2

F-111 WING COMMANDER

A Mprolog Expert System

by

John T. Minor & Grant Wright

CSR-90-044

MASTER COPY - KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1/91	3. REPORT TYPE AND DATES COVERED Technical

4. TITLE AND SUBTITLE F-111 Wing Commander - A Mprolog Expert System	5. FUNDING NUMBERS DAAL03-87-G-0004
6. AUTHOR(S) John T. Minor and Grant Wright	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Nevada, Las Vegas 4505 Maryland Parkway Las Vegas, NV 89154	8. PERFORMING ORGANIZATION REPORT NUMBER CSR-90-044
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Army Research Office P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ARO 24960.67-MA-REP

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

MPROLOG is used to implement the expert system discussed in this paper. A description of the expert system, "a F-111 Wing Commander Consultant", and its implementation, starting with the knowledge acquisition and design phases, are discussed in part 2. The problems encountered developing the expert system and problems due to MPROLOG are also presented. Finally, an evaluation of MPROLOG as an expert system development tool will make up part 3.

14. SUBJECT TERMS MPROLOG Expert System, Expert Systems	15. NUMBER OF PAGES
	16. PRICE CODE

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

Acknowledgements

The authors wish to thank Col. Edward Kowalczyk, USAF retired, for the valuable assistance that he gave us on this project. As our domain expert, he generously gave of his time so that this research could be completed.

This work has been funded by the Army Research Office under grant #DAAL03-87-G-0004.

Contents

1 Introduction

1.1 Expert Systems

Expert systems are designed to perform the tasks currently performed by human experts. The specialized knowledge that a human expert has is placed in the expert system program. If constructed properly, the expert system will request the same information that an expert would request, and then output the same decision that the expert would. These programs are useful because there is usually a shortage of qualified human experts in a given field. The computer can be fully informed of changing facts at all times and will never get fatigued or temperamental. Also an expert system can apply the expertise of several human experts, if so programmed, and can free human experts to do more creative work. Many expert systems have already been developed to perform tasks like medical diagnosis, geological analysis, computer configuration design and more[8,9].

Expert systems are divided into a knowledge base and an inference part. The knowledge base is kept separate because it must be allowed to grow and change, while the inference program part is not changed. The knowledge base is made up of decision rules and facts, which the program uses when a result is to be output. Since an expert must make decisions based on data with questionable accuracy, an uncertainty factor is often associated with knowledge base information. For instance, a medical diagnosis program may have a rule stating "the patient has Disease_1 with a confidence level of 90% if x-rays show condition A with a confidence level of 85% and symptoms B and C are present". The role of the inference part would be to ask the user for input, process the proper rules and output an acceptable result.

The construction of expert systems is called knowledge engineering. Acquiring the knowledge is the most important step because the expert system will only be as good as the knowledge base it uses. The programmer, or knowledge engineer, must have some familiarity with the domain of the expert system. Then the knowledge engineer must tap the expertise of the domain expert, a person that has expertise in the given area, who may or may not be able to convey his knowledge readily. The knowledge engineer must organize the information into a set of facts and decision rules, which can be used to output answers that the expert thinks are feasible. Problems arise in rule design because of uncertainty, since an expert must make decisions based on facts with confidence levels less than 100%. When the confidence levels of facts and rules in the knowledge base have been determined with accuracy, the likelihood of the expert system doing the intended job is much greater.

Choosing the tool of implementation is the next problem in expert system design. Early expert systems were designed using ad hoc methods. They were designed from the ground up, using system and domain dependent strategies. There are now many useful expert system design tools, among which are KEE (knowledge engineering environment [6]) and ART (automated reasoning tool

[1,2]). The main problems with these tools are cost and complexity. An university researcher has developed a free expert system tool called MPROLOG[4]. This tool provides a designer with the programming power of PROLOG (a logic programming language[3]), as well as providing the representation power of fuzzy confidence levels in the knowledge base. A description of this expert system design tool is presented below.

MPROLOG is used to implement the expert system discussed in this paper. A description of the expert system, "a F-111 Wing Commander Consultant", and its implementation, starting with the knowledge acquisition and design phases, are discussed in part 2. The problems encountered developing the expert system and problems due to MPROLOG are also presented. Finally, an evaluation of MPROLOG as an expert system development tool will make up part 3.

1.2 MPROLOG

MPROLOG is an expert system development tool designed by Dr. John Minor and implemented by Martin Flatebo in 1988 to complete a masters degree in computer science at the University of Nevada, Las Vegas[4]. MPROLOG is written in Common LISP on the Symbolics 3600 series machines. The purpose of MPROLOG is to increase the capabilities of the powerful programming language PROLOG, by supplementing the language with multi-valued logic capabilities. Standard PROLOG uses simple two-valued logic leaving no room for uncertainty. MPROLOG allows programmers to store information with uncertainties based on minimal-bounded fuzzy logic. The usefulness of PROLOG as an expert system development tool is therefore greatly enhanced since uncertainties are put directly into the knowledge base and manipulated automatically by the language interpreter. This allows the knowledge engineer a mechanism for implementing the full knowledge of the expert, even in uncertain cases.

The fundamental piece of knowledge in MPROLOG is the fact. Facts are the same as single predicates in first-order logic. For instance, likes(john,mary) could mean "john likes mary". A group of facts with the same predicate name could be thought of as a database relation. The following example demonstrates this:

Group of Facts

```
plane( 111, fighter, damaged ).
plane( 121, fighter, available ).
plane( 211, bomber, damaged ).
plane( 221, bomber, available ).
```

Plane database relation

plane#	*type*	*status*
111	fighter	damaged
121	fighter	available
211	bomber	damaged
221	bomber	available

MPROLOG is not frame-based like ART and KEE. The inheritable properties from other relations must be extracted with rules. Rules are Horn clauses which are processed using backward-chaining.

MPROLOG facts look like:

$$p\{c\}(t_1, t_2, ..., t_n).$$

where p is a predicate (the fact name),
c is the optional uncertainty factor, or truth value ($0 \leq c \leq 1$),
$t_1, ..., t_n$ are terms (attributes of the fact).

MPROLOG rules look like:

$$\begin{aligned} p\{c_0\}(t_1, t_2, ..., t_n) \text{ :- } & q_1\{c_1\}(s_{11}, s_{12}, ..., s_{1m_1}), \\ & q_2\{c_2\}(s_{21}, s_{22}, ..., s_{2m_2}), \\ & \vdots \\ & q_k\{c_k\}(s_{k1}, s_{k2}, ..., s_{km_k}). \end{aligned}$$

where p is the head predicate of the rule,
c_i's are the optional uncertainty factors ($0 \leq c_i \leq 1$),
q_i's are predicates that make up the tail of the rule,
t_i's and s_{ij}'s are the terms for the respective predicates.

If an uncertainty factor is absent, MPROLOG assumes, like PROLOG, that the fact or rule is 100% true. Otherwise the uncertainty factor is treated as a minimal bound on the truthfulness of the fact or rule.

MPROLOG works by resolving rules against rules and facts. The first rule is invoked at the MPROLOG prompt "?-". At this point, the question calls the system interpreter. For example, a system with the above rule could be invoked with "?- p{.7}($a_1, a_2, ..., a_n$)." A rule p is satisfied (the head is true to at least level c_0) if each q_i is true with a truth value of at least c_i. Any q_i that matches a fact is true, and any q_i that matches the head of a rule will cause MPROLOG to attempt to satisfy the tail of its rule in this same way. When a rule's tail cannot be satisfied, backtracking occurs in an attempt to resatisfy the rule in a different way.

2 The F-111 Wing Commander: a MPROLOG Expert System

2.1 The Expert System Domain

The domain of this expert system is an F-111 wing commander circa 1975 (Vietnam War era). The wing commander is expected to assign planes, weapons and weapon fusings to missions which field personnel want to see implemented. The conditions of the mission affect the number of planes needed, the types of weapons used on the mission, and the fusing of those weapons. The commander issues a mission assignment if the needed planes and weapons are available. The wing commander does not assign the individual pilots because this is done at a lower command level.

The calculation of the number of planes and their weapon loads is done by first considering certain factors like low altitude cover and terrain (affecting target visibility and hit/miss probability), target materials (affecting load types and fusing), defensive positions and anti-aircraft weaponry (affecting the number of planes needed), etc. These factors are looked up in mission implementation manuals, the suggestions are noted, and the commander estimates the number of planes needed and the type of loads needed for a mission consisting of the given factors. An intelligence officer keeps the commander briefed on current conditions on all missions.

The expert system is assist with the duties of the wing commander. It will interact with the field personnel by querying about the type of mission, problems which can possibly affect the mission, and the desired completion level for the mission. If the mission can be implemented, the expert system should make a suggestion by listing the number of planes, types of weapons, and types of weapon fusings needed to carry out the request. If the mission cannot be implemented, the system should cite possible factors for this, such as insufficient supplies or too difficult conditions.

The expert system is expected to maintain databases for missions, planes and weapons, so field personnel can check the status of these items. The expert system also allows changes and deletions, to a certain degree, to the mission database. Changes to the plane and weapon databases are unlimited, but there is a password system protecting access to these databases. A standard file is used when starting up the expert system from scratch. The field personnel can also edit a saved session of the expert system. Changes can be made to the system to reflect more current conditions, but no special capability for the opinions of an intelligence officer are provided.

