequent chapters) many of the relationships are simply fiction and therefore up to the programmer's imagination. Does anyone really know what happens when a photon torpedo hits a star?

In other simulations we try to accurately represent physical phenomena. When we are uncertain or have no basis for a representation we make a guess. After we implement the guess and run the simulation we might be able to judge whether the results are reasonable, and, if not, to change the approach to the problem. In this way a simulation can help us try out several approaches to a given problem without actually having to do the physical experiments.

To be sure, this is the purpose of most industrial applications of computer simulations. When Walt Disney Productions was planning their monorail system they first designed a computer program that simulated the arrivals and departures of the individual cars, the number of anticipated passengers, the locations of the loading ramps, and so on. By running the program the engineers could determine whether their hardware designs were sufficient and make appropriate modifications.

DESIGNING A SIMULATION

A simulation program, as with any piece of software, should be well thought out and designed in a step-by-step, straightforward manner. This should not be thought of as imposing restrictions upon the programmer. Quite the contrary. It frees the programmer to concentrate more on the details of the program and logic and worry less about producing a program so intertwined and complex as to be confusing, unmanageable, and at best, impossible to debug.

The steps in producing any piece of software are the same whether it be for a small program to generate random numbers or an extensive simulation system. They are:

1. Define the overall objective.
2. Identify the major functions.
3. Define the objectives of each of the major functions.
4. Define the interface between the major functions.
5. Define the logic flow.
6. Code the logic flow.
7. Implement and test the logic flow.

While these constitute the major steps in producing software they are not necessarily to be thought of as discrete steps. Although the general flow of work is from top to bottom (from number 1 to number 7) there is quite often an overlapping of progress. Parts 1 through 4 would probably be done together since the definitions required often expect the predefinition of some other part. It is a recursive type of work where one part is tentatively defined, some other part is defined based on the first, and then the first is redefined based on the further understanding of the second part. It means simply that the project must be viewed as a whole and that understanding must be established for all of the parts working together as well as each part working individually. The more time that is spent on preparation, the less time will be needed for debugging later on.

Define the Overall Objective

It should be obvious that an overall objective must be clearly understood before undertaking any task, yet all too often this step is overlooked. Many programmers begin with coding (writing the program), having only a vague idea of what the program is supposed to do. It is a sort of trial and error method in which much time is spent in needless testing, code-writing, and subsequent rewriting. Having a clearly specified objective is therefore of utmost importance.

Identify the Major Functions

A *major function* can be thought of as "a portion or section of the entire system which accomplishes some predefined goal and which is logically distinct from the other functions." For example, in a chess playing program there might be several main functions including displaying the game board, accepting player moves, piece evaluation, and so on.

The next task is to define the major functions for the simulation. Defining these functions can be done in several ways. A large programming task may employ several software and hardware specialists so that a joint effort by all, probably in a common meeting, might be required to talk over the objectives. Each member may have expertise in a particular phase of the project or have been assigned to research some aspect. In the

joint meeting each one can express his ideas and then defend them as the others pick them apart to find flaws.

Most programmers, however, particularly hobbyists and personal computer users, will not have access to a software team and must rely upon their own resources and ingenuity. But that should not stop them from taking on a large task. They should simply partition the task into workable pieces so that when it comes to the point of defining the logic flow the functions can be tackled one at a time. The STAR SHIP simulation was that type of project with the roles of system designer, programmer, etc. all done by the author. It is by dividing the project into workable pieces (functions) that allows one person to take on such a task.

Define the Objectives of Each of the Major Functions

This portion of the system design is closely related to the previous one. Obviously when attempting to identify the major functions you must have a good idea of the objectives of each. This part simply helps to further clarify the task by writing down exactly what you want to accomplish.

This may involve subdividing the major functions into smaller subfunctions. A piece-position evaluation function for a chess-playing program may contain subfunctions to evaluate starting positions, middle-game positions, and ending-game positions.

The definition of the function need not be restricted to strictly software. If, for example, CRT type displays are involved in the task, then preliminary display formats might be drawn up at this stage. The main objective here is to section the task into logical, workable pieces so that each one can be handled individually and the programmer does not have to keep track of the entire system at one time. In the STAR SHIP project I generally use *module* interchangeably with function. A module is easily thought of as a subroutine and, indeed, is often implemented as such. Just as a computer may be made up of several hardware modules, e.g. the CPU board, memory modules, I/O modules, so software is made up of functional modules.

Define the Interface Between the Major Functions

An interface must be defined between the several major modules. The interface is the method by which the modules exchange information.

A simple method of transferring information is through an area of resident (or offline) memory specified as a COMMON area. This means that all modules have access to it. Each module monitors the variables in common and when a change which is significant to a particular module occurs it (the module) takes the appropriate action. If the programming language being used is BASIC then the problem is simple since all variables are accessible to every portion of the program, subroutines included. When using FORTRAN the programmer must explicitly put them into a common area. This is easily done with the COMMON statement. Every language has some means by which all modules may have access to a common data area.

Defining the interface not only means defining the method of interchange but also explicitly naming the variables which will reside in the common area and stating their meanings. One suggested method is to compile a list of the variables, their names, their meanings, and their range of values, grouped logically. They might also be included in a chart showing which modules can read the variable values and which modules can alter (or write) values into the variables.

Define the Logic Flow

It is always advisable to design the logic flow in a structured manner. Without a defined structure there is a tendency to write code concurrent with the development of the logic. This all too often leads to poor coding practices such as employing *tricks*, losing track of the logic flow, and unmanageable code. I always suggest that a program first be written using only comment statements (REMARK statements in BASIC). There should be no actual code included, yet the entire program should be defined. That is the important point. There should be a complete definition of the program before any code is written. Working this way eliminates much of the debugging and subsequent rewriting.

The next chapter presents one form of structuring a program.

Code the Logic Flow

At some point early in the design of a program the decision should be made as to which programming language is to be used for implementation. Knowing the instruction repertoire that will

be available often has an effect on the way the logic flow is developed. For most hobbyists the choices are limited to BASIC and the assembly language of their computer, but for others the choices may be quite varied. Certainly you will want to choose the one that will make your task easiest.

There are many tradeoffs to be considered. If speed of execution is of utmost importance, then an assembly language such as 6502 ASM would be best. But the coding will probably take longer and generally be more tedious. If ease of programming is important then a high-level language such as BASIC or FORTRAN might be a good choice, understanding that it may run slower than its assembly language counterpart. The evaluation of the tradeoffs and their respective importance is up to the programmer.

When the logic has been thoroughly defined the actual code (of the selected language) can be written. It is the programmer's job to develop the code that executes the functions of each structured programming construct. This is quite simple for most languages. As an example, look at the BASIC program of figure 1.1. (Double quotes signify the beginning of the comment field, allowed in some versions of BASIC.)

```
10  IF C <>0 THEN GOTO 40          "IF CODE = 0 :
20  F1=1                           "SET INITIALIZATION FLAG .
30  GOTO 100
40  IF C <>1 THEN GOTO 80          "ORIF CODE = 1 :
50  F1=0                           "CLEAR INITIALIZATION FLAG .
60  GO TO 100
70  REM                            "ELSE CODE IS BAD :
80  PRINT 'BAD CODE'               "SO NOTIFY OPERATOR .
90  REM                            "ENDIF
100 GOSUB 1000                     "GET INPUT FROM OPERATOR ;
110 REM                            "< PRINT SQUARES OF 1 TO 10
120 FOR I = 1 TO 10                "FOR I = 1 TO 10
130 PRINT I↑2                      "PRINT SQUARE OF I .
140 NEXT I                         "ENDFOR
```

Figure 1.1 Example of Implementing Structured Code in BASIC

The comments, signified by double quotes, are written first. The actual executable code is then added which implements the functions described in the structured programming comment statements and are written parallel to them.

The structured comments make it clear what the executable statements are accomplishing. Without them, understanding the function of any section of code becomes a task in itself, and a

programmer wants to program, not to be an interpreter of confusing code.

Some programmers are fortunate to have access to a piece of software called a *structured programming preprocessor*. It allows the programmer to write source code using structured constructs not normally allowed in a particular language. The source code is then used as input to the preprocessor which interprets it and produces as output a new source-code listing with each of the structured constructs converted into its corresponding code in the selected language. Figures 1.2 and 1.3 serve as an example of a FORTRAN program before and after preprocessing (FORTRAN normally does not allow structured constructs).

While the code after preprocessing looks quite involved the programmer never needs to look at it. He can work directly from the original (Figure 1.2) for all of his implementation and debugging. The processed code is simply used as the input to the compiler.

```
      FOR I = 1 , 10
      WRITE (1,10)A
10    FORMAT('HELLO  THERE')
      ENDFOR
      X=0
      REPEAT
      X=X+.1
      UNTIL (X.EQ.1)
      Q=100
      WHILE (Q.GT.7)
      Q=Q-.3
      ENDWHILE
      IF (CODE.EQ.1)
      P=7
      ORIF (CODE.EQ.2)
      H=98
      ORIF (CODE.EQ.3)
      J=4
      ELSE
      K=0
      ENDIF
      END
```

Figure 1.2 FORTRAN Example: Before Preprocessing

Actual structured programming constructs are allowed within the instructions.

```
      DO 99999 I = 1 , 10
      WRITE (1,10)A
10    FORMAT('HELLO THERE')
99999 CONTINUE
      X=0
99998 CONTINUE
      X=X+.1
      IF(.NOT.(X.EQ.1)) GOTO 99998
      Q=100
      GOTO 99997
99996 CONTINUE
      Q=Q-.3
99997 CONTINUE
      IF(Q.GT.7) GOTO 99996
      IF(.NOT.(CODE.EQ.1)) GOTO 99995
      P=7
      GOTO 99994
99995 CONTINUE
      IF(.NOT.(CODE.EQ.2)) GOTO 99993
      H=98
      GOTO 99994
99993 CONTINUE
      IF(.NOT.(CODE.EQ.3)) GOTO 99992
      J=4
      GOTO 99994
99992 CONTINUE
      K=0
99994 CONTINUE
      END
```

Figure 1.3 FORTRAN Example After Preprocessing

The structured programming conventions have been replaced by comparable FORTRAN expressions. This is the source used as the input to the compiler.

Implement and Test the Logic Flow

All too often the only test procedure followed is *run it until it fails, then debug*. The more advisable method is to plan a set of test procedures while the software is being developed (steps 1 to 5), when you know exactly what each section of the code should produce. This may include such things as a sample input

with known results (best used in numerically processed data) or a scenario in the case of a simulator.

Including trace functions in the code is a very good practice, too. A trace function is one or more lines of code that print out the values of critical variables at certain points during the execution of the program. It may include messages such as "GOT TO THE FIRST SUBROUTINE OK WITH K = 7" to let the operator follow the logic flow during the run. When an unexpected value or message turns up or the logic flow doesn't follow the expected path the programmer will know exactly where in the program to look for the error. After all of the problems are resolved the trace functions are removed for the final program version.

Some programming systems include a built-in trace function which usually prints out more information than is really necessary. I suggest that the program developer utilize his own tracing methods rather than try to wade through the massive printouts produced by the standard trace functions. He can include only what is sufficient to test his program and not have to cope with so many extraneous trace outputs.

2 Program Structure

I present here a description of the program structuring methods that I have utilized in the STAR SHIP project. What I describe is a nondedicated, structured program logic definition tool. It is nondedicated in the sense that it is not associated with any particular programming language. It is a method of defining the logic flow of a program using a set of Englishlike grammatical constructions. Since the STAR SHIP system is written utilizing these constructions it will help to read through this chapter once and then refer back to it when reading through the STAR SHIP logic flow definitions in the next chapter.

Structured programming is a method of program development. It is not so much a language as it is a technique for utilizing whatever computer language you are working with. It aids the programmer in approaching his task in a logical, straightforward, break-it-apart-and-tackle-it-piece-by-piece manner.

This discussion relates more to the structured design of software and the use of comment statements employing the structured constructs than to the use of a particular language. This means that software should be designed, logically thought out, and completely written down in the form of structured comment statements before any actual code is written. After that has been done the executable code is added which implements the functional descriptions given in the structured comments. With this in mind, we can discuss the details of structured programming.

Structured programming allows the following types of constructions

IF-ORIF-ELSE-ENDIF
REPEAT-UNTIL
WHILE-ENDWHILE
FOR-ENDFOR
DECLARATIVE
RETURN
COMMENT
STATEMENT SEQUENCE

Note particularly that there is no such thing as a GO TO or jump type statement. On the surface it might appear that you can't do without some form of GO TO statement. Consider the fact that it is theoretically possible to write any program in straight-line code with only one type of statement, the decision statement, which states *if such-and-such is true then do this*. Certainly such a program would be difficult to develop, might require a computer with infinite memory, and the decision conditions would get quite involved, but it is possible to develop any program using only the IF statement. Any other additions to the repertoire of instructions are simply embellishments to make it easy for the programmer. The constructions listed above are quite sufficient for any programming task. Now to the descriptions.

IF–ORIF–ELSE–ENDIF

This is the decision-making construction and its general form is as follows:

```
IF (condition 1) :
  statement sequence 1 .
ORIF (condition 2) :
  statement sequence 2 .
ORIF (condition 3) :
  statement sequence 3 .
------
------
------
ORIF (condition n) :
  statement sequence n .
ELSE (comment) :
  statement sequence for else condition .
ENDIF
```


In this construct the first conditional statement is evaluated. If it evaluates to true, then statement sequence 1 is executed and the remainder of the IF construction is skipped i.e. control is passed to the statement following the ENDIF. (The term *statement sequence* will be fully defined later. It is sufficient now to consider it to be a set of statements to be executed.) If conditional statement 1 evaluates to false, then conditional statement 2 is evaluated. If it evaluates to true, then statement sequence 2 is executed and the remainder of the IF construction is skipped. If conditional statement 2 evaluates to false, then conditional statement 3 is evaluated, and so on.

In other words, the first conditional statement within the IF/ORIF/ELSE/ENDIF construction that evaluates to true causes its corresponding statement sequence to be executed and the remainder to be skipped. If none of the conditions evalute to a logical true, then the statement sequence following the ELSE is executed.

Note that any number of ORIFs are allowed, but only one ELSE, and the ELSE must be the last part of the construction before the ENDIF.

That is the general form of the IF. The ORIFs and their associated statement sequences are optional. Likewise, the ELSE and its associated statement sequence is optional. The comment on the ELSE line is also optional but is often included to make the program more readable. Notice, too, that the IF, ORIFs, ELSE, and ENDIF all start in the same character position (card column), and that the statement sequences are indented one character position. The other forms of the IF statement, then, are as follows:

IF–ORIF–ENDIF

```
IF (condition 1) :
  statement sequence 1 .
ORIF (condition 2) :
  statement sequence 2 .
------
------
------
ORIF (condition n) :
  statement sequence n .
ENDIF
```

IF–ELSE–ENDIF

```
IF (condition) :
 statement sequence .
ELSE (comment) :
 statement sequence for ELSE .
ENDIF
```

IF–ENDIF

```
IF (condition) :
 statement sequence .
ENDIF
```

Perhaps a word about CONDITIONs is in order. A condition is a logical statement. That should be familiar to anyone who has used any sort of IF statement, no matter what the language used. It could be of the form IF (X.EQ.7) for those familiar with FORTRAN, or IF A<=B for those familiar with BASIC. Or it could be of a more functional Englishlike form such as IF SWITCH 6 WAS ACTIVATED. The structured programming constructions should tell not just what variables are being used but also the functional objectives of the code, i.e. what is being accomplished.

One last word about the IF-ORIF-ELSE-ENDIF. It is an ideal construction for the SELECT-CASE type of routine. Here is a simple example:

```
IF DIAL 1 WAS TURNED :
 THEN DO SO-AND-SO .
ORIF DIAL 2 WAS TURNED :
 THEN DO SUCH-AND-SUCH .
ORIF DIAL 3 WAS TURNED :
 THEN DO THIS-AND-THAT .
ORIF DIAL 4 WAS TURNED :
 THEN DO SUCH-AND-SO .
ELSE NO DIAL WAS TURNED :
 < SO THIS SUBROUTINE SHOULD NOT HAVE BEEN
 < CALLED.
 NOTIFY THE OPERATOR WITH A MESSAGE .
ENDIF
```


In general, note that each IF and ORIF line is terminated with a colon (:). This is quite helpful in reading the program when a (condition) continues on to subsequent lines.

REPEAT–UNTIL

This construction is used to repeat a section of code (statement sequence) until some specified condition is true. The general form is as follows

```
REPEAT
 statement sequence .
UNTIL (condition) :
```

Generally, a condition is set up to be logically false before the REPEAT-UNTIL is executed. Then the statement sequence is executed over and over again until that condition is logically true. Obviously there must be some statement within the statement sequence which sets the condition true or it will result in an endless loop. Here is an example which prints out the square of a number input by an operator. When the input number is equal to 999 the program stops.

```
REPEAT
 GET NUMBER FROM THE OPERATOR ;
 IF THE NUMBER IS NOT 999 :
  PRINT OUT THE SQUARE OF THE NUMBER .
 ENDIF
UNTIL THE NUMBER INPUT BY THE OPERATOR IS 999 :
```

In the above example the statement sequence which is repeatedly executed is all of the statements between the REPEAT and the UNTIL. Note that the statement sequence may contain subsequent structured programming constructions, in this case the simple IF-ENDIF.

It is important to note that the statement sequence within a REPEAT-UNTIL will always be executed at least once regardless of the evaluation of the condition. This is because the evaluation is done after the first execution of the statement sequence.

WHILE–ENDWHILE

This construction is very similar to the REPEAT-UNTIL except that the test as to whether to execute the statement sequence is done before its execution. This means that although in the REPEAT-UNTIL case the statement sequence is always done at least once (since the test is not done until after the first execution), in the WHILE-ENDWHILE case the statement sequence might not be done at all, depending upon the first evaluation of the condition. Also note that the statement sequence is executed as long as the condition is true which is the opposite of the REPEAT-UNTIL which executes its statement sequence as long as the condition is false. The general form is as follows:

```
WHILE (condition) :
  statement sequence .
ENDWHILE
```

As with the REPEAT-UNTIL care must be taken to avoid endless loops caused by the condition never evaluating to false.

FOR–ENDFOR

This construction is very similar to the FOR statement in BASIC or the DO statement in FORTRAN. It is used when a section of a program is to be repeated a certain number of times and always at least once. Its general form is as follows:

```
FOR i = n TO m STEP j :
  statement sequence .
ENDFOR
```

As in most constructions of this type the *i* is the index variable, *n* is the initial value assigned to the index variable, *m* is the final test value which is used to determine when the statement sequence should stop being executed, and *j* is the increment to be added to the variable after each execution of the statement sequence.

For example, to execute some sequence of statements exactly ten times, the following could be done:


```
FOR I = 1 TO 10 :
  statement sequence .
ENDFOR
```

Note that the step is optional and when not specified is assumed to be 1.

DECLARATIVE

A declarative is any statement which specifies some operation (other than decision or looping) to be executed. Sample declaratives would be as follows:

```
PRINT THE VALUE OF X ;
DISPLAY THE CURVE ON THE DISPLAY ;
X=A+B*SIN(R)/LOG(D-C) ;
CALL THE INPUT ROUTINE ;
INCREMENT THE VALUE OF W BY 1 ;
```

In particular, note that each declarative is terminated by a semi-colon (;), just as a colon (:) is used to terminate conditions in the IF, ORIF, FOR, UNTIL, and WHILE constructions. This becomes helpful when the declarative continues on to more than one line (card). Also notice that calls to subroutines are specified in declarative statements.

The use of the word *DECLARATIVE* is in the English grammar sense of a command to do something. It applies equally well to both data specification statements and to executable statements.

RETURN

The RETURN statement is used at the end of a subroutine to denote, quite obviously, its end. The general (and only) form is:

```
RETURN//
```

The double slash is included as part of the return statement simply to make it easily recognizable in a program.

COMMENT

The COMMENT statement is used to describe what is being done functionally by successive statements. It is used to clarify what is being done by some section of code so that the reader does not have to read through all of the successive statements to determine what is going on. The general form is as follows:

< (comment)

The left angle bracket denotes that a comment follows or can be used to leave a line blank to enhance readability. Note that the word *COMMENT* is not part of the statement. Here is an example:

```
<
<WRITE OUT ALL OF THE VALUES IN THE DATA
<ARRAY WHICH ARE POSITIVE
<
FOR I = 1 TO THE NUMBER OF ELEMENTS IN THE
ARRAY :
 IF THE ITH ELEMENT IS POSITIVE :
  WRITE IT OUT .
 ENDIF
ENDFOR
```

STATEMENT SEQUENCE

A statement sequence is simply a set of properly implemented structured programming constructions, including any of the decision and looping constructs, declaratives, comments, or additional statement sequences. For examples of all of these constructs see the STAR SHIP logic definitions in chapter 3.

GENERAL RULES OF STRUCTURED PROGRAMMING

1. Every IF must have an ENDIF.
2. Every REPEAT must have an UNTIL.
3. Every FOR must have an ENDFOR.
4. Every WHILE must have an ENDWHILE.
5. Indentation rules:

   a. Indent after each IF.
   b. Indent after each ORIF.
   c. Indent after each ELSE.
   d. Indent after each REPEAT.
   e. Indent after each FOR.
   f. Indent after each WHILE.
6. Every (condition) must be terminated with a colon (:).
7. Every declarative of a statement sequence is terminated with a semi-colon (;) except for the last declarative of a statement sequence which is terminated with a period (.).
8. Every subroutine is terminated with a RETURN//.
9. Every subroutine has one entry point and one exit point. (This is a direct consequence of the application of the structured programming conventions.)

Remember that there is no set programming language associated with structured program design. Indeed, the statements look more like comments than a distinct set of programming language elements. Structured programming is more of an aid to programming and as such is much more flexible in its applications. As long as the statements are clear and there is no ambiguity as to their meanings then they are acceptable. As the programmer continues to develop his code he will add more and more concise statements until the point is reached where all executable statements are elements of the programming language he is using.

3 The Star Ship Simulation Project

In the original published version of this STAR SHIP simulation program which appeared in the August, September, and October 1977 issues of *Interface Age* magazine, I stated that "while this description is extensive it is not exhaustive." That situation is now changed. The version presented in this chapter leaves little left to be done, short of actually implementing it for a particular hardware/software setup. Details are given for every major function aboard a star ship as well as the intelligence of enemy and friendly space craft, and celestial objects. What follows, then, is an example of the principles discussed in the previous two chapters.

SYSTEM OBJECTIVE

A. To develop a software system which completely simulates the hardware associated with the navigation, communications, helm, medical, engineering, and sciences functions of a Constitution class Starship as defined on the television program STAR TREK and in the documents *Star Fleet Technical Manual* and *Star Trek Blueprints*, both published by Franz Joseph Designs, 1975.[*]

B. To simulate the actions (intelligence and tactical maneuvers) of a number of enemy and Federation space craft within a simulated three-dimensional universe.

C. To develop the software in a flexible manner so that it may be implemented either for a multi-operator mode in which

[*] These books are available at bookstores, nationwide.

several people operate the system with each one assuming the role of a particular officer on the bridge including captain, navigator, communications officer, engineer, etc. and each one having his own display and input terminal, or for a single operator-single terminal implementation.

D. To provide a simple means by which specific tactical situations may be defined both dynamically during the running of the simulation or offline for future recall.

MAJOR FUNCTION IDENTIFICATION

The major functions for this simulation are as follows:
1. Simulation controller
2. Communications
3. Navigation
4. Sciences
5. Engineering
6. Medical
7. Helm

Each of these major functions comprise one main module of the system.

MAJOR FUNCTIONS OBJECTIVES

Simulation Controller

The simulation controller function is the module which controls the operation of all of the remaining modules. During the running of the simulation there is one person who has the role of controller. This is not an Enterprise officer position and has no counterpart on the television show STAR TREK except, perhaps, in the role of director or producer. The controller dictates what will happen during the running of the simulation. He has complete control over all of the variables which reside in common memory (to be described later). He can create enemy and Federation craft, as well as celestial objects. He specifies the task of the Enterprise. He can cause malfunctions to any of the ship's hardware such as computers, life-support systems, warp engines, and so on. He has the ability to set into action a predefined set of events (a scenario) for the officers. In effect, the controller runs the show.

The scenario, as mentioned above, and its associated module, the SIMULATION CONTROLLER, make it possible for a given sequence of events to be run more than once, allowing the operators to try different methods of handling the given tactical situations. This is different from the average STAR TREK computer game which has no well-defined sequence of events. They are usually controlled by operator actions and certain pseudo-random variables. In the simulation, on the other hand, the simulation controller has access to all of the variables throughout all of the modules and thus controls the tactical situations presented to the operators.

In its simplest form (and the form in which it is used in this simulation) a scenario is a set of events and the times at which they are to occur. A sample scenario is shown in figure 3.1.

This scenario is, in effect, a set of instructions to load certain values into the variables in the common data area. It may be thought of as a program, prepared offline, that runs concurrently (or parallel) with the simulation. For instance, to cause damage to the life support system on the Enterprise the scenario might specify to change the variable called FB from whatever its current status is to zero, indicating severe damage. The module which monitors the variable, in this case the Engineering module, would soon recognize the change in status and report it, via an appropriate display, to the operator.

The scenario is considered to be input to the SIMULATION CONTROLLER. The actual form of the scenario might be a paper tape, a cassette, a magtape, diskette, or other type of storage which can be read in during the execution of the simulation. The specific hardware/software installation will determine the form and whether it is read one line (record) at a time or all at once. The function of the SIMULATION CONTROLLER is to execute the commands in the scenario at the appropriate times. This eliminates the need to write an entirely new simulation each time a different tactical situation is to be presented to the operators. The scenario, then, actually defines the mission of the Enterprise. By writing an appropriate scenario just about any of the tactical situations presented on the STAR TREK television program may be simulated. Indeed, the only limit to the kinds of missions that this system can simulate is the user's imagination.

An expanded implementation might allow conditional statements within the scenario providing greater control over the

TIME	STATEMENT	COMMENT
0000	CV=2001	SET DESTINATION OF THE ENTERPRISE TO ENEMY CRAFT NUMBER 1
0010	HW(1)=1	SET MISSION OF ENEMY CRAFT NUMBER 1 TO UNCONDITIONAL ATTACK
0025	B7(2)=32	SET RELIABILITY OF THE ENTERPRISE'S PHASER STATION NUMBER 2 TO POOR
0030	HO(1)=4000	SET THE DESTINATION OF ENEMY CRAFT 1 TO THE ENTERPRISE
0032	DP(3)=4	MAKE SHUTTLECRAFT NUMBER 3 SEEK THE ENTERPRISE SHUTTLE BAY
0100	F2=45763	SET THE ENTERPRISE TOTAL ENERGY SUPPLY
0103	AH(2)=2	MAKE CELESTIAL OBJECT NUMBER 2 INTO A BLACK HOLE
0123	CH=0	MAKE THE SPACE/WARP ENGINES INOPERATIVE (SHOULD REDUCE THE ENTERPRISE'S ABILITY TO OUTRUN AN ENEMY CRAFT)
0245	DC=100	SET THE PATIENT CAPACITY OF THE MEDICAL SECTION
0256	EO=65	SET THE ENERGY REQUIREMENT OF THE TRANSPORTER (THIS WILL CAUSE IT TO USE UP MORE ENERGY EACH TIME IT IS USED)
0270	BS(42)=3	SEND PERSONNEL NUMBER 42 TO THE BRIG
0301	EV=52	SET THE FOOD NUTRITION LEVEL TO POOR (THIS WILL CAUSE ILLNESS TO SOME OF THE CREW)

Figure. 3.1 SAMPLE SCENARIO

The scenario defines the tactical situations presented to the operators. It is a dynamic type of definition and is, in effect, a program which runs concurrently with the simulation, creating enemy and Federation space craft, causing time-warps or system malfunctions, inflicting damage to planetary civilizations, and so on.

FUNCTION	COMMAND FORMAT
Initialize All modules	I A
Intialize Engineering module	I E
Initialize Communications module	I C
Initialize Navigation module	I N
Initialize Sciences module	I S
Initialize Medical module	I M
Initialize Helm module	I H
Execute All modules	E A
Execute Communications module	E C
Execute Navigation module	E N
Execute Sciences module	E S
Execute Medical module	E M
Execute Helm module	E H
Halt All modules	H A
Halt Engineering module	H E
Halt Communications module	H C
Halt Navigation module	H N
Halt Sciences module	H S
Halt Medical module	H M
Halt Helm module	H H
Display Module variables	D M module name
Display Variable	D V variable name
Assign Value to variable	A V value, variable name
Create Enemy	C E enemy name, mission, x, y, z
Create Federation craft	C F craft name, mission, x, y, z
Create Celestial object	C C object name, x, y, z, radius, mass, radiation type
Input a scenario	I
Run the scenario	R
Halt the scenario	H

Figure 3.2 SIMULATION CONTROLLER COMMANDS

events and situations. Since the STAR SHIP simulation is completely modularized, adding this feature should be quite easy.

Besides providing the ability to run scenarios the simulation controller is a powerful debugging device. It allows the operator to initialize and run specific modules of the system and to watch the changing values of the common data variables as the modules are being executed. It also allows each module to be initialized to some predefined state.

A suggested set of simulation controller commands are given in figure 3.2. The actual format for the commands will, of course, depend upon the specific software implementations. The conventions established here, however, should serve as a good example.

Communications

The communications module is responsible for maintaining the status of intra-ship as well as inter-space communications and the location of personnel. It receives and displays messages coming from the various stations throughout the ship and from other craft and planets. It also enables the Enterprise to send and receive messages (e.g., distress calls) and initiate the transfer of personnel and cargo from one place to another.

This last function is of particular importance since it is through the communications officer that maintenance personnel are sent to repair damaged hardware, medical personnel are dispatched to the intensive care unit, search parties are sent to the shuttlecraft, food supplies are sent to planetary civilizations, and so on.

The valid communications commands are shown in figure 3.4.

When a person or cargo is given a destination figure 3.6 is consulted. If the destination is within the same turbo/elevator service area as the person's or cargo's current location, then the time required to effect the transfer is one minute and will be handled completely by the communications module.

If the destination is in a turbo/elevator service area different from the person's or cargo's current location then the person or cargo will be placed at the turbo/elevator station of the service area in which the cargo or person currently resides. This is done by the communications module. The engineering module then takes over by sending the person or cargo, via the turbo/elevator system, to the T/E station which services the area for the person's or cargo's destination.

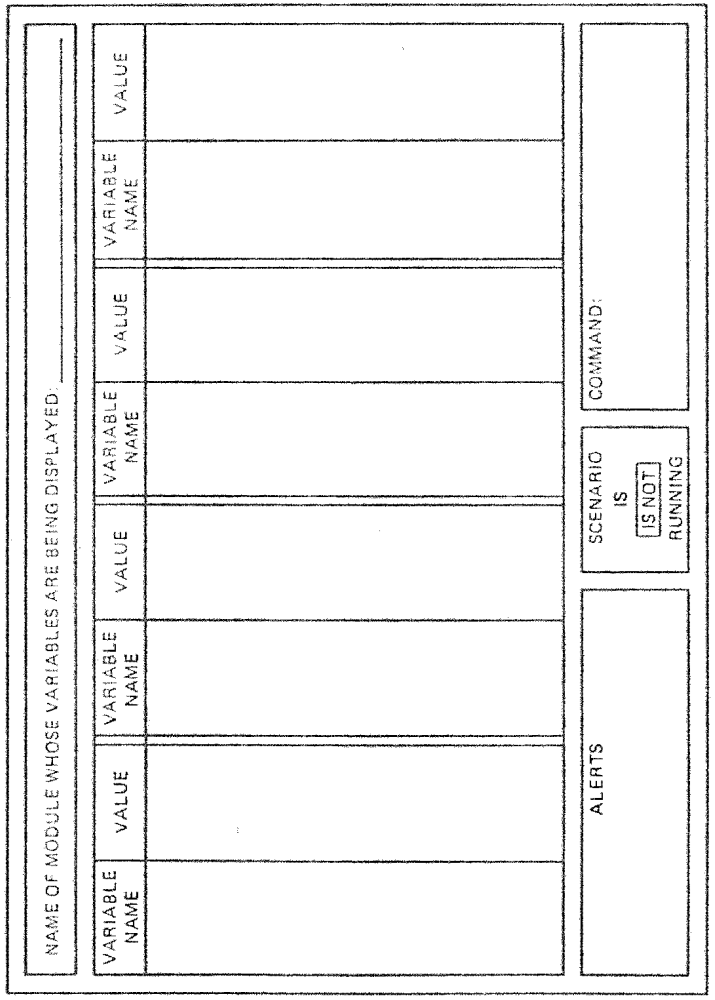


Figure 3.3 SIMULATION CONTROLLER DISPLAY: PRELIMINARY FUNCTIONAL DESIGN

Once the person or cargo is at the T/E station which services his destination the communications module takes over and sends him or it to the proper destination.

If a person or cargo is given a destination outside of the Enterprise (such as a shuttlecraft already on a mission, a planet, another Federation craft) when its current location is within the Enterprise then it is first sent to one of the transporters or one of the shuttlecraft, depending upon the distance to the destination. When the person or cargo arrives at the transporter or shuttlecraft bay the engineering module takes over to complete the transfer.

All of this occurs as the result of one command by the communications officer to transfer some person or cargo. After the command, everything is taken care of automatically by the simulation software.

FUNCTION	COMMAND FORMAT
Display next Message	D M
Send Distress message	S D
Request Personnel movement	R P personnel name, destination
Request Cargo movement	R C cargo type, destination
Request Aid	R A federation craft name
Send Peace Offer to enemy craft	S P O craft name
Accept Peace Offer from enemy craft	A P O craft name
Reject Peace Offer from enemy craft	R P O craft name

Figure 3.4 COMMUNICATIONS MODULE COMMANDS

The communications officer can send several types of messages to Federation and enemy craft and can receive messages from any number of positions aboard the Enterprise as well as from other craft and planets. The preliminary design for the communications display shows that a convention has been established for receiving messages. The communications officer may display only one incoming message at a time. The messages are kept in a list (a first-in-first-out queue). If the list gets filled then subsequent messages will be lost. The officer displays the next message on the list by executing the D M command which effectively removes the next message from the list and displays it and makes room for another message on the list.

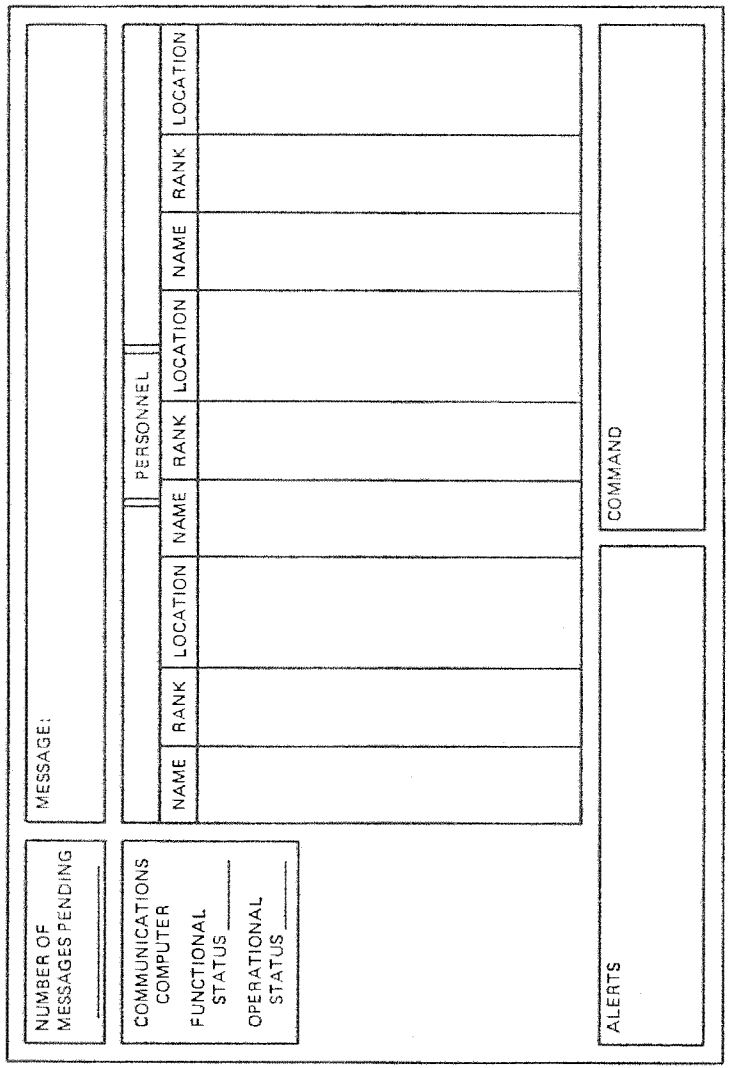


Figure 3.5 COMMUNICATIONS DISPLAY: PRELIMINARY FUNCTIONAL DESIGN

TURBO ELEVATOR STATION NUMBER (SERVICE AREA)	AREAS SERVED
1	BRIDGE SENSOR STATIONS TRANSPORTER #1 PHASER STATION #1
2	SCIENCES LABORATORY TRANSPORTER #2 TORPEDO TUBE #1 DEFLECTOR SHIELD STATION #1
3	ENGINEERING TRANSPORTER #3 PHASER STATION #2
4	NAVIGATION COMPUTER TRANSPORTER #4 TORPEDO TUBE #2 DEFLECTOR SHIELD STATION #2
5	MEDICAL RESEARCH LABORATORY MEDICAL COMPUTER INTENSIVE CARE UNIT TRANSPORTER #5 PHASER STATION #3
6	TURBO/ELEVATOR COMPUTER FOOD PROCESSING PLANT OXYGEN DISTRIBUTION AND RECYCLING CENTER WATER DISTRIBUTION AND RECYCLING CENTER TRANSPORTER #6 TORPEDO TUBE #3 DEFLECTOR SHIELD STATION #3
7	TRACTOR BEAM TRANSPORTER #7 PHASER STATION #4
8	ENERGY SUPPLY TRANSPORTER #8 TORPEDO TUBE #4 DEFLECTOR SHIELD STATION #4
9	TRANSPORTER #9 PHASER STATION #5 DEFLECTOR SHIELD STATION #5
10	TRANSPORTER #10 SHUTTLE BAY (ALL SHUTTLECRAFT) PHASER STATION #6 DEFLECTOR SHIELD STATION #6

Figure 3.6 AREAS SERVED BY TURBO/ELEVATORS

When a person or cargo is given a destination the communications and engineering modules effect the transfers. The time for transfer within a T/E service area is one minute. The time for transfer between T/E service areas is the difference in area numbers times one-half minute.

Obviously, the officer must respond quickly so he does not lose any messages. While that may sound like a poor way to handle incoming messages it is certainly comparable to real-life situations.

PERSONNEL	YELLOW ALERT	GREEN ALERT	RED ALERT
SCIENCE	10% SCIENCE LAB 90% CREW'S QUARTERS	50% SCIENCE LAB 50% CREW'S QUARTERS	100% SCIENCE LAB
ENGINEERING	10% ENGINEERING 90% CREW'S QUARTERS	50% ENGINEERING 50% CREW'S QUARTERS	100% ENGINEERING
MEDICAL	20% MEDICAL LAB 20% INTENSIVE CARE 60% CREW'S QUARTERS	40% MEDICAL LAB 40% INTENSIVE CARE 20% CREW'S QUARTERS	50% MEDICAL LAB 50% INTENSIVE CARE
CHIEF MEDICAL OFFICER	INTENSIVE CARE	INTENSIVE CARE	INTENSIVE CARE
SECURITY	20% SECURITY 80% CREW'S QUARTERS	50% SECURITY 50% CREW'S QUARTERS	100% SECURITY
MAINTENANCE	20% EVENLY DISTRIBUTED	50% EVENLY DISTRIBUTED	100% EVENLY DISTRIBUTED AMONG AREAS REQUIRING REPAIR
GENERAL CREW	30% EVENLY DISTRIBUTED 70% CREW'S QUARTERS	50% EVENLY DISTRIBUTED 50% CREW'S QUARTERS	90% EVENLY DISTRIBUTED 10% CREW'S QUARTERS

Figure 3.7 PERSONNEL ASSIGNED LOCATIONS

Each personnel has an assigned location associated with the current alert status. For normal alert each personnel remains where he is assigned by the communications officer. For the other alerts they are distributed as specified in this chart.

Navigation

The navigation module provides the facilities for laying in courses for point-to-point travel. It calculates, utilizing the navigation *computer*, courses for establishing orbits around

planets or other celestial objects. It provides a display of the projected universe and the local sector including the ship's current position and course as well as the course of enemy and friendly craft. It maintains the ship's chronometer (real-time clock). It simulates the intelligence and tactical maneuvers of enemy and Federation space craft and the Enterprise shuttlecraft. It simulates the natural physical interactions of celestial objects (gravity, etc.). Unlike most STAR TREK games where the enemy craft sit in one spot and fire only when the Enterprise makes a move, the enemy craft in the simulation roam throughout the universe, some looking to attack Federation space ships, others trying to conquer civilizations, deliver goods or weapons, or seeking peace treaties.

There are Federation craft as well. They, too, are intelligent. Indeed, there can be battles waged between enemy and Federation craft at one end of the universe as the Enterprise traverses the other end completely unaware of the encounter. A war may be won without the Enterprise even knowing about it.

FUNCTION	COMMAND FORMAT
Set Course to Coordinates	S C C x, y, z
Set Course to specified Object	S C O object name
Evasive Tactics	E T
Set Velocity	S V velocity value
Set Direction	S D x/y angle x/z angle
Establish Orbit	E O

Figure 3.8 NAVIGATION COMMANDS

Sciences

The sciences module monitors the functional status of the life support systems, including food, air, and water supply. It also monitors the life forms, gravity, radiation, functional status, and fire power of planets and other space craft via the sensors on board the Enterprise.

The science officer can request sensor data of celestial objects or general sensor scan of areas of space. The type of data displayed is determined by the object being scanned. The amount and accuracy of the displayed data is related to the functional status of the sensors, the science computer, the distance to the

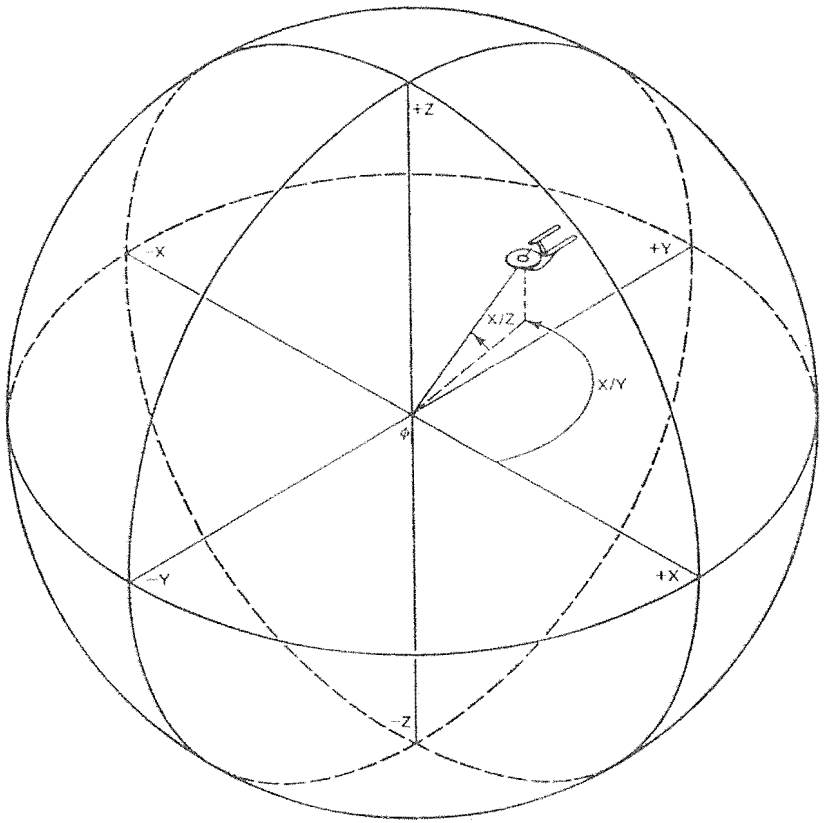


Figure 3.9 THREE-DIMENSIONAL COORDINATE SYSTEM

Unique positions within the universe are specified by a set of x, y, and z coordinates, each value being within its defined minimum and maximum limits (see the common data table). It is assumed that each unit of value represents 100 meters. This means that the smallest resolution within the universe is 100 meters.

To specify a direction of the Enterprise within the universe, as in setting the course with the S D command, two angles must be specified. The angles assume that a local coordinate system exists with its 0, 0, 0 point at the center of the Enterprise and its positive x-axis directed straight ahead. The x/y angle denotes the positive angle measured from the positive x-axis towards the positive y-axis. The x/z angle denotes the positive angle measured from the positive x/y plane towards the positive z-axis.

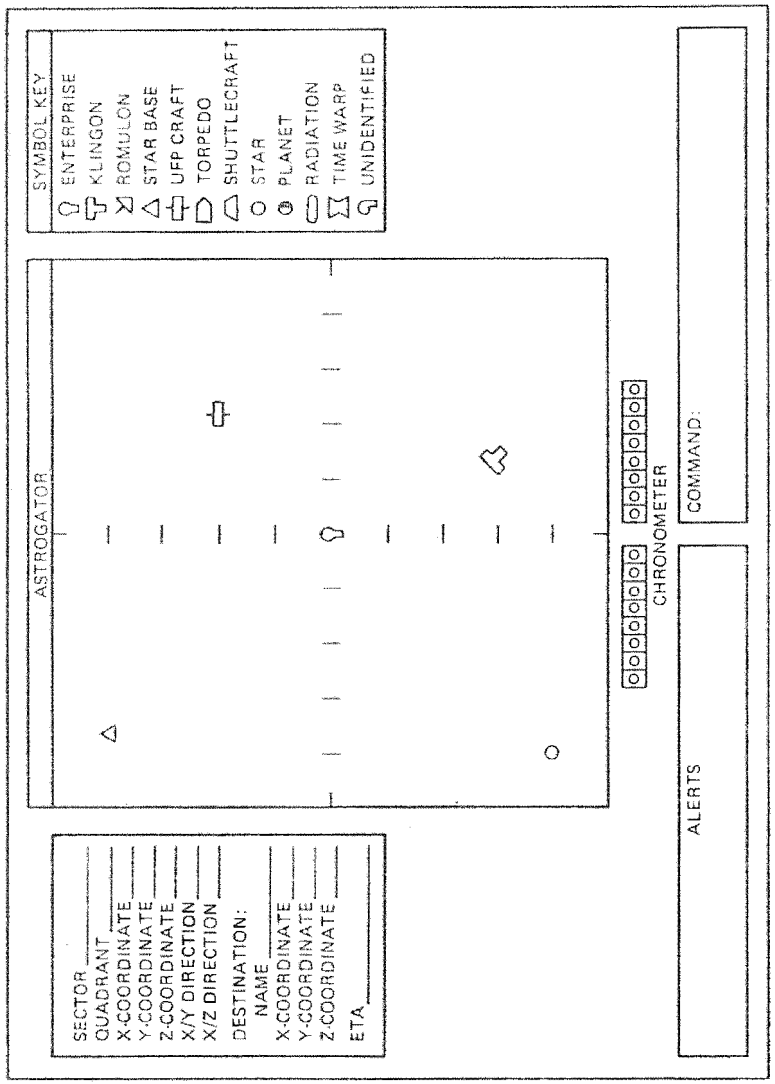


Figure 3.10 NAVIGATION DISPLAY: PRELIMINARY FUNCTIONAL DESIGN

object, and the presence of sensor-data-seeking shuttlecraft (i.e. if such a shuttlecraft is near the object then the sensor data quality is proportionally increased).

FUNCTION	COMMAND FORMAT
Scan Specified Object	S S O object name
Scan Area	S A x, y, z
Scan Randomly	S R

Figure 3.11 SCIENCES COMMANDS

Engineering

The engineering module maintains the status of the shuttlecraft, transporters, energy supply, space/warp and impulse engines, main craft structural damage, turbo/elevators, etc. It allows the engineer to specify the distribution of energy to the various sections of the ship.

Each of the shuttlecraft has an assigned mission, initially set to *none*. The valid mission types are DELIVER CARGO, SEEK SENSOR DATA, TRANSPORT PERSONNEL, and SEEK THE ENTERPRISE. By means of the communications module personnel may be sent to the shuttlecraft and cargo may be loaded onto them, provided, of course, that the specified shuttlecraft is in the shuttle bay. Once a shuttlecraft is assigned a mission, it departs from the shuttle bay, attempts to perform its mission, and then returns to the Enterprise.

For a DELIVER CARGO mission the craft will travel to the specified destination, normally another Federation craft or an inhabited planet, transfer the cargo, and then return to the Enterprise.

If the craft is seeking sensor data then it will travel to the specified destination and send back data to the Enterprise. This is accomplished by increasing the functional status of the Enterprise's sensors as the shuttle nears its destination.

The TRANSPORT PERSONNEL mission is similar to the transport of cargo except that several personnel may be aboard the shuttle, each with a different destination. In this case the personnel will each be delivered to his respective destination before the shuttle returns to the Enterprise.

SEEK THE ENTERPRISE will cause the specified shuttle to return to the shuttle bay.

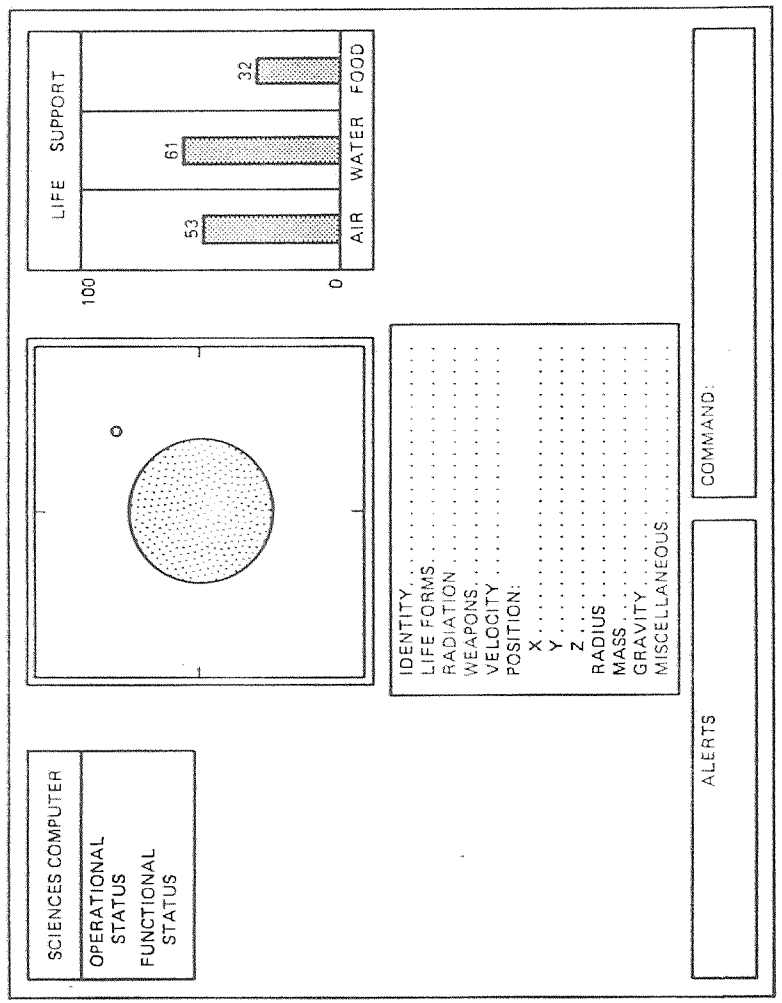


Figure 3.12 SCIENCES DISPLAY: PRELIMINARY FUNCTIONAL DESIGN

Whenever a command is given to one of the shuttlecraft, whether or not it receives the command is dependent upon the functional status of the Enterprise's communications systems and the distance to the specified shuttle.

Another function of the engineering module is to allow energy to be distributed to the various devices and sections of the ship. Initially every device receives an equal share. However, in situations where the total energy supply is low and the Enterprise is in a combat situation the captain might decide to divert more energy to the life-support systems, deflector shields, and defensive weapons. Increasing the energy supply for one device proportionally decreases the supply to all other devices.

Setting an energy level to a ship section sets all of the energy supply levels for the associated devices. See figure 3.13.

Note that in general the more energy available to a device the more powerful it is and the quicker it will be repaired when damaged (although this last function is also related to other attributes such as the number of maintenance personnel at the device location).

The last command of the engineering module is to display information about the various devices on board the Enterprise. This is accomplished with the REPORT STATUS OF DEVICE command.

The engineering task of maintaining the transporters involves not only keeping them in working order but also recognizing when there are personnel or cargo to be transported. When such a situation exists, as a result of the communications officer sending someone or something to the transporter or to a location outside of the Enterprise but within transport distance, the engineering module does the actual transport. Of course, the success of the transport depends upon the functional status of the transporter. It is possible to be injured or killed, or cargo to be destroyed by a malfunctioning transporter. Particularly note that if explosive cargo is being transported the resultant explosion can also damage the transporter.

FUNCTION	COMMAND FORMAT
Report Status of device	R S device code
Set Shuttlecraft mission	S S shuttlename, mission, destination
set Energy supply to Device	E D device code

DEVICE CODE	MEANING	SHIP SECTION
T	torpedo tube	A
P	phaser station	A
D	deflector shield	A
W	space warp engines	B
I E	impulse engines	B
N C	navigation computer	B
R	research laboratory	C
I C	intensive care unit	C
M C	medical computer	C
T C	T/E computer	D
T	transporter	D
T B	tractor beam	D
O D	oxygen distribution	E
O R	oxygen recycling	E
W D	water distribution	E
W R	water recycling	E
C C	communication computer	F
S	security	G
R S	radiation sensor	H
G S	gravitation sensor	H
L S	life forms sensor	H
A S	atmospheric sensor	H

Figure 3.13 ENGINEERING COMMANDS

Medical Module

The medical module maintains the health status of all of the crew members. It keeps track of the number and status of its patients, the patient capacity of the medical stations, the medical personnel, the functional status of its intensive care units and other hardware.

While there is no operator input to the medical module, and hence no need for a medical officer, there is a display of the medical data. The only operator input to this module is indirectly through the communications officer. When crew members (not the actual operators) are found to be sick (diminishing health status) the communications officer sends them a message

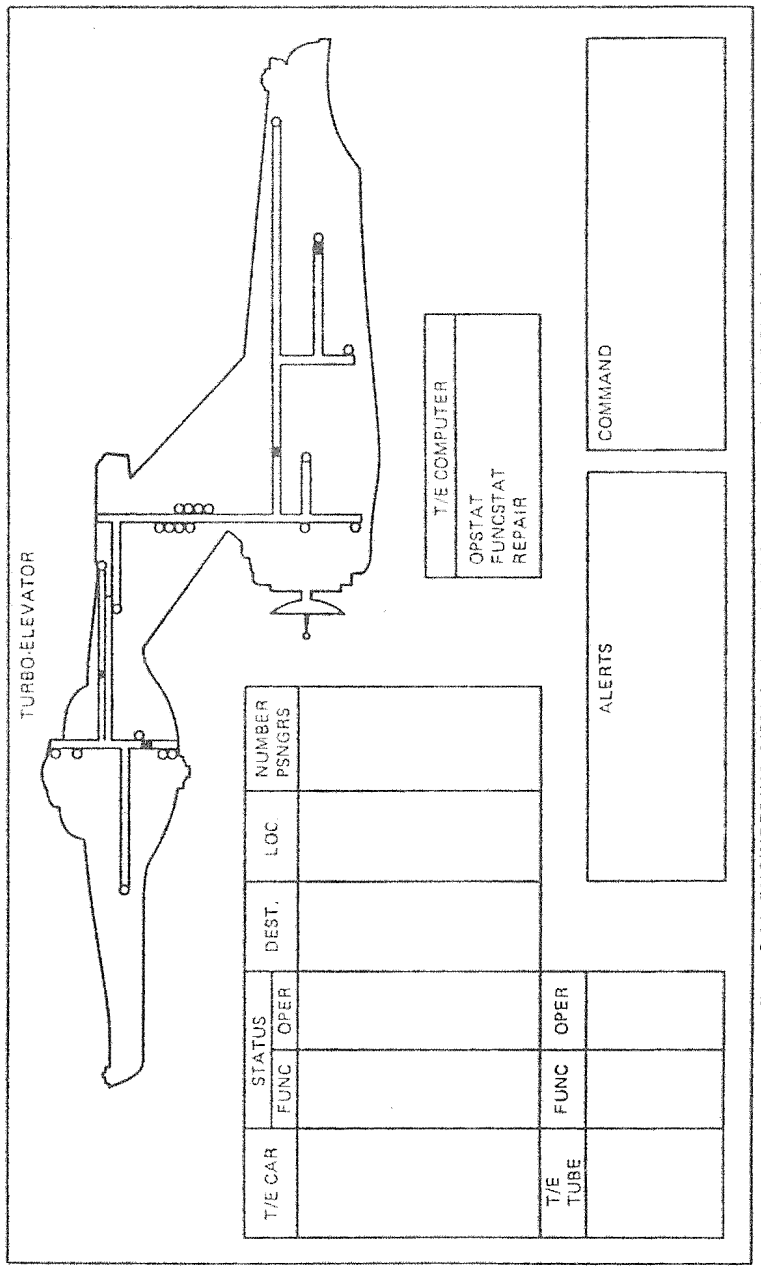
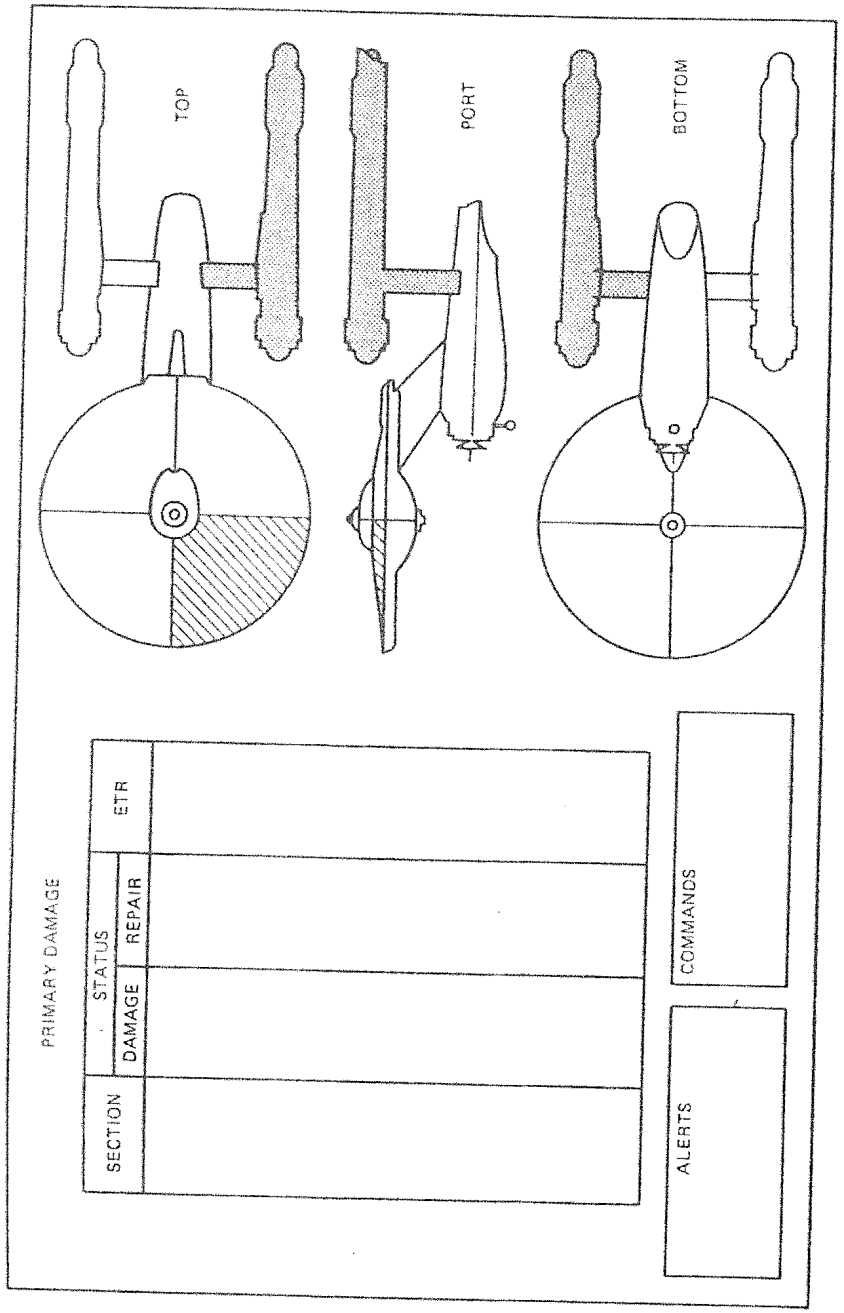


Figure 3.14 ENGINEERING DISPLAYS: PRELIMINARY FUNCTIONAL DESIGNS



to proceed to the intensive care unit. While they are there their health condition will improve (or worsen) depending upon the functional and operational status of the intensive care unit, the medical computer, and medical research laboratory, and the available energy. When, through the application of expert medical care, a patient recovers, he is sent to his assigned station (see figure 3.7).

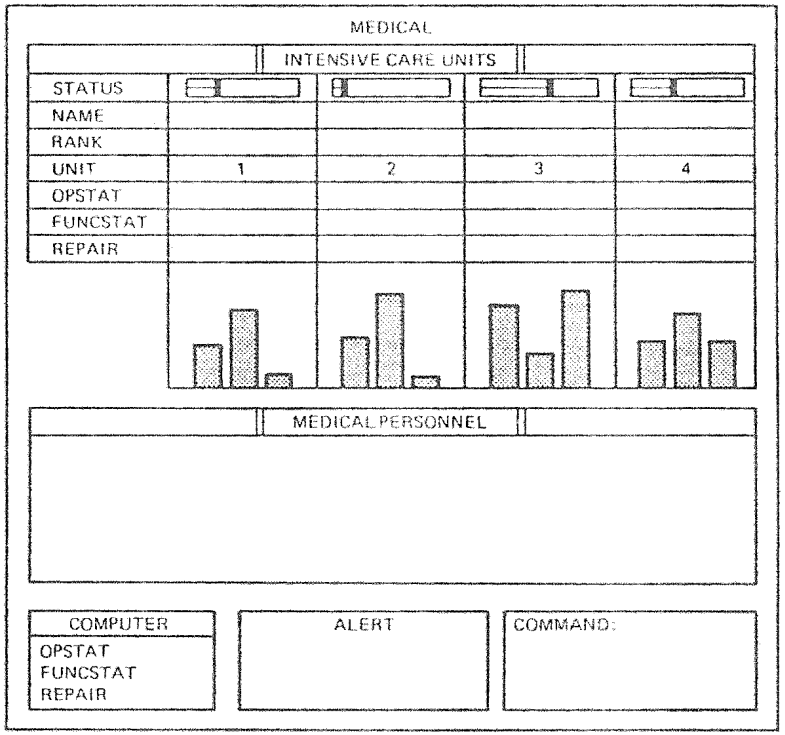


Figure 3.15 MEDICAL DISPLAY: PRELIMINARY FUNCTIONAL DESIGN

Helm

The helm module maintains the status of and provides the means of operating the offensive and defensive weapons.

For the phasers the spatial relationship between the Enterprise and the target is first determined by the module. This

means that it looks to see if the target is fore, aft, above, below, port, or starboard in relation to the Enterprise. Then it checks to see whether the corresponding phaser bank is operational. If not, then the helmsman gets notified. If it is, then the weapon is fired and the appropriate damage (based on distance to the target, status of the target's deflector shields, etc.) is registered to the target.

What this means is that the phasers have limited direction capability and if only the forward banks are functioning it will not be possible to fire at a target following the Enterprise. The Enterprise would first have to turn completely around.

The photon torpedo, on the other hand, will track a target. That is, it may be fired from any of the six torpedo tubes, will *look for* the target, and will zero in on it even if the target changes course. It (the torpedo) is intelligent enough to know not to hit the Enterprise.

For both the phasers and the photon torpedos the specified target must be within sensor range and not have a cloaking device in operation. Otherwise the weapon will not fire.

Note that there is a limited number of torpedos. However, delivery may be received from another Federation craft or by docking with a star base.

There are six deflector shields on the Enterprise, one each for fore, aft, top, bottom, port, and starboard, and their operational levels may be individually set.