The domain expert, Colonel E. Kowalczyk, USAF retired, was a pilot of F-111 aircraft during the Vietnam War. He has extensive knowledge of missions implemented during this era, and knows the expected effectiveness for these missions. His knowledge is largely declarative, and he had no problems relating heuristic values or describing critical factors. He was very patient in explaining much of the military jargon and semantic knowledge, and was helpful in eliminating unimportant factors and establishing restrictions on other factors.

2.2 Design of the Expert System

The expert system consists of databases for planes, weapons, missions, and mission components. The plane database keeps track of plane status and availability. The weapon database holds the amount available of each weapon type. The mission database consists of many attributes: mission number, type, desired completion level, start time, location, and number of planes. Mission component databases also exist to keep track of data associated with each mission, including lists of the number of planes carrying a certain load with a certain weapon fusing.

The expert system is menu driven. The menu breakdown was developed by Dr. Minor, who also created the idea of the F-111 wing commander expert system. The top level menu of the expert system has 5 choices: plan mission, change mission, check status, change database and quit the system. The plan mission selection is where the expert knowledge is used. The other selections are simply used to configure, change and check the databases, and will not be discussed further.

The plan mission section is used to develop a mission frame. A mission frame example follows:

```
mission number : 1
location: locname
start-time: 1200
mission type: interdiction
number of planes assigned: 33
desired completion percentage: 60-75%
```

Also associated with this frame is a mission-component frame and a list of assigned planes with corresponding weapon loads and fusings.

The plan mission section starts by querying for location and start-time of the mission. The mission number is supplied by the system. Then the user is asked to select from the following mission types: interdiction against a target, area preparation, close-air support, and 24 hour alert. The interdiction selection also brings up a menu on target types: personnel concentration, unarmoured vehicles, armoured vehicles, building complex, roads/railroads, and bridges. Selection of any mission type results in a series of menus appearing which are designed to get a proper description of important factors for the chosen mission (see figure 1 for a list of these factors). All of these factors are represented as menus except for the 24 hour alert selection, which simply asks for the number of planes. A final menu records the desired completion percentage for the mission. This completion level specifies to the expert system the importance

Mission Type	Affecting Factors
interdiction: personnel	force size of targeted personnel, terrain type, protective cover, defensive positions, anti-aircraft guns, surface-to-air missile sites, visibility
interdiction: unarmored vehicles	size of vehicle convoy, terrain type, protective cover, anti-aircraft guns, surface-to-air missile sites, visibility
interdiction: armored vehicles	number of vehicles, terrain type, protective cover, anti-aircraft guns, surface-to-air missile sites, visibility
interdiction: building complex	area of complex, building materials, anti-aircraft guns, surface-to-air missile sites, visibility
interdiction: road/railroad	number of cuts, road/railroad-bed type, anti-aircraft guns, surface-to-air missile sites, visibility
interdiction: bridge	length of bridge, building materials, anti-aircraft guns, surface-to-air missile sites, visibility
area preparation	size of area to prepare, terrain type, protective cover, defensive positions, armored vehicles, anti-aircraft guns, surface-to-air missile sites, visibility
close-air support	force-size to support, terrain type, protective cover, defensive positions, armored vehicles, anti-aircraft guns, surface-to-air missile sites, visibility
24 Hour Alert	number of planes to put on alert.

Figure 1

and priority level of the mission. (The desired completion percentage for a 24 hour alert is automatically assumed to be 100%.)

The expert system takes all these mission factors and resolves them against rules to figure out the number of planes, load types and fusings needed for the mission to meet the desired completion level. If there are enough planes and weapons to meet mission specifications, the system suggests the number of planes and weapons required. The system user can then assign or scrap this suggested mission. Assigning the mission causes the system to subtract the number of planes and amount of each weapon used from the appropriate databases, as well as adding the mission to the mission database. If the number of planes or the amount of any weapon type is insufficient for the mission, then the system tells the user the mission cannot be implemented. Factors are then cited as to why the mission cannot be implemented:

> Mission requires X planes but only Y available.
> Not enough of weapon X to implement mission.

Some important factors limiting the mission completion may also be cited:

> Heavy foliage makes completion level difficult.
> Urban cover makes completion level difficult.
> Dense Fog makes completion level difficult.
> Mountainous terrain makes completion level difficult.
> Lowering the desired completion level may help.

2.3 Knowledge Acquisition Phase

The knowledge acquisition stage was interleaved with the design of the program. Initially Dr. Minor worked on the design of the menu framework with the domain expert, Col. Kowalczyk. The mission types were established, and the factors involved in each mission were listed. A method of supplying values for these factors was needed, and so menus were chosen because they would limit difficulties in user input. Menus also allowed the programmer a simple way of defining ranges of values to be input. The information flow for the expert system program had been established, but more interviewing was needed to get menu ranges. The expert supplied much semantic information that was used in developing the menus, such as dividing force sizes into units like squads, platoons and battalions, and dividing anti-aircraft guns and SAM-site sizes into sections, batteries and battalions.

The most important menu established was the desired completion percentage. This menu required many adjustments. The expert recalled that missions with above 60% completion were considered satisfactory, missions with over 75% completion were considered excellent, but missions with over 90% completion were rare. This led to a final percentage range breakdown of : 20-44%, 45-59%, 60-74%, 75-89%, 90-100%. The 60-74% range was considered the basis value for the facts in the rule-base. Since above 90% completion was very difficult, our expert said the number of planes needed to implement a mission at that level would be 3 times the number needed at the 60-74% level. The 75-89% level was considered 1.5 times as difficult, while 45-59% and 20-44% were considered 0.8 and 0.4 times as difficult respectively.

Other factors were decided declaratively. These included things like anti-aircraft guns and SAM-sites. The expert estimated the number of planes these devices could eliminate at each of their respective menu levels. The expert also estimated multiplier values for these other menus: terrain, protective cover and low-altitude visibility. These multipliers were created to increase the number of planes to a level that would make the desired completion level attainable. The number of planes required to implement every mission type at the basis level was recorded and placed in the rule-base. The load types and fusings associated with each mission type was also recorded. Certain mission types could be affected by defensive positions and protective cover factors, which could also require different weapon loads.

There were few decision rule requirements for the system. Since the number of planes for each mission type and each menu level were in the database, alternative rules were no problem[7]. Weapon loads were determined in a similar fashion and again alternative rules did not occur. This was due to the menu implementation which narrowed the weapon load assignments by supplying specific menu values.

2.4 Implementing the Knowledge Base

The implementation of the knowledge base was done as follows by Grant Wright. Each mission had specific plane and weapon assignments for each menu value. The basis completion percentage value was 60-74%. Thus if the menu for mission type type_x had menu values 1, 2, and 3, and these menu values required 5, 10, and 20 planes to allow completion at the basis level then the facts would look like this:

```
assn{.60} (type_x,1,5).
assn{.60} (type_x,2,10).
assn{.60} (type_x,3,20).
```

To find the values of these at different percentages, the following rules would be used:

```
assn{.20} (type_x,M,NP) :- assn{.60} (type_x,M,N), NP is N*0.4.
assn{.45} (type_x,M,NP) :- assn{.60} (type_x,M,N), NP is N*0.8.
assn{.75} (type_x,M,NP) :- assn{.60} (type_x,M,N), NP is N*1.5.
assn{.90} (type_x,M,NP) :- assn{.60} (type_x,M,N), NP is N*3.0.
```

This is how all initial plane assignments are handled. The actual number of planes can be higher because of multipliers from other factors, as well as the addition of planes because of factors like anti-aircraft guns and SAM-sites.

Load types and fusings are decided by asserting appropriate percentages of weapons for each mission type. The weapon types and fusings vary on specific menu criteria that characterize a mission. Load rules look like this:

```
load_missiontype( MissionNum, Factors, Percentages ) :-
```

```
assert( loadpc( MissionNum, Load1, Fuse1, Percent1) ),
assert( loadpc( MissionNum, Load2, Fuse2, Percent2) ),
                        .
                        .
                        .
assert( loadpc( MissionNum, Loadn, Fusen, Percentn) ).
```

The Percentages variable is needed for missions that have more than one set of load constraints. Area preparation missions, for example, are affected by defensive positions and protective cover.

Implementing the weapon loads as assertions to a new loadpc database was necessary for many reasons. The most important reason was the absence of list access in MPROLOG. The easiest way of implementing the loads would have been to construct a list of load and fuse tuples. This would have avoided costly assertions and deletions to unnecessary databases.

A similar database called load_amt was created later to find the actual amounts of each weapon for the mission. Then a database with tuples for mission number, number of planes, plane loads, load fusings, and load amount was also created to store the actual mission data. Even this could have been implemented more efficiently as lists.

Certain allowances had to be made for underflow. Missions against small forces with low desired completion levels tended to need less than 2 planes. This caused zero planes carrying nothing to be asserted into the database because most missions needed more than 2 weapon types. This was solved using a maximum attribute attractiveness rule. The expert supplied weapon types for each mission that were considered most essential. Only these weapons were asserted in cases of underflow.