FUNCTION	COMMAND FORMAT
Fire Phaser at named Object	F P O object name
Fire Phaser at Coordinates	F P C x, y, z
Fire photon Torpedo at named Object	F T O object name
Fire photon Torpedo at Coordinates	F T C x, y, z
Set Deflector shield level	S D shield location, level

Figure 3.16 HELM COMMANDS

MAJOR FUNCTIONS INTERFACE

Each module in this simulation has access to a common data area. It is assumed that all modules reside in main memory and are executed sequentially under control of the Simulation Con-

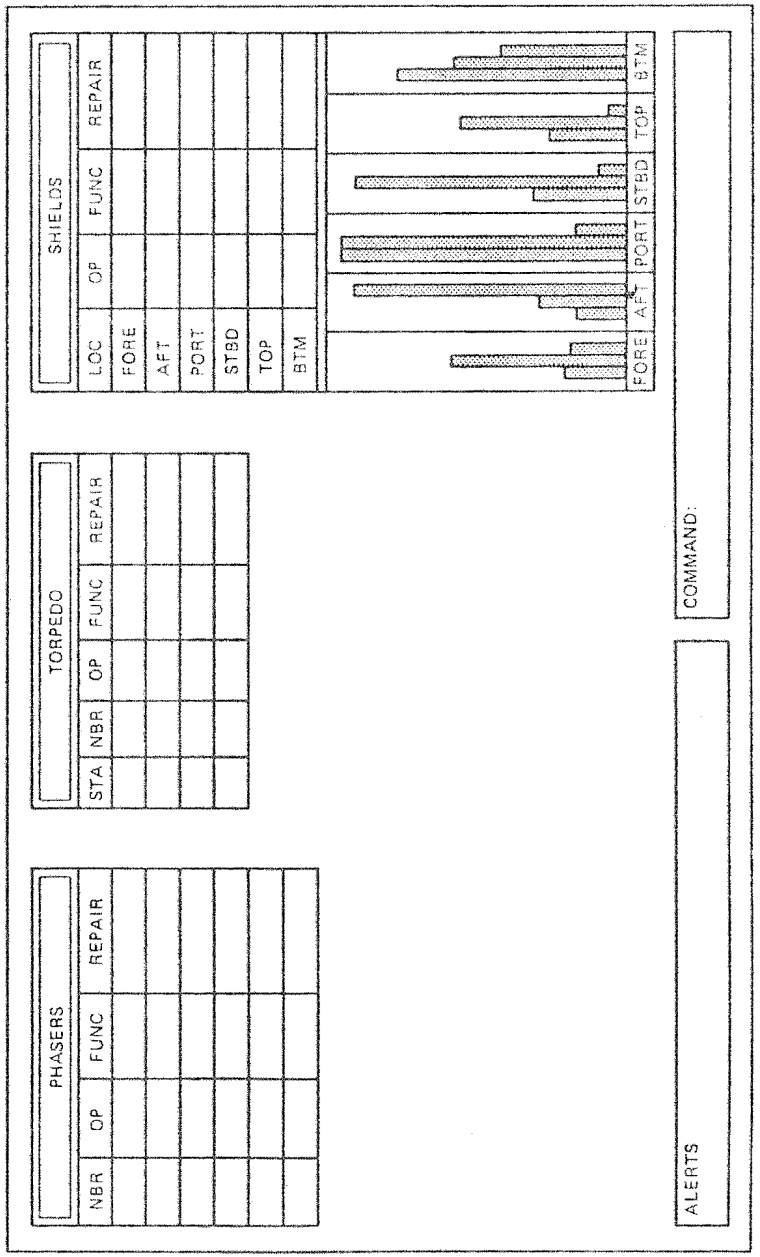


Figure 3.17 HELM DISPLAY: PRELIMINARY FUNCTIONAL DESIGN

troller although other methods are possible and discussed in the chapter on implementation. A portion of main memory is reserved as a common data area and all modules have read/write access to it. What follows is a list of the variables in this common data area and a description of how to read the table.

Text enclosed in angle brackets (<) define the major classification of the variables that follow. Text followed by a colon (:) indicates a subclassification. Each actual variable is preceeded by a two-character variable name. Data identifications followed by a set of parentheses () indicate an array variable. A number within the parenetheses specifies the dimension of the array. Where no number appears within the parentheses the array size is unspecified (or random) indicating that the actual dimension is determined by the programmer or imposed by the limits of the computer memory. Commas within parentheses indicate additional dimensions (e.g. (,) means a two dimensional array with each dimension unspecified). Text preceeded by a square bracket ([) denotes information about the variable such as the numerical range or units.

For example, under ENTERPRISE COMMUNICATIONS DATA the line MESSAGE STACK(10,100) indicates an array with ten messages of one hundred characters each. The SHUTTLECRAFT PROPULSION TUBES RELIABILITY FACTOR (6,2) is a two dimensioned array. Each of the six shuttlecraft has two propulsion tubes. So DT(1,1) holds the reliability factor of tube number one on shuttlecraft number one, DT(1,2) holds the reliability factor of tube two craft one, etc.

In some cases the subscripts are given as variable names. Under CELESTIAL DATA the number of celestial objects is given as the variable named AG. Subsequently, the CLASSIFICATION has a subscript of AG. This indicates that the dimension of the classification variable is the same as the number of celestial objects, and that classification(1) is for object number 1, CLASSIFICATION(2) is for object 2, and so on.

COMMON DATA AREA

<CELESTIAL DATA>

UNIVERSE SIZE

AA X-MAXIMUM	[DETERMINED BY THE NUMBER OF
AB X-MIN	[DIGITS OF ACCURACY FOR THE
AC Y-MAX	[PARTICULAR IMPLEMENTATION OF
AD Y-MIN	[THIS SIMULATION. ONE UNIT =
AE Z-MAX	[100 METERS
AF Z-MIN	

CELESTIAL OBJECTS

AG NUMBER	[RANDOM-DETERMINED BY
	[COMPUTER MEMORY CAPACITY
AH CLASSIFICATION(AG)	[1 = STAR
	[2 = BLACK HOLE
	[3 = PARTICLE CLOUD
	[4 = TIME/WARP
	[5 = PLANET
	[6 = MOON
	[7 = NOVA
AI NAME(AG,10)	[ALPHABETIC-10 CHARACTERS
	[PER NAME
JV CHARTED(AG)	[0 = NOT CHARTED
	[1 = CHARTED

LOCATION

AJ X-COORDINATE(AG)	[- INFINITY TO + INFINITY
AK Y-COORDINATE(AG)	
AL Z-COORDINATE(AG)	

VELOCITY VECTOR

AM X/Y DIRECTION(AG)	
AN X/Z DIRECTION(AG)	
AO SPEED(AG)	[0 - .99 WARP
AP RADIUS(AG)	[METERS

RADIATION

AQ TYPE(AG)	[0 = NONE
	[1 = LIGHT
	[2 = RADIOACTIVE
AR INTENSITY(AG)	[0 - 1000
AS MASS(AG)	[KILOTONS

LIFE FORMS

AT QUANTITY(AG)	[0 - INFINITE
AU CLASSIFICATION(AG)	[0 = NONE
	[1 = HUMANOID
	[2 = VEGETATION
	[3 = AQUATIC
	[4 = OBJECT ITSELF IS INTELLIGENT
AV INTELLIGENCE QUOTIENT(AG)	[0 - 300

DEFENSIVE WEAPONS

AW	NUMBER(AG)	[0 OR 1 (MAY BE EXPANDED)
AX	TYPE(1,AW)	[0 = NONE
		[1 = SHIELD SCREEN
AY	FUNCTIONAL STATUS(1,AW)	[0 – 100%
AZ	OPERATIONAL STATUS(1,AW)	[REL TYPE
A1	RELIABILITY FACTOR(1,AW)	[0 – 100%
A2	ENERGY REQUIREMENT(1,AW)	[UNITS PER UNIT–TIME

OFFENSIVE WEAPONS

A3	NUMBER(AG)	[0 – 3
A4	TYPE(3,AG)	[0 = NONE
		[1 = PHASER
		[2 = PHOTON TORPEDO
		[3 = COMMUNICATIONS DISRUPTER
		[4 = ULTIMATE DESTRUCT
A5	FUNCTIONAL STATUS(3,AG)	[0 – 100%
A6	OPERATIONAL STATUS(3,AG)	[REL TYPE
A7	RELIABILITY FACTOR(3,AG)	[0 – 100%
A8	ENERGY REQUIREMENT(3,AG)	[UNITS PER UNIT–TIME
A9	FIRED UPON FLAG(AG)	[0 = NOT FIRED UPON
		[1 = FIRED UPON
BA	PEACE TREATY OFFER(AG)	[0 = OFFERED
		[NON–ZERO = CODE OF CRAFT THAT
		[OFFERED
BB	PEACE TREATY REQUEST	[0 = NOT REQUESTED
		[1 = REQUESTED

ROMULON EMPIRE
LOCATION

BC	X–COORDINATE	[–INFINITY TO + INFINITY
BD	Y–COORDINATE	[–INFINITY TO + INFINITY
BE	Z–COORDINATE	[–INFINITY TO + INFINITY
BF	RADIUS	[0 – INFINITY

KLINGON EMPIRE
LOCATION

BG	X–COORDINATE	[–INFINITY TO + INFINITY
BH	Y–COORDINATE	[–INFINITY TO + INFINITY
BI	Z–COORDINATE	[–INFINITY TO + INFINITY
BJ	RADIUS	[0 – INFINITY

<DATA ASSOCIATED WITH THE ENTERPRISE>

<ENTERPRISE PERSONNEL DATA>

PERSONNEL

BK	NUMBER	[43 OFFICERS, 387 CREW ENSIGN
		[GRADE
BL	NAME(BK,10)	[RANDOM NAMES, ALPHABETIC
		[10 CHARACTERS PER NAME
BM	RANK(BK)	[0 = NONEXISTENT
		[1 = SCIENCE OFFICER
		[2 = ENGINEERING OFFICER
		[3 = MEDICAL OFFICER
		[4 = CHIEF MEDICAL OFFICER
		[5 = SECURITY OFFICER
		[6 = MAINTENANCE CREW
		[7 = GENERAL CREW
BN	INTELLIGENCE(BK)	[0 – 300 SEE INTELLIGENCE CHART
BO	LOCATION CODE(BK)	[–1 = LOCATION IS SPECIFIED BY
		[THE X,Y,Z COORDINATES
		[0 = BRIDGE
		[1 = SCIENCES LABORATORY
		[2 = ENGINEERING
		[3 = BRIG (PRISON)
		[4 = SECURITY
		[5 = NAVIGATION COMPUTER
		[6 = MEDICAL RESEARCH
		[LABORATORY
		[7 = MEDICAL COMPUTER
		[8 = TURBO–ELEVATOR COMPUTER
		[9 = TRACTOR BEAM
		[10 = FOOD PROCESSING PLANT
		[11 = OXYGEN DISTRIBUTION AND
		[RECYCLING
		[12 = WATER DISTRIBUTION AND
		[RECYCLING
		[13 = ENERGY SUPPLY
		[14 = INTENSIVE CARE UNIT
		[15 = SENSOR STATIONS (ALL 4
		[SENSOR TYPES)
		[16 = CREW'S QUARTERS
		[17 = SHUTTLEBAY
		[20 – 29 = TRANSPORTER
		[STATION 0 – 9
		[30 – 49 = TURBO–ELEVATOR
		[STATION 0 – 19
		[50 – 69 = TURBO–ELEVATOR 0 – 19
		[70 – 79 = SHUTTLECRAFT 0 – 9
		[80 – 83 = PHOTON TORPEDO TUBE
		[STATION 0 – 3
		[90 – 95 = PHASER STATION 0 – 5
		[PORT, STARBOARD, TOP, BOTTOM,
		[FORE, AND AFT
		[100 – 105 = DEFLECTOR SHIELD
		[STATION 0 – 5 PORT, STARBOARD,
		[TOP, BOTTOM, FORE, AND AFT
		[1000 – 1NNN = CELESTIAL OBJECT
		[0 – NNN
		[2000 – 2NNN = ENEMY CRAFT NNN
		[3000 – 3NNN = FEDERATION
		[NNN
		[4000 = ENTERPRISE
BP	X–COORDINATE	
BQ	Y–COORDINATE	
BR	Z–COORDINATE	
BS	DESTINATION(Y)	[SAME CODES AS LOCATION

BT FUNCTIONAL STATUS(Y) | 0 – 100%, 0 = DEAD
BU HEALTH STATUS(Y) | 0 – 10, 0 = DEAD
BV FOOD CONSUMPTION | KG/HOUR AVERAGE PER INDIVIDUAL
BW WATER CONSUMPTION | LITERS/HOUR AVERAGE PER INDIVIDUAL
BX OXYGEN CONSUMPTION | LITERS/HOUR AVERAGE PER INDIVIDUAL

<ENTERPRISE WEAPONS DATA>

<ENTERPRISE OFFENSIVE WEAPONS>

PHOTON TORPEDO TUBES | 6 STATIONS, PORT, STARBOARD, TOP, BOTTOM, FORE, AND AFT
BY FUNCTIONAL STATUS(6) | 0 – 100%
BZ RELIABILITY FACTOR(6) | 0 – 100%
B1 ENERGY REQUIREMENT
B2 NUMBER OF PHOTON TORPEDOS(6) | 0 – 20 PER STATION
B3 LOCATION OF TORPEDOS(6,20) | –1 = NO LOCATION(NON–EXISTENT) SEE PERSONNEL LOCATION CODES
B4 DESTINATION(6,20) | SAME AS ABOVE
PHASER STATIONS | 6 STATIONS, PORT, STARBOARD, TOP, BOTTOM, FORE, AND AFT
B5 FUNCTIONAL STATUS(6) | 0 – 100%
B6 OPERATIONAL STATUS(6) | 0 – 100%
B7 RELIABILITY FACTOR(6) | 0 – 100%
B8 ENERGY REQUIREMENT | UNITS PER UNIT–TIME

<ENTERPRISE DEFENSIVE WEAPONS>

DEFLECTOR SHIELDS | FORE, AFT, PORT, STARBOARD, TOP, BOTTOM
B9 FUNCTIONAL STATUS(6) | 0 – 100%
CA OPERATIONAL STATUS(6) | 0 – 100%
CB RELIABILITY FACTOR(6) | 0 – 100%
CD ENERGY REQUIREMENT | UNITS PER UNIT–TIME;

CLOAKING DEVICE

CE FUNCTIONAL STATUS | 0 – 100%
CF OPERATIONAL STATUS | 0 – 100%
CG RELIABILITY FACTOR | 0 – 100%

<ENTERPRISE PROPULSION DATA>

SPACE/WARP ENGINES
CH FUNCTIONAL STATUS(2) | 0 – 100%
CI OPERATIONAL STATUS(2) | 1 – 20 WARP
CJ RELIABILITY FACTOR(2) | 0 – 100%
CK ENERGY REQUIREMENT | UNITS PER UNIT–TIME
IMPULSE ENGINES | 4 ENGINES

CL FUNCTIONAL STATUS(4) | 0 – 100%
CM OPERATIONAL STATUS(4) | 0 – .99 WARP
CN RELIABILITY FACTOR(4) | 0 – 100%
CO ENERGY REQUIREMENT | UNITS PER UNIT–TIME;

<ENTERPRISE NAVIGATION DATA>

LOCATION

CP X–COORDINATE | –INFINITY TO + INFINITY
CQ Y–COORDINATE | –INFINITY TO + INFINITY
CR Z–COORDINATE | –INFINITY TO + INFINITY

VELOCITY VECTOR

CS X/Y DIRECTION | 0 – 360 DEGREES
CT X/Z DIRECTION | 0 – 360 DEGREES
CU SPEED | 0 – INFINITE WARP
CV DESTINATION CODE | SUBSET OF PERSONNEL LOCATION CODES. INCLUDES CELESTIAL OBJECTS, ENEMY AND FEDERATION CRAFT, SHUTTLECRAFT, AND X,Y,Z COORDINATES. 0 = NO DESTINATION (DEAD STOP) –1 = X,Y,Z COORDINATE

DESTINATION

CW X–COORDINATE | –INFINITY TO + INFINITY
CX Y–COORDINATE | –INFINITY TO + INFINITY
CY Z–COORDINATE | –INFINITY TO + INFINITY

NAVIGATION COMPUTER

CZ FUNCTIONAL STATUS | 0 – 100%
C1 OPERATIONAL STATUS | 0 – 100%
C2 RELIABILITY FACTOR | 0 – 100%
C3 ENERGY REQUIREMENT | UNITS PER UNIT–TIME

<ENTERPRISE MEDICAL SECTION DATA>

MEDICAL RESEARCH LAB

C4 FUNCTIONAL STATUS | 0 – 100%
C5 OPERATIONAL STATUS | 0 – 100%
C6 RELIABILITY FACTOR | 0 – 100%
C7 ENERGY REQUIREMENT | UNITS PER UNIT–TIME

INTENSIVE CARE UNIT

C8 FUNCTIONAL STATUS | 0 – 100%
C9 OPERATIONAL STATUS | 0 – 100%
DA RELIABILITY FACTOR | 0 – 100%
DB ENERGY REQUIREMENT | UNITS PER UNIT–TIME;
DC PATIENT CAPACITY | (TBD)

MEDICAL COMPUTER

DD FUNCTIONAL STATUS | 0 – 100%
DE OPERATIONAL STATUS | 0 – 100%
DF RELIABILITY FACTOR | 0 – 100%

```
DG  ENERGY REQUIREMENT          | UNITS PER UNIT-TIME
                                | AND NUMBER OF PATIENTS
  MEDICAL SUPPLY
DH  QUANTITY                    | UNITS
```

<ENTERPRISE SHUTTLECRAFT DATA>

```
DI  OPERATIONAL STATUS(6)       | 0 = IN SHUTTLE BAY
                                | 1 = ON MISSION
  LOCATION
DJ  X-COORDINATE(6)             | -INFINITY TO + INFINITY
DK  Y-COORDINATE(6)             | -INFINITY TO + INFINITY
DL  Z-COORDINATE(6)             | -INFINITY TO + INFINITY
  VELOCITY VECTOR
DM  X/Y DIRECTION(6)            | 0 - 360 DEGREES
DN  X/Z DIRECTION(6)            | 0 - 360 DEGREES
DO  SPEED(6)                    | KILOMETERS PER SECOND
DP  MISSION(6)                  | 0 = NONE
                                | 1 = SEEK SENSOR DATA
                                | 2 = DELIVER CARGO
                                | 3 = TRANSPORT PERSONNEL
                                | 4 = SEEK SHUTTLE BAY
DQ  DESTINATION(6)              | SAME AS ENTERPRISE LOCATION
                                | CODES
  SHUTTLECRAFT PROPULSION       | 2 TUBES FOR EACH OF 6
    TUBES                       | SHUTTLECRAFT
DR  FUNCTIONAL STATUS(6,2)      | 0 - 100%
DS  OPERATIONAL STATUS(6,2)     | 0 - 100%
DT  RELIABILITY FACTOR(6,2)     | 0 - 100%
DU  ENERGY REQUIREMENT          | UNITS PER UNIT-TIME;
DV  CARGO(6)
  SHUTTLECRAFT SENSOR ARRAY
DW  FUNCTIONAL STATUS(6)        | 0 - 100%
DX  OPERATIONAL STATUS(6)       | 0 - 100%
DY  RELIABILITY FACTOR(6)       | 0 - 100%
DZ  ENERGY REQUIREMENT          | UNITS PER UNIT-TIME;
  SHUTTLECRAFT DEFENSIVE WEAPONS
  PHASER                        | ONE PHASER PER SHUTTLECRAFT
D1  FUNCTIONAL STATUS(6)        | 0 - 100%
D2  OPERATIONAL STATUS(6)       | 0 - 100%
D3  RELIABILITY(6)              | 0 - 100%
D4  ENERGY REQUIREMENT          | UNITS PER UNIT-TIME
  SHUTTLECRAFT OFFENSIVE WEAPONS
  DEFLECTOR SHIELDS             | ONE SHIELD PER CRAFT
D5  FUNCTIONAL STATUS(6,2)      | 0 - 100%
D6  OPERATIONAL STATUS(6,2)     | 0 - 100%
D7  RELIABILITY FACTOR(6,2)     | 0 - 100%
D8  ENERGY REQUIREMENT          | UNITS PER UNIT-TIME;
```


<ENTERPRISE INTRA-SHIP TRANSPORTATION DATA>

```
  TURBO-ELEVATOR STATIONS       | 10 STATIONS
D9  FUNCTIONAL STATUS(10)       | 0 - 100%
EA  OPERATIONAL STATUS(10)      | 0 - 100%
EB  RELIABILITY FACTOR(10)      | 0 - 100%
  TURBO-ELEVATOR CARS           | 6 TURBO-ELEVATOR CARS
EC  FUNCTIONAL STATUS(6)        | 0 - 100%
ED  LOCATION(6)                 | SEE PERSONNEL LOCATION CODES
                                | 0 THROUGH 15
EF  DESTINATION(6)              | SAME AS ABOVE
EG  ARRIVAL TIME(6)             | UNITS
  TURBO-ELEVATOR COMPUTER
EH  FUNCTIONAL STATUS           | 0 - 100%
EI  OPERATIONAL STATUS          | 0 - 100%
EJ  RELIABILITY FACTOR          | 0 - 100%
EK  ENERGY REQUIREMENT          | UNITS PER PASSENGER
```

<ENTERPRISE TRANSPORTER DATA>

```
  STATIONS                      | 10 STATIONS
EL  FUNCTIONAL STATUS(10)       | 0 - 100%
EM  OPERATIONAL STATUS(10)      | 0 - 100%
EN  RELIABILITY FACTOR(10)      | 0 - 100%
EO  ENERGY REQUIREMENT          | UNITS PER TRANSPORT; REL
                                | DISTANCE
```

<ENTERPRISE TRACTOR BEAM DATA>

```
TRACTOR BEAM
EP  FUNCTIONAL STATUS           | 0 - 100%
EQ  OPERATIONAL STATUS          | 0 - 100%
ER  RELIABILITY FACTOR          | 0 - 100%
ES  INDENTITY OF OBJECT BEING   | SEE PERSONNEL LOCATION CODES
    PULLED                      | FOR CELESTIAL OBJECTS, ENEMY
                                | AND FEDERATION CRAFT, AND
                                | SHUTTLECRAFT.
                                | 0 = NO OBJECT .
ET  ENERGY REQUIREMENT          | UNITS PER UNIT-TIME; REL OBJECT
                                | MASS, AND VELOCITY VECTOR,
                                | AND OPERATIONAL STATUS OF
                                | OBJECT'S ENGINES IF APPLICABLE
```

<ENTERPRISE LIFE SUPPORT SYSTEMS DATA>

```
  FOOD SUPPLY
EU  QUANTITY                    | 0 TO 100000 KILOGRAMS
EV  NUTRITION LEVEL             | 0 - 100%
EW  MAXIMUM QUANTITY            | UNITS
EX  POLLUTION LEVEL             | 0 - 100%
```

FOOD RECYCLE SYSTEM

EY	MAXIMUM CAPACITY	[FOOD UNITS
EZ	FUNCTIONAL STATUS	[0 – 100%
E1	OPERATIONAL STATUS	[0 – 100%
E2	RELIABILITY FACTOR	[0 – 100%
E3	ENERGY REQUIREMENT	[0 – 100%

OXYGEN

E4	QUANTITY	[0 – 1 BILLION CUBIC FEET
E5	MAXIMUM QUANTITY	[UNITS, CUBIC FEET
E6	POLLUTION LEVEL	[0 – 100%

OXYGEN DISTRIBUTION SYSTEM

E7	FUNCTIONAL STATUS	[0 – 100%
E8	OPERATIONAL STATUS	[0 – 100%
E9	RELIABILITY FACTOR	[0 – 100%
FA	ENERGY REQUIREMENT	[UNITS PER UNIT–TIME;

OXYGEN RECYCLE SYSTEM

FB	FUNCTIONAL STATUS	[0 – 100%
FC	OPERATIONAL STATUS	[0 – 100%
FD	RELIABILITY FACTOR	[0 – 100%
FE	ENERGY REQUIREMENT	[UNITS PER UNIT–TIME

WATER

FF	QUANTITY	[0 – 1 MILLION KILOLITERS
FG	MAXIMUM QUANITTY	[KILOLITERS
FH	POLLUTION LEVEL	[0 – 100%

WATER DISTRIBUTION SYSTEM

FI	FUNCTIONAL STATUS	[0 – 100%
FJ	OPERATIONAL STATUS	[0 – 100%
FK	RELIABILITY FACTOR	[0 – 100%
FL	ENERGY REQUIREMENT	[UNITS PER UNIT–TIME

WATER RECYCLE SYSTEM

FM	FUNCTIONAL STATUS	[0 – 100%
FN	OPERATIONAL STATUS	[0 – 100%
FO	RELIABILITY FACTOR	[0 – 100%
FP	ENERGY REQUIREMENT	[UNITS PER UNIT–TIME

<ENTERPRISE COMMUNICATION DATA>

<INTRA– AND INTER–SHIP COMMUNICATIONS DATA>

	MESSAGES	[ONE PER COMMUNICATIONS [STATION
FQ	MESSAGE COUNT	[0 – 10
FR	MESSAGE STACK(10,100)	[ALPHA–NUMERICS, MAXIMUM [OF 10 100 CHARACTER [MESSAGES

<INTER–CELESTIAL COMMUNICATIONS DATA>

ENTERPRISE COMMUNICATIONS COMPUTER

FS	FUNCTIONAL STATUS	[0 –100%
FT	OPERATIONAL STATUS	[0 – 100%
FU	RELIABILITY FACTOR	[0 – 100%
FV	ENERGY REQUIREMENT	[UNITS PER UNIT–TIME

<ENTERPRISE SECURITY DATA>

	DETENTION CELL	[ONE DETENTION CELL
FW	FUNCTIONAL STATUS	[0 – 100%
FX	OPERATIONAL STATUS	[0 – 100%
FY	RELIABILITY FACTOR	[0 – 100%
FZ	ENERGY REQUIREMENTS	[UNITS PER PRISONER
F1	MAXIMUM PRISONER CAPACITY	[30

<ENTERPRISE ENERGY SUPPLY DATA>

ENERGY

F2	QUANTITY	[0 – MAXIMUM
F3	MAXIMUM QUANTITY	[20 BILLION UNITS

ENERGY SUPPLY INTERCONNECT SYSTEM 22 STATIONS

	CONNECTION STATION CODES	[1 = PHOTON TORPEDO TUBES [2 = PHASER STATIONS [3 = DEFLECTOR SHIELDS [4 = SPACE/WARP ENGINES [5 = IMPULSE ENGINES [6 = NAVIGATION COMPUTER [7 = RESEARCH LAB [8 = INTENSIVE CARE UNIT [9 = MEDICAL COMPUTER [10 = TURBO–ELEVATOR COMPUTER [11 = TRANSPORTERS [12 = TRACTOR BEAM [13 = OXYGEN DISTRIBUTION SYSTEM [14 = OXYGEN RECYCLE SYSTEM [15 = WATER DISTRIBUTION SYSTEM [16 = WATER RECYCLE SYSTEM [17 = COMMUNICATIONS COMPUTER [18 = SECURITY [19 = RADIATION SENSOR [20 = GRAVITY SENSOR [21 = LIFE FORMS SENSOR [22 = ATMOSPHERIC SENSOR
F4	FUNCTIONAL STATUS(22)	[0 – 100%
F5	OPERATIONAL STATUS(22)	[0 – 100%
F6	RELIABILITY FACTOR(22)	[0 – 100%
F7	ENERGY SUPPLY(22)	[UNITS OF ENERGY AVAILABLE TO [THE SPECIFIED DEVICE

<ENTERPRISE SENSOR ARRAY DATA>

RADIATION SENSOR

```
F8  FUNCTIONAL STATUS              [ 0 - 100%
F9  OPERATIONAL STATUS             [ 0 - 100%
GA  RELIABILITY FACTOR             [ 0 - 100%
GB  ENERGY REQUIREMENT             [ UNITS PER UNIT-TIME;
  GRAVITY SENSOR
GC  FUNCTIONAL STATUS              [ 0 - 100%
GD  OPERATIONAL STATUS             [ 0 - 100%
GE  RELIABILITY FACTOR             [ 0 - 100%
GF  ENERGY REQUIREMENT             [ UNITS PER UNIT-TIME;
  LIFE FORMS SENSOR
GH  FUNCTIONAL STATUS              [ 0 - 100%
GI  OPERATIONAL STATUS             [ 0 - 100%
GJ  RELIABILITY FACTOR             [ 0 - 100%
GK  ENERGY REQUIREMENT             [ UNITS PER UNIT-TIME;
  ATMOSPHERIC SENSORS
GL  FUNCTIONAL STATUS              [ 0 - 100%
GM  OPERATIONAL STATUS             [ 0 - 100%
GN  RELIABILITY FACTOR             [ 0 - 100%
GO  ENERGY REQUIREMENT             [ UNITS PER UNIT-TIME
```

<ENTERPRISE CARGO DATA>

```
GP  CARGO()                        [ 0 = FOOD
                                   [ 1 = WATER
                                   [ 2 = OXYGEN
                                   [ 3 = ENERGY
                                   [ 4 = BOMB
GQ  QUANTITY()
GR  LOCATION()                     [ SEE PERSONNEL LOCATION
                                   [ CODES
```

<ENTERPRISE GENERAL DATA>

```
G3  ALERT STATUS                   [ 0 = NORMAL
                                   [ 1 = YELLOW
                                   [ 2 = GREEN
                                   [ 3 = RED
```

<DATA ASSOCIATED WITH ENEMY SHIPS>

```
GT  NUMBER OF ENEMY SHIPS          [ 0 - 1000
GU  NAME(1000,10)                  [ ALPHANUMERIC, 10 CHARACTERS
                                   [ EACH
GV  EXISTENCE(1000)                [ 0 = DESTROYED
                                   [ 1 = EXISTS
```

<DEFENSIVE WEAPONS DATA>

```
  DEFENSIVE WEAPONS
GW  NUMBER(1000)                   [ 0 - 2
GX  TYPE(1000,2)                   [ 1 = CLOAKING DEVICE
```



```
                                   [ 2 = DEFLECTOR SHIELDS
GY  FUNCTIONAL STATUS(1000,2)      [ 0 - 100%
GZ  OPERATIONAL STATUS(1000,2)     [ 0 - 100%
G1  RELIABILITY FACTOR(1000,2)     [ 0 - 100%
G2  ENERGY REQUIREMENT(1000,2)     [ REL TYPE
  OFFENSIVE WEAPONS
G3  NUMBER(1000)                   [ 0 - 2
G4  TYPE(1000,2)                   [ 1 = PHASERS
                                   [ 2 = PHOTON TORPEDOS
G5  FUNCTIONAL STATUS(1000,2)      [ 0 - 100%
G6  OPERATIONAL STATUS(1000,2)     [ 0 - 100%
G7  RELIABILITY FACTOR(1000,2)     [ 0 - 100%
G8  ENERGY REQUIREMENT(1000,2)     [ REL TYPE
```

<ENEMY SHIPS LIFE FORMS DATA>

```
HA  TYPE OF LIFE FORM(1000)        [ 0 = NONE
                                   [ 1 = HUMANOID
                                   [ 2 = VEGETATION
                                   [ 3 = AQUATIC
                                   [ 4 = INTELLIGENT SHIP (NO LIFE
                                   [ BUT SELF-SUFFICIENT CRAFT, A
                                   [ ROBOT SHIP)
HB  NUMBER(1000)                   [ 0 - INFINITE
HC  INTELLIGENCE LEVEL(1000)       [ 0 - 300 AVERAGE INTELLIGENCE
HD  FUNCTIONAL STATUS(1000)        [ 0 - 100%
HE  OPERATIONAL STATUS(1000)       [ 0 - 100%
HF  RELIABILITY FACTOR(1000)       [ 0 - 100%
HG  HEALTH STATUS(1000)
```

<ENEMY SHIPS NAVIGATION DATA>

```
  LOCATION
HH  X-COORDINATE(1000)             [ -INFINITY TO + INFINITY
HI  Y-COORDINATE(1000)             [ -INFINITY TO + INFINITY
HJ  Z-COORDINATE(1000)             [ - INFINITY TO + INFINITY
  VELOCITY VECTOR
HK  X/Y DIRECTION(1000)            [ 0 - 360 DEGREES
HL  X/Z DIRECTION(1000)            [ 0 - 360 DEGREES
HM  SPEED(1000)                    [ 0 - 20 WARP
HN  MAXIMUM SPEED
      CAPABILITY(1000)             [ 0 - 20 WARP
HO  DESTINATION(1000)              [ SEE ENTERPRISE PERSONNEL
                                   [ LOCATION CODES FOR CELESTIAL
                                   [ OBJECTS, ENEMY AND FEDERATION
                                   [ CRAFT, AND SHUTTLECRAFT
                                   [ -1 = DESTINATION IS SPECIFIED BY
                                   [      THE X,Y, AND Z COORDINATES
HP  X-COORDINATE(1000)             [ -INFINITY TO + INFINITY
HQ  Y-COORDINATE(1000)             [ -INFINITY TO + INFINITY
HR  Z-COORDINATE(1000)             [ -INFINITY TO + INFINITY
```

```
 NAVIGATION COMPUTER
HS  FUNCTIONAL STATUS(1000)      [ 0 - 100%
HT  OPERATIONAL STATUS(1000)     [ 0 - 100%
HU  RELIABILITY FACTOR(1000)     [ 0 - 100%
HV  ENERGY REQUIREMENT(1000)     [ UNITS PER UNIT-TIME;
HW  MISSION(1000)                [ 0 = NO MISSION, SHIP
                                 [       NON-EXISTENT
                                 [ 1 = CONDITIONAL ATTACK
                                 [ 2 = UNCONDITIONAL ATTACK
                                 [ 3 = ESTABLISH PEACE TREATY
                                 [ 4 = SEARCH AND CONQUER
                                 [       CIVILIZATION
                                 [ 5 = WEAPONS DELIVERY
                                 [ 6 = PEACEFUL CARGO DELIVERY

 CARGO
HX  CARGO(1000)                  [ SEE ENTERPRISE CARGO CODES
HY  QUANTITY(1000)               [ UNITS
HZ  LOCATION(1000)               [ SEE ENTERPRISE PERSONNEL
                                 [ LOCATION CODES
H1  DESTINATION(1000)            [ SAME AS ABOVE
                                 [ -1 = DESTINATION SPECIFIED IN
                                 [      THE FOLLOWING X,Y,Z
                                 [      COORDINATES
H2  X-COORDINATE (1000)          [ -INFINITY TO + INFINITY
H3  Y-COORDINATE (1000)          [ -INFINITY TO + INFINITY
H4  Z-COORDINATE (1000)          [ -INFINITY TO + INFINITY
H5  PEACE TREATY OFFERED         [ 0 = NOT OFFERED BY OTHER CRAFT
      FLAG(1000)                 [ NON-ZERO = PEACE TREATY
                                 [ OFFERED BY OTHER CRAFT. THE
                                 [ CODE IN THIS VARIABLE IS THE
                                 [ CODE OF THE CRAFT OFFERING
                                 [ THE PEACE TREATY. SEE PERSON-
                                 [ NEL LOCATION CODES 1000 TO 4000.
H6  PEACE TREATY REQUEST         [ 1 = REQUESTED BY THIS CRAFT
      FLAG(1000)                 [ 0 = NOT REQUESTED
```

<ENEMY SHIP POWER SUPPLY>

```
 ENERGY
H7  QUANTITY(1000)               [ 0 - 10↑10 UNITS
H8  FIRED UPON FLAG(1000)        [ 0 = NOT FIRED UPON
                                 [ NON-ZERO = CODE OF WHO FIRED.
                                 [ SEE ENTERPRISE PERSONNEL
                                 [ LOCATION CODES 1000 - 3000
```

<DATA ASSOCIATED WITH FEDERATION SHIPS>

```
H9  NUMBER OF FEDERATION SHIPS   [ 0 - 1000
IA  NAME(1000,10)                [ ALPHANUMERIC, 10 CHARACTERS
                                 [ EACH
IB  EXISTENCE(1000)              [ 0 = DESTROYED
                                 [ 1 = EXISTS
```


<DEFENSIVE WEAPONS DATA>

```
 DEFENSIVE WEAPONS
IC  NUMBER(1000)
ID  TYPE(1000,2)                 [ 1 = DEFLECTOR SHIELDS
                                 [ 2 = CLOAKING DEVICE
IE  FUNCTIONAL STATUS(1000,2)    [ 0 - 100%
IF  OPERATIONAL STATUS(1000,2)   [ 0 - 100%
IG  RELIABILITY FACTOR(1000,2)   [ 0 - 100%
IH  ENERGY REQUIREMENT(1000,2)   [ REL TYPE
 OFFENSIVE WEAPONS
IJ  NUMBER(1000)                 [ 0 - 2
IK  TYPE(1000,2)                 [ 1 = PHASERS
                                 [ 2 = PHOTON TORPEDO
IL  FUNCTIONAL STATUS(1000,2)    [ 0 - 100%
IM  OPERATIONAL STATUS(1000,2)   [ 0 - 100%
IN  RELIABILITY FACTOR(1000,2)   [ 0 - 100%
IO  ENERGY REQUIREMENT(1000,2)   [ 0 - 100%
```

<FEDERATION SHIPS LIFE FORMS DATA>

```
IP  NUMBER(1000)                 [ 0 TO 10000 PERSONS
IQ  INTELLIGENCE(1000)           [ AVERAGE
IS  OPERATIONAL STATUS(1000)     [ 0 - 100%
IT  RELIABILITY FACTOR(1000)     [ 0 - 100%
IU  HEALTH STATUS(1000)          [ 0 - 100%
```

<FEDERATION SHIPS NAVIGATION DATA>

```
 LOCATION
IV  X-COORDINATE(1000)           [ -INFINITY TO + INFINITY
IW  Y-COORDINATE(1000)           [ -INFINITY TO + INFINITY
IX  Z-COORDINATE(1000)           [ -INFINITY TO + INFINITY
 VELOCITY VECTOR
IY  X/Y DIRECTION(1000)          [ 0 - 360 DEGREES
IZ  X/Z DIRECTION(1000)          [ 0 - 360 DEGREES
I1  SPEED(1000)                  [ 0 - 20 WARP
I2  MAXIMUM SPEED(1000)          [ 0 - 20 WARP
I3  DESTINATION CODE(1000)       [ SEE ENTERPRISE PERSONNEL
                                 [ LOCATION CODES FOR
                                 [ CELESTIAL OBJECTS, ENEMY AND
                                 [ FEDERATION CRAFT, AND
                                 [ SHUTTLECRAFT;
                                 [ -1 = DESTINATION SPECIFIED IN
                                 [      THE FOLLOWING X,Y,Z
I4  X-COORDINATE(1000)           [ -INFINITY TO + INFINITY
I5  Y-COORDINATE(1000)           [ -INFINITY TO + INFINITY
I6  Z-COORDINATE(1000)           [ -INFINITY TO + INFINITY
 NAVIGATION COMPUTER
I7  FUNCTIONAL STATUS(1000)      [ 0 - 100%
I8  OPERATIONAL STATUS(1000)     [ 0 - 100%
```

```
I9  RELIABILITY FACTOR(1000)        | 0 - 100%
JA  ENERGY REQUIREMENT(1000)        | UNITS PER UNIT-TIME;
JB  MISSION(1000)                   | 0 = NO MISSION, SHIP
                                    |     NON-EXISTENT
                                    | 1 = CONDITIONAL ATTACK
                                    | 2 = UNCONDITIONAL ATTACK
                                    | 3 = ESTABLISH PEACE TREATY
                                    | 4 = SEARCH AND CONQUER
                                    |     CIVILIZATION
                                    | 5 = WEAPONS DELIVERY
                                    | 6 = PEACEFUL CARGO DELIVERY
JC  CARGO(1000)                     | SEE ENTERPRISE CARGO CODES
JD  QUANTITY(1000)
JE  LOCATION(1000)                  | SEE ENTERPRISE PERSONNEL
                                    | LOCATION CODES 1000 - 4000
                                    | -1 = LOCATION SPECIFIED BY
                                    |      THE FOLLOWING X,Y,Z
JF  X-COORDINATE(1000)              | -INFINITY TO + INFINITY
JG  Y-COORDINATE(1000)              | -INFINITY TO + INFINITY
JH  Z-COORDINATE(1000)              | -INFINITY TO + INFINITY
JI  DESTINATION(1000)               | SAME AS ABOVE
JK  X-COORDINATE(1000)              | -INFINITY TO + INFINTIY
JL  Y-COORDINATE(1000)              | -INFINITY TO + INFINITY
JM  Z-COORDINATE(1000)              | -INFINITY TO + INFINITY
JN  PEACE TREATY OFFER(1000)        | SEE ENEMY CRAFT
JO  PEACE TREATY REQUEST(1000)      | SEE ENEMY CRAFT
```

<FEDERATION SHIP POWER SUPPLY>

ENERGY

```
JP  QUANTITY(1000)                  | 0 - 10↑10 UNITS
JQ  FIRED UPON FLAG(1000)           | 0 = NOT FIRED UPON
                                    | NON-ZERO = CODE OF WHO FIRED
                                    | SEE PERSONNEL LOCATION CODES
                                    | 1000 - 4000
```

<GENERAL DATA>

```
JR  MODULE INITIALIZATION FLAGS(6)
JS  MODULE RUN FLAGS(6)
JT  REAL TIME CLOCK                 | STARDATE = YEAR, MONTH, DAY,
                                    | HOURS, MINUTES, SECONDS,
                                    | MILLISECONDS
JU  SCENARIO RUN FLAG               | 0 = HALT
                                    | 1 = RUN
```

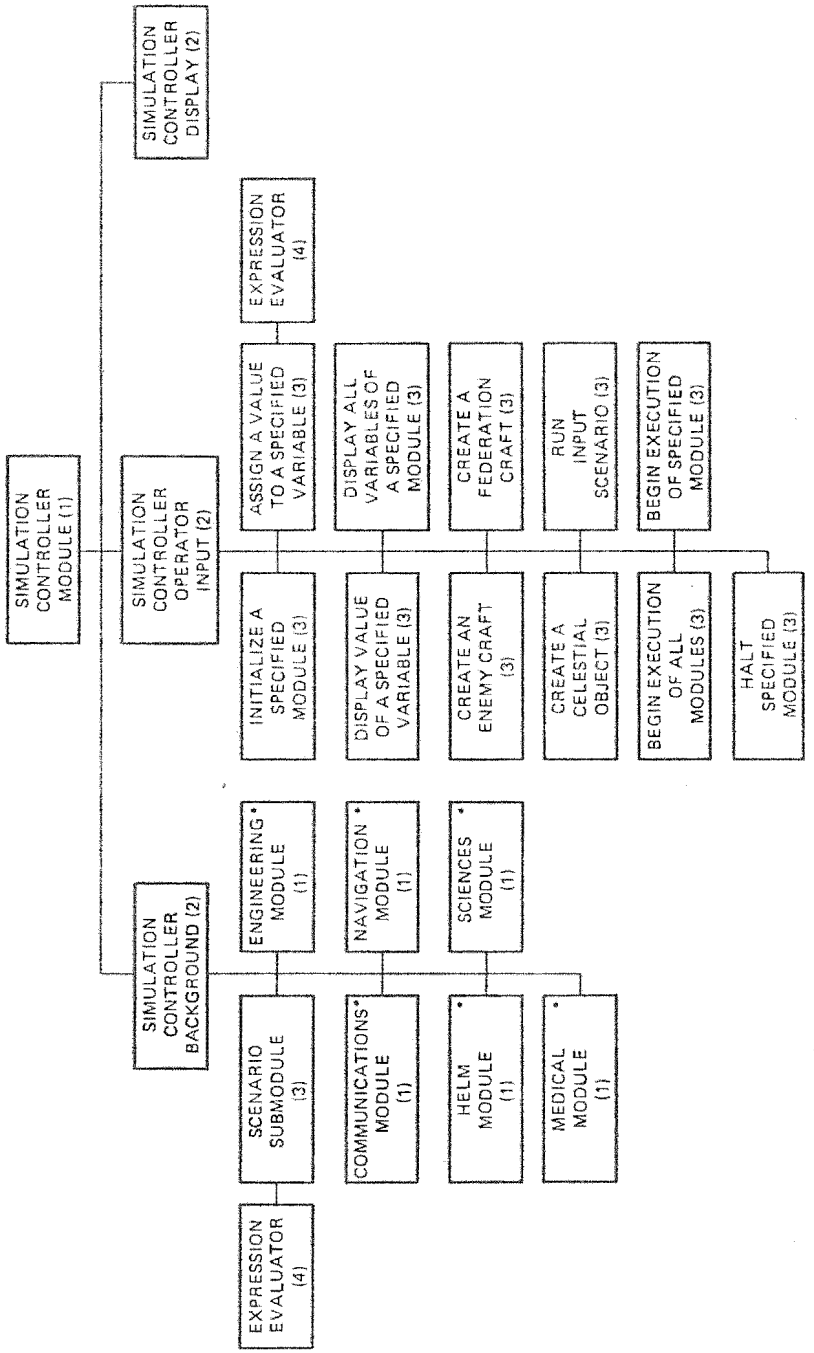



Figure 3.18 SIMULATION CONTROLLER: SUBMODULE INTERRELATIONSHIPS

LOGIC FLOW DEFINITIONS

LOCAL DATA

```
VARIABLE DISPLAY FLAGS()      [ ONE FLAG FOR EACH COMMON
                              [ DATA VARIABLE
MODULE DISPLAY FLAG(6)        [ 0 = DO NOT DISPLAY
                              [ 1 = DISPLAY ALL VARIABLES
                              [   ASSOCIATED WITH THE MODULE
SCENARIO HAS BEEN INPUT FLAG  [ 0 = NOT INPUT
                              [ 1 = INPUT
SCENARIO IS RUNNING FLAG      [ 0 = NOT RUNNING
                              [ 1 = RUNNING
```

Figure 3.19 SIMULATION CONTROLLER

```
< ********************************************************** >
< SIMULATION CONTROLLER MODULE (1) >
REPEAT
   CALL THE SIMULATION CONTROLLER BACKGROUND SUBMODULE ;
   CALL THE SIMULATION CONTROLLER OPERATOR INPUT SUBMODULE ;
   CALL THE SIMULATION CONTROLLER DISPLAY SUBMODULE .
UNTIL THE SYSTEM IS TURNED OFF :
RETURN//


< ********************************************************** >
< SIMULATION CONTROLLER OPERATOR INPUT SUBMODULE (2) >
IF THE SIMULATION CONTROLLER ENTERED A COMMAND :
   IF THE COMMAND IS INITIALIZE ALL MODULES :
      SET THE INITIALIZE FLAG FOR EACH MODULE TO 1 .
   ORIF THE COMMAND IS INITIALIZE A SPECIFIED MODULE :
      CALL THE INITIALIZE A SPECIFIED MODULE SUBMODULE .
   ORIF THE COMMAND IS ASSIGN A VALUE TO A SPECIFIED COMMON
   DATA VARIABLE :
      CALL THE ASSIGN A VALUE TO A SPECIFIED VARIABLE SUBMODULE .
   ORIF THE COMMAND IS DISPLAY THE VALUE OF A SPECIFIED COMMON
   DATA VARIABLE :
      CALL THE DISPLAY VALUE OF SPECIFIED VARIABLE SUBMODULE .
   ORIF THE COMMAND IS DISPLAY THE VALUES OF ALL COMMON DATA
   VARIABLES OF A SPECIFIED MODULE :
      CALL THE DISPLAY ALL VARIABLES OF A SPECIFIED MODULE
      SUBMODULE .
   ORIF THE COMMAND IS CREATE AN ENEMY CRAFT :
      CALL THE CREATE AN ENEMY CRAFT SUBMODULE .
   ORIF THE COMMAND IS CREATE A FEDERATION CRAFT :
      CALL THE CREATE A FEDERATION CRAFT SUBMODULE .
   ORIF THE COMMAND IS CREATE A CELESTIAL OBJECT :
      CALL THE CREATE A CELESTIAL OBJECT SUBMODULE .
   ORIF THE COMMAND IS INPUT A SCENARIO :
      READ IN THE SCENARIO ;
      SET THE 'SCENARIO HAS BEEN INPUT' FLAG .
```



```
   ORIF THE COMMAND IS RUN THE INPUT SCENARIO :
      CALL THE RUN INPUT SCENARIO COMMAND SUBMODULE .
   ORIF THE COMMAND IS HALT THE SCENARIO :
      CLEAR THE 'SCENARIO RUNNING FLAG' .
   ORIF THE COMMAND IS BEGIN EXECUTION OF A SPECIFIED MODULE :
      CALL THE BEGIN EXECUTION OF SPECIFIED MODULE SUBMODULE .
   ORIF THE COMMAND IS BEGIN EXECUTION OF ALL MODULES :
      CALL THE BEGIN EXECUTION OF ALL MODULES SUBMODULE .
   ORIF THE COMMAND IS HALT EXECUTION OF A SPECIFIED MODULE :
      CALL THE HALT EXECUTION OF A SPECIFIED MODULE SUBMODULE .
   ORIF THE COMMAND IS HALT EXECUTION OF ALL MODULES :
      CLEAR EXECUTION FLAGS FOR ALL MODULES .
   ELSE THE COMMAND IS UNKNOWN :
      SO NOTIFY THE SIMULATION CONTROLLER OPERATOR .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< SIMULATION CONTROLLER BACKGROUND SUBMODULE (2) >
CALL THE SIMULATION CONTROLLER SCENARIO SUBMODULE ;
< NOTE THAT IN THE FOLLOWING SIX CALLS 'MODULES' ARE CALLED,
< NOT 'SUBMODULES'. EACH OF THE SIX MAJOR FUNCTION MODULES
< ARE ACTUALLY LEVEL ONE MODULES EVEN THOUGH THEY ARE
< CALLED FROM THE LEVEL 2 BACKGROUND SUBMODULE OF THE
< SIMULATION CONTROLLER MODULE.
CALL THE ENGINEERING MODULE ;
CALL THE COMMUNICATIONS MODULE ;
CALL THE NAVIGATION MODULE ;
CALL THE HELM MODULE ;
CALL THE SCIENCES MODULE ;
CALL THE MEDICAL MODULE .
RETURN//


< ********************************************************** >
< SIMULATION CONTROLLER DISPLAY SUBMODULE (2) >
< NOTE: THE DISPLAYED VALUES WILL CONTINUALLY BE UPDATED ON
< THE DISPLAY SINCE THE FLAGS STAY SET UNTIL DYNAMICALLY
< CHANGED BY THE SIMULATION CONTROLLER OPERATOR.
<
< DISPLAY THE VALUES OF ALL COMMON DATA VARIABLES SPECIFIED
< BY THE SIMULATION CONTROLLER OPERATOR.
FOR I = 1 TO THE NUMBER OF COMMON DATA VARIABLES :
   IF THE ITH VARIABLE'S 'DISPLAY FLAG' IS SET :
      THEN DISPLAY ITS NAME AND CURRENT VALUE .
   ENDIF
ENDFOR
IF A MODULE DISPLAY FLAG IS SET :
   DISPLAY ALL VARIABLES ASSOCIATED WITH THAT MODULE .
ENDIF
RETURN//
```

```
< ********************************************************** >
< SIMULATION CONTROLLER SCENARIO SUBMODULE >
IF THE SCENARIO RUN FLAG IS SET :
   IF THE TIME SPECIFIED ON THE CURRENT SCENARIO LINE IS LESS
   THAN OR EQUAL TO THE REAL TIME :
      < THEN IT IS TIME TO EXECUTE THE CURRENT SCENARIO LINE
      IF THE SPECIFIED COMMON DATA VARIABLE NAME IS KNOWN :
         < EVALUATE THE EXPRESSION
         CALL THE SIMULATION CONTROLLER EXPRESSION EVALUATER ;
         IF THE EXPRESSION EVALUATED OK :
            ASSIGN THE VALUE TO THE SPECIFIED COMMON DATA
            VARIABLE .
         ELSE THE EXPRESSION DID NOT EVALUATE ALL RIGHT :
            SO NOTIFY THE SIMULATION CONTROLLER .
         ENDIF
      ELSE THE SPECIFIED COMMON DATA VARIABLE NAME IS NOT
      KNOWN :
         SO NOTIFY THE SIMULATION CONTROLLER .
      ENDIF
      POINT TO THE NEXT SCENARIO LINE ;
      IF THERE IS NO NEXT SCENARIO LINE :
         CLEAR THE SCENARIO RUN FLAG ;
         NOTIFY THE SIMULATION CONTROLLER OPERATOR .
      ENDIF
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< INITIALIZE SPECIFIED MODULE SUBMODULE (3) >
IF THE SPECIFIED MODULE NAME IS KNOWN :
   SET THE INITIALIZATION FLAG FOR THE SPECIFIED MODULE TO 1 .
ELSE THE SPECIFIED MODULE NAME IS NOT KNOWN :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//


< ********************************************************** >
< ASSIGN VALUE TO SPECIFIED COMMON DATA VARIABLE
SUBMODULE (3) >
IF THE SPECIFIED COMMON DATA VARIABLE NAME IS KNOWN :
   CALL THE EXPRESSION EVALUATION SUBMODULE ;
   IF THE EVALUATION WAS GOOD :
      ASSIGN THE VALUE TO THE SPECIFIED COMMON DATA VARIABLE .
   ELSE THE EVALUATION WAS BAD :
      SO NOTIFY THE SIMULATION CONTROLLER .
   ENDIF
ELSE THE SPECIFIED COMMON DATA VARIABLE NAME IS NOT KNOWN :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//
```



```
< ********************************************************** >
< DISPLAY SPECIFIED MODULE COMMON DATA VARIABLES
SUBMODULE (3) >
IF THE SPECIFIED MODULE NAME IS KNOWN :
   SET A FLAG TO INDICATE THAT THE SPECIFIED MODULE'S COMMON
   DATA IS TO BE DISPLAYED BY THE SIMULATION CONTROLLER
   BACKGROUND SUBMODULE ;
   CLEAR ALL OTHER MODULE DISPLAY FLAGS .
ELSE THE SPECIFIED MODULE NAME IS NOT KNOWN :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//


< ********************************************************** >
< CREATE ENEMY SUBMODULE (3) >
IF THE SPECIFIED X, Y, Z COORDINATES VALUES ARE WITHIN THE
UNIVERSE :
   IF THE SPECIFIED ENEMY CRAFT NAME IS NOT ALREADY AN EXISTING
   NAME :
      IF THE SPECIFIED MISSION IS A VALID MISSION TYPE :
         IF THERE IS ROOM FOR AN ADDITIONAL ENEMY CRAFT WITHIN
         THE COMMON DATA ARRAY RESERVED FOR ENEMY CRAFT :
            < THEN CREATE THE ENEMY CRAFT AS SPECIFIED BY
            < INITIALIZING THE APPROPRIATE COMMON DATA
            < VARIABLES
            INCREMENT THE COUNT OF ENEMY CRAFT .
         ELSE THERE IS NO ROOM FOR ANOTHER ENEMY CRAFT :
            SO NOTIFY THE SIMULATION CONTROLLER .
         ENDIF
      ELSE IT IS AN INVALID MISSION TYPE :
         SO NOTIFY THE SIMULATION CONTROLLER .
      ENDIF
   ELSE THE SPECIFIED CRAFT NAME IS ALREADY USED :
      SO NOTIFY THE SIMULATION CONTROLLER .
   ENDIF
ELSE THE SPECIFIED X, Y, Z COORDINATES ARE NOT WITHIN THE
UNIVERSE :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//


< ********************************************************** >
< CREATE FEDERATION CRAFT SUBMODULE (3) >
IF THE SPECIFIED X, Y, Z COORDINATES ARE WITHIN THE UNIVERSE :
   IF THE SPECIFIED FEDERATION CRAFT NAME IS NOT ALREADY AN
   EXISTING CRAFT NAME :
      IF THE SPECIFIED MISSION IS A VALID TYPE :
         IF THERE IS ROOM FOR AN ADDITIONAL FEDERATION CRAFT
         WITHIN THE COMMON DATA ARRAY RESERVED FOR
         FEDERATION CRAFT :
            < THEN CREATE THE FEDERATION CRAFT AS SPECIFIED BY
```

```
            < INITIALIZING THE APPROPRIATE COMMON DATA VARIABLES
            INCREMENT THE COUNT OF FEDERATION CRAFT .
         ELSE THERE IS NO ROOM FOR ANOTHER FEDERATION CRAFT :
            SO NOTIFY THE SIMULATION CONTROLLER .
         ENDIF
      ELSE THE SPECIFIED MISSION TYPE IS INVALID :
         SO NOTIFY THE SIMULATION CONTROLLER
      ENDIF
   ELSE THE SPECIFIED CRAFT NAME IS ALREADY USED :
      SO NOTIFY THE SIMULATION CONTROLLER .
   ENDIF
ELSE THE SPECIFIED X, Y, Z COORDINATES ARE NOT WITHIN THE
UNIVERSE :
   SO NOTIFY THE SIMULATION CONTROLLER
ENDIF
RETURN//


< ********************************************************** >
< CREATE A CELESTIAL OBJECT SUBMODULE (3) >
IF THE SPECIFIED X, Y, Z COORDINATE IS WITHIN THE UNIVERSE :
   IF THE SPECIFIED NAME IS NOT ALREADY USED :
      IF THE SPECIFIED RADIUS IS VALID (REASONABLE) :
         IF THERE IS ROOM FOR ANOTHER CELESTIAL OBJECT WITHIN
         THE COMMON DATA ARRAY RESERVED FOR CELESTIAL OBJECTS :
            < THEN CREATE THE SPECIFIED CELESTIAL OBJECT BY
            < INITIALIZING THE APPROPRIATE COMMON DATA
            < VARIABLES
            INCREMENT THE COUNT OF CELESTIAL OBJECTS .
         ELSE THERE IS NO ROOM FOR ANOTHER CELESTIAL OBJECT :
            SO NOTIFY THE SIMULATION CONTROLLER .
         ENDIF
      ELSE THE SPECIFIED RADIUS IS INVALID :
         SO NOTIFY THE SIMULATION CONTROLLER .
      ENDIF
   ELSE THE SPECIFIED NAME IS ALREADY USED :
      SO NOTIFY THE SIMULATION CONTROLLER .
   ENDIF
ELSE THE SPECIFIED X, Y, Z COORDINATES ARE OUTSIDE THE UNIVERSE :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//


< ********************************************************** >
< RUN THE INPUT SCENARIO COMMAND SUBMODULE (3) >
IF A SCENARIO HAS BEEN INPUT :
   SET THE SCENARIO POINTER TO THE FIRST LINE IN THE SCENARIO ;
   SET THE FLAG INDICATING THAT A SCENARIO IS RUNNING .
ELSE THERE IS NO SCENARIO TO RUN :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//
```



```
< ********************************************************** >
< BEGIN EXECUTION OF THE SPECIFIED MODULE (3) >
IF THE SPECIFIED MODULE NAME IS KNOWN :
   IF THE INITIALIZATION FLAG IS CLEAR FOR THE SPECIFIED MODULE :
      < THEN IT IS READY TO RUN
      SET THE RUN FLAG FOR THE SPECIFIED MODULE .
   ELSE THE INITIALIZATION FLAG FOR THE SPECIFIED MODULE IS SET :
      < SO IT IS NOT READY TO RUN
      NOTIFY THE SIMULATION CONTROLLER .
   ENDIF
ELSE THE SPECIFIED MODULE NAME IS NOT KNOWN :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//


< ********************************************************** >
< BEGIN EXECUTION OF ALL MODULES SUBMODULE (3) >
IF ALL MODULE INITIALIZATION FLAGS ARE CLEAR :
   SET THE MODULE EXECUTION FLAGS FOR ALL MODULES .
ELSE NOT ALL MODULE INITIALIZATION FLAGS ARE CLEAR :
   SO NOTIFY THE SIMULATION CONTROLLER OF WHICH FLAGS ARE NOT
   READY .
ENDIF
RETURN//


< ********************************************************** >
< HALT SPECIFIED MODULE SUBMODULE (3) >
IF THE SPECIFIED MODULE NAME IS KNOWN :
   IF THE SPECIFIED MODULE EXECUTION FLAG IS NOT CLEAR :
      CLEAR THE EXECUTION FLAG FOR THE SPECIFIED MODULE .
   ELSE THE EXECUTION FLAG FOR THE SPECIFIED MODULE IS ALREADY
   CLEARED :
      SO NOTIFY THE SIMULATION CONTROLLER .
   ENDIF
ELSE THE SPECIFIED MODULE NAME IS NOT KNOWN :
   SO NOTIFY THE SIMULATION CONTROLLER .
ENDIF
RETURN//


< ********************************************************** >
< SIMULATION CONTROLLER EXPRESSION EVALUATOR (4) >
< THIS SUBMODULE EVALUATES AN EXPRESSION INPUT AS A SCENARIO
< LINE.
< CURRENTLY IT ONLY ALLOWS TWO TYPES OF EXPRESSIONS, SIMPLE
< NUMERICAL VALUES LIKE 32 OR 45, AND SINGLE COMMON DATA
< VARIABLE NAMES (INCLUDING ARRAYS) .
< THIS SUBMODULE SHOULD ENTER WITH THE EXPRESSION AND
< RETURN WITH A NUMERICAL VALUE AND A FLAG EITHER CLEARED TO
< INDICATE A GOOD EVALUATION OR SET TO INDICATE A BAD
```

```
< EVALUATION. A BAD EVALUATION CAN BE CAUSED BY AN UNKNOWN
< VARIABLE NAME, A SUBSCRIPT EVALUATED BAD, OR A BAD
< NUMERICAL CHARACTER. IT IS LEFT UP TO THE READER TO EXPAND
< THIS SUBMODULE INTO A WORKING SUBROUTINE.
RETURN//


< ********************************************************** >
< COMMUNICATIONS MODULE (1) >
IF THE INITIALIZATION FLAG IS SET :
    CALL THE COMMUNICATIONS INITIALIZATION SUBMODULE .
ELSE THE INITIALIZATION FLAG IS CLEAR :
    IF THE RUN FLAG IS SET :
        CALL THE COMMUNICATIONS EXECUTION SUBMODULE .
    ENDIF
ENDIF
RETURN//


< ********************************************************** >
< COMMUNICATIONS INITIALIZATION SUBMODULE (2) >
SET ALL COMMUNICATIONS DATA TO NOMINAL ;
CLEAR THE COMMUNICATIONS INTIALIZATION FLAG .
RETURN//


< ********************************************************** >
< COMMUNICATIONS EXECUTION SUBMODULE (2) >
CALL THE COMMUNICATIONS BACKGROUND SUBMODULE ;
CALL THE COMMUNICATIONS OFFICER INPUT SUBMODULE ;
CALL THE COMMUNICATIONS DISPLAY SUBMODULE .
RETURN//


< ********************************************************** >
< COMMUNICATIONS BACKGROUND SUBMODULE (3) >
UPDATE THE OPERATIONAL STATUS OF THE ENTERPRISE'S COMPUTER ;
DECREASE THE ENERGY SUPPLY BY AN AMOUNT CORRESPONDING TO
THE COMMUNICATIONS ENERGY REQUIREMENT ;
< UPDATE THE STATUS OF ALL PERSONNEL AND CARGO ON THE
ENTERPRISE
CALL THE COMMUCICATIONS PERSONNEL LOCATION UPDATE
SUBMODULE ;
< UPDATE THE SECURITY SECTION
CALL THE SECURITY UPDATE SUBMODULE .
RETURN//


< ********************************************************** >
< COMMUNICATIONS DISPLAY SUBMODULE (3) >
UPDATE THE COMMUNICATIONS DISPLAY .
RETURN//
```