2.5 Examples

All examples have been tested using the Wing Commander expert system. The mission suggestions and failure factors are those provided by the expert system.

The menu ordering is that found when running the Wing Commander. A listing of menus used by the expert system is given in Appendix A, and the small letters used below refer to that list. These examples all start by choosing "plan mission" from the top-level menu (a).

Example 1

Close air support mission: a company is to be provided with close air support. The company is travelling over rough terrain under a jungle cover. Enemy positions will be underground (in tunnels). No armored vehicles, anti-aircraft guns, SAM-sites are expected to be encountered. Low altitude visibility is clear and the mission should be carried out to 75% satisfaction level. Set location as "loc 1" and time at 0830 hours.

Menu (appendix letter)	**Selection**
Select Mission Type(b)	close-air support
Select Principle Terrain Type(e)	mountainous / rough

Select Protective Cover Type(f)	heavy forest / jungle
Select Type of Defensive Position(g)	tunnels
Est. Number of Armored Vehicles(m)	none
Est. Number of Anti-aircraft Guns(h)	none
Est. Number of SAM-sites(i)	none
Low Altitude Cover(j)	clear
Desired Completion Percentage(k)	75 - 89%

The Wing Commander system outputs the following suggestion:

5 planes will carry 60 napalm bombs with impact fusing
5 planes will carry 100 500lb HD bombs with impact fusing
1 planes will carry 16 cluster bombs with proximity fusing
4 planes will carry 16 2000lb bombs with delay fusing
All 15 planes will carry a total of 30000 rounds of 20mm

Example 2

Bridge mission: A large concrete bridge is to be destroyed. The bridge crosses a gorge of approximately 250 feet. Expect a battery of anti-aircraft guns and a section of SAM-sites. Visibility is hazy due to heavy rain in the area. A 50% completion level will be satisfactory. Set location as "loc 2" and time at 1130 hours.

Menu (appendix letter)	**Selection**
Select Mission Type(b)	interdiction
Select Principle Target Type (c)	bridge
Estimate Length of Bridge in Feet(r)	200-300
Select Bridge Material(s)	concrete
Est. Number of Anti-aircraft Guns(h)	3 - 6 (battery)
Est. Number of SAM-sites(i)	1 - 2 (section)
Low Altitude Cover(j)	hazy / heavy rain
Desired Completion Percentage(k)	45 - 59%

The Wing Commander system outputs the following suggestion:

2 planes will carry 32 750lb bombs with delay fusing
2 planes will carry 8 2000lb bombs with delay fusing
4 planes will carry 96 air-to-ground missiles with impact fusing
All 8 planes will carry a total of 16000 rounds of 20mm

Example 3

Area preparation: an area of more than one square mile has been requested to be cleared. The area is in a mountainous area in thick jungle. Enemy positions include a set of bunkers and a platoon of armored vehicles. No anti-aircraft devices are expected and visibility will likely be clear. Completion level of 70% is satisfactory. Set location as "loc 3" and time at 0310 hours.

Menu (appendix letter)	**Selection**
Select Mission Type(b)	area preparation
Select Area in Square Miles (t)	1 - 1 1/2
Select Principle Terrain Type (e)	mountainous
Select Protective Cover Type(f)	heavy forest / jungle
Select Type of Defensive Position(g)	concrete bunkers
Est. Number of Armored Vehicles(m)	1 - 4 (platoon)
Est. Number of Anti-aircraft Guns(h)	none
Est. Number of SAM-sites(i)	none
Low Altitude Cover(j)	clear
Desired Completion Percentage(k)	60 - 74%

This mission will fail. The Wing Commander states the following:

Not enough planes to implement mission. 57 needed but only 50 available.
Mountainous terrain makes completion level difficult.
Heavy foliage makes completion level difficult.
Lowering the desired completion level may help.

If the mission is attempted again at 45-59% completion, the result is:

12 planes will carry 192 cluster bombs with proximity fusing

```
2 planes will carry 40 500lb HD bombs with impact fusing
2 planes will carry 32 750lb bombs with impact fusing
2 planes will carry 8 2000lb with impact fusing
15 planes will carry 60 2000lb bombs with delay fusing
15 planes will carry 240 750lb bombs with delay fusing
All 48 planes will carry a total of 96000 rounds of 20mm
```

2.6 Limitations of the Expert System

The framework of the program is very restrictive. In the era of the wing commander, planes could carry mixed loads, although only certain loads could be mixed on any one plane[5]. This could have been implemented if the list notation of PROLOG existed in MPROLOG.

The system automatically quits resolving against rules when there is an insufficient amount of any weapon load. Suggesting the mission anyway and simply telling the user that it cannot be implemented would have been better.

The numbers for most of the mission suggestions seem to "round". This problem occurs with floating point numbers, which must be represented as strings in MPROLOG. Multiplication of these numbers, which lose accuracy being represented as strings, compounds the numerical error. MPROLOG also had no way of providing an upper-bound limit on the completion percentage.

Verifying the expert systems operation could not be carried out fully. The only source to verify the system was the domain expert since access to the military information needed was impossible. The verification consisted, more or less, of showing the domain expert a series of missions suggested by the system for given inputs. These simulations were carried out on a variety of different mission types, and in each case the domain expert evaluated the results of the program and suggested corrections. Adjustments to the rule base were made until the expert verified that subsequent simulations produced satisfactory results.

3 Evaluation of MPROLOG

3.1 Advantages of MPROLOG

MPROLOG has a distinct advantage over other expert system design tools since it is based on PROLOG. PROLOG is familiar to everyone in the artificial intelligence community, and thus most knowledge engineers would be familiar with it. In MPROLOG one can perform non-monotonic reasoning, a feature present in both KEE and ART. Database facts are basically the same in all three systems. The rule types in KEE and ART can be modeled in MPROLOG, but both KEE and ART have messier LISP-like implementations. A person designing an expert system in KEE or ART must have a good understanding of LISP, must learn the syntax of the specific tool, and must learn to work with a new system-dependent environment. MPROLOG only requires the designer to understand PROLOG. MPROLOG does not require specialized training of system personnel to keep it running. It can simply be loaded onto a system and run like any PROLOG implementation.

MPROLOG features a way of updating information in the database with the built-in predicates *support* and *detract*, as well as the standard PROLOG predicates *assert* and *retract*. These operators work by "adding" some confidence percentage to a given rule when *support* is used, and "subtracting" some confidence percentage when *detract* is used. In diagnostic expert systems applications, this would be incredibly useful.

3.2 Problems with MPROLOG

Some of MPROLOG's advantages become its problems. The fact that MPROLOG is based on PROLOG gives it the power of that programming language, but also the problems associated with PROLOG are inherited: basic counting is a task in PROLOG; asserting and deleting database items is not order preserving; infinite looping and recursion can occur easily in PROLOG; some predicates allow the programmer to write rules that are unsafe.

List notation was left out of MPROLOG, and implementing list operations using the *lisp* predicate is incredibly difficult. Using the LISP *quote* function seems impossible, and the lack of the "|" operator from PROLOG is a major drawback. Floating point handling must also be improved and expressions should be evaluated as arguments to functions and predicates before being passed. Another helpful fix would involve the uncertainty values. If the system provided ways to allow variables in the uncertainty value field, MPROLOG would be more powerful.

Procedural attachments available in ART and KEE are not available in MPROLOG. Through these procedural attachments, both ART and KEE support object oriented programming. The inheritance properties in ART and KEE provide easy classification methods not present in MPROLOG. Of course a programmer has the power in MPROLOG to create these properties and attachments, although they are not built-in.

Overall, the problems of MPROLOG may hinder the knowledge engineer, but all of these problems are correctable. If mended, MPROLOG will be very useful.

4 Bibliography

1. **ART Reference Manual: ART version 3.0**, Inference Corp., Los Angeles, Jan. 1987.

2. **ART Programming Tutorial, Vol 1**, Inference Corp., Los Angeles.

3. Clocksin, W.F. and C.S. Mellish, **Programming in Prolog: Third Edition**, Springer-Verlag, Berlin, 1987.

4. Flatebo, Martin, "MPROLOG Manual and Source Code", Dept. of Computer Science Report CSR-88-003, University of Nevada, Las Vegas, Jan. 1988.

5. **Flight Manual T.O. 1F-111A-1, for USAF Series Aircraft F-111A**, United States Air Force, Sept. 1974.

6. Hedberg, Sara and Marilyn Seltzner, **Knowledge Engineering Environment (KEE) System: Summary of Release 3.1**, Intellicorp, Mountain View, CA, 1988.

7. McGraw, Karen L. and Karan Harbison-Briggs, **Knowledge Acquisition: Principles and Guidelines**, Prentice-Hall, NJ, 1989.

8. Rich, Elaine, **Artificial Intelligence**, McGraw-Hill, New York, NY, 1983.

9. Weiss, Sholom and Casimir Kulikowski, **A Practical Guide to Designing Expert Systems**, Rowman & Allanheld, Totowa, NJ, 1984.