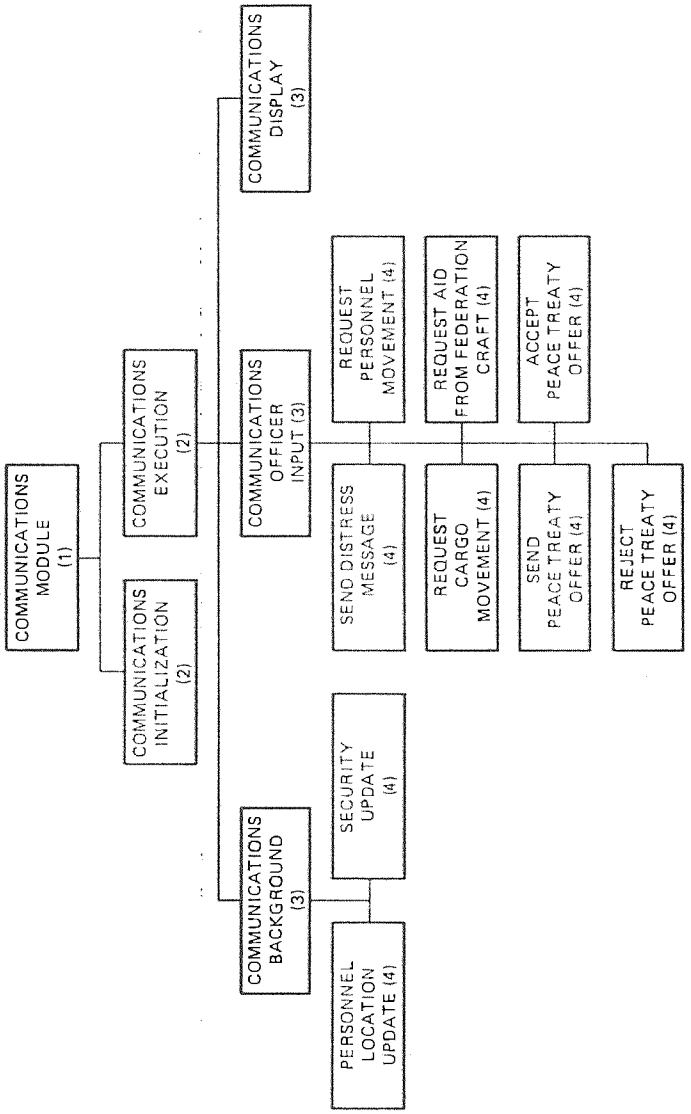



Figure 3.20 COMMUNICATIONS: SUBMODULE INTERRELATIONSHIPS

```
< ********************************************************** >
< COMMUNICATIONS OPERATOR INPUT SUBMODULE (3) >
IF THE COMMUNICATIONS OPERATOR ENTERED A COMMAND :
   IF THE COMMAND IS DISPLAY THE NEXT MESSAGE :
      SET FLAG TO DISPLAY THE NEXT MESSAGE .
   ORIF THE COMMAND IS TO SEND DISTRESS MESSAGE :
      CALL THE SEND DISTRESS MESSAGE SUBMODULE .
   ORIF THE COMMAND IS TO REQUEST PERSONNEL MOVEMENT :
      CALL THE REQUEST PERSONNEL MOVEMENT SUBMODULE .
   ORIF THE COMMAND IS REQUEST CARGO MOVEMENT :
      CALL THE REQUEST CARGO MOVEMENT SUBMODULE .
   ORIF THE COMMAND IS TO REQUEST AID FROM A FEDERATION CRAFT:
      CALL THE REQUEST AID FROM FEDERATION CRAFT SUBMODULE .
   ORIF THE COMMAND IS TO SEND PEACE TREATY OFFER TO ENEMY
   CRAFT :
      CALL THE SEND PEACE TREATY OFFER TO ENEMY CRAFT
      SUBMODULE .
   ORIF THE COMMAND IS TO ACCEPT PEACE TREATY OFFER FROM
   ENEMY CRAFT :
      CALL THE ACCEPT PEACE TREATY OFFER FROM ENEMY CRAFT
      SUBMODULE .
   ORIF THE COMMAND IS TO REJECT PEACE TREATY OFFER FROM
   ENEMY CRAFT :
      CALL THE REJECT PEACE TREATY OFFER FROM ENEMY CRAFT
      SUBMODULE .
   ELSE THE COMMAND IS UNKNOWN :
      SO NOTIFY THE COMMUNICATIONS OFFICER .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< COMMUNICATIONS PERSONNEL AND CARGO LOCATION UPDATE
SUBMODULE (4) >
< ALL PERSONNEL AND CARGO WHOSE LOCATION IS NOT THE SAME AS
< THEIR DESTINATION MUST BE UPDATED
FOR I = 1 TO THE NUMBER OF PERSONNEL ON THE ENTERPRISE :
   IF HIS DESTINATION IS WITHIN THE SAME TURBO-ELEVATOR SERVICE
   AREA :
      THEN SET HIS LOCATION TO HIS DESTINATION ;
      SET HIS DESTINATION TO 'NONE' ;
      NOTIFY THE COMMUNICATIONS OFFICER THAT HE HAS ARRIVED .
   ORIF HIS DESTINATION IS WITHIN ONE OF THE OTHER T/E SERVICE
   AREAS AND HE IS NOT ALREADY AT THE T/E STATION WITHIN HIS
   CURRENT SERVICE AREA :
      THEN SET HIS LOCATION TO THE T/E STATION WITHIN THE T/E
      SERVICE AREA IN WHICH HE IS CURRENTLY LOCATED .
   ORIF HIS DESTINATION IS OUTSIDE OF THE ENTERPRISE AND HE IS
   NOT ALREADY AT A TRANSPORTER OR IN THE SHUTTLEBAY :
      IF HIS DESTINATION IS WITHIN TRANSPORTING DISTANCE (TBD) OF
      THE ENTERPRISE :
         SET HIS LOCATION TO THE TRANSPORTER WITHIN THE T/E
```



```
         SERVICE AREA IN WHICH HE CURRENTLY RESIDES .
      ELSE THE DESTINATION IS NOT WITHIN TRANSPORTING DISTANCE
      (TBD) :
         < SO HE WILL HAVE TO USE A SHUTTLECRAFT
         SET HIS LOCATION TO THE SHUTTLEBAY .
      ENDIF
   ENDIF
ENDFOR
FOR I = 1 TO THE NUMBER OF CARGO ON THE ENTERPRISE :
   < DO THIS THE SAME AS FOR PERSONNEL, ABOVE .
ENDFOR
RETURN//


< ********************************************************** >
< SECURITY UPDATE SUBMODULE (4) >
< DETERMINE THE NUMBER OF PRISONERS
COUNT = 0 ;
FOR I = 1 TO THE NUMBER OF PERSONNEL ON THE ENTERPRISE :
   IF THE ITH PERSON'S LOCATION IS THE BRIG :
      COUNT = COUNT + 1 .
   ENDIF
ENDFOR
< UPDATE THE DRAIN ON THE MAIN ENERGY SUPPLY PROPORTIONAL TO
< THE NUMBER OF PRISONERS, FUNCTIONAL, AND OPERATIONAL
< STATUS OF THE SECURITY SECTION .
< PRISONER ESCAPE IS A FUNCTION OF THE NUMBER OF PRISONERS,
< FUNCTIONAL STATUS AND RELIABILITY OF THE SECURITY SECTION,
< AND THE NUMBER OF SECURITY PERSONNEL AT THE SECURITY
< SECTION .
< COMPUTE AN ESCAPE FACTOR (AS ABOVE) .
< DETERMINE THE NUMBER OF SECURITY PERSONNEL AT THE SECURITY
< SECTION
COUNTA = 0 .
FOR I = 1 TO THE NUMBER OF PERSONNEL ON THE ENTERPRISE :
   IF THE ITH PERSON IS A SECURITY OFFICER AND IS AT THE SECURITY
   SECTION (NOT IN THE BRIG) :
      COUNTA = COUNTA + 1 .
   ENDIF
ENDFOR
IF THE ESCAPE FACTOR IS ABOVE SOME DEFINED LIMIT (TBD) :
   THEN SET THE DESTINATION OF ONE OF THE PRISONERS TO A
   RANDOM DESTINATION OUTSIDE OF THE ENTERPRISE (OTHER CRAFT
   OR CELESTIAL OBJECT ;
   <THE DETECTION OF THE ESCAPE IS A RANDOM FUNCTION
   IF THE ESCAPE IS DETECTED :
      NOTIFY THE COMMUNICATIONS OFFICER .
   ENDIF
ENDIF
< UPDATE THE FUNCTIONAL STATUS OF THE DETENTION CELL
< RELATIVE TO THE NUMBER OF MAINTENANCE PERSONNEL THERE
< (NOT IN THE BRIG BUT AT THE SECURITY SECTION) .
RETURN//
```

```
< ********************************************************** >
< SEND DISTRESS MESSAGE SUBMODULE (4) >
< A DISTRESS MESSAGE IS NOT SENT TO ANY PARTICULAR FEDERATION
< CRAFT BUT IS A GENERAL CALL FOR HELP. WHETHER IT IS RECEIVED
< DEPENDS UPON THE DISTANCE TO OTHER FEDERATION CRAFT, AND
< THE FUNCTIONAL STATUS AND RELIABILITY FACTOR OF THE
< ENTERPRISE'S COMMUNICATIONS COMPUTER.
< CALCULATE THE 'RECEPTION FACTOR' BASED UPON THE STATUS AND
< RELIABILITY FACTOR OF THE ENTERPRISE'S COMPUTER. (SHOULD BE
< IN THE RANGE 0 - 100 WHERE ZERO MEANS MOST UNLIKELY TO BE
< RECEIVED AND 100 MEANS MOST LIKELY TO BE RECEIVED.
FOR I = 1 TO THE NUMBER OF FEDERATION CRAFT :
   DETERMINE THE DISTANCE BETWEEN THE ITH FEDERATION CRAFT
   AND THE ENTERPRISE ;
   SET THE RECEPTION FACTOR FOR THE ITH FEDERATION CRAFT
   EQUAL TO THE GENERAL RECEPTION FACTOR (CALCULATED ABOVE)
   MINUS A FACTOR DIRECTLY PROPORTIONAL TO THE DISTANCE
   (I.E. THE FARTHER AWAY THE LOWER THE RECEPTION FACTOR) ;
   IF THE CALCULATED RECEPTION FACTOR FOR THE ITH FEDERATION
   CRAFT IS ABOVE SOME LEVEL (TBD) :
      < THEN THE ITH FEDERATION CRAFT RECEIVES THE MESSAGE SO
      < MAKE IT BRING HELP TO THE ENTERPRISE
      SET THE DESTINATION OF THE ITH FEDERATION CRAFT TO THE
      ENTERPRISE ;
      SET THE MISSION OF THE ITH FEDERATION CRAFT TO
      CONDITIONAL ATTACK .
   ENDIF
ENDFOR
RETURN//


< ********************************************************** >
< REQUEST PERSONNEL MOVEMENT SUBMODULE (4) >
IF THE SPECIFIED PERSONNEL NAME IS KNOWN :
   IF THE SPECIFIED PERSONNEL IS ALIVE :
      IF THE SPECIFIED DESTINATION IS KNOWN :
         SET THE DESTINATION OF THE SPECIFIED PERSONNEL TO THE
         SPECIFIED DESTINATION .
      ELSE THE SPECIFIED DESTINATION IS NOT KNOWN :
         SO NOTIFY THE COMMUNICATIONS OFFICER .
      ENDIF
   ELSE THE SPECIFIED PERSONNEL IS DEAD :
      SO NOTIFY THE COMMUNICATIONS OFFICER .
   ENDIF
ELSE THE SPECIFIED PERSONNEL NAME IS NOT KNOWN :
   SO NOTIFY THE COMMUNICATIONS OFFICER .
ENDIF
RETURN//


< ********************************************************** >
< REQUEST CARGO MOVEMENT SUBMODULE (4) >
IF THE SPECIFIED CARGO IS KNOWN :
   IF THE SPECIFIED DESTINATION IS KNOWN :
```

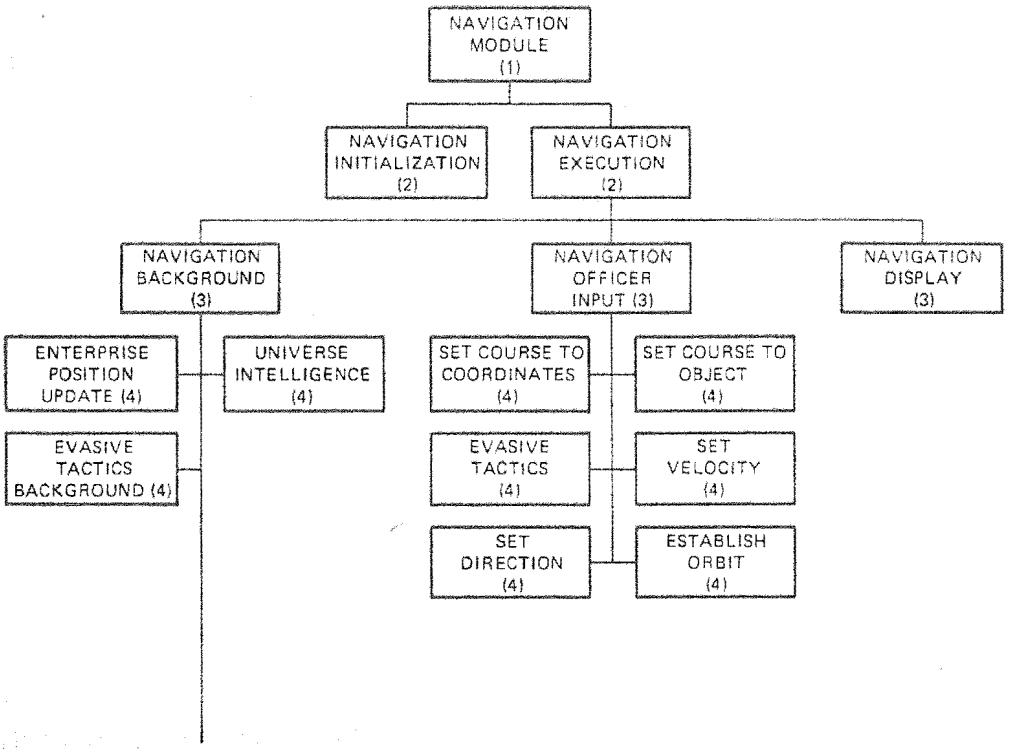


```
      SET THE DESTINATION OF THE SPECIFIED CARGO TO THE
      SPECIFIED DESTINATION .
   ELSE THE SPECIFIED DESTINATION IS NOT KNOWN :
      SO NOTIFY THE COMMUNICATIONS OFFICER .
   ENDIF
ELSE THE SPECIFIED CARGO IS NOT KNOWN :
   SO NOTIFY THE COMMUNICATIONS OFFICER .
ENDIF
RETURN//


< ********************************************************** >
<REQUEST AID FROM A FEDERATION CRAFT SUBMODULE (4) >
IF THE SPECIFIED FEDERATION CRAFT NAME IS KNOWN :
   < CALCULATE THE RECEPTION FACTOR AS IN THE 'SEND DISTRESS
   < MESSAGE' SUBMODULE .
   IF THE RECEPTION FACTOR IS ABOVE A (TBD) LEVEL :
      < THEN THE SPECIFIED CRAFT RECEIVES THE MESSAGE .
      SET THE DESTINATION OF THE SPECIFIED FEDERATION CRAFT TO
      THE ENTERPRISE ;
      SEND A MESSAGE (VIA THE COMMUNICATIONS MODULE MESSAGE
      AREA) NOTIFYING THE COMMUNICATIONS OFFICER THAT THE
      FEDERATION CRAFT IS ON ITS WAY .
   ELSE THE MESSAGE IS NOT RECEIVED :
      SO NOTIFY THE COMMUNICATIONS OFFICER .
   ENDIF
ELSE THE SPECIFIED FEDERATION CRAFT NAME IS NOT KNOWN :
   SO NOTIFY THE COMMUNICATIONS OFFICER .
ENDIF
RETURN//


< ********************************************************** >
< SEND PEACE TREATY OFFER TO ENEMY CRAFT SUBMODULE (4) >
IF THE SPECIFIED ENEMY CRAFT NAME IS KNOWN :
   < CALCULATE A RECEPTION FACTOR AS IN THE 'SEND DISTRESS
   < MESSAGE' SUBMODULE .
   IF THE RECEPTION FACTOR IS ABOVE A (TBD) LEVEL :
      <THEN THE ENEMY CRAFT RECEIVES THE PEACE TREATY OFFER .
      SO SET THE 'PEACE TREATY OFFER FLAG' OF THE SPECIFIED
      ENEMY CRAFT TO THE LOCATION CODE FOR THE ENTERPRISE(4000)
      SO HE KNOWS WHO MADE THE OFFER .
   ENDIF
ELSE THE SPECIFIED ENEMY CRAFT NAME IS NOT KNOWN :
   SO NOTIFY THE COMMUNICATIONS OFFICER .
ENDIF
RETURN//


< ********************************************************** >
< ACCEPT PEACE TREATY FROM ENEMY CRAFT SUBMODULE (4) >
IF THE SPECIFIED ENEMY CRAFT NAME IS KNOWN :
   IF THE 'REQUEST PEACE TREATY FLAG' FOR THE SPECIFIED ENEMY
   CRAFT IS SET :
```



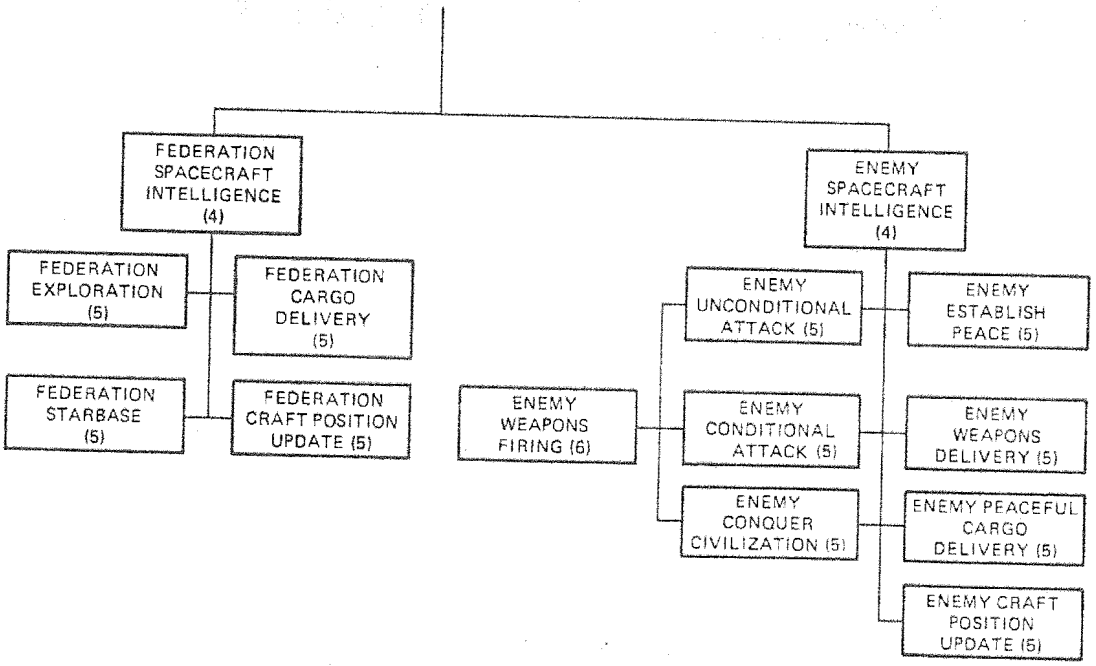


Figure 3.21 NAVIGATION: SUBMODULE INTERRELATIONSHIPS

```
      < THEN A PEACE TREATY IS ESTABLISHED .
      CLEAR THE 'REQUEST PEACE TREATY FLAG' OF THE SPECIFIED
      ENEMY CRAFT ;
      SET THE MISSION OF THE SPECIFIED ENEMY CRAFT TO ANY
      MISSION OTHER THAN UNCONDITIONAL ATTACK ;
      SET THE DIRECTION OF THE SPECIFIED ENEMY CRAFT AWAY FROM
      THE ENTERPRISE ;
      SEND A MESSAGE FROM THE SPECIFIED ENEMY CRAFT, VIA THE
      COMMUNICATIONS MODULE MESSAGE AREA, TO THE ENTERPRISE
      OF THE ACCEPTANCE OF THE PEACE TREATY .
   ENDIF
ELSE THE SPECIFIED ENEMY CRAFT NAME IS NOT KNOWN :
   SO NOTIFY THE COMMUNICATIONS OFFICER .
ENDIF
RETURN//


< ********************************************************** >
< REJECT PEACE TREATY OFFER FROM ENEMY CRAFT SUBMODULE (4) >
IF THE SPECIFIED ENEMY CRAFT NAME IS KNOWN :
   CLEAR THE 'REQUEST PEACE TREATY FLAG' FOR THE SPECIFIED
   ENEMY CRAFT ;
   < REJECTING AN OFFER OF PEACE TENDS TO MAKE THE ENEMY MAD
   < SO HE WILL ATTEMPT TO GET HELP FROM OTHER ENEMY CRAFT .
   GENERATE A RANDOM NUMBER ;
   IF IT IS OVER SOME (TBD) LEVEL :
      < THEN THE ENEMY IS MAD ENOUGH TO CALL FOR HELP.
      CALCULATE A 'RECEPTION FACTOR' AS IN THE 'SEND DISTRESS
      MESSAGE' SUBMODULE (EXCEPT DO IT FOR THIS ENEMY, NOT THE
      ENTERPRISE) ;
      FOR I = 1 TO THE NUMBER OF ENEMY CRAFT :
         DETERMINE THE DISTANCE BETWEEN THE ITH ENEMY CRAFT
         AND THE SPECIFIED ENEMY CRAFT ;
         SET THE RECEPTION FACTOR FOR THE ITH ENEMY CRAFT
         EQUAL TO THE RECEPTION FACTOR CALCULATED ABOVE MINUS
         A FACTOR DIRECTLY RELATED TO THE DISTANCE ;
         IF THE CALCULATED RECEPTION FACTOR FOR THE ITH ENEMY
         CRAFT IS ABOVE A (TBD) LEVEL :
            < THEN THE ITH ENEMY CRAFT RECEIVES THE CALL FOR
            < HELP .
            SO SET THE DESTINATION OF THE ITH ENEMY CRAFT TO THE
            LOCATION OF THE ENTERPRISE AND SET ITS MISSION TO
            UNCONDITIONAL ATTACK .
         ENDIF
      ENDFOR
   ENDIF
ELSE THE SPECIFIED ENEMY CRAFT NAME IS NOT KNOWN :
   SO NOTIFY THE COMMUNICATIONS OFFICER .
ENDIF
RETURN//
```


LOCAL DATA

EVASIVE TACTICS FLAG	[0 = NORMAL NAVIGATION [1 = COMPUTER CONTROLLED EVASIVE TACTICS
NEAREST CELESTIAL OBJECT CODE	
TIME OF LAST CALL	
ENEMY OFFENSIVE WEAPONS WAIT TIME (1000)	[0 – INFINITE SECONDS
ENEMY CRAFT COURSE CHANGE TIME (1000)	[0 – INFINITE SECONDS
FEDERATION CRAFT COURSE CHANGE TIME (1000)	[0 – INFINITE SECONDS
CELESTIAL OBJECT OFFENSIVE WEAPONS WAIT TIME (1000)	[0 – INFINITE SECONDS

Figure 3.22 NAVIGATION MODULE

```
< ********************************************************** >
< NAVIGATION MODULE (1) >
IF THE INITIALIZATION FLAG IS SET :
   CALL THE NAVIGATION INITIALIZATION SUBMODULE .
ELSE THE INITIALIZATION FLAG IS CLEAR :
   IF THE RUN FLAG IS SET :
      CALL THE NAVIGATION EXECUTION SUBMODULE .
   ENDIF
ENDIF
SET THE TIME OF LAST CALL EQUAL TO THE CURRENT TIME .
RETURN//


< ********************************************************** >
< NAVIGATION INITIALIZATION SUBMODULE (2) >
SET ALL NAVIGATION VARIABLES TO NOMINAL ;
CLEAR THE NAVIGATION MODULE INITIALIZATION FLAG .
RETURN//


< ********************************************************** >
< NAVIGATION EXECUTION SUBMODULE (2) >
CALL THE NAVIGATION BACKGROUND SUBMODULE ;
CALL THE NAVIGATION OFFICER INPUT SUBMODULE ;
CALL THE NAVIGATION DISPLAY SUBMODULE .
RETURN//


< ********************************************************** >
< NAVIGATION BACKGROUND SUBMODULE (3) >
UPDATE THE REAL-TIME CLOCK ;
CALL THE ENTERPRISE NAVIGATION POSITION UPDATE SUBMODULE ;
CALL THE NAVIGATION ENEMY SPACECRAFT INTELLIGENCE SUBMODULE ;
```

```
CALL THE NAVIGATION FEDERATION SPACECRAFT INTELLIGENCE
SUBMODULE ;
CALL THE NAVIGATION UNIVERSE INTELLIGENCE SUBMODULE ;
CALL THE NAVIGATION EVASIVE TACTICS BACKGROUND SUBMODULE .
RETURN//


< ********************************************************** >
< NAVIGATION OFFICER INPUT SUBMODULE (3) >
IF THE NAVIGATION OFFICER HAS GIVEN A COMMAND :
   IF THE COMMAND IS NOT EVASIVE TACTICS :
      CLEAR THE EVASIVE TACTICS FLAG .
   ENDIF
   IF THE COMMAND IS SET COURSE TO COORDINATES :
      CALL THE NAVIGATION SET COURSE TO COORDINATES
      SUBMODULE .
   ORIF THE COMMAND IS SET COURSE TO OBJECT :
      CALL THE NAVIGATION SET COURSE TO OBJECT SUBMODULE .
   ORIF THE COMMAND IS EVASIVE TACTICS :
      CALL THE NAVIGATION EVASIVE TACTICS SUBMODULE .
   ORIF THE COMMAND IS SET VELOCITY :
      CALL THE NAVIGATION SET VELOCITY SUBMODULE .
   ORIF THE COMMAND IS SET DIRECTION :
      CALL THE NAVIGATION SET DIRECTION SUBMODULE .
   ORIF THE COMMAND IS ESTABLISH ORBIT :
      CALL THE NAVIGATION ESTABLISH ORBIT SUBMODULE .
   ELSE THE COMMAND IS NOT RECOGNIZED :
      SO NOTIFY THE NAVIGATION OFFICER .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION DISPLAY SUBMODULE (3) >
UPDATE THE NAVIGATION DISPLAY ;
< SEE FIGURE 3.10
UPDATE THE MAIN SCREEN DISPLAY .
< THE MAIN SCREEN IS THE ONE WHICH, ON THE TELEVISION SHOW
< 'STAR TREK', IS POSITIONED DIRECTLY IN FRONT OF THE CAPTAIN,
< I.E. THE LARGE, 8-FOOT WIDE DISPLAY
RETURN//


< ********************************************************** >
< NAVIGATION SET COURSE TO COORDINATES SUBMODULE (4) >
IF THE SPECIFIED COORDINATES ARE WITHIN THE UNIVERSE :
   DETERMINE THE DIRECTION NECESSARY TO REACH THE SPECIFIED
   COORDINATES ;
   SET THE DIRECTION OF THE ENTERPRISE ACCORDINGLY ;
   IF THE SPECIFIED COORDINATES ARE WITHIN KLINGON TERRITORY :
      NOTIFY THE NAVIGATOR .
   ORIF THE SPECIFIED COORDINATES ARE WITHIN ROMULON
   TERRITORY :
```



```
      NOTIFY THE NAVIGATOR .
   ENDIF
ELSE THE SPECIFIED COORDINATES ARE OUTSIDE THE UNIVERSE :
   SO NOTIFY THE NAVIGATOR .
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION SET COURSE TO OBJECT SUBMODULE (4) >
< THE OBJECT MUST BE SPECIFIED BY NAME .
< SEE IF THE SPECIFIED OBJECT IS A CELESTIAL OBJECT .
I = 1 ;
REPEAT
   CHECK THE ITH CELESTIAL OBJECT ;
   I = I + 1 .
UNTIL THE NAMED OBJECT IS FOUND WHICH IS ALSO CHARTED OR ALL
CELESTIAL OBJECTS ARE CHECKED :
IF THE SPECIFIED OBJECT IS NOT A CELESTIAL OBJECT :
   < SEE IF IT IS AN ENEMY CRAFT .
   I = 1;
   REPEAT
      CHECK THE ITH ENEMY CRAFT NAME ;
      I = I + 1 .
   UNTIL THE NAMED ENEMY CRAFT FOUND WHICH IS ALSO DETECTABLE
   AND WITHIN SENSOR DISTANCE OR ALL ENEMY CRAFT CHECKED :
   IF THE SPECIFIED OBJECT IS NOT AN ENEMY CRAFT :
      < SEE IF IT IS A FEDERATION CRAFT .
      I = 1 ;
      REPEAT
         CHECK THE ITH FEDERATION CRAFT ;
         I = I + 1 .
      UNTIL THE NAMED FEDERATION CRAFT FOUND WHICH IS WITHIN
      SENSOR RANGE OR ALL FEDERATION CRAFT CHECKED :
      IF THE SPECIFIED OBJECT IS NOT A FEDERATION CRAFT :
      < SEE IF IT IS A SHUTTLECRAFT .
      I = 1 ;
      REPEAT
         CHECK THE ITH SHUTTLECRAFT ;
         I = I + 1 .
      UNTIL THE NAMED SHUTTLECRAFT FOUND THAT IS WITHIN SENSOR
      RANGE OR ALL SHUTTLECRAFT CHECKED :
      ENDIF
   ENDIF
ENDIF
IF THE NAMED OBJECT WAS FOUND :
   THEN SET THE ENTERPRISE DESTINATION CODE TO THE CODE OF THE
   OBJECT .
   < RATHER THAN SPECIFYING A COORDINATE LOCATION WHICH
   < MIGHT CAUSE THE ENTERPRISE TO LOSE TRACK OF THE OBJECT
ELSE THE SPECIFIED OBJECT WAS NOT FOUND :
   SO NOTIFY THE NAVIGATOR .
ENDIF
RETURN//
```

```
< ********************************************************** >
< NAVIGATION EVASIVE TACTICS SUBMODULE (4) >
SET THE EVASIVE TACTICS FLAG .
< NOTE: THE ACTUAL EVASIVE TACTICS ARE HANDLED BY THE
< NAVIGATION BACKGROUND MODULE. THIS FLAG MERELY TELLS IT
< THAT THE NAVIGATION COMPUTER IS TAKING OVER CONTROL OF
< NAVIGATION AND OTHER RELATED FUNCTIONS.
RETURN//


< ********************************************************** >
< NAVIGATION SET VELOCITY SUBMODULE (4) >
IF THE SPECIFIED VELOCITY IS SUBLIGHT (0 – .99) :
   TURN OFF THE SPACE/WARP ENGINES ;
   SET THE OPERATIONAL STATUS OF THE IMPULSE ENGINES TO THE
   REQUESTED SPEED ;
   SET THE ENTERPRISE VELOCITY TO THE REQUESTED SPEED .
ORIF THE SPECIFIED VELOCITY IS GREATER THAN LIGHT :
   TURN OFF THE IMPULSE ENGINES ;
   SET THE OPERATIONAL STATUS OF THE SPACE/WARP ENGINES TO THE
   SPECIFIED SPEED ;
   SET THE ENTERPRISE VELOCITY TO THE SPECIFIED SPEED .
ELSE THE REQUESTED VELOCITY IS INVALID :
   SO NOTIFY THE NAVIGATOR .
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION SET DIRECTION SUBMODULE (4) >
< THE DIRECTION MUST BE GIVEN AS TWO ANGLES; X/Y AND X/Z .
IF BOTH ANGLES ARE WITHIN 0 TO 360 DEGREES :
   SET THE X/Y AND X/Z DIRECTION VECTORS OF THE ENTERPRISE TO
   THE SPECIFIED VECTORS ;
   SET THE DESTINATION CODE FOR THE ENTERPRISE TO –1 .
ELSE NOT BOTH DIRECTION ANGLES ARE WITHIN 0 – 360 DEGREES :
   SO NOTIFY THE NAVIGATOR .
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION ESTABLISH ORBIT SUBMODULE (4) >
< THIS ALWAYS ESTABLISHES THE ORBIT AROUND THE NEAREST
< CELESTIAL OBJECT WHICH IS WITHIN ORBITING RANGE .
<
< DETERMINE THE NEAREST CELESTIAL OBJECT .
SET 'NEAREST CELESTIAL OBJECT' TO 'NONE' ;
SET DISTANCE TO NEAREST CELESTIAL OBJECT TO INFINITE ;
FOR I = 1 TO THE NUMBER OF CELESTIAL OBJECTS :
   CALCULATE THE DISTANCE BETWEEN THE ENTERPRISE AND THE ITH
   CELESTIAL OBJECT ;
   IF THE ITH CELESTIAL OBJECT IS CLOSER THAN THE CURRENTLY
   CALCULATED NEAREST CELESTIAL OBJECT :
```



```
      THEN SET THE ITH CELESTIAL OBJECT AS THE NEAREST CELESTIAL
      OBJECT ;
      SET THE DISTANCE OF THE NEAREST CELESTIAL OBJECT TO THE
      DISTANCE OF THE ITH CELESTIAL OBJECT .
   ENDIF
ENDFOR
IF THE NEAREST CELESTIAL OBJECT IS CLOSE ENOUGH TO ESTABLISH
ORBIT (TBD) :
   THEN SET THE ENTERPRISE VELOCITY VECTOR (DIRECTION AND
   SPEED) SO THAT ORBIT WILL BE ESTABLISHED ;
   < THE ABOVE REQUIRES CONSIDERATION OF SIZE, MASS, AND TYPE
   < OF CELESTIAL OBJECT .
   SET THE ENTERPRISE DESTINATION CODE TO –1 .
ELSE THE NEAREST CELESTIAL OBJECT IS TOO FAR AWAY TO ESTABLISH
ORBIT :
   SO NOTIFY THE NAVIGATOR .
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION POSITION UPDATE SUBMODULE (4) >
< UPDATE THE X, Y, AND Z COORDINATES OF THE ENTERPRISE .
< COMPUTE THE NEW X, Y, AND Z COORDINATES USING THE PREVIOUS
< COORDINATE POSITION, THE VELOCITY VECTOR, GRAVITATIONAL
< FIELDS OF NEARBY CELESTIAL OBJECTS, THE EFFECT OF TRACTOR
< BEAMS AND THE TIME SINCE THE LAST UPDATE .
IF ANY OBJECT EXISTS BETWEEN THE PREVIOUS POSITION AND THE NEW
POSITION :
   THEN THE ENTERPRISE HAS BEEN DESTROYED ;
   NOTIFY THE CAPTAIN OF THE END OF THE MISSION ;
   NOTIFY THE SIMULATION CONTROLLER TO TERMINATE THE MISSION .
ORIF THE ENTERPRISE IS ON A COLLISION COURSE WITH SOME OBJECT :
   NOTIFY THE NAVIGATION OFFICER OF THE ESTIMATED TIME OF
   IMPACT .
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION ENEMY SPACECRAFT INTELLIGENCE SUBMODULE (4) >
< PROCESS ALL ENEMY SPACECRAFT .
FOR I = 1 TO THE NUMBER OF ENEMY SPACECRAFT :
   IF THE ITH SPACECRAFT EXISTS :
      IF ITS MISSION IS UNCONDITIONAL ATTACK :
         CALL THE ENEMY UNCONDITIONAL ATTACK SUBMODULE .
      ORIF ITS MISSION IS CONDITIONAL ATTACK :
         CALL THE ENEMY CONDITIONAL ATTACK SUBMODULE .
      ORIF ITS MISSION IS ESTABLISH PEACE TREATY :
         CALL THE ENEMY ESTABLISH PEACE TREATY SUBMODULE .
      ORIF ITS MISSION IS CONQUER CIVILIZATION :
         CALL THE ENEMY CONQUER CIVILIZATION SUBMODULE .
      ORIF ITS MISSION IS WEAPONS DELIVERY :
         CALL THE ENEMY WEAPONS DELIVERY SUBMODULE .
```

```
      ORIF ITS MISSION IS PEACEFUL CARGO DELIVERY :
         CALL THE ENEMY PEACEFUL CARGO DELIVERY SUBMODULE .
      ELSE ITS MISSION IS UNKNOWN :
         SO NOTIFY THE SIMULATION CONTROLLER OF AN ERROR IN
         THE SIMULATION .
      ENDIF
   ENDIF
   < UPDATE ALL OF ITS OFFENSIVE WEAPONS STATUS.
   < USE A GENERAL INCREASE IN STATUS OVER TIME LESS THE
   < RELIABILITY FACTOR.
   <
   < UPDATE ALL OF ITS DEFENSIVE WEAPONS.
   < UPDATE THE INTELLIGENCE OF ITS LIFE FORMS. CAN ALSO
   < RANDOMLY INCREASE THE NUMBER OF LIFE FORMS (LIFE FORMS
   < DO REPRODUCE, YOU KNOW).
   < UPDATE ITS NAVIGATION COMPUTER STATUS.
   <
   < UPDATE THE QUANTITY OF ITS ENERGY (SLOWLY INCREASE OVER
   < TIME, CONSIDER A REGENERATIVE POWER SOURCE).
   CALL THE ENEMY CRAFT POSITION UPDATE SUBMODULE .
ENDFOR
RETURN//


< ********************************************************** >
< NAVIGATION FEDERATION SPACECRAFT INTELLIGENCE
SUBMODULE (4) >
IF THERE ARE ANY FEDERATION CRAFT IN THE UNIVERSE (OTHER THAN
THE ENTERPRISE) :
   FOR I = 1 TO THE NUMBER OF FEDERATION CRAFT :
      IF THE MISSION OF THE ITH FEDERATION CRAFT IS EXPLORATION :
         THEN CALL THE FEDERATION EXPLORATION SUBMODULE .
      ORIF ITS MISSION IS CARGO DELIVERY :
         THEN CALL THE FEDERATION CARGO DELIVERY SUBMODULE .
      ORIF ITS MISSION IS STARBASE :
         THEN CALL THE FEDERATION STARBASE SUBMODULE .
      ELSE THE MISSION IS UNKNOWN :
         SO NOTIFY THE SIMULATION CONTROLLER OF AN ERROR IN
         THE SIMULATION .
      ENDIF
      < UPDATE THE POSITION OF THE ITH FEDERATION CRAFT .
      CALL THE FEDERATION CRAFT POSITION UPDATE SUBMODULE .
   ENDFOR
ENDIF
RETURN//


< ********************************************************** >
< NAVIGATION UNIVERSE UPDATE SUBMODULE (4) >
FOR I = 1 TO THE NUMBER OF PLANETS IN THE UNIVERSE :
   UPDATE THE VELOCITY VECTOR ;
   UPDATE THE X, Y, AND Z COORDINATES;
   UPDATE THE FUNCTIONAL STATUS OF ALL THE DEFENSIVE AND
   OFFENSIVE WEAPONS ON THE ITH CELESTIAL OBJECT ;
```



```
   IF THE 'FIRED UPON' FLAG FOR THE ITH CELESTIAL OBJECT IS SET :
      CALL THE CELESTIAL OBJECT DEFENSE SUBMODULE .
   ENDIF
ENDFOR
RETURN//


< ********************************************************** >
< NAVIGATION EVASIVE TACTICS BACKGROUND SUBMODULE (5) >
IF THE EVASIVE TACTICS FLAG IS SET :
   IF THE ENTERPRISE'S DEFLECTOR SHIELDS ARE NOT UP :
      SET THEM ALL TO THEIR MAXIMUM CAPACITY .
   ENDIF
   SET THE VELOCITY VECTOR OF THE ENTERPRISE AWAY FROM THE
   NEAREST ENEMY CRAFT ;
   < NOTE: IT IS SUGGESTED THAT THE PROGRAMMER TRY SEVERAL
   < METHODS AND STRATEGIES FOR THIS SUBMODULE. THE PURPOSE
   < HERE IS TO PROVIDE THE CAPTAIN WITH A COMPUTER CONTROLLED
   < EVASIVE TACTICS CAPABILITY. IT MIGHT BE THOUGHT OF AS SIMILAR
   < TO THE M5 UNIT ON ONE OF THE 'STAR TREK' EPISODES WHICH HAD
   < THE CAPACITY TO CONTROL THE ENTIRE SHIP. I HAD ORIGINALLY
   < THOUGHT OF THIS SUBMODULE MERELY AS TACTICAL MANEUVERS
   < BUT YOU MIGHT ALSO INCLUDE WEAPON FIRING CAPABILITY AND
   < SO ON.
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< ENEMY SPACECRAFT CONDITIONAL ATTACK SUBMODULE (5) >
IF THE CRAFT WAS FIRED UPON :
   < THEN FIRE BACK, CALL FOR HELP, AND TURN ON OFFENSIVE
   < WEAPONS .
   CALL THE ENEMY WEAPONS FIRING SUBMODULE ;
   IF THE CRAFT HAS A CLOAKING DEVICE :
      TURN IT ON .
   ENDIF
   IF THE CRAFT HAS DEFLECTOR SHIELDS :
      TURN THEM ON .
   ENDIF
   < SEND OUT A CALL FOR HELP TO OTHER ENEMY SPACECRAFT .
   < NOTE THAT RECEPTION IS RANDOM .
   GENERATE A RANDOM NUMBER ;
   IF THE RANDOM NUMBER IS ABOVE SOME (TBD) LEVEL :
      < THEN ASSUME THAT THE CALL FOR HELP WILL BE RECEIVED .
      < DETERMINE CLOSEST ENEMY CRAFT (WHICH, OF COURSE, IS
      < CONSIDERED A FRIENDLY TO THIS CRAFT) .
      CLOSEST = 'NONE' ;
      DISTANCE = INFINITE ;
      FOR I = 1 TO THE NUMBER OF ENEMY CRAFT :
         DETERMINE ITS DISTANCE TO THE CURRENT CRAFT ;
         IF THE ITH ENEMY CRAFT EXISTS AND IS CLOSER THAN THE
         CURRENT CLOSEST ENEMY CRAFT :
```

```
            SET THE ITH ENEMY CRAFT CLOSEST ;
            SET THE DISTANCE TO THE DISTANCE OF THE ITH ENEMY CRAFT .
         ENDIF
      ENDFOR
      IF THERE IS A CLOSEST CRAFT :
         SET ITS DESTINATION TO BE THE FEDERATION CRAFT WHICH
         SET THE ENEMY'S FIRED UPON FLAG ;
         SET ITS MISSION TO UNCONDITIONAL ATTACK .
      ENDIF
   ENDIF
   < REQUEST A PEACE TREATY (BASED ON DAMAGE LEVEL AND A
   < RANDOM FUNCTION) .
   DETERMINE OVERALL DAMAGE STATUS INCLUDING SUCH THINGS AS
   NUMBER OF DEAD OR INJURED PERSONNEL, OFFENSIVE AND
   DEFENSIVE WEAPONS STATUS, NAVIGATIONAL COMPUTER STATUS,
   AND A RANDOM FUNCTION ;
   IF THE OVERALL DAMAGE STATUS IS ABOVE SOME (TBD) LEVEL :
      THEN NOTIFY THE COMMUNICATIONS OFFICER OF THE
      ENTERPRISE OF A REQUEST FOR A PEACE TREATY ;
      SET THE REQUEST PEACE TREATY FLAG FOR THIS ENEMY CRAFT .
   ELSE THE DAMAGE STATUS IS BELOW THE (TBD) LEVEL :
      SO CLEAR THE PEACE TREATY REQUEST FLAG FOR THIS ENEMY
      CRAFT .
      < THIS CAN CAUSE DECEPTIVE PEACE TREATY OFFERS .
      < IF THE FLAG IS NOT SET WHEN THE ENTERPRISE ACCEPTS THE
      < OFFER THE ENEMY EFFECTIVELY RENEGS ON THE DEAL .
   ENDIF
ELSE THE ENEMY CRAFT HAS NOT BEEN FIRED UPON :
   < SO CONTINUE ON ITS WAY ROAMING THE UNIVERSE .
   IF COURSE CHANGE TIME HAS PASSED :
      RANDOMLY CHANGE DIRECTION AND SPEED OR DESTINATION ;
      SET NEW COURSE CHANGE TIME TO THE CURRENT TIME PLUS SOME
      RANDOM AMOUNT OF TIME (THE RANGE OF WHICH IS TBD) .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< ENEMY SPACECRAFT UNCONDITIONAL ATTACK SUBMODULE (5) >
IF THE CRAFT WAS FIRED UPON :
   < THIS SECTION IS THE SAME AS WITH THE CONDITIONAL ATTACK
   < TYPE FOR FIRED UPON STATUS EXCEPT THAT ITS CONDITION FOR
   < OFFERING A PEACE TREATY SHOULD BE MORE STRINGENT (TO
   < CAUSE A 'HOLD OUT TO THE VERY LAST MAN' ATTITUDE).
ELSE THE CRAFT WAS NOT FIRED UPON :
   < SO CONTINUE LOOKING FOR FRIENDLY CRAFT TO ATTACK .
   < DETERMINE WHICH FEDERATION CRAFT IS CLOSEST .
   CLOSEST = 'NONE' ;
   DISTANCE = INFINITE ;
   FOR I = 1 TO THE NUMBER OF FEDERATION CRAFT :
      DETERMINE THE DISTANCE BETWEEN THIS ENEMY CRAFT AND THE
      ITH FEDERATION CRAFT ;
```



```
      IF THE ITH FEDERATION CRAFT IS CLOSER THAN THE CURRENT
      CLOSEST :
         SET THE ITH FEDERATION CRAFT AS CLOSEST ;
         SET THE DISTANCE EQUAL TO THE DISTANCE BETWEEN THIS
         ENEMY CRAFT AND THE ITH FEDERATION CRAFT .
      ENDIF
   ENDFOR
   < DETERMINE IF A SHUTTLECRAFT IS CLOSER .
   FOR I = 1 TO THE NUMBER OF SHUTTLECRAFT :
      DETERMINE THE DISTANCE TO THE ITH SHUTTLECRAFT FROM THE
      CURRENT ENEMY CRAFT ;
      IF THE DISTANCE IS CLOSER THAN THE CURRENT CLOSEST :
         THEN SET THE ITH SHUTTLECRAFT AS CLOSEST ;
         SET THE DISTANCE EQUAL TO THE DISTANCE BETWEEN THE ITH
         SHUTTLECRAFT AND THE CURRENT ENEMY CRAFT .
      ENDIF
   ENDFOR
   < DETERMINE IF THE ENTERPRISE IS CLOSEST .
   DETERMINE THE DISTANCE FROM THE CURRENT ENEMY CRAFT TO
   THE ENTERPRISE ;
   IF THE ENTERPRISE IS CLOSER THAN THE CURRENT CLOSEST :
      THEN SET THE ENTERPRISE AS CLOSEST ;
      SET THE DISTANCE EQUAL TO THE DISTANCE TO THE ENTERPRISE .
   ENDIF
   < NOW THAT IT IS KNOWN WHO IS CLOSEST
   IF THERE IS A CLOSEST :
      IF THE CLOSEST CRAFT IS WITHIN FIRING RANGE (TBD) :
         < THEN START A BATTLE .
         IF THE ENEMY CRAFT HAS A CLOAKING DEVICE :
            THEN TURN IT ON .
         ENDIF
         IF THE ENEMY CRAFT HAS DEFLECTOR SHIELDS :
            THEN TURN THEM ON .
         ENDIF
         CALL THE ENEMY WEAPONS FIRING SUBMODULE ;
         < HAVE THE ENEMY TAKE EVASIVE TACTICS .
         IF THE COURSE CHANGE TIME FOR THE CURRENT ENEMY CRAFT
         HAS PASSED :
            THEN RANDOMLY CHANGE COURSE BUT STAY NEAR THE
            FEDERATION CRAFT WITH WHICH THE BATTLE IS ENGAGED ;
            SET NEW COURSE CHANGE TIME FOR THE CURRENT ENEMY
            CRAFT TO THE CURRENT TIME PLUS A RANDOM AMOUNT OF
            TIME .
            < (THE RANGE OF THE RANDOM TIME TO ADD IS TBD.)
         ENDIF
      ORIF THE CLOSEST FEDERATION CRAFT IS WITHIN DETECTION
      DISTANCE (TBD) I. E. ABLE TO BE DETECTED BY THE CURRENT
      ENEMY CRAFT. (DETECTION RANGE MUST BE GREATER THAN
      FIRING RANGE) :
         THEN SET THE DESTINATION OF THE CURRENT ENEMY CRAFT
         TO THE CLOSEST FEDERATION CRAFT .
      ELSE NOT WAGING A BATTLE AND NO FEDERATION CRAFT IN
      SIGHT :
```

```
        < SO CONTINUE TO ROAM THE UNIVERSE.
        IF THE COURSE CHANGE TIME FOR THE CURRENT ENEMY CRAFT
        HAS PASSED :
            RANDOMLY CHANGE COURSE ;
            SET NEW COURSE CHANGE TIME FOR THE CURRENT ENEMY
            CRAFT TO THE CURRENT TIME PLUS A RANDOM AMOUNT OF
            TIME ;
            < THE RANGE OF THE TIME TO ADD IS TBD.
        ENDIF
      ENDIF
   ENDIF
RETURN//


< ********************************************************** >
< ENEMY SPACECRAFT ESTABLISH PEACE TREATY SUBMODULE (5) >
< PEACE TREATY SPACECRAFT SHOULD HAVE NO WEAPONS.
IF THIS ENEMY CRAFT WAS FIRED UPON :
   SET PEACE TREATY REQUEST FLAG FOR THE CURRENT ENEMY CRAFT ;
   IF THE CRAFT THAT FIRED WAS THE ENTERPRISE :
      NOTIFY THE COMMUNICATIONS OFFICER OF THE PEACE TREATY
      REQUEST.
   ENDIF
   SET THE COURSE OF THE ENEMY CRAFT AWAY FROM THE CRAFT
   THAT FIRED ;
   < REQUEST AID FROM ANOTHER ENEMY CRAFT.
   < SEE 'SEND OUT CALL FOR HELP' IN ENEMY CONDITIONAL ATTACK
   < SUBMODULE.
   CLEAR THE FIRED UPON FLAG FOR THE CURRENT ENEMY CRAFT.
ELSE NOT FIRED UPON :
   < DETERMINE THE CLOSEST FEDERATION CRAFT.
   < NOTE: SEE 'DETERMINE WHICH FRIENDLY IS CLOSEST' IN THE
   < UNCONDITIONAL ATTACK SUBMODULE.
   IF THERE IS A CLOSEST FEDERATION CRAFT :
      IF IT IS WITHIN DETECTION RANGE (TBD) :
         SET THE DESTINATION OF THE CURRENT ENEMY TO THE
         CLOSEST FEDERATION CRAFT.
      ENDIF
   ENDIF
RETURN//


< ********************************************************** >
< ENEMY CRAFT CONQUER CIVILIZATION SUBMODULE (5) >
IF THIS ENEMY CRAFT WAS FIRED UPON :
   SET MISSION OF THE CURRENT ENEMY TO UNCONDITIONAL ATTACK.
ELSE NOT FIRED UPON :
< SO CONTINUE SEARCHING FOR A CIVILIZATION TO CONQUER.
< DETERMINE CLOSEST CELESTIAL OBJECT.
IF THERE ARE ANY CELESTIAL OBJECTS :
   CLOSEST = 'NONE' ;
   DISTANCE = INFINITE ;
   FOR I = 1 TO THE NUMBER OF CELESTIAL OBJECTS :
      CALCULATE THE DISTANCE FROM THE ITH CELESTIAL OBJECT TO
```



```
      THE CURRENT ENEMY CRAFT ;
      IF THE DISTANCE IS LESS THAN THE CURRENT CLOSEST :
         SET THE ITH CELESTIAL OBJECT AS THE CLOSEST ;
         SET THE DISTANCE EQUAL TO THE DISTANCE BETWEEN THE ITH
         CELESTIAL OBJECT AND THE CURRENT ENEMY CRAFT.
      ENDIF
   ENDFOR
   IF THE CLOSEST CELESTIAL OBJECT IS WITHIN SENSOR RANGE (TBD)
   OF THE CURRENT ENEMY CRAFT :
      < THEN IT IS A LIKELY CANDIDATE FOR ATTACK.
      IF THE CLOSEST CELESTIAL OBJECT HAS LIFE FORMS :
         SET THE DESTINATION OF THE ENEMY CRAFT TO THE CLOSEST
         CELESTIAL OBJECT.
      ELSE IT HAS NO LIFE FORMS :
         < SO SEEK OUT SOME OTHER CELESTIAL OBJECT.
         SET THE VELOCITY VECTOR OF THE CURRENT ENEMY CRAFT
         AWAY FROM THE CLOSEST CELESTIAL OBJECT.
      ENDIF
   ORIF THE CLOSEST CELESTIAL OBJECT IS WITHIN FIRING RANGE (TBD)
   OF THE CURRENT ENEMY CRAFT (FIRING RANGE MUST BE CLOSER
   THAN SENSOR DETECTION RANGE) :
      CALL THE ENEMY WEAPONS FIRING SUBMODULE.
   ORIF THE CLOSEST CELESTIAL OBJECT IS WITHIN ORBITTING RANGE
   (TBD) OF THE CURRENT ENEMY CRAFT :
      THEN SET THE VELOCITY VECTOR OF THE CURRENT ENEMY CRAFT
      SO THAT ORBIT WILL BE ESTABLISHED.
      < NOTE THAT DETERMINATION OF THE VELOCITY VECTOR TO
      < ESTABLISH ORBIT IS BASED UPON THE SIZE, MASS, AND TYPE
      < OF CELESTIAL OBJECT, AND ALSO IS RELATED TO THE
      < FUNCTIONAL STATUS OF THE NAVIGATION COMPUTER ON
      < BOARD THE CURRENT ENEMY CRAFT.
   ELSE THE CLOSEST CELESTIAL OBJECT IS STILL TOO FAR AWAY :
      IF COURSE CHANGE TIME FOR THE CURRENT ENEMY CRAFT HAS
      PASSED :
         CHANGE THE COURSE OF THE CURRENT ENEMY CRAFT
         RANDOMLY ;
         SET THE COURSE CHANGE TIME OF THE CURRENT ENEMY CRAFT
         TO THE CURRENT TIME PLUS SOME RANDOM AMOUNT OF TIME.
         < THE RANGE OF THE TIME TO BE ADDED IS TBD.
      ENDIF
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< ENEMY SPACECRAFT WEAPONS DELIVERY SUBMODULE (5) >
IF FIRED UPON :
   CHANGE MISSION TO UNCONDITIONAL ATTACK.
ELSE NOT FIRED UPON :
   < SO CONTINUE WITH THE MISSION.
   DETERMINE DISTANCE TO DESTINATION ;
   < NOTE: ALL WEAPONS DELIVERY ENEMY CRAFT MUST HAVE
   < ANOTHER ENEMY CRAFT AS ITS DESTINATION.
```

```
    IF WITHIN TRANSFER DISTANCE (TBD) :
        <THEN TRANSFER WEAPONS TO THE DESTINATION ENEMY CRAFT.
        <THIS MEANS INCREASING THE FUNCTIONAL STATUS OF THE
        <DESTINATION CRAFT'S WEAPONS, ADDING WEAPONS TO THE
        <OTHER CRAFT, INCREASING THE ENERGY SUPPLY OF THE OTHER
        <CRAFT, AND PROPORTIONALLY DECREASING THE CARGO OF
        <THE DELIVERY CRAFT.
        SET THE DESTINATION OF THE DELIVERY CRAFT RANDOMLY TO
        ONE OF THE OTHER ENEMY CRAFT.
    ENDIF
ENDIF
RETURN//


<********************************************************>
<ENEMY SPACECRAFT PEACEFUL CARGO DELIVERY SUBMODULE (5) >
IF FIRED UPON :
    CHANGE THE MISSION OF THE CURRENT ENEMY CRAFT TO
    UNCONDITIONAL ATTACK.
ELSE NOT FIRED UPON :
    <SAME AS WEAPONS DELIVERY EXCEPT THAT CARGO TRANSFERRED
    <IS ENERGY AND UPDATE OF PERSONNEL HEALTH STATUS AND
    <NUMBER OF PERSONNEL.
ENDIF
RETURN//


<********************************************************>
<ENEMY SPACECRAFT POSITION UPDATE SUBMODULE (5) >
FOR I = 1 TO THE NUMBER OF ENEMY CRAFT :
    IF THE DESTINATION OF THE ITH ENEMY CRAFT IS SOME SPECIFIC
    OBJECT :
        <NOTE: THIS ALLOWS A CRAFT TO ATTACK ITS DESTINATION.
        IF THE SPECIFIED OBJECT STILL EXISTS :
            SET THE DIRECTION OF THIS ENEMY CRAFT TOWARD THE
            SPECIFIED OBJECT ;
            SET THE VELOCITY OF THIS ENEMY CRAFT TO MID-RANGE.
        ELSE THE SPECIFIED DESTINATION OBJECT NO LONGER EXISTS :
            SO SET THE DESTINATION OF THIS ENEMY CRAFT TO A RANDOM
            X, Y, Z COORDINATE AND SET ITS DESTINATION CODE TO -1.
        ENDIF
    ENDIF
    UPDATE THE X, Y, Z POSITION OF THIS ENEMY CRAFT RELATIVE TO
    ITS DIRECTION, VELOCITY, AND TIME SINCE THE LAST POSITION
    UPDATE ;
    IF THERE IS AN OBJECT BETWEEN ITS PREVIOUS LOCATION AND ITS
    CURRENT LOCATION :
        <KAPOW!!
        DELETE THE ENEMY CRAFT FROM THE UNIVERSE ;
        REGISTER APPROPRIATE DAMAGE TO THE OBJECT IT HIT.
    ENDIF
ENDFOR
RETURN//
```



```
<********************************************************>
<FEDERATION EXPLORATION SUBMODULE (5) >
IF THIS FEDERATION CRAFT WAS FIRED UPON :
    <SEND OUT A CALL FOR HELP.
    <NOTE: THIS IS DONE IN THE SAME WAY AS FOR THE 'SEND DISTRESS'
    <MESSAGE FOR THE ENTERPRISE.
    SET THE VELOCITY VECTOR OF THIS FEDERATION CRAFT AWAY FROM
    THE CRAFT THAT FIRED AT MAXIMUM SPEED.
ELSE THIS FEDERATION CRAFT WAS NOT FIRED UPON :
    <SO CONTINUE EXPLORING. WHENEVER AN EXPLORATION CRAFT
    <GETS WITHIN SENSOR RANGE (TBD) OF A CELESTIAL OBJECT THE
    <OBJECT BECOMES CHARTED.
    FOR I = 1 TO THE NUMBER OF CELESTIAL OBJECTS :
        DETERMINE THE DISTANCE BETWEEN THIS FEDERATION CRAFT
        AND THE ITH CELESTIAL OBJECT ;
        IF THE ITH CELESTIAL OBJECT IS WITHIN SENSOR RANGE (TBD) OF
        THIS FEDERATION CRAFT :
            SET THE 'CHARTED' FLAG OF THE ITH CELESTIAL OBJECT.
        ENDIF
    ENDFOR
    <WHENEVER AN ENEMY CRAFT COMES WITHIN SENSOR RANGE OF A
    <FEDERATION EXPLORATION CRAFT A MESSAGE IS SENT TO THE
    <ENTERPRISE.
    FOR I = 1 TO THE NUMBER OF ENEMY CRAFT :
        DETERMINE THE DISTANCE BETWEEN THIS FEDERATION CRAFT
        AND THE ITH ENEMY CRAFT ;
        IF THE ITH ENEMY CRAFT IS WITHIN SENSOR RANGE OF THIS
        FEDERATION EXPLORATION CRAFT :
            SEND A MESSAGE TO THE ENTERPRISE TO NOTIFY IT OF THE
            LOCATION AND TYPE OF ENEMY CRAFT.
            <NOTE: RECEPTION OF THE MESSAGE BY THE COMMUNICATIONS
            <OFFICER IS DETERMINED AS IN THE 'SEND DISTRESS MESSAGE'
            <SUBMODULE. THE AMOUNT AND QUALITY OF INFORMATION
            <RECEIVED DEPENDS UPON THE DISTANCE TO THE ENEMY,
            <WHETHER CLOAKING DEVICES ARE IN EFFECT, AND SO ON.
    ENDIF
ENDFOR
ENDIF
RETURN//


<********************************************************>
<FEDERATION CARGO DELIVERY SUBMODULE (5) >
<NOTE: ALL CARGO DELIVERY FEDERATION CRAFT MUST HAVE
<ANOTHER FEDERATION CRAFT AS ITS DESTINATION.
IF THIS FEDERATION CRAFT HAS BEEN FIRED UPON :
    <RESPOND IN THE SAME WAY AS FOR FEDERATION EXPLORATION
    <CRAFT.
ELSE THIS FEDERATION CRAFT HAS NOT BEEN FIRED UPON :
    IF THIS FEDERATION CRAFT IS WITHIN CARGO TRANSFER DISTANCE
    (TBD) OF ITS DESTINATION CRAFT (AND ITS DESTINATION CRAFT IS
    NOT A STAR BASE) :
```

```
      < THEN EFFECT A TRANSFER BY INCREASING THE ENERGY,
      < WATER, ETC. OF THE DESTINATION CRAFT ACCORDING TO THE
      < TYPE OF CARGO ON THE DELIVERY CRAFT AND REDUCE THE
      < CARGO PROPORTIONALLY ON THE DELIVERY CRAFT.
      IF THE CARGO ON THE DELIVERY CRAFT IS NOW ZERO :
         SET THE DESTINATION OF THE DELIVERY CRAFT TO A STAR
         BASE .
      ELSE THE CARGO QUANTITY IS NOT ZERO :
         SO SET THE DESTINATION OF THIS DELIVERY CRAFT TO ONE OF
         THE OTHER FEDERATION CRAFT (NOT THE ONE JUST
         DELIVERED TO) .
      ENDIF
   ORIF THIS CRAFT IS WITHIN CARGO TRANSFER DISTANCE (TBD) OF ITS
   DESTINATION CRAFT AND THE DESTINATION CRAFT IS A STAR BASE :
      < THEN TRANSFER CARGO FROM THE STAR BASE TO THE
      < DELIVERY CRAFT .
      SET THE DESTINATION OF THE DELIVERY CRAFT TO ONE OF THE
      OTHER FEDERATION CRAFT .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< FEDERATION STAR BASE SUBMODULE (5) >
IF THIS STAR BASE HAS BEEN FIRED UPON :
   THEN SEND OUT A DISTRESS MESSAGE AS IN THE 'SEND DISTRESS
   MESSAGE' SUBMODULE FOR THE ENTERPRISE .
   < NOTE THAT IF THE ENTERPRISE CAN RECEIVE THE DISTRESS CALL
   < THEN THE ENTERPRISE IS NOTIFIED VIA THE MESSAGE AREA IN
   < THE COMMUNICATIONS MODULE. THE ENTERPRISE'S COURSE
   < SHOULD NOT AUTOMATICALLY BE CHANGED, HOWEVER, AS WOULD
   < BE THE CASE IF SOME OTHER FEDERATION CRAFT RECEIVED THE
   < DISTRESS CALL.
   CALL THE FEDERATION FIRE WEAPONS SUBMODULE ;
   CLEAR THE FIRED UPON FLAG .
ENDIF
RETURN//


< ********************************************************** >
< FEDERATION CRAFT POSITION UPDATE SUBMODULE (5) >
IF THIS FEDERATION CRAFT HAS A SPECIFIED OBJECT AS ITS
DESTINATION :
   THEN SET THE VELOCITY VECTOR OF THIS FEDERATION CRAFT
   TOWARD THE SPECIFIED OBJECT .
   < NOTE: THIS ALLOWS A FEDERATION CRAFT TO 'TRACK' AN OBJECT .
ENDIF
UPDATE THE X, Y, Z POSITION OF THIS FEDERATION CRAFT THE SAME AS
UPDATING ENEMY CRAFT X, Y, Z POSITIONS .
RETURN//
```



```
< ********************************************************** >
< CELESTIAL OBJECT DEFENSE SUBMODULE (5) >
< NOTE: THIS SUBMODULE IS CALLED ONLY IF THE CELESTIAL OBJECT
< HAS BEEN FIRED UPON .
IF THE CELESTIAL OBJECT HAS DEFLECTOR SHIELDS :
   TURN THEM ON TO THEIR MAXIMUM POWER .
ENDIF
IF THE CELESTIAL OBJECT HAS OFFENSIVE WEAPONS :
   FOR I = 1 TO THE NUMBER OF OFFENSIVE WEAPONS ON THIS
   CELESTIAL OBJECT :
      IF THE WAIT TIME HAS PASSED FOR THE ITH WEAPON :
         < THEN IT IS READY TO FIRE AGAIN .
         < SO FIRE THE WEAPON AT THE CRAFT WHICH FIRED UPON THIS
         < CELESTIAL OBJECT .
         INFLICT APPROPRIATE DAMAGE TO THE TARGET ;
         < SEE THE ENTERPRISE INFLICT DAMAGE SUBMODULE FOR
         < DAMAGE INFLICTION INFORMATION .
         REDUCE THE ENERGY SUPPLY FOR THIS CELESTIAL OBJECT
         ACCORDINGLY;
         RESET THE WEAPON FIRING WAIT TIME FOR THIS OFFENSIVE
         WEAPON .
      ENDIF
   ENDFOR
ENDIF
RETURN//


< ********************************************************** >
< ENEMY WEAPONS FIRING SUBMODULE (6) >
< TRY TO FIRE ALL OF THE WEAPONS ON THE CURRENT ENEMY CRAFT .
IF THERE ARE ANY OFFENSIVE WEAPONS ON THE CURRENT ENEMY
CRAFT :
   FOR I = 1 TO THE NUMBER OF OFFENSIVE WEAPONS :
      IF THE ASSOCIATED WEAPON FIRING WAIT TIME HAS PASSED :
         < FIRE AT THE TARGET .
         < DAMAGE MAY OCCUR TO ANY OF THE TARGET'S DEVICES .
         < FOR THE ENTERPRISE AS A TARGET ANY DEVICE WITH A
         < 'FUNCTIONAL STATUS' OR 'RELIABILITY FACTOR', IS A VALID
         < CANDIDATE FOR DAMAGE. THE DAMAGE IS INDICATED BY
         < REDUCING FUNCTIONAL STATUS OR RELIABILITY FACTOR.
         < SUCH OTHER THINGS AS WATER POLLUTION LEVEL,
         < PERSONNEL HEALTH STATUS, (THEY MAY BE KILLED) ETC.
         < MAY ALSO BE DAMAGED. THE CHOICE AND NUMBER OF
         < TARGET DEVICES WHICH RECEIVE DAMAGE IS RELATED TO
         < THE TYPE OF WEAPON FIRED, THE FUNCTIONAL STATUS AND
         < RELIABILITY OF THE WEAPON, THE DISTANCE TO THE
         < TARGET, AND THE DEFLECTOR SHIELDS STATUS OF THE
         < TARGET.
         RESET THE WEAPON FIRING WAIT TIME ;
         SET THE FIRED UPON FLAG OF THE TARGET .
      ENDIF
   ENDFOR
RETURN//
```

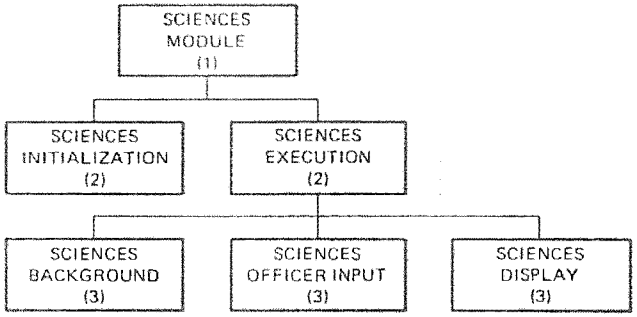


Figure 3.23 SCIENCES: SUBMODULE INTERRELATIONSHIPS

LOCAL DATA

SCAN POINTER	[SEE ENTERPRISE PERSONNEL [DESTINATION CODES FOR [CELESTIAL OBJECTS, FEDERATION [AND ENEMY CRAFT, AND [SHUTTLECRAFT [−1 = LOCATION IS BEING SCANNED
LOCATION BEING SCANNED	
X-COORDINATE	[- INFINITY TO + INFINITY
Y-COORDINATE	[- INFINITY TO + INFINITY
Z-COORDINATE	[- INFINITY TO + INFINITY
RANDOM SCAN WAIT TIME	
RANDOM SCAN FLAG	[0 = NOT SCANNING RANDOMLY [1 = SCANNING RANDOMLY