A Appendix: Menus from Wing Commander

a) Fighter Mission Dispatching System:

plan mission
change mission
check status
change database
quit

b) Select Mission Type:

interdiction
area preparation
close-air support
on 24-hour alert

c) Select Interdiction Target Type:

personnel camp
unarmored vehicles (convoy)
armored vehicles
building complex
roads/railroads
bridge

d) Select Approximate Force Size:

squad (1 - 10)
platoon (11 - 30)
company (31 - 100)
battalion (101 - 350)
regiment (351 - 1000)
larger (over 1000)

e) Select Principle Terrain Type:

flat
rolling hills
mountainous / rough

f) Select Protective Cover Type:

open

light trees / scattered buildings
heavy forest / jungle
city / urban area

g) Select Principle Type of Defensive Position:

none
trenches/earth works
tunnels
reinforced concrete bunkers/caves

h) Estimate Number of Anti-aircraft Guns:

none
1 - 2 (section)
3 - 6 (battery)
7 - 18 (battalion)
over 18

i) Estimate Number of SAM Sites:

none
1 - 2 (section)
3 - 6 (battery)
7 - 18(battalion)
over 18

j) Low Altitude Cover:

clear
light rain / drizzle
haze / heavy rain / fog patches
dense fog / smoke

k) Estimate Desired Mission Completion Level:

20 - 44%
45 - 59%
60 - 74%
75 - 89%
90 - 100%

l) Estimate Number of Unarmoured Vehicles:

less than 5

5 to 10
11 to 15
16 to 20
more than 20

m) Estimate Number of Armored Vehicles:

none
1 - 4 (platoon)
5 - 12 (company)
13-36 (battalion)
more than 36

n) Estimate Size of Buildings in Square Feet:

under 1000
1000 to 2500
2500 to 5000
5000 to 10,000
10,000 to 20,000
over 20,000

o) Select Building Material Type:

wood, straw, or tents
sandbag reinforced hut
brick
reinforced concrete

p) Select Road/Railroad Target Type:

simple cut
double cut
intersection/fork
major junction

q) Road Type / Railroad Underlining:

dirt
macadam / rock
concrete

r) Estimate Length of Bridge in Feet:

under 50

50 - 100
100 - 200
200 - 300
over 300

s) Select Bridge Material:

wood
concrete
steel

t) Select Area in Square Miles:

0 - 1/4
1/4 - 1/2
1/2 - 1
1 - 1 1/2
1 1/2 - 2

B Appendix: Source Code for Wing Commander

```
/*                     Structure of Wing Commander expert system

        level   description                      predicates
        -----   -----------                      ----------
        1       Plan Mission                     process(1,..)
        1.1       Interdictions                  process11
        1.1.1       personnel concentration      process11(pc,..)
        1.1.2       unarmored vehicles           process11(uv,..)
        1.1.3       armored vehicles             process11(av,..)
        1.1.4       building complex             process11(bc,..)
        1.1.5       roads / railroads            process11(rr,..)
        1.1.6       bridges                      process11(bridge,..)
        1.2       Area Preparation               process1(areaPrep,..)
        1.3       Close-air Support              process1(closeAirSupport,..)
        1.4       On 24 Hour Alert               process1(on24HrAlert,..)
        2       Change Mission                   process(2,..)
        2.1       Delete                         process2(Mnum,del)
        2.2       Location                       process2(Mnum,loc)
        2.3       Time                           process2(Mnum,time)
        2.4       Completion %                   process2(Mnum,cmp)
        3       Check Status                     process(3,..)
        3.1       Status of Planes               process3(1)
        3.2       Status of Weapon Loads         process3(2)
        3.3       Status of Missions             process3(3)
        4       Change Database                  process(4,..)
        4.0       Change Password for Access     process4(0)
        4.1       Change Plane Amounts           process4(1)
        4.1.1       number of unassigned         process41(ua)
        4.1.2       number on maintenance        process41(sm)
        4.1.3       number with battle damage    process41(bd)
        4.1.4       number malfunctioning        process41(mf)
        4.2       Change weapon amounts          process4(2)
        4.2.1       amount of 20mm               process42(mm20)
        4.2.2       amount of Cluster Bombs      process42(cb)
        4.2.3       amount of 500lb high drag    process42(hd500)
        4.2.4       amount of 500lb low drag     process42(ld500)
        4.2.5       amount of 750lb              process42(m750)
        4.2.6       amount of 2000lb             process42(m2000)
        4.2.7       amount of napalm             process42(napalm)
        4.2.8       amount of air-to-ground      process42(atg)
        5       Quit                             process(5,..)                    */

round(X,Y) :-
        W is X + 0.5,
        lisp(Y,truncate,W).

trunc(X,Y) :- lisp(Y,truncate,X).

bk_amt(mm20,0).         /*  Keeps track of load usage during weapon assignment */
bk_amt(cb,0).           /*  In case same weapon used with different fusing     */
bk_amt(hd500,0).        /*  Must be reset every time a mission is planned      */
bk_amt(ld500,0).
bk_amt(m750,0).
bk_amt(m2000,0).
bk_amt(napalm,0).
bk_amt(atg,0).

name_of(mm20,"20 mm") :- !.
name_of(cb,"cluster bombs") :- !.
name_of(hd500,"500lb high drag bombs") :- !.
name_of(ld500,"500lb low drag bombs") :- !.
name_of(m750,"750lb bombs") :- !.
name_of(m2000,"2000lb bombs") :- !.
name_of(napalm,"napalm") :- !.
name_of(atg,"air to ground missiles") :- !.

expert :-                /* Start the database by typing expert at the '?-' prompt */
        clear_db,
        lisp(Ans,newdb),
        open_proper( Ans ), !,
        do(0).

clear_db :-
        mission(M, _, Sel, _, _, _ ),
        retract( mission(M,_Sel,_,_,_) ),
        kill_loadamts(M), kill_sugg(M), kill_loadpc(M),
        removeTypeAttr(M,Sel),
        clear_db.

clear_db :- cpass, ccounts, cplanes, cweaps.

cpass :-
        password(_),
        replace( password(_), password(secret) ).
cpass.
ccounts :-
        count(_), miscount(_),
        retract( count(_) ),
        retract( miscount(_) ).
ccounts.
cplanes :-
        uaplanes(_),
        retract( uaplanes(_) ), retract( asplanes(_) ), retract( bdplanes(_) ),
        retract( mfplanes(_) ), retract( smplanes(_) ).
cplanes.
cweaps :-
        weapon( mm20, _ ),
        retract( weapon( mm20, _ ) ), retract( weapon( cb, _ ) ), retract( weapon( hd500, _)),
        retract( weapon(ld500,_) ), retract( weapon( m750, _ ) ), retract( weapon( m2000, _)),
        retract( weapon( napalm, _ ) ), retract( weapon( atg, _ ) ).
cweaps.

open_proper( n ) :-
        ask('Supply file name of your database (in quotes with .mprolog):',Name),
        [Name],nl,
        write(' File ',Name,' is in the work space...'),nl.
```

```
open_proper( y ) :-
        ["standards.mprolog"].

do(Activity) :-
        Activity = quit,
        lisp(Ans, exitMenu),
        saveDB(Ans).

saveDB(Ans) :-          /*   Save session in a file */
        Ans = 'y',
        write(" Supply prefix file name (without .mprolog) to save database in:"),
        read(SaveMe), nl,
        tell(SaveMe),
        count_listing,
        missionListing,
        planeListing,
        weaponListing,
        list1, list2, list3,
        list4, list5, list6, list7, list8,
        list9, list10, list11, list12,
        list13, list14, list15,
        told.

saveDB(Ans) :- told.

count_listing :- listing( count ), listing( miscount ).
count_listing.

planeListing :-
        listing(uaplanes),listing(smplanes),listing(bdplanes),
        listing(mfplanes),listing(asplanes).
planeListing.

weaponListing :- listing(weapon).
weaponListing.

missionListing :- listing(mission).
missionListing.

list1  :- listing(new_pl_amt).
list1.
list2  :- listing( bk_amt ).
list2.
list3  :- listing( password ).
list3.
list4  :- listing(personnel).
list4.
list5  :- listing(uVehicles).
list5.
list6  :- listing(aVehicles).
list6.
list7  :- listing(buildings).
```

```
list7.
list8  :- listing(cuts).
list8.
list9  :- listing(bridge).
list9.
list10 :- listing(areaPrep).
list10.
list11 :- listing(closeAirSupport).
list11.
list12 :- listing(on24HrAlert).
list12.
list13 :- listing(loadpc).
list13.
list14 :- listing(load_amt).
list14.
list15 :- listing(sugg_miss).
list15.


do(Activity) :- !, process(Activity, Next), do(Next).

process(0, Next) :-        /* Process selections from menu */
        lisp(Selection, menu0),
        process(Selection, Next).


process(1, 0) :-nl,                /* Plan Mission level 1 */
        write('Plan mission'), nl,
        missionNum(Mnum),
        resetbk,
        ask('Enter mission location:', Mloc),
        ask('Enter mission starting time:', Mstart),
        lisp(Selection, selectMission),
        process1(Selection, Mnum, Inter),
        get_cp( Sel, Comp ),
        assignPlanes(Mnum, Selection, Inter, Comp, NPs), !,
        round( NPs, Planes ),
        !,
        implement_mission( Mnum, Mloc, Mstart, Planes, Selection, Inter, Comp ).

get_cp( on24HrAlert, 0).
get_cp( Sel, Comp ) :-
        lisp(Comp,compperc).

resetbk :-
        replace(bk_amt(mm20,_),bk_amt(mm20,0)),
        replace(bk_amt(cb,_),bk_amt(cb,0)),
        replace(bk_amt(hd500,_),bk_amt(hd500,0)),
        replace(bk_amt(ld500,_),bk_amt(ld500,0)),
        replace(bk_amt(m750,_),bk_amt(m750,0)),
        replace(bk_amt(m2000,_),bk_amt(m2000,0)),
        replace(bk_amt(napalm,_),bk_amt(napalm,0)),
        replace(bk_amt(atg,_),bk_amt(atg,0)).
```

```
/*   Now, assign planes, loads & fusings !!!

        assignPlanes gets total # of planes needed and computes load factors
        by asserting a new database. Then loadmtype is called to assign
        those loads to each plane for missions of type mtype...          */

assignPlanes(M, on24HrAlert, none, Comp, Nplanes) :-
        on24HrAlert( M, Nplanes ),
        load_24hr( M ).

assignPlanes(M, closeAirSupport, none, Comp, TotPlanes) :-
        closeAirSupport( M, FS, Terr, PC, DP, AV, AA, SS, LC),
        assnpc( Comp, closeAirSupport, FS, Nplanes ),
        assign_std_factors( AA, SS, AddPlanes ),
        assign_avs( AV, AddMorePlanes ),
        compute_terr_factors( Terr, PC, LC, TFactor ),!,

        X is Nplanes * TFactor,        Y is AddPlanes + AddMorePlanes,
        TotPlanes is X + Y,

        cond_load( M, closeAir, DP).
/*      load_closeAir( M, 0.5 ),                   Cover only for area prep
        cov_and_def( M, PC, DP ).        */

cond_load( M, closeAir, none ) :-
        load_closeAir( M, 1.0 ).

cond_load( M, closeAir, DP ) :-
        load_closeAir( M, 0.75 ),
        defpos_loads( M, 0.25, DP ).

assignPlanes(M, areaPrep, none, Comp, TotPlanes) :-
        areaPrep( M, AR, Terr, PC, DP, AV, AA, SS, LC ),
        assnpc( Comp, areaprep, AR, Nplanes),
        assign_std_factors( AA, SS, AddPlanes ),
        assign_avs( AV, AddMorePlanes ),
        compute_terr_factors( Terr, PC, LC, TFactor ), !,

        X is Nplanes * TFactor,        Y is AddPlanes + AddMorePlanes,
        TotPlanes is X + Y,

        load_ap( M, 0.75 ),
        cov_and_def( M, PC, DP ).

assignPlanes(M, interdiction, pc, Comp, TotPlanes) :-
        personnel( M, FS, Terr, Pcov, DP, AA, SS, LC ),
        assnpc( Comp, pc, FS, Nplanes ),
        assign_std_factors( AA, SS, AddPlanes ),
        compute_terr_factors( Terr, Pcov, LC, TFactor ), !,

        X is Nplanes * TFactor,
        TotPlanes is X + AddPlanes,
        cond_load( M, pc, DP ).
/*       load_pers( M, 0.5 ),                    Cover only for area Prep
        cov_and_def( M, PC, DP ).        */

cond_load( M, pc, none ) :-
        load_pers( M, 1.0 ).

cond_load( M, pc, DP ) :-
        load_pers( M, 0.75 ),
        defpos_loads( M, 0.25, DP ).

cov_and_def( M, PC, none ) :-
        cover_loads( M, 0.25, PC ).

cov_and_def( M, PC, DP ) :-
        cover_loads( M, 0.125, PC ),
        defpos_loads( M, 0.125, DP ).

assignPlanes(M, interdiction, uv, Comp, TotPlanes) :-
        uVehicles( M, UV, Terr, PC, AA, SS, LC ),
        assnpc( Comp, uv, UV, Nplanes ),
        assign_std_factors( AA, SS, AddPlanes ),
        compute_terr_factors( Terr, PC, LC, TFactor ), !,

        X is Nplanes * TFactor,
        TotPlanes is X + AddPlanes,

        load_uv( M, 1.0 ).
/*       load_uv( M, 0.5 ),                      cover shouldn't affect
        cover_loads( M, 0.5, PC ). */

assignPlanes(M, interdiction, av, Comp, TotPlanes) :-
        aVehicles( M, AV, Terr, PC, AA, SS, LC ),
        assnpc( Comp, av, AV, Nplanes ),
        assign_avs( AV, AddPlanes ),
        compute_terr_factors( Terr, PC, LC, TFactor ), !,

        X is Nplanes * TFactor,
        TotPlanes is X + AddPlanes,
        load_av( M, 1.0 ).
/*       load_av( M, 0.5 ),                      cover shouldn't affect
        cover_loads( M, 0.5, PC ). */

assignPlanes(M, interdiction, bc, Comp, TotPlanes) :-
        buildings( M, Area, Mats, AA, SS, LC ),
        assnpc( Comp, bc, Area, Nplanes ),
        assign_std_factors( AA, SS, AddPlanes ), !,

        TotPlanes is Nplanes + AddPlanes,

        load_buildmats( M, 1.0, Mats ).
```

```
assignPlanes(M, interdiction, rr, Comp, TotPlanes) :-
        cuts( M, NC, RT, AA, SS, LC ),
        assnpc( Comp, rr, NC, Nplanes ),
        assign_std_factors( AA, SS, AddPlanes ), !,

        TotPlanes is Nplanes + AddPlanes,

        load_cuts( M, 1.0, RT).

assignPlanes(M, interdiction, bridge, Comp, TotPlanes) :-
        bridge( M, Len, Mat, AA, SS, LC ),
        assnpc( Comp, bridge, Len, Nplanes ),
        assign_std_factors( AA, SS, AddPlanes ), !,

        TotPlanes is Nplanes + AddPlanes,

        load_bridgemats( M, 1.0, Mat ).

ask(Question, Response) :- write(Question),
        read(Response),nl,write("response:<",Response,">"),nl.

process1(interdiction, Mnum, Selection) :- lisp(Selection, targetType),
        process11(Selection, Mnum).

process11(pc,Mnum) :-
        lisp(ES,forceSize),
        lisp(Terrain,terrain),
        lisp(Pcov,pcover),
        lisp(DefPos,defpos),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(personnel(Mnum,_,_,_,_,_,_,_),
                personnel(Mnum, ES, Terrain,Pcov, DefPos, AA, SS, LC)).

replace(C1,C2) :- C1, retract(C1), assert(C2).
replace(C1,C2) :- assert(C2).

process11(uv,Mnum) :-
        uavehicles(UV),
        lisp(Terrain,terrain),
        lisp(Pcov,pcover),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(uVehicles(Mnum,_,_,_,_,_,_), uVehicles(Mnum,UV,Terrain,Pcov,AA,SS,LC)).

process11(av,Mnum) :-
        lisp(AV,avehicles),
        lisp(Terrain,terrain),
        lisp(Pcov,pcover),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(aVehicles(Mnum,_,_,_,_,_,_), aVehicles(Mnum,AV,Terrain,Pcov,AA,SS,LC)).
```

```
process11(bc,Mnum) :-
        lisp(SF,buildingarea),
        lisp(Material, buildingMats),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(buildings(Mnum,_,_, _, _, _), buildings(Mnum. SF, Material, AA, SS,LC)).


process11(rr,Mnum) :-
        lisp(FC,ncuts),
        lisp(RT,roadtype),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(cuts(Mnum,_,_,_,_,_), cuts(Mnum,FC,RT,AA,SS,LC)).

process11(bridge,Mnum) :-
        lisp(Len,bridgelen),
        lisp(Material, bridgeMats),
        airdef(AA.SS),
        lisp(LC,locover),
        replace(bridge(Mnum,_,_,_,_,_), bridge(Mnum, Len, Material, AA, SS, LC)).

process1(areaPrep, Mnum, none) :-
        lisp(Area, area),
        lisp(Terrain, terrain),
        lisp(Pcov, pcover),
        lisp(DefPos, defPos),
        armoredVehicles(AV),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(areaPrep(Mnum,_,_,_,_,_,_,_,_),
                areaPrep(Mnum, Area, Terrain, Pcov, DefPos, AV, AA, SS, LC)).

armoredVehicles(AV) :- lisp(AV,av2).
uavehicles(UV) :- lisp(UV,uvehicles).

airdef(X,Y) :- aaGuns(X), samSites(Y).
aaGuns(AAguns) :- lisp(AAguns,aadef).
samSites(SAMsites) :- lisp(SAMsites,samsites).

process1(closeAirSupport, Mnum, none) :-
        lisp(ForceSize, forceSize),
        lisp(Terrain, terrain),
        lisp(Pcov, pcover),
        lisp(DefPos, defPos),
        armoredVehicles(AV),
        airdef(AA,SS),
        lisp(LC,locover),
        replace(closeAirSupport(Mnum,_,_,_,_,_,_,_,_),
                closeAirSupport(Mnum,ForceSize,Terrain,Pcov,DefPos,AV,AA,SS,LC)).

process1(on24HrAlert, Mnum, none) :-
```

```
        write('24 hour alert'), nl, uaplanes(Count),
        nl, write('Number of unassigned planes = '),
        write(Count), nl,
        write('How many planes are to be put on 24 hour alert? '),
        read(NumPlanes), nl,
        replace(on24HrAlert(Mnum,_),on24HrAlert(Mnum,NumPlanes)).

suggest( M ) :-          /* Output a "suggestion" for the given mission M */
        sugg_miss( M, Load, Fuse, Amt, Planes ),
        name_of(Load,Lname),
        print_line( Load, Planes, Amt, Lname, Fuse ),
        fail.

suggest( M ).

print_line( mm20, Planes, Amt, Lname, Fuse ) :-
        write("-----------------------------------------------"),nl,
        write("All ",Planes," will also carry ",Amt," rounds of 20mm "),nl,nl,nl.

print_line( Load, Planes, Amt, Lname, Fuse ) :-
        Load |= mm20,
        write(Planes," planes will carry ",Amt," units of ",Lname," with a ",Fuse," fusing."),
        nl.

/*      implement_mission( Mnum, Nplanes, MissionType, InterdictionType )

        At this point, an attempt to implement the mission is put forth.
        Failure in implement mission occurs if the number of planes available
        is insufficient.  Failure in calculations results if an augmented
        plane amount is still insufficient.  Failure in more_calculations
        results when there are insufficient weapon loads of any kind.        */

implement_mission( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ) :-
        uaplanes( X ),
        X >= NP,
        !,
        calculations( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ).

implement_mission( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ) :-
        uaplanes(X),
        write("Not enough planes in database for mission ",Mnum),nl,
        write("        Mission requires ",NP," planes, only ",X," available."),nl,nl,
        excuse( Mnum, Sel, Inter, Comp, NP ),
        finally_implement( 2, Mnum, Sel, Inter, NP ).

calculations( Mnum, Mloc, Mstart,  NP, on24HrAlert, none, Comp ):-
        uaplanes( X ),
        X >= NP,
        special_24( Mnum, NP ),
        more_calculations( Mnum, Mloc, Mstart, NP, on24HrAlert, none, Comp ).

calculations( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ):-
        Sel |= on24HrAlert,
        calc_load_amts( Mnum, NP ),
        new_pl_amt( Mnum, NNP ),
        uaplanes( X ),
        X >= NNP,
        retract( new_pl_amt( Mnum, NNP ) ),
        !,
        more_calculations( Mnum, Mloc, Mstart, NNP, Sel, Inter, Comp ).

calculations( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ) :-
        not( new_pl_amt(_,_) ),
        too_few( Mnum, Sel, Inter, NP),
        more_calculations( Mnum, Mloc, Mstart, NP, Sel, Inter, Comp

calculations( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ) :- /*  Not enough of  weapon */
        uaplanes(X),
        new_pl_amt( Mnum, NNP ),
        retract( new_pl_amt( Mnum, NNP ) ),
        write("Not enough planes in database for mission ",Mnum),nl,
        write("         Mission requires ",NNP," planes, only ",X," available."),nl,nl,
        excuse( Mnum, Sel, Inter, Comp, NP ),
        finally_implement( 2, Mnum, Sel, Inter, NP ).

special_24( Mnum, NP ) :-       /*  Kludge to avoid approximations on 24Hr Alert mission */
        L1 is .45*NP,
        L3 is .10*NP,
        trunc(L1,R1),
        trunc(L1,R2),
        round(L3,R3),
        figure(NP,R1,R2,R3,X,Y),
        loadpc( Mnum, LD1, F1, PC1 ),
        retract( loadpc( Mnum, LD1, F1, PC1 ) ),
        loadpc( Mnum, LD2, F2, PC2 ),
        retract( loadpc( Mnum, LD2, F2, PC2 ) ),
        loadpc( Mnum, LD3, F3, PC3 ),
        retract( loadpc( Mnum, LD3, F3, PC3 ) ),
        std_load_amts( LD1, Amt1 ),
        std_load_amts( LD2, Amt2 ),
        std_load_amts( LD3, Amt3 ),
        AN1 is Amt1*X,
        AN2 is Amt2*Y,
        AN3 is Amt3*R3,
        load_and_suggest( Mnum, AN1, LD1, F1, X ),
        load_and_suggest( Mnum, AN2, LD2, F2, Y ),
        load_and_suggest( Mnum, AN3, LD3, F3, R3 ).

figure( NP, X, Y, Z, NX, NY ) :-
        T is X + Y + Z,
        NP = T,
        NX is X,
        NY is Y.

figure( NP, X, Y, Z, NX, NY ) :-
        X = Y, XXX is  X + 1,
        figure( NP, XXX, Y, Z, NX, NY ).

figure( NP, X, Y, Z, NX, NY ) :- YYY is Y + 1, figure( NP, X, YYY, Z, NX, NY ).

set_npa( A, B, C ) :- B > C, A is B.
set_npa( A, B, C ) :- A is C.

too_few( Mnum, Sel, Inter, NP) :-          /* Too few is evoked when underflow occurs */
        kill_loadamts( Mnum ), kill_sugg( Mnum ),
        princ_load( Mnum, Sel, Inter, Lode, Fuse ),
        std_load_amts( Lode, Amt ),
        weapon( Lode, AmtAvail ),
        AmtNeeded is NP * Amt,
        AmtAvail > AmtNeeded,
        assert( load_amt( Mnum, Lode, Fuse, AmtNeeded ) ),
        assert( sugg_miss( Mnum, Lode, Fuse, AmtNeeded, NP ) ).


too_few( Mnum, Sel, Inter, NP) :-
        princ_load( Mnum, Sel, Inter, Lode, Fuse ),
        assert( load_amt( Mnum, Lode, Fuse, 0 ) ).

too_few( A, B,     C,    D ).

more_calculations( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ) :-
        calc_20( Mnum, NP ),
        not( load_amt( Mnum,_,_,0 ) ),nl,nl,nl,
        write("   Here is a suggestion for implementing mission ",Mnum),nl,
        write("----------------------------------------------"),nl,nl,
        suggest( Mnum ),
        lisp( Ans, assnmission ),
        finally_implement( Ans, Mnum, Sel, Inter, NP ),
        assert_if_necessary( Ans, Mnum, Mloc, Sel, Mstart, NP, Comp ).


more_calculations( Mnum, Mloc, Mstart,  NP, Sel, Inter, Comp ) :- /* Not enough of  weapon */
        report_failures( Mnum ),
        excuse( Mnum, Sel, Inter, Comp, NP),
        !,
        finally_implement(2, Mnum, Sel, Inter, NP ).

load_20( Mnum, NP ) :-
        std_load_amts( mm20, Amt ),
        weapon( mm20, Old ),
        SP is Amt*NP,
        NewAmt is Old - SP,
        replace( weapon(mm20,Old), weapon(mm20,NewAmt) ).

assert_if_necessary( 1, Mnum, Mloc, Sel, Mstart, NP, Comp ) :-
        assert(mission( Mnum, Mloc, Sel, Mstart, NP, Comp )),
        uaplanes( X ),  asplanes( Z ),  miscount( R ),
        Y is X - NP,    W is Z + NP, S is R + 1,
        replace( uaplanes(X), uaplanes(Y) ),
          replace( asplanes(Z), asplanes(W) ),
          replace( miscount(R), miscount(S) ).

assert_if_necessary( 2, Mnum, _, _, _, _, _ ) :- kill_sugg( Mnum ).

/*  excuse cover possible reasons for a missions failure */

excuse( Mnum, closeAirSupport, none, Comp, N ) :- !,
          closeAirSupport( Mnum, _, Terrain, Pcover, _, _, _, _, LC),
          terrExcuse( Terrain ),
          pcovExcuse( Pcover ),
          compExcuse( Comp ).

excuse( Mnum,areaPrep , none, Comp, N ) :- !,
          areaPrep( Mnum, _, Terrain, Pcover, _, _, _, _, LC),
          terrExcuse( Terrain ),
          pcovExcuse( Pcover ),
          compExcuse( Comp ).

excuse( Mnum, interdiction, pc, Comp, N ) :- !,
          personnel( Mnum, _, Terrain, Pcover, _, _, _, LC),
          terrExcuse( Terrain ),
          pcovExcuse( Pcover ),
          compExcuse( Comp ).

excuse( Mnum, interdiction, uv, Comp, N ) :- !,
          uVehicles( Mnum, _, Terrain, Pcover, _, _, LC),
          terrExcuse( Terrain ),
          pcovExcuse( Pcover ),
          compExcuse( Comp ).

excuse( Mnum, interdiction, av, Comp, N ) :- !,
          aVehicles( Mnum, _, Terrain, Pcover, _, _LC),
          terrExcuse( Terrain ),
          pcovExcuse( Pcover ),
          compExcuse( Comp ).

excuse( Mnum, interdiction, bc, Comp, N ) :- !,
          buildings( Mnum, _, _, _, _, LC ),
          locovExcuse( LC ),
          compExcuse( Comp ).

excuse( Mnum, interdiction, rr, Comp, N ) :- !,
          cuts( Mnum, _, _, _, _, LC ),
          locovExcuse( LC ),
          compExcuse( Comp ).

excuse( Mnum, interdiction, bridge, Comp, N ) :- !,
          bridge( Mnum, _, _, _, _, LC ),
          locovExcuse( LC ),
          compExcuse( Comp ).

excuse( Mnum, X, Y, Z, W ) :- !.
```

```
compExcuse( 0 ).
compExcuse( 1 ).
compExcuse( 2 ).
compExcuse( 3 ) :- nl, write("Attempting to drop completion percentage may aid you.").
compExcuse( 4 ) :- nl, write("Attempting to drop completion percentage can aid you.").
compExcuse( 5 ) :- nl, write("Attempting to drop completion percentage will aid you.").

terrExcuse(mountains) :- nl, write("Mountainous terrain makes completion level difficult.").
terrExcuse(open).
terrExcuse(hilly).

pcovExcuse(open).
pcovExcuse(sb).
pcovExcuse(jungle) :- nl, write("Heavy foliage makes completion level difficult.").
pcovExcuse(urban) :- nl, write("Urban cover makes completion level difficult.").

locovExcuse(1).
locovExcuse(2).
locovExcuse(3) :- nl, write("Haze/Heavy Rain/Fog Patches make completion level difficult.").
locovExcuse(4) :- nl, write("Dense Fog/Smoke make completion level very difficult.").

calc_20( Mnum, NP ) :-          /*   Find amount of 20mm needed to implement mission   */
        std_load_amts( mm20, Amt ),
        AmtNeeded is NP * Amt,
        weapon( mm20, AmtAvail ),
        AmtNeeded =< AmtAvail,
        assert( sugg_miss(Mnum, mm20, impact, AmtNeeded, NP ) ).

calc_20( Mnum, NP) :-
        assert( load_amt(Mnum, mm20, impact, 0) ).

calc_load_amts( Mnum, NP ) :-   /*  Find amount of any weapon needed to implement mission */
        loadpc( Mnum, Load, Fuse, P ),
        retract( loadpc( Mnum, Load, Fuse, P ) ),
        !,
        X is P*NP,
        round(X,Y),
        add_new_tot( Mnum, Y ),
        std_load_amts( Load, AmtStd ),
        AmtNeeded is Y*AmtStd,            /*  Y is # of planes */
        load_and_suggest( Mnum, AmtNeeded, Load, Fuse, Y ),!,
        calc_load_amts( Mnum, NP ).

calc_load_amts( Mnum, NP ).

add_new_tot( M, 0 ).

add_new_tot( M, X ) :-
        new_pl_amt( M, Y ),
        Z is X + Y,
        replace( new_pl_amt(M,Y), new_pl_amt(M,Z) ).
```

```
add_new_tot( M, X ) :- assert( new_pl_amt( M, X ) ).

load_and_suggest( Mnum, AmtNeeded, Load, Fuse, 0 ).     /* do not assert 0 planes !!! */
load_and_suggest( Mnum, AmtNeeded, Load, Fuse, Y ) :-
        weapon( Load, AmtAvail ),
        bk_amt( Load, Used ),              /* Load Total used on same weapon */
        Total is Used + AmtNeeded,         /* (if weapon uses with different fusings) */
        Total =< AmtAvail,
        replace( bk_amt(Load,Used), bk_amt(Load,Total) ),
        assert_load_and_sugg( Mnum, AmtNeeded, Load, Fuse, Y ).

assert_load_and_sugg( Mnum, AmtNeeded, Load, Fuse, Y ) :-
        load_amt( Mnum, Load, Fuse, Other_Amt ),
        sugg_miss( Mnum, Load, Fuse, Old_Amt, OY ),
        NY is OY + Y,                      /* Changing #'s in case weapon already asserted */
        Total_Amt is Other_Amt + AmtNeeded,
        replace( load_amt( Mnum, Load, Fuse, Other_Amt ),
                 load_amt( Mnum, Load, Fuse, Total_Amt ) ),
        replace( sugg_miss( Mnum, Load, Fuse, Other_Amt, OY ),
                 sugg_miss( Mnum, Load, Fuse, Total_Amt, NY ) ).

assert_load_and_sugg( Mnum, AmtNeeded, Load, Fuse, Y ) :-
        assert( load_amt( Mnum, Load, Fuse, AmtNeeded ) ),
        assert( sugg_miss( Mnum, Load, Fuse, AmtNeeded, Y ) ).

load_and_suggest( Mnum, AmtNeeded, Load, Fuse, Y ) :-
        assert( load_amt( Mnum, Load, Fuse, 0 ) ).

report_failures( Mnum ) :-
        load_amt( Mnum, Load, _, Amt ),
        Amt > 0,
        retract( load_amt( Mnum, Load, _, Amt ) ),
        report_failures( Mnum ).

report_failures( Mnum ) :-
        load_amt( Mnum, Load, _, Amt ),
        Amt = 0,
        name_of( Load, WeaponName ),
        write("Not enough ",WeaponName," to implement mission ",Mnum),
        retract( load_amt( Mnum, Load, _, Amt )),
        report_failures( Mnum ).

report_failures( Mnum ).

finally_implement( 1, M, S, I, NP ) :-
        load_20( Mnum, NP ),
        actually_load( M ).

actually_load( Mnum ) :-
        load_amt( Mnum, Load, Fuse, Amt ),
        Amt > 0,
        weapon( Load, X ),
        Y is X - Amt,
        replace( weapon(Load,X), weapon(Load,Y) ),
        retract( load_amt( Mnum, Load, Fuse, Amt ) ),
        actually_load( Mnum ).

actually_load( Mnum ).

finally_implement( 2, M, S, I, NP ) :-
        deletemtype( M, S, I ),
        count( X ),
        Z is X - 1,
        replace( count( X ), count( Z ) ),
        kill_loadpc( M ),          /*     Remove all scratch work for mission */
        kill_sugg( M ),
        kill_loadamts( M ).

kill_loadpc( M ) :- loadpc( M,_,_,_), retract( loadpc( M, _, _, _ ) ), kill_loadpc( M ).
kill_loadpc( M ).

kill_loadamts( M ) :-
        load_amt( M,_,_,_ ),
        retract( load_amt( M, _, _, _ ) ),
        kill_loadamts( M ).

kill_loadamts( M ).

kill_sugg( M ) :-
        sugg_miss( M, _, _, _, _ ),
        retract( sugg_miss( M, _, _, _, _ ) ),
        kill_sugg( M ).

kill_sugg( M ).

deletemtype( M, interdiction, pc ) :- retract( personnel( M,_,_,_,_,_,_,_,_ ) ).
deletemtype( M, interdiction, uv ) :- retract( uVehicles( M,_,_,_,_,_,_,_ ) ).
deletemtype( M, interdiction, av ) :- retract( aVehicles( M,_,_,_,_,_,_,_ ) ).
deletemtype( M, interdiction, bc ) :- retract( buildings( M,_,_,_,_,_,_ ) ).
deletemtype( M, interdiction, rr ) :- retract( cuts( M,_,_,_,_,_,_ ) ).
deletemtype( M, interdiction, bridge ) :- retract( bridge( M,_,_,_,_,_,_ ) ).
deletemtype( M, closeAirSupport, none ) :- retract( closeAirSupport( M,_,_,_,_,_,_,_,_,_,_ ) ).
deletemtype( M, areaPrep, none ) :- retract( areaPrep( M,_,_,_,_,_,_,_,_,_,_ ) ).
deletemtype( M, on24HrAlert, none ) :- retract( on24HrAlert( M,_ ) ).

std_load_amts( cb, 16 ).          /*  in fours */
std_load_amts( mm20, 2000 ).
std_load_amts( hd500, 20 ).
std_load_amts( ld500, 24 ).
std_load_amts( m750, 16 ).
std_load_amts( m2000, 4 ).
std_load_amts( napalm, 12 ).
std_load_amts( atg, 24 ).         /*  in sizes */

        /*     princ_load( Sel, Inter, Lode, Fuse )          */
princ_load( M, interdiction, pc, hd500, impact ).
princ_load( M, interdiction, uv, atg, impact ).
princ_load( M, interdiction, av, atg, impact ).
princ_load( M, interdiction, bc, Lode, Fuse ) :-
        buildings( M, _, Mats, _, _, _ ),
        pla_appropo( bc, Mats, Lode, Fuse ).

princ_load( M, interdiction, rr, Lode, Fuse ) :-
        cuts( M, _, Mats, _, _, _ ),
        pla_appropo( rr, Mats, Lode, Fuse ).

princ_load( M, interdiction, bridge , Lode, Fuse ) :-
        bridge( M, _, Mats, _, _, _ ),
        pla_appropo( bridge, Mats, Lode, Fuse ).

princ_load( M, closeAirSupport, none, hd500, impact ).
princ_load( M, areaPrep, none, m750, delayed ).
princ_load( M, on24HrAlert, none, hd500, impact ).

pla_appropo( bc, wood, napalm, impact ).
pla_appropo( bc, rhut, m750, proximity ).
pla_appropo( bc, brick, m2000, proximity ).
pla_appropo( bc, rc, m2000, delayed ).

pla_appropo( rr, 1, m2000, delayed ).
pla_appropo( rr, 2, m750, impact ).
pla_appropo( rr, 3, m750, impact ).

pla_appropo( bridge, wood, m750, impact ).
pla_appropo( bridge, concrete, m2000, delay ).
pla_appropo( bridge, steel, m2000, impact ).

missionNum(Mnum) :-
        count(X),
        Mnum is X + 1,
        not(mission(Mnum,_,_,_,_)),
        replace(count(X),count(Mnum)).
missionNum(1).

process(5, quit) :- write('Quit').
```

```
level(complete, 5,"90 - 100%").
level(complete, 4,"75 - 89%").
level(complete, 3,"60 - 74%").
level(complete, 2,"45 - 59%").
level(complete, 1,"20 - 44%").
level(complete, 0,"100%").

location(Loc) :-
        nl, write('Where are the planes to be based? '),
        read(Loc),nl.

process(2, 0) :-            /* Make changes to missions */
        write('Change mission'), nl,nl,
        miscount(Nmiss),
        Nmiss > 0,
        write('There are ',Nmiss),write(' mission(s) in the database.'),nl,nl,
        listmissions,
        ask('Which mission would you like to change (enter number):',MN),
        showMission(MN),
        lisp(Which.attribute),      nl,
        process2(MN,Which).

process(2, 0) :- miscount(Nmiss), Nmiss = 0, nl, write("There are no planned missions!!!"),nl.

process(2, 0) :- nl, write('mission not changed...'), nl.

continue(M) :-
        mission(M, _, _, _, _),
        write('New Mission Status:  Mission ',M),nl,nl,
        showMission(M).

continue(M).

process2(M,none).

process2(M,del) :-          /* Delete this mission */
        return_weapons(M),
        remove_book_keeping(M),
        miscount(X), Y is X - 1,
        replace( miscount(X), miscount(Y) ),
        retract(mission(M,_,_,_,_)).

return_weapons(M) :-
        pull_and_replace_loads(M),!,
        return_weapons( M ).

return_weapons(M):-
        mission( M, _,_,NP,_ ),
        uaplanes( X ),
        Y is X + NP,
        replace( uaplanes( X ), uaplanes( Y ) ),
        asplanes( R ),
        S is R - NP,
        replace( asplanes( R ), asplanes( S ) ).

pull_and_replace_loads(M):-
        sugg_miss( M, Lode, Fusing, Amt, _ ),!,
        weapon( Lode, Old ),
        New is Old + Amt,
        replace( weapon( Lode, Old ), weapon( Lode, New ) ),
        retract( sugg_miss( M, Lode, Fusing, Amt, _ ) ).

pull_and_replace_loads(M):- fail.

remove_book_keeping(M) :-
        mission(M,_,Type,_,_,_),
        removeTypeAttr(M,Type).

process2(M,loc) :-        /* Change location name of this mission */
        ask("Enter new mission location",Nloc),nl,
        mission(M,_,W,X,Y,Z),
        replace(mission(M,_,W,X,Y,Z),mission(M,Nloc,W,X,Y,Z)),
        continue(M).

process2(M,time) :-       /* Change time of this mission */
        ask("Enter new mission start time",Time),nl,
        mission(M,W,X,_,Y,Z),
        replace(mission(M,W,X,_,Y,Z),mission(M,W,X,Time,Y,Z)),
        continue(M).

process2(M,cmp) :-        /* Change desired completion % for this mission */
        mission(M,W,X,Y,Z,_),
        not( on24HrAlert( M, _ )),
        lisp(CP,comppercc),
        return_weapons(M),!,
        miscount(J), K is J - 1,
        replace( miscount(J), miscount(K) ),
        retract(mission(M,W,X,Y,Z,_)),
        resetbk,
        type_inter(M,X,Inter),
        assignPlanes(M,X,Inter,CP,NP),!,
        round( NP, NPlanes ),!,
        implement_mission( M, W, Y, NPlanes, X, Inter, CP ),
        continue(M).

process2(M,cmp) :-
        on24HrAlert(M),
        write('Completion percentage for 24 Hour Alert can only be 100%.').

process2(M,cmp) :- write("Error in changing mission",M),nl.

type_inter(M, areaPrep, none).
type_inter(M, on24HrAlert, none).
type_inter(M, closeAirSupport, none).
type_inter(M, interdiction, pc) :- personnel(M,_,_,_,_,_).
type_inter(M, interdiction, uv) :- uVehicles(M,_,_,_,_,_).
type_inter(M, interdiction, av) :- aVehicles(M,_,_,_,_,_,_).
type_inter(M, interdiction, bc) :- buildings(M,_,_,_,_,_).
type_inter(M, interdiction, rr) :- cuts(M,_,_,_,_,_).
type_inter(M, interdiction, bridge) :- bridge(M,_,_,_,_,_).
type_inter(M, X, Y) :- write('error finding mission type : ',X),nl.

removeTypeAttr(M,interdiction) :- personnel(M,_,_,_,_,_,_,_),interKill(M,pc).
removeTypeAttr(M,interdiction) :- uVehicles(M,_,_,_,_,_,_),interKill(M,uv).
removeTypeAttr(M,interdiction) :- aVehicles(M,_,_,_,_,_,_),interKill(M,av).
removeTypeAttr(M,interdiction) :- buildings(M,_,_,_,_,_),interKill(M,bc).
removeTypeAttr(M,interdiction) :- cuts(M,_,_,_,_,_),interKill(M,rr).
removeTypeAttr(M,interdiction) :- bridge(M,_,_,_,_,_),interKill(M,bridge).

interKill(M,pc) :- retract(personnel(M,_,_,_,_,_,_,_)).
interKill(M,uv) :- retract(uVehicles(M,_,_,_,_,_,_)).
interKill(M,av) :- retract(aVehicles(M,_,_,_,_,_,_)).
interKill(M,bc) :- retract(buildings(M,_,_,_,_,_)).
interKill(M,rr) :- retract(cuts(M,_,_,_,_,_)).
interKill(M,bridge) :- retract(bridge(M,_,_,_,_,_)).
removeTypeAttr(M,areaPrep) :- retract(areaPrep(M,_,_,_,_,_,_,_,_)).
removeTypeAttr(M,closeAirSupport) :- retract(closeAirSupport(M,_,_,_,_,_,_,_,_)).
removeTypeAttr(M,on24HrAlert) :- retract(on24HrAlert(M,_)).


process(3,0) :- lisp(Selection,selectStatus), process3(Selection).

process3(1)  :-            /* Check status of planes */
        uaplanes(X1),smplanes(X2),bdplanes(X3),mfplanes(X4),asplanes(X5),
        nl,nl,nl,
        write('planes unassigned            :',X1),nl,
        write('planes on scheduled maintenance:',X2),nl,
        write('planes with battle damage    :',X3),nl,
        write('planes completly malfunctioning:',X4),nl,
        write('planes assigned to missions   :',X5),nl,
        write('--------------------------------------'),nl,
        Tot is X1+X2+X3+X4+X5,
        write('Total number of planes        :',Tot),nl,process(3,0).

process3(2) :- nl,nl,nl,           /* check weapon status */
        write('  Weapon Status Report'),nl,
        write('---------------------'), nl,
        weap0,weap1,weap2,weap3,weap4,weap5,weap6,weap7,process(3,0).

weap0 :-
        weapon(mm20, Amt20),
        write('20mm:                  '), write(Amt20),
        write(' rounds in supply'), nl.
weap1 :-
        weapon(cb, AmtCB),
        write('Cluster bombs:         '), write(AmtCB),
        write(' rounds in supply'), nl.
weap2 :-
        weapon(hd500, Amt500HD),
        write('500 HD:                     '), write(Amt500HD),
        write(' rounds in supply'), nl.
weap3 :-
        weapon(ld500, Amt500LD),
        write('500 LD:                     '), write(Amt500LD),
        write(' rounds in supply'), nl.
weap4 :-
        weapon(m750, Amt750),
        write('750:                        '), write(Amt750),
        write(' rounds in supply'), nl.
weap5 :-
        weapon(m2000, Amt2000),
        write('2000:                       '), write(Amt2000),
        write(' rounds in supply'), nl.
weap6 :-
        weapon(napalm, AmtNapalm),
        write('Napalm (200 gal):           '), write(AmtNapalm),
        write(' rounds in supply'), nl.
weap7 :-
        weapon(atg,Amt),
        write('Air to ground:              '), write(Amt),
        write(' rounds in supply'), nl.

process3(3) :-
        miscount(Mcount),
        Mcount > 0,
        write('There are '), write(Mcount),
        write(' planned missions: '),nl,listmissions,
        write('Which mission number? '),
        read(M),
        showMission(M),
        process(3,0).

process3(3) :-
        miscount(Mcount),
        Mcount = 0,
        write('No planned missions at this time...'),nl.

process3(3) :- write('Supply an existant mission next time').

showMission(M) :-
        mission(M, Loc, Type, ST, NP, Comp),
        write('Mission type:        '), write(Type), nl,
        write('Mission location:    '), write(Loc), nl