Figure 3.24 SCIENCES MODULE

```
< ********************************************************** >
< SCIENCES MODULE (1) >
IF SCIENCES INITIALIZATION FLAG IS SET :
   CALL THE SCIENCES INITIALIZATION SUBMODULE .
ELSE THE SCIENCES INTIALIZATION FLAG IS CLEAR :
   IF THE RUN FLAG IS SET :
      CALL THE SCIENCES EXECUTION SUBMODULE .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< SCIENCES INITIALIZATION SUBMODULE (2) >
SET ALL SCIENCES VARIABLES TO NOMINAL ;
CLEAR THE SCIENCES MODULE INTIALIZATION FLAG .
RETURN//
```



```
< ********************************************************** >
< SCIENCES EXECUTION SUBMODULE (2) >
CALL THE SCIENCES BACKGROUND SUBMODULE ;
CALL THE SCIENCES OFFICER INPUT SUBMODULE ;
CALL THE SCIENCES DISPLAY SUBMODULE .
RETURN//


< ********************************************************** >
< SCIENCES BACKGROUND SUBMODULE (3) >
< UPDATE THE STATUS OF THE SENSOR COMPUTER AND EACH OF THE
< ENTERPRISE SENSORS. THE STATUS OF EACH IS DEPENDENT UPON THE
< TIME SINCE THE LAST UPDATE AND THE NUMBER OF MAINTENANCE
< PERSONNEL AT EACH SENSOR STATION .
UPDATE THE ENERGY DRAIN OF THE SENSORS ;
IF THE RANDOM SCAN FLAG IS SET :
   < THEN THE SCANNING IS RANDOM
   IF THE WAIT TIME HAS PASSED :
      < THEN IT IS TIME TO LOOK AT ANOTHER AREA .
      SET THE X, Y, Z COORDINATES TO A RANDOM POINT WITHIN
      SENSOR RANGE (TBD) ;
      SET THE WAIT TIME TO THE CURRENT TIME PLUS A RANDOM
      WAITING PERIOD .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< SCIENCES OFFICER INPUT SUBMODULE (4) >
< NOTE: SENSOR RANGE IS A FUNCTION OF THE FUNCTIONAL STATUS
< OF THE SENSORS, THE ENERGY SUPPLY AVAILABLE TO THE SENSORS,
< ETC.
IF THE SCIENCE OFFICER HAS GIVEN A COMMAND :
   IF IT IS A COMMAND TO SCAN A SPECIFIED OBJECT :
      IF THE SPECIFIED OBJECT NAME IS KNOWN :
         SET THE SCAN POINTER TO THE CODE OF THE SPECIFIED OBJECT;
         CLEAR THE RANDOM FLAG .
      ELSE THE SPECIFIED OBJECT NAME IS NOT KNOWN :
         SO NOTIFY THE SCIENCE OFFICER .
      ENDIF
   ORIF THE COMMAND IS TO SCAN AN AREA :
      IF THE SPECIFIED AREA IS WITHIN THE UNIVERSE :
         SET THE COORDINATES OF THE AREA TO BE SCANNED ;
         SET THE SCAN POINTER TO −1 TO INDICATE AN AREA IS BEING
         SCANNED ;
         CLEAR THE RANDOM FLAG .
      ELSE THE SPECIFIED AREA IS NOT WITHIN THE UNIVERSE :
         SO NOTIFY THE SCIENCE OFFICER .
      ENDIF
   ORIF THE COMMAND IS TO SCAN RANDOMLY :
      SET THE SCAN COORDINATES TO A RANDOM AREA WITHIN SENSOR
      RANGE ;
      SET THE RANDOM FLAG ;
```

```
        SET THE WAIT TIME (TBD) ;
        SET THE POINTER TO –1 TO INDICATE THAT AN AREA IS BEING
        SCANNED .
    ELSE THE COMMAND IS UNKNOWN :
        NOTIFY THE SCIENCES OFFICER THAT HE HAS GIVEN AN UNKNOWN
        COMMAND .
    ENDIF
ENDIF
RETURN//


< ********************************************************** >
< SCIENCES DISPLAY SUBMODULE (4) >
< NOTE: THE QUALITY AND AMOUNT OF INFORMATION DISPLAYED IS
< RELATED TO THE DISTANCE TO THE SCANNED AREA OR OBJECT,
< SHIELD AND CLOAKING DEVICE STATUS OF THE OBJECT, AND THE
< FUNCTIONAL STATUS OF THE ENTERPRISE'S SENSORS.
IF SCANNING AN OBJECT :
    DISPLAY APPROPRIATE INFORMATION ABOUT THE OBJECT .
```

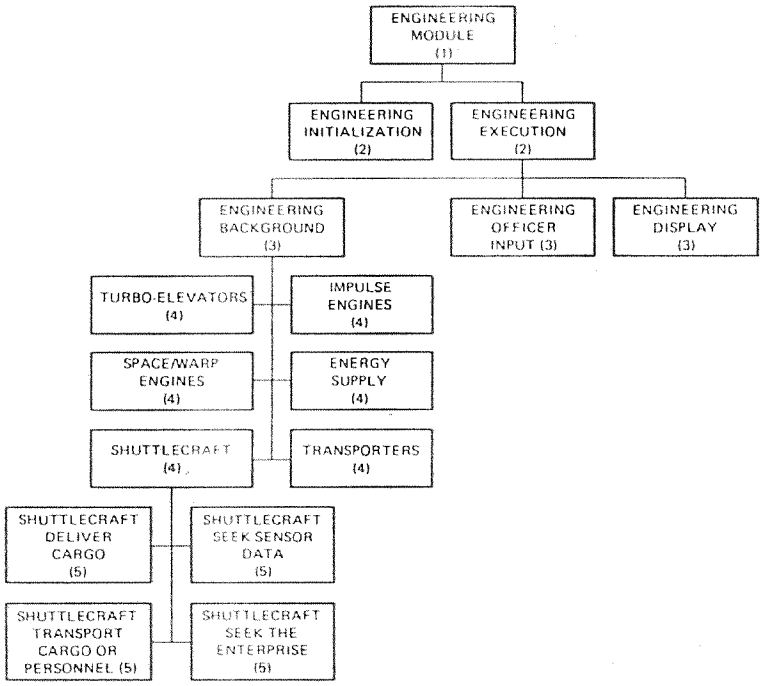


Figure 3.25 ENGINEERNG: SUBMODULE INTERRELATIONSHIPS


```
ORIF SCANNING AN AREA :
    IF AN OBJECT IS WITHIN THE AREA (OF RANDOM RADIUS) :
        DISPLAY THE APPROPRIATE INFORMATION ABOUT THE OBJECT ;
        IF RANDOM SCANNING IS IN EFFECT :
            THEN SET THE SCAN WAIT TIME LONG ENOUGH SO THAT THE
            OPERATOR CAN HAVE TIME TO VIEW THE INFORMATION ;
            NOTIFY THE SCIENCE OFFICER THAT DATA ARE BEING
            DISPLAYED .
        ENDIF
    ELSE THERE IS NO OBJECT IN THE AREA :
        SO NOTIFY THE SCIENCE OFFICER THAT NO OBJECT WAS FOUND .
    ENDIF
ENDIF
RETURN//
```

LOCAL DATA

DISPLAY FLAGS FOR EACH OF THE ENGINEERING DEVICES

Figure 3.26 ENGINEERING MODULE

```
< ********************************************************** >
< ENGINEERING MODULE (1) >
IF THE INITIALIZATION FLAG IS SET :
    CALL THE ENGINEERING INITIALIZATION SUBMODULE .
ELSE THE INITIALIZATION FLAG IS CLEAR :
    IF THE RUN FLAG IS SET :
        CALL THE ENGINEERING EXECUTION SUBMODULE .
    ENDIF
ENDIF
RETURN//


< ********************************************************** >
< ENGINEERING INITIALIZATION SUBMODULE (2) >
SET ALL COMMON DATA ASSOCIATED WITH THE ENGINEERING MODULE
TO NOMINAL ;
CLEAR THE ENGINEERING MODULE INTIALIZATION FLAG .
RETURN//


< ********************************************************** >
< ENGINEERING EXECUTION SUBMODULE (2) >
CALL THE ENGINEERING BACKGROUND SUBMODULE ;
CALL THE ENGINEERING DISPLAY SUBMODULE ;
CALL THE ENGINEERING OPERATOR INPUT SUBMODULE .
RETURN//
```

```
< ********************************************************** >
< ENGINEERING BACKGROUND SUBMODULE (3) >
CALL THE ENGINEERING SHUTTLECRAFT SUBMODULE ;
CALL THE ENGINEERING TRANSPORTERS SUBMODULE ;
CALL THE ENGINEERING ENERGY SUPPLY SUBMODULE ;
CALL THE ENGINEERING SPACE/WARP ENGINES SUBMODULE ;
CALL THE ENGINEERING IMPULSE ENGINES SUBMODULE ;
CALL THE ENGINEERING TURBO-ELEVATORS SUBMODULE .
RETURN//


< ********************************************************** >
< ENGINEERING DISPLAY SUBMODULE (3) >
< UPDATE THE ENGINEERING DISPLAY .
< SEE THE PRELIMINARY FUNCTIONAL DISPLAY, FIGURE 3.14 .
RETURN//


< ********************************************************** >
< ENGINEERING OPERATOR INPUT SUBMODULE (3) >
IF THE ENGINEERING OFFICER HAS GIVEN A COMMAND :
   IF THE COMMAND IS SET ENERGY SUPPLY TO A DEVICE :
      IF DEVICE CODE IS KNOWN :
         SET THE ENERGY SUPPLY TO THAT DEVICE .
      ELSE THE SPECIFIED DEVICE CODE IS NOT KNOWN :
         SO NOTIFY THE ENGINEER .
      ENDIF
   ORIF THE COMMAND IS SET ENERGY SUPPLY TO A SHIP SECTION :
      IF THE SPECIFIED SHIP SECTION CODE IS KNOWN :
         SET THE ENERGY SUPPLY TO ALL DEVICES IN THE SPECIFIED
         SHIP SECTION .
         < SEE FIGURE 3.13 .
      ELSE THE SPECIFIED SHIP SECTION IS NOT KNOWN :
         SO NOTIFY THE ENGINEER .
      ENDIF
   ORIF THE COMMAND IS REPORT STATUS OF A SPECIFIED DEVICE :
      IF THE DEVICE CODE IS KNOWN :
         CLEAR ALL DEVICE DISPLAY FLAGS ;
         SET THE DEVICE DISPLAY FLAG FOR THE SPECIFIED DEVICE .
      ELSE THE SPECIFIED DEVICE FLAG IS NOT KNOWN :
         SO NOTIFY THE ENGINEER .
      ENDIF
   ORIF THE COMMAND IS SET SHUTTLECRAFT MISSION :
      IF SPECIFIED SHUTTLECRAFT IS KNOWN :
         IF SPECIFIED MISSION CODE IS KNOWN :
            SET THE SPECIFIED SHUTTLECRAFT TO THE SPECIFIED
            MISSION .
         ELSE THE SPECIFIED MISSION CODE IS NOT KNOWN :
            SO NOTIFY THE ENGINEER .
         ENDIF
      ELSE THE SPECIFIED SHUTTLECRAFT IS NOT KNOWN :
         SO NOTIFY THE ENGINEER .
      ENDIF
   ELSE THE COMMAND IS UNKNOWN :
```



```
      SO NOTIFY THE ENGINEERING OFFICER .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< ENGINEERING SHUTTLECRAFT SUBMODULE (4) >
< UPDATE THE OPERATIONAL AND FUNCTIONAL STATUS OF EACH OF
< THE SYSTEMS ON ALL OF THE SHUTTLECRAFT .
< UPDATE THE X, Y, Z LOCATION OF ALL OF THE SHUTTLECRAFT BASED
< ON THEIR CURRENT LOCATION AND DESTINATION .
< EACH SHUTTLECRAFT HAS A PARTICULAR MISSION. EACH MISSION
< TYPE IS HANDLED SEPARATELY .
FOR I = 1 TO THE NUMBER OF SHUTTLECRAFT :
   IF THE SHUTTLECRAFT'S MISSION IS DELIVER CARGO :
      CALL THE SHUTTLECRAFT DELIVER CARGO SUBMODULE .
   ORIF THE SHUTTLECRAFT'S MISSION IS SEEK SENSOR DATA :
      CALL THE SHUTTLECRAFT SEEK SENSOR DATA SUBMODULE .
   ORIF THE SHUTTLECRAFT'S MISSION IS TO TRANSPORT PERSONNEL
   OR CARGO :
      CALL THE SHUTTLECRAFT TRANSPORT PERSONNEL OR CARGO
      SUBMODULE .
   ORIF THE SHUTTLECRAFT'S MISSION IS SEEK THE ENTERPRISE :
      CALL THE SHUTTLECRAFT SEEK THE ENTERPRISE SUBMODULE .
   ENDIF
ENDFOR
RETURN//


< ********************************************************** >
<ENGINEERING TRANSPORTER SUBMODULE (4) >
FOR I = 1 TO THE NUMBER OF TRANSPORTERS ON THE ENTERPRISE :
   < CHECK TO SEE IF THERE ARE ANY PERSONNEL AT TRANSPORTER
   <STATION I WITH A DESTINATION OTHER THAN SOMEPLACE ON THE
   <ENTERPRISE .
   J = 1 ;
   REPEAT
      CHECK THE LOCATION AND DESTINATION OF THE JTH CREW
      MEMBER ;
      J = J + 1 .
   UNTIL A PERSON FOUND AT THE TRANSPORTER WITH DESTINATION
   OUTSIDE OF THE ENTERPRISE OR ALL PERSONNEL CHECKED :
   < CHECK CARGO IN THE SAME WAY IF NO PERSONNEL FOUND .
   IF THERE IS SOMETHING TO BE TRANSPORTED :
      < ATTEMPT TO TRANSPORT .
      < THE SUCCESS OF ANY TRANSPORT IS RELATED TO THE
      < FUNCTIONAL STATUS OF THE TRANSPORTER, THE ASSOCIATED
      < RELIABILITY FACTOR, AND THE DISTANCE TO THE DESTINATION .
      IF THE TRANSPORT WAS NOT SUCCESSFUL :
         DELETE THE OBJECT (CARGO OR PERSONNEL) FROM THE
         UNIVERSE ;
         IF THE OBJECT TRANSPORTED WAS A BOMB :
            < INFLICT DAMAGE TO THE TRANSPORTER .
```

```
        ENDIF
        NOTIFY THE ENGINEER .
      ELSE THE TRANSPORT WAS SUCCESSFUL :
        UPDATE THE LOCATION OF THE OBJECT ;
        UPDATE MAIN POWER SUPPLY DUE TO DRAIN FROM TRANSPORT .
      ENDIF
   ENDIF
   < UPDATE THE TRANSPORTER OPERATIONAL STATUS, FUNCTIONAL
   < STATUS, AND RELIABILITY FACTOR .
ENDFOR
RETURN//


< ********************************************************** >
< ENGINEERING ENERGY SUPPLY SUBMODULE (4) >
FOR I = 1 TO THE NUMBER OF STATIONS IN THE ENERGY SUPPLY
NETWORK :
   UPDATE THE FUNCTIONAL STATUS OF THE STATION .
ENDFOR
RETURN//


< ********************************************************** >
< ENGINEERING SPACE/WARP ENGINES SUBMODULE (4) >
IF THE SPACE/WARP ENGINES ARE ON :
   UPDATE THE ENERGY SUPPLY .
   < THE ENERGY CONSUMPTION OF THE SPACE/WARP ENGINES IS
   < DIRECTLY RELATED TO THE VELOCITY AND INVERSELY RELATED
   < TO THE FUNCTIONAL STATUS OF THE ENGINES .
   UPDATE THE FUNCTIONAL STATUS OF THE SPACE/WARP ENGINES .
ENDIF
RETURN//


< ********************************************************** >
< ENGINEERING IMPULSE ENGINES SUBMODULE (4) >
IF THE IMPULSE ENGINES ARE ON :
   UPDATE THE ENERGY SUPPLY .
   < THE ENERGY CONSUMPTION OF THE IMPULSE ENGINES IS
   < RELATED TO THE FUNCTIONAL STATUS OF THE IMPULSE ENGINE
   < AND VELOCITY OF THE CRAFT, AND TIME SINCE LAST UPDATE .
   UPDATE THE FUNCTIONAL STATUS AND OPERATIONAL STATUS .
ENDIF
RETURN//


< ********************************************************** >
< ENGINEERING TURBO-ELEVATORS SUBMODULE (4) >
< UPDATE THE LOCATION OF ALL T/E CARS .
< UPDATE THE FUNCTIONAL STATUS OF THE TURBO-ELEVATORS .
FOR I = 1 TO THE NUMBER OF TURBO-ELEVATOR CARS :
   IF T/E CAR I HAS A DESTINATION (T/E STATION) :
      IF THE ARRIVAL TIME FOR THIS CAR HAS PASSED :
         SET ITS LOCATION TO ITS DESTINATION ;
```



```
         SET ITS DESTINATION TO 'NONE' ;
         < DETERMINE WHETHER THERE WAS A PASSENGER ON THIS
         <CAR .
         J = 1 ;
         REPEAT
            CHECK PERSONNEL NUMBER J TO SEE IF HE IS ON
            T/E CAR .
         UNTIL ALL PERSONNEL CHECKED OR A PERSON FOUND ON THIS
         T/E CAR :
         IF NO PERSONNEL FOUND ON THIS T/E CAR :
            < CHECK TO SEE IF THERE IS CARGO .
            J = 1 ;
            REPEAT
               CHECK THE JTH CARGO TO SEE IF IT IS ON THIS T/E CAR .
            UNTIL ALL CARGO CHECKED OR A CARGO FOUND ON THIS
            T/E CAR :
         ENDIF
         IF THERE IS A PASSENGER OR CARGO ON THIS T/E CAR :
            < THEN HE/IT HAS ARRIVED AT THE APPROPRIATE T/E
            < STATION .
            SET ITS (HIS) LOCATION TO THE LOCATION OF THE T/E CAR
            (I. E. THE T/E STATION, NOT THE CAR) .
         ENDIF
      ENDIF
   ENDIF
ENDFOR
< CHECK ALL T/E STATIONS TO SEE IF PERSONNEL OR CARGO ARE
< WAITING TO USE A T/E .
<
FOR I = 1 TO THE NUMBER OF STATIONS :
   < SEE IF THERE IS A PERSON AT THE ITH STATION .
   J = 1 ;
   REPEAT
      CHECK THE JTH PERSONNEL TO SEE IF HIS LOCATION IS AT THIS
      T/E STATION BUT HIS DESTINATION IS NOT WITHIN THE AREA
      SERVED BY THIS T/E STATION .
      < THAT IS, CHECK TO SEE IF HE WANTS TO USE THE T/E TO GET TO
      < ANOTHER STATION .
   UNTIL ALL PERSONNEL CHECKED OR A PERSON FOUND AT THIS T/E
   STATION WHOSE DESTINATION IS NOT AT THIS T/E STATION (AND IS
   NOT 'NONE') :
   IF THERE IS NO PERSONNEL WAITING TO USE THE T/E :
      < CHECK TO SEE IF THERE IS CARGO WAITING .
      J = 1 ;
      REPEAT
         CHECK THE JTH CARGO .
      UNTIL ALL THE CARGO CHECKED OR A CARGO FOUND AT THIS T/E
      STATION WHOSE DESTINATION IS NOT WITHIN THE AREA SERVED
      BY THIS T/E STATION :
   ENDIF
   IF THERE IS A PASSENGER OR CARGO WAITING AT THIS T/E
   STATION TO USE THE T/E CAR :
      < FIND OUT IF THERE IS A T/E CAR AT THIS STATION THAT DOES
      < NOT ALREADY HAVE A PASSENGER .
```

```
      J = 1 ;
      REPEAT
         CHECK THE JTH T/E CAR .
      UNTIL ALL T/E CARS CHECKED OR A T/E CAR FOUND AT THIS
      STATION WHOSE DESTINATION IS 'NONE' :
      IF A T/E CAR WAS FOUND AT THE STATION WITH A  DESTINATION
      OF 'NONE' :
         THEN IT IS AVAILABLE TO TAKE ON THE PASSENGER ;
         SET ITS DESTINATION CODE EQUAL TO THE T/E SERVICE
         STATION SERVING THE DESTINATION OF THE PERSONNEL OR
         CARGO ;
         SET ITS ARRIVAL TIME EQUAL TO THE CURRENT TIME PLUS
         TRAVEL TIME (SEE FIGURE 3.6) .
      ELSE THERE IS NO T/E CAR AT THIS STATION :
         < SO LOOK FOR A FREE T/E AT SOME OTHER STATION .
         J = 1 ;
         REPEAT
            CHECK THE JTH T/E CAR .
         UNTIL ALL T/E CARS CHECKED OR ONE FOUND WITH
         DESTINATION EQUAL TO 'NONE' :
            SET ITS DESTINATION EQUAL TO THE STATION WHERE THE
            PERSON OR CARGO IS WAITING ;
            SET ITS ARRIVAL TIME (SEE FIGURE 3.6 FOR TRAVEL TIMES) .
         ENDIF
      ENDIF
   ENDIF
ENDFOR
RETURN//


< ********************************************************** >
< SHUTTLECRAFT DELIVER CARGO SUBMODULE (5) >
IF THE SHUTTLECRAFT IS WITHIN CARGO TRANSFER DISTANCE (TBD) OF
THE DESTINATION :
   < TRANSFER THE CARGO .
   SET THE LOCATION OF THE CARGO TO ITS DESTINATION ;
   SET THE DESTINATION OF THE CARGO TO 'NONE' ;
   SET THE MISSION OF THIS SHUTTLECRAFT TO SEEK THE ENTERPRISE .
ENDIF
RETURN//


< ********************************************************** >
< SHUTTLECRAFT SEEK SENSOR DATA SUBMODULE (5) >
SET THE ENTERPRISE'S SENSOR'S FUNCTIONAL STATUS INVERSELY
PROPORTIONAL TO THE DISTANCE BETWEEN THIS SHUTTLECRAFT AND
ITS DESTINATION (I.E. AS THE SHUTTLECRAFT GETS CLOSER THE
ENTERPRISE GETS BETTER DATA) ;
< DON'T LET THE SHUTTLECRAFT CRASH INTO ITS DESTINATION .
IF THE SHUTTLECRAFT IS WITHIN AN OPTIMUM SAFETY RANGE (TBD) :
   THEN SET ITS LOCATION TO THE OPTIMUM SAFETY RANGE .
ENDIF
RETURN//
```



```
< ********************************************************** >
< SHUTTLECRAFT TRANSPORT PERSONNEL SUBMODULE (5) >
IF THE SHUTTLECRAFT IS WITHIN PERSONNEL TRANSFER DISTANCE
(TBD) OF THE DESTINATION :
   < TRANSFER THE PERSONNEL .
   SET THE  LOCATION OF THE PERSONNEL TO HIS DESTINATION ;
   SET THE DESTINATION OF THE PERSONNEL TO 'NONE' ;
   SET THE MISSION OF THIS SHUTTLECRAFT TO SEEK THE ENTERPRISE .
ENDIF
RETURN//


< ********************************************************** >
< SHUTTLECRAFT SEEK ENTERPRISE SUBMODULE (5) >
IF THE SHUTTLECRAFT IS WITHIN DOCKING DISTANCE (TBD) OF THE
ENTERPRISE :
   SET THE SHUTTLECRAFT'S MISSION TO 'NONE' ;
   SET THE SHUTTLECRAFT'S LOCATION TO THE SHUTTLEBAY ;
   NOTIFY THE ENGINEERING OFFICER THAT THE CRAFT HAS
   RETURNED .
ENDIF
RETURN//
```

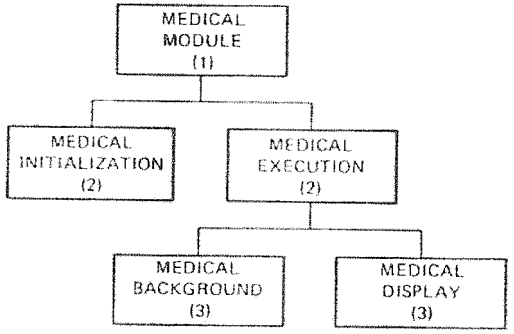


Figure 3.27 MEDICAL: SUBMODULE INTERRELATIONSHIPS

```
< ********************************************************** >
< MEDICAL MODULE (1) >
IF THE MEDICAL INITIALIZATION FLAG IS SET :
   CALL THE MEDICAL INITIALIZATION SUBMODULE .
ELSE THE MEDICAL INITIALIZATION FLAG IS CLEAR :
   IF THE MEDICAL MODULE RUN FLAG IS SET :
      CALL THE MEDICAL EXECUTION SUBMODULE .
   ENDIF
ENDIF
RETURN//
```

```
< ********************************************************** >
< MEDICAL INITIALIZATION SUBMODULE (2) >
SET ALL OF THE MEDICAL DATA TO NOMINAL ;
CLEAR THE MEDICAL MODULE INTIALIZATION FLAG .
RETURN//


< ********************************************************** >
< MEDICAL EXECUTION SUBMODULE (2) >
CALL THE MEDICAL BACKGROUND SUBMODULE ;
CALL THE MEDICAL DISPLAY SUBMODULE .
< NOTE THAT THERE IS NO MEDICAL OFFICER INPUT MODULE SINCE
< THERE IS NOTHING A MEDICAL OFFICER COULD DO FROM AN INPUT
< CONSOLE. IT MIGHT BE POSSIBLE, HOWEVER, FOR A MEDICAL OFFICER
< TO BE USED. HIS INPUT WOULD BE AN INTERACTION WITH A DISPLAY
< CONSOLE IN WHICH HE WOULD ANSWER MEDICAL QUESTIONS. HIS
< ABILITY TO ANSWER THE QUESTIONS CORRECTLY COULD THEN BE
< USED AS A FACTOR IN THE RECOVERY OF PATIENTS IN THE INTENSIVE
< CARE UNIT. IT IS LEFT UP TO THE PROGRAMMER TO ADD THIS
< CAPABILITY IF DESIRED .
RETURN//


< ********************************************************** >
< MEDICAL BACKGROUND SUBMODULE (3) >
UPDATE THE STATUS OF THE MEDICAL SECTION COMPUTER ;
UPDATE THE STATUS OF THE INTENSIVE CARE UNITS AND THE
FUNCTIONAL STATUS OF THE MEDICAL PERSONNEL IN THE MEDICAL
SECTION ;
< UPDATE THE STATUS OF THE PATIENTS IN THE MEDICAL SECTION .
FOR I = 1 TO THE NUMBER OF PATIENTS IN THE MEDICAL SECTION :
   UPDATE HIS FUNCTIONAL AND HEALTH STATUS ;
   < NOTE: THIS IS RELATIVE TO THE NUMBER OF MEDICAL PERSONNEL
   < AT THE INTENSIVE CARE UNIT, THE STATUS OF THE MEDICAL
   < COMPUTER, AND SO ON .
   IF HIS FUNCTIONAL AND HEALTH STATUS IS SATISFACTORY :
      SEND HIM TO HIS ASSIGNED STATION ;
         < SEE FIGURE 3.7 FOR ASSIGNED STATIONS .
      NOTIFY THE COMMUNICATIONS OFFICER OF THE RECOVERY OF
      THIS PATIENT .
   ENDIF
ENDFOR
RETURN//


< ********************************************************** >
< MEDICAL DISPLAY SUBMODULE (3) >
UPDATE THE MEDICAL DISPLAY .
RETURN//
```

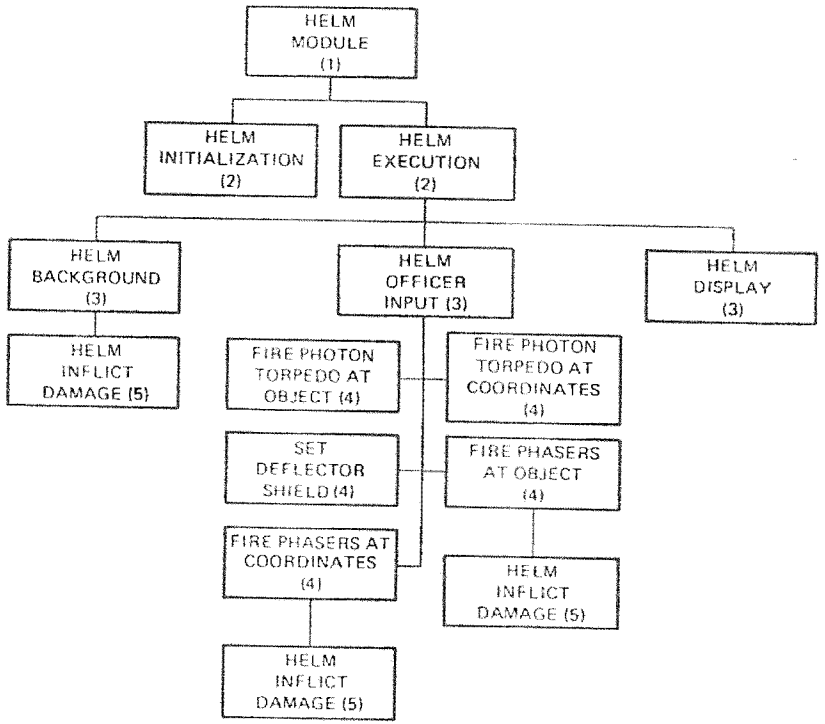



Figure 3.28 HELM: SUBMODULE INTERRELATIONSHIPS

LOCAL DATA

TENTATIVE DAMAGE LEVELS ( )

Figure 3.29 HELM MODULE

```
< ********************************************************** >
< HELM MODULE (1) >
IF THE HELM INITIALIZATION FLAG IS SET :
   CALL THE HELM INITIALIZATION SUBMODULE .
ELSE THE HELM MODULE INITIALIZATION FLAG IS CLEAR :
   IF THE HELM SUBMODULE RUN FLAG IS SET :
   CALL THE HELM EXECUTION SUBMODULE .
   ENDIF
ENDIF
RETURN//
```

```
< ********************************************************** >
< HELM INITIALIZATION SUBMODULE (2) >
SET ALL HELM DATA TO NOMINAL ;
CLEAR THE HELM MODULE INITIALIZATION FLAG .
RETURN//


< ********************************************************** >
< HELM EXECUTION SUBMODULE (2) >
CALL THE HELM BACKGROUND SUBMODULE ;
CALL THE HELM OFFICER INPUT SUBMODULE ;
CALL THE HELM DISPLAY SUBMODULE .
RETURN//


< ********************************************************** >
< HELM BACKGROUND SUBMODULE (3) >
< THE HELM MODULE HANDLES THE OFFENSIVE AND DEFENSIVE
< WEAPONS, UPDATES STATUS OF PHOTON TORPEDO TUBES, PHASER
< BANKS, DEFLECTOR SHIELDS, ETC.
< UPDATE THE STATUS OF ALL FIRED PHOTON TORPEDOES .
FOR I = 1 TO THE NUMBER OF PHOTON TORPEDOES :
   IF THE ITH TORPEDO EXISTS AND HAS A DESTINATION :
      IF ITS DESTINATION IS A SPECIFIED OBJECT :
         IF THE SPECIFIED OBJECT STILL EXISTS :
            SET THE VELOCITY VECTOR OF THE ITH PHOTON TORPEDO
            TOWARDS THE SPECIFIED OBJECT DESTINATION .
         ELSE THE SPECIFIED OBJECT NO LONGER EXISTS :
            SO DELETE THE ITH PHOTON TORPEDO FROM THE UNIVERSE .
            < DO NOT INFLICT ANY DAMAGE .
         ENDIF
      ENDIF
      IF THIS PHOTON TORPEDO STILL EXISTS :
         UPDATE ITS X, Y, Z POSITION ;
         IF THE PHOTON TORPEDO IS WITHIN EXPLOSION DISTANCE (TBD)
         OF ITS TARGET :
            THEN CALL THE HELM INFLICT DAMAGE SUBMODULE ;
            DELETE THIS PHOTON TORPEDO FROM THE UNIVERSE .
         ENDIF
      ENDIF
   ENDIF
ENDFOR
RETURN//


< ********************************************************** >
< HELM OFFICER INPUT SUBMODULE (3) >
IF THE HELMSMAN GAVE A COMMAND :
   IF THE COMMAND IS FIRE PHASERS AT SPECIFIED OBJECT :
      CALL THE FIRE PHASERS AT SPECIFIED OBJECT SUBMODULE .
   ORIF THE COMMAND IS FIRE PHASERS AT COORDINATES :
      CALL THE FIRE PHASERS AT COORDINATES SUBMODULE .
   ORIF THE COMMAND IS FIRE PHOTON TORPEDO AT SPECIFIED OBJECT:
```



```
      CALL THE FIRE PHOTON TORPEDO AT SPECIFIED OBJECT
      SUBMODULE .
   ORIF THE COMMAND IS FIRE PHOTON TORPEDO AT COORDINATES :
      CALL THE FIRE PHOTON TORPEDO AT COORDINATES SUBMODULE .
   ORIF THE COMMAND IS SET DEFLECTOR SHIELD LEVEL :
      CALL THE SET DEFLECTOR SHIELD LEVEL SUBMODULE .
   ELSE THE COMMAND IS UNKNOWN :
      SO NOTIFY THE HELMSMAN .
   ENDIF
ENDIF
RETURN//


< ********************************************************** >
< HELM DISPLAY SUBMODULE (3) >
UPDATE THE HELM DISPLAY .
RETURN//


< ********************************************************** >
< HELM FIRE PHASER AT SPECIFIED OBJECT SUBMODULE (4) >
< DETERMINE WHETHER THE SPECIFIED OBJECT IS KNOWN. IT MAY BE A
< CELESTIAL OBJECT, FEDERATION OR ENEMY SPACECRAFT, OR A
< SHUTTLECRAFT .
IF THE SPECIFIED OBJECT IS KNOWN :
   < DETERMINE THE LOCATION OF THE OBJECT RELATIVE TO THE
   < ENTERPRISE (PORT, STARBOARD, TOP, BOTTOM, FORE, OR AFT) .
   IF THE APPROPRIATE PHASER BANK IS OPERATIONAL :
      < INFLICT DAMAGE ON THE TARGET .
      CALL THE HELM INFLICT DAMAGE SUBMODULE ;
      REDUCE THE ENTERPRISE'S ENERGY SUPPLY ACCORDINGLY .
   ELSE THE APPROPRIATE PHASER BANK IS NOT FUNCTIONING :
      SO NOTIFY THE HELMSMAN .
   ENDIF
ELSE THE SPECIFIED OBJECT IS NOT KNOWN :
   SO NOTIFY THE HELMSMAN .
ENDIF
RETURN//


< ********************************************************** >
< HELM FIRE PHASER AT COORDINATES SUBMODULE (4) >
IF THE COORDINATES ARE WITHIN THE UNIVERSE :
   DETERMINE THE POSITION OF THE SPECIFIED COORDINATES
   RELATIVE TO THE ENTERPRISE (PORT, STARBOARD, TOP, BOTTOM,
   FORE, OR AFT) ;
   IF THE APPROPRIATE PHASER BANK IS OPERATIONAL :
      DETERMINE WHETHER THERE IS AN OBJECT AT THE SPECIFIED
      COORDINATES (COULD BE CELESTIAL OBJECT, FEDERATION OR
      ENEMY CRAFT, OR SHUTTLECRAFT) .
      IF THERE IS AN OBJECT AT THE SPECIFIED COORDINATES :
         CALL THE HELM INFLICT DAMAGE SUBMODULE .
      ENDIF
      REDUCE THE ENTERPRISE'S ENERGY SUPPLY ACCORDINGLY .
```

```
    ELSE THE APPROPRIATE PHASER BANK IS NOT OPERATIONAL :
       SO NOTIFY THE HELMSMAN .
    ENDIF
ELSE THE SPECIFIED COORDINATES ARE NOT WITHIN THE UNIVERSE :
    SO NOTIFY THE HELMSMAN .
ENDIF
RETURN//


< ********************************************************** >
< HELM FIRE PHOTON TORPEDO AT SPECIFIED OBJECT SUBMODULE (4) >
DETERMINE WHETHER THE SPECIFIED OBJECT IS KNOWN ;
< IT MAY BE A CELESTIAL OBJECT, A FEDERATION OR ENEMY SPACE
< CRAFT OR A SHUTTLECRAFT .
IF THE SPECIFIED OBJECT IS KNOWN :
    IF THERE IS AN OPERATIONAL TORPEDO STATION WITH AN UNUSED
    PHOTON TORPEDO :
       SET THE DESTINATION OF THE TORPEDO TO THE SPECIFIED OBJECT ;
       REDUCE THE ENTERPRISE'S ENERGY SUPPLY ACCORDINGLY .
    ELSE THERE IS NO OPERATIONAL TORPEDO STATION WITH AN
    UNUSED PHOTON TORPEDO :
       SO NOTIFY THE HELMSMAN .
    ENDIF
ELSE THE SPECIFIED OBJECT IS NOT KNOWN :
    SO NOTIFY THE HELMSMAN .
ENDIF
RETURN//


< ********************************************************** >
< HELM FIRE PHOTON TORPEDO AT COORDINATES SUBMODULE (4) >
< NOTE: AN UNUSED PHOTON TORPEDO IS ONE WHOSE LOCATION IS AT
< A TORPEDO STATION BUT WHOSE DESTINATION IS 'NONE' .
IF THE SPECIFIED COORDINATES ARE WITHIN THE UNIVERSE :
    IF THERE IS AN OPERATIONAL TORPEDO STATION WITH AN UNUSED
    PHOTON TORPEDO :
       SET THE DESTINATION FOR THE PHOTON TORPEDO TO THE
       SPECIFIED COORDINATES ;
       SET ITS DESTINATION CODE TO -1 ;
       UPDATE THE ENTERPRISE'S ENERGY SUPPLY ACCORDINGLY .
    ELSE THERE IS NO OPERATIONAL PHOTON TORPEDO STATION WITH
    AN UNUSED PHOTON TORPEDO :
       SO NOTIFY THE HELMSMAN .
    ENDIF
ELSE THE SPECIFIED COORDINATES ARE NOT WITHIN THE UNIVERSE :
    SO NOTIFY THE HELMSMAN .
ENDIF
RETURN//


< ********************************************************** >
< HELM SET DEFLECTOR SHIELD SUBMODULE (4) >
IF ALL SHIELDS WAS SPECIFIED IN THE COMMAND :
    IF THE SPECIFIED SHIELD LEVEL IS WITHIN LIMITS :
```



```
       THEN SET THE OPERATIONAL STATUS OF ALL SIX DEFLECTOR
       SHIELDS TO THE SPECIFIED LEVEL .
    ELSE THE SPECIFIED LEVEL IS NOT WITHIN LIMITS (0 - 100%) :
       SO NOTIFY THE HELMSMAN .
    ENDIF
ORIF A SPECIFIC SHIELD WAS SPECIFIED :
    IF THE SPECIFIED SHIELD LEVEL IS WITHIN LIMITS (0 - 100%) :
       THEN SET THE OPERATIONAL STATUS OF THE SPECIFIED SHIELD TO
       THE SPECIFIED LEVEL .
    ELSE THE SPECIFIED LEVEL WAS NOT WITHIN LIMITS (0 - 100%) :
       SO NOTIFY THE HELMSMAN .
    ENDIF
ELSE THE SPECIFIED SHIELD IS UNKNOWN :
    SO NOTIFY THE HELMSMAN .
ENDIF
RETURN//


< ********************************************************** >
< HELM INFLICT DAMAGE SUBMODULE (5) >
< THE TARGET THAT RECEIVES THE DAMAGE AND THE WEAPON THAT IS
< INFLICTING THE DAMAGE IS SPECIFIED BY THE LEVEL 4 SUBMODULE
< WHICH CALLS THIS SUBROUTINE.
< IN GENERAL, THE DAMAGE INFLICTED DEPENDS UPON THE FOLLOWING:
<
< 1. TYPE, FUNCTIONAL AND OPERATIONAL STATUS, AND RELIABILITY
<    FACTOR OF THE WEAPON.
< 2. DISTANCE FROM THE WEAPON TO THE TARGET.
<    NOTE THAT A PHOTON TORPEDO MAY DETONATE NEAR A TARGET,
<    NOT NECESSARILY DIRECTLY ON THE TARGET.
< 3. THE SHIELD LEVEL OF THE TARGET. IF THE SHIELDS ARE UP THEN
<    A PROPORTIONATE LEVEL OF THE DAMAGE WILL BE APPLIED TO
<    THE SHIELDS.
<
DETERMINE THE ATTRIBUTES OF THE TARGET WHICH ARE SUBJECT TO
DAMAGE ;
< FUNCTIONAL STATUS, HEALTH, INTELLIGENCE, POLLUTION LEVEL,
< QUANTITY, QUALITY, AND RELIABILITY FACTOR ATTRIBUTES.
SET THE 'TENTATIVE DAMAGE' OF EACH OF THE TARGET ATTRIBUTES
IDENTIFIED ABOVE TO 100% ;
DETERMINE THE 'BLAST IMPACT' OF THE WEAPON ;
< THIS IS DIRECTLY PROPORTIONAL TO THE FUNCTIONAL STATUS OF
< THE WEAPON, INVERSELY PROPORTIONAL TO THE RELIABILITY
< FACTOR OF THE WEAPON, INVERSELY PROPORTIONAL TO THE
< DISTANCE BETWEEN THE ENTERPRISE AND THE TARGET (FOR PHASERS)
< OR BETWEEN THE LOCATION OF THE PHOTON TORPEDO WARHEAD
< AND THE TARGET (FOR PHOTON TORPEDOES) .
DETERMINE THE NUMBER OF TARGET ATTRIBUTES WHICH WILL BE
AFFECTED BY THE BLAST ;
< THIS WILL BE RELATED TO THE DISTANCE (AS NOTED ABOVE), THE
< BLAST IMPACT, A RANDOM FUNCTION, AND THE OPERATIONAL
< STATUS OF THE TARGET DEFLECTOR SHIELDS .
DETERMINE WHICH OF THE TARGET ATTRIBUTES ARE TO BE DAMAGED ;
< THIS IS A RANDOM CHOICE BUT IF THE DEFLECTOR SHIELDS OF THE
```

```
< TARGET ARE ON THEN THEY MUST BE INCLUDED IN THE ATTRIBUTES
< TO BE AFFECTED.
SET THE 'TENTATIVE DAMAGE' FOR ALL OF THE TARGET ATTRIBUTES
NOT SELECTED TO RECEIVE DAMAGE TO 0% ;
DISTRIBUTE THE 'BLAST EFFECT' RANDOMLY OVER THE TARGET
ATTRIBUTES WHICH WERE SELECTED TO RECEIVE DAMAGE ;
< (E.G. IF THE BLAST EFFECT IS 50% AND THERE ARE TWO TARGET
< ATTRIBUTES WHICH WERE SELECTED TO RECEIVE DAMAGE THEN ONE
< MIGHT RECEIVE 20% AND THE OTHER 30%.
< IF THE TARGET DEFLECTOR SHIELDS ARE ON THEN THE TARGET'S
< DEFLECTOR SHIELD ATTRIBUTE SHOULD RECEIVE A LARGE SHARE
< OF THE 'BLAST EFFECT'.
APPLY THE BLAST EFFECT BY CHANGING THE TARGET ATTRIBUTES
PROPORTIONAL TO THE ASSOCIATED 'TENTATIVE DAMAGE' ;
< E.G. FUNCTIONAL STATUS WOULD BE DECREASED 15% FOR AN
< ASSOCIATED TENTATIVE DAMAGE OF 15%, RELIABILITY FACTOR
< WOULD INCREASE 7% FOR AN ASSOCIATED TENTATIVE DAMAGE OF 7%.
RETURN//
```

GENERAL NOTES

1. *Setting to nominal* means setting the associated variables to some predetermined value, often a mid-range or maximum value. For example, when setting the celestial data to nominal it might entail creating the maximum number of celestial objects with a roughly even distribution throughout the universe.
2. Many subroutines specify to *NOTIFY THE OFFICER*. This is a general term used to indicate that some message should be transmitted to the named officer. It may be done by displaying the message on the console in the *alerts* section or by announcing it orally if a computer-controlled voice-synthesizer is available.

Generally those messages which are warnings to the officer, such as when he inputs an invalid command, would be displayed in the *ALERTS* area of his display. Those messages which are meant to come from other Federation or enemy craft and the like would be sent to the communications module message area.
3. The abbreviation *TBD* is used throughout the logic flow definitions. TBD stands for To Be Determined and is used to describe those constants, variables, and relationships which can only be determined by running the simulation. It is left to the programmer to define them after he has exercised the routines.

LOGIC FLOW CODE

The purpose of this book has been to present the methods employed in designing a simulation and then to present the descriptions of the STAR SHIP simulation as an example. Since

there are a great variety of programming languages and hardware systems available no attempt is made here to define the actual executable code. It would be an impossible task to provide an actual implementation which would satisfy every reader.

The logic flow definitions, however, have been given in such detail and been written utilizing English-form structured programming constructs that it should be a relatively easy and straightforward exercise for the reader to make the transformation to his particular software. Several examples have already been given in CODING THE LOGIC FLOW in chapter 2 of how to make the translation. Before writing any code, however, it is advisable to read through the next section on testing and the final chapter on implementation.

TEST PROCEDURES

The modules should be coded top to bottom. That is, the Simulation Controller and Common Data areas first. Testing of the top level can then be done by coding stubs for each of the remaining modules. A *stub* is a module or subroutine which has an entry point and an exit point (return) but no actual code except, perhaps, for a tracing output message such as *GOT TO THE MEDICAL MODULE*. Each of the subfunctions and operator commands should be tested separately, utilizing tracing functions for each. One particular type of trace would be one which writes out the common data variable values at critical points throughout the module so that it can be determined whether they are being updated properly.

Only after the Simulation Controller has been thoroughly checked should another module be added to replace its stub and then be checked out.

After all modules have been thoroughly debugged using tracing features, a sample scenario which will exercise all functions should be run (with traces still intact). The first scenario run should be done without any operator input. The test is to see if the module responds correctly to the scenario by itself. Observations are done by watching the trace output and the module displays. When it is certain that everything is working properly the full scale running of the system with or without the scenario and with full operator input can begin.

Trace functions should be removed as the programmer gains confidence in the associated modules. The final phase of testing

is adjusting the constants and relationships which were initially educated guesses. This will include such things as how much damage should be done when a target gets hit by a phaser, how quickly the food supply should be consumed, the relationship of target mass to tractor beam operational status, and so on. Since in this simulation much of this information was not available during the design phase it now becomes the task of the programmer in conjunction with the operators to adjust the constants and relationships until the responses are *realistic*. In this case realistic means whatever the operators feel is reasonable and fair. Certainly allowing the STAR SHIP infinite energy or undamagable shield screens would be both unreasonable and unfair.

In other simulations which are designed to imitate actual known physical phenomena this part of the task is simplified since there is a set standard with which to compare the simulation output. For the STAR SHIP it is a bit more challenging and certainly more rewarding.

4 Implementation

For those readers who wish to implement the STAR SHIP simulation so that they may run it on their own computer I have a few suggestions and helpful hints. You will first have to decide whether it is to be implemented as a multi-operator simulation or a single-operator one.

SINGLE–OPERATOR MODE

In the single operator mode the entire simulation requires only one CRT-type display. Obviously only one of the several displays (engineering, navigation, etc.) can be displayed at one time. The programmer has two alternatives.

The first is to combine as many of the displays as possible into one all-encompassing display and simply disregard those displays that won't fit. This has the advantage of allowing the operator to see a lot of information at the same time but has the disadvantage of possibly producing a cluttered, confusing display.

The other possibility is to allow the operator to dynamically choose which display he wants to view at any particular time. While he is maneuvering the Enterprise he may want to have the navigation display; while in a battle situation he would probably choose the helm display; and so on. No matter what display he currently has up he could type in a command to select an alternative display.

This brings up another aspect of the single-operator mode. Regardless of how the displays are handled all operator input commands should be available at all times. This means, for

instance, that if the operator is looking at a medical display he should still be able to issue a navigation command. What this means for the programmer is a change in the way input commands are handled. A suggested method is to remove the operator input submodules from each of the several major modules and put them all together in the simulation controller module. When a command is entered, the simulation controller module would determine which submodule should be called to execute it, no matter what major function it is related to.

MULTI–OPERATOR MODE

Obviously the simulation was designed with a multi-operator mode in mind. It is this mode which provides the most realistic interaction. As described in chapter 3, there would be a separate CRT display and keyboard for each of the seven major functions.

The person operating the simulation controller console (keyboard and display) would have control over the entire simulation. Six other people would be involved in the operation of the Enterprise; a navigator, communications officer, science officer, an engineer, and a helmsman, each with his own console, plus the captain seated in his position of command but with no console. A typical setup is shown in figure 4.1.

After the simulation controller has initialized and started all of the modules, and possibly begun a scenario, then the captain takes over and the Enterprise is on her way to parts unknown.

HARDWARE

The hardware for this simulation will be a major consider. I present here several possible designs.

Single-Processor Single-Console

If your only access is to a computer employing one processor and one display, as in most hobbyist systems, or if it is a time-sharing system in which you cannot communicate between consoles, then your choice is obviously limited. You will have to settle for a single operator mode.

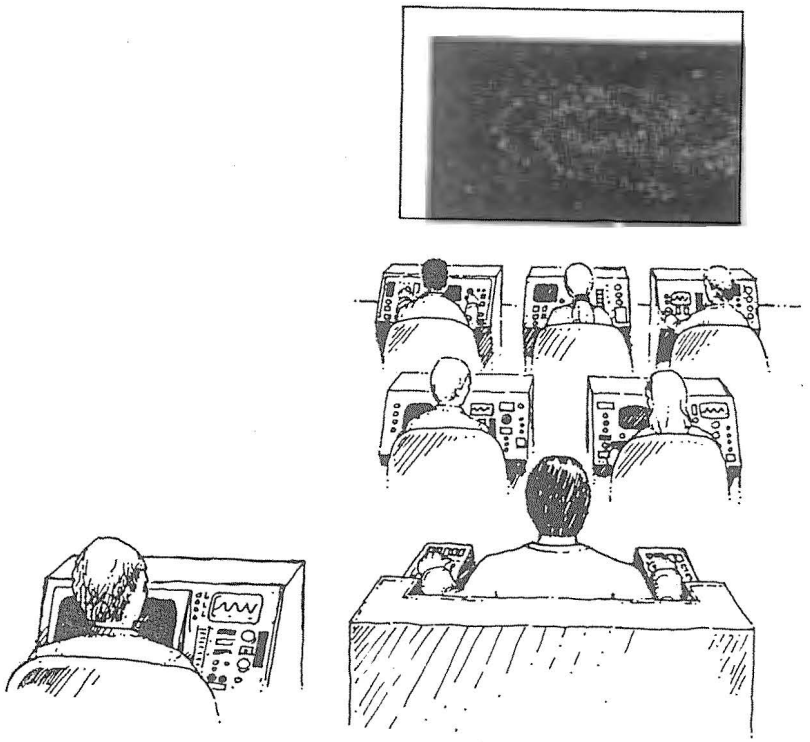


Figure 4.1 TYPICAL MULTI–OPERATOR SETUP

Single-Processor Multi-Console

Some systems, although run by one processor, allow access to several consoles. This might be a personal computer which has several consoles connected, each with its own dedicated I/O port. The simulation software will probably be resident in main memory at all times and as each module gets executed it will check its appropriate I/O port (console) to see if any command has been typed in by the associated officer.

Another single-processor multi-console system would be typified by a large time-sharing system that has several consoles. On some of these systems it is possible to send messages from a

program running at one console to a program running at another console. It might also be possible for each of the several programs to access the same disk file which could be thought of as common memory. Each main module of the STAR SHIP simulation could then be written as a separate program and each one would run at its own console, communicating with the other modules (programs) by sending messages and via the common disk file. Considerable changes would have to be made to the logic flow definitions in order to accomodate the unusual form of communications, however. Yet it is possible to do it this way due to the modularity of the simulation design. It is advisable, however, to have several years of programming experience before attempting this type of setup.

I should point out, also, that the simulation will not generally run any faster on a time-sharing system than on the other single-processor multi-console system mentioned above. This is because the modules will still get executed in a one-at-a-time sequential fashion. Indeed, it may even run slower on the time-shared system since there is overhead time (bookkeeping, disk access time, etc.) which interrupts the programs periodically.

Multi-Processor Single-Console

This is where extreme processor speed and, hence, realistic interaction can be accomplished. A multi-processor system is one in which there is, in effect, more than one computer running at the same time. This may sound as if it is out of the realm of the average computer hobbyist but there are such systems available. One company that produces such a system in kit form is Ohio Scientific Incorporated; 11679 Hayden Street; Hiram, Ohio; 44234.

The ideal multi-processor system would be one in which one processor is dedicated to each of the seven major modules. In this way each module is executed continually rather than sequentially as in the single-processor systems. Obviously this speeds up the simulation seven times, a fantastic savings in time. This time-saving means that the whole simulation runs faster and, therefore, can run in real-time. Now, it may be possible to have the simulation run at a reasonable speed even when implemented on a single-processor system and it should not be assumed that you must have a multi-processing setup in order to

have a reasonably realistic star ship. The savings in processing time when using the multi-processing system means that you can more easily expand the simulation to include, for example, more intelligence for the enemy and Federation craft, or more navigation command formats, or what have you.

As mentioned above, the ideal situation would be to have a separate processor for each major module. However, even if only two processors are available it would be possible to realize a saving in processor time by dividing the modules between the two processors. In any case each of the processors will have to access to a common memory area. A suggested method, assuming one processor per module, would be to have the module program stored on ROM (read-only-memory) and be local to the associated processor. Each processor would also have its own local RAM (random-access memory) for those variables which are used only by the module. Every processor, however, would have read-write access to the common RAM memory which would hold, quite obviously, the common data variables.

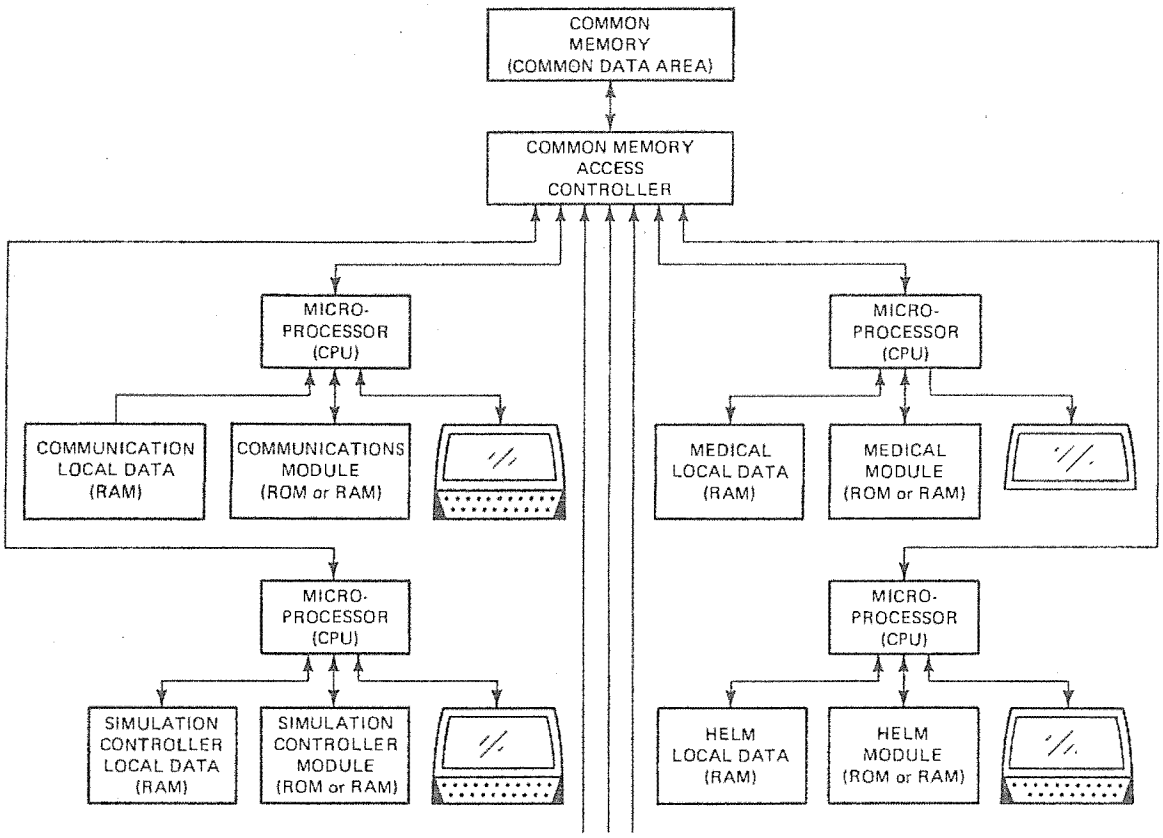
As with the single-processor systems described above it is possible to have a multi-processor system set up with only one console, and hence only one operator, or with two or more consoles. In the single-console setup there would be the same considerations as for the single-processor single-console design.

Multi-Processor Multi-Console

With the addition of a separate console for each operator plus a dedicated processor for each major module we would have the ultimate STAR SHIP simulation. Not only would each processor work only upon its own module it would also access its own console. This system is shown in block diagram in figure 4.2. By implementing it in this manner you obtain the fastest processor speeds, the closest approach to real-time interaction, and the most powerful simulation.

Switches and Dials

So far this book has assumed that the operator interaction with the system would be through fairly standard CRT-type displays and alphanumeric keyboards. But anyone who has seen



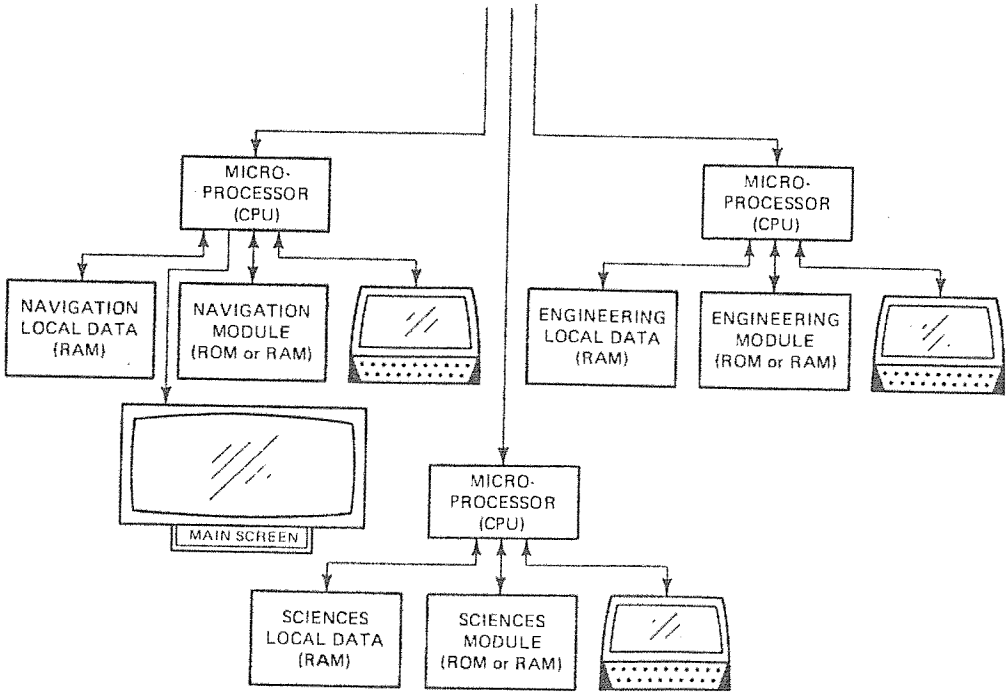


Figure 4.2 MULTI—PROCESSOR MULTI—CONSOLE BLOCK DIAGRAM

even one episode of STAR TREK knows that the Enterprise is controlled by turning dials, pushing buttons, and so on. For the adventurous programmer with a good deal of hardware expertise and not a little bit of money it is quite possible to build a full scale mock-up of the bridge of the Enterprise complete with dials, buttons, switches, color displays, and sound effects, all controlled by the simulation programs.* The only basic difference in the logic flow definitions would be that operator input would come in via switches and analog-to-digital convertors and output would go to sophisticated displays and vocal output units.

To be sure, it may be unreasonable to expect anyone to attempt developing the project to this extent, but it is exciting to think of the possibilities. One last thought before leaving this section; if any reader seriously considers attempting this (and could possibly afford it) you can contact me via the dilithium Press to discuss it. So much for wishful thinking.

SOFTWARE

Certainly a major concern to anyone attempting to implement the STAR SHIP project is that of which software to use. There are a variety of options for the programmer, none of which will satisfy everyone.

High Level Languages

Any high level language will certainly make it easier to program the modules. However, if you use a language which is implemented as an interpreter, as BASIC often is, then the completed simulation will run unreasonably slow. Attempting to turn the

[*] See the article "SFS Walletsize: Spaceship Simulator" by Lance E. Strickler, John H. Buffington, III, and Ivy Fleck Strickler in the February and March 1978 issues of *Interface Age* magazine. It describes a truly impressive full scale spaceship control panel which was built from spare parts within a budget of $500. Back issues of the article may be obtained for $2.50 each from *Interface Age*, 16704 Marquardt Avenue, Cerritos, California, 90701. Or you can write directly to the authors at 2005 Naudain Street, Philadelphia, Pennsylvania, 19146.

Enterprise around to run away from an enemy vessel could take precious minutes, while the enemy is firing at you. The various displays at the operators' consoles should appear to update continuously. Using an interpretive language it would be far too obvious that each display is laboriously and slowly being updated. So an interpreter would be your last choice.

The other alternative in high level languages is the compiler. This type of language takes your written program and turns it into directly-executable machine language code and is certainly a step up from the interpreter. The difficulty with this option is that it tends to use a lot of main memory and unless you are operating on a system which allows swapping in (from disk or other off-line device) segments of executable programs only as they are needed, or have access to a very large amount of memory, then a compiler language may not be the best choice either.

A third high level option would be a language which is somewhere between an interpreter and a compiler. This type of language first translates the source programs into an intermediate form of coded information and then interprets this code in order to do the actual execution. Its applicability to the STAR SHIP simulation lies somewhere between interpreters and compilers.

Low Level Languages

The low level language, while possibly a bit more difficult to implement than a high level language, probably offers the best option. A low level language, typically called an assembler language, is one in which the program writing is done very close to the machine level code. It allows the programmer to produce compact (using a minimum of main memory), fast-running code. From this standpoint it is the best option. The only possible disadvantage is that it must be written with a particular processor in mind and, therefore, may not be easily transferred to a different computer. But, then, all of the logic definitions are written in a nondedicated Englishlike manner so that translating the program into the language of a different computer should not be all that difficult. Indeed, it is for this reason that I did not write the logic flow in a particular language. As it stands it has the best flexibility and should provide the greatest ease in implementation.

Fitting It All In

Assuming the decision has been made about what hardware and software will be used, the programmer will have to concern himself with fitting the entire simulation into the available memory space. One solution to the too small memory space problem has already been suggested; that of storing the executable modules on some off-line memory device, such as a high-speed disk and bringing each module into main memory only as it is needed. The obvious problem with this is that most programmers do not have access to such a piece of harware.

Let's assume that you have a set amount of main memory and no off-line storage capabilities. If the entire simulation program simply cannot fit, then it will have to be trimmed here and there. Since the whole project is defined in modular fashion, cutting out excess portions is made quite easy. You could, for example, simply do away with the simulation controller module altogether. The implication would be that the program would have to be restarted each time an entirely different tactical situation was desired. It might mean setting certain initial conditions as random so that the same tactical situation does not appear at each startup. It would also imply that scenarios and run-time debugging would not be possible.

You could eliminate portions of the major modules such as the shuttlecraft capabilities, some of the enemy and Federation craft mission types, transporter functions, and so on. The medical module is a prime candidate for total elimination. Again, due to the modularity of the simulation, the programmer can cut and trim here and there quite easily, obviously reducing the simulation capabilities, but maintaining the basic structural integrity of the project.

Before eliminating actual functions, however, it is advisable to limit the amount of memory required by the common data area since array variables are notorious consumers of memory. In many places in common data I specify array sizes of 1000, for example in the number of celestial objects. It would be quite simple to limit this to, perhaps, 20 celestial objects. The memory saving will be quite high. And this is not limited to celestial objects. I suggest that the programmer browse through the common data area and he should find many places where he can trim the size of the program simply by reducing array sizes. An added

advantage of this type of trimming is that the program will run faster since it will not have to process so many spacecraft, celestial objects, and so on. I set the arrays as large as they are simply to indicate a reasonable upper limit.

Expansion and the Future

The inverse of fitting it all in is expanding it. Once you have written the programs and have run the simulation for some time you, no doubt, will discover other functions which I either have overlooked or couldn't fit into the book. Please feel free to experiment with the simulation. Add functions, make the enemy craft more intelligent, expand the evasive tactics function of the Enterprise so that it is totally self-sufficient, and so on. Have fun with it. Your only limit is your imagination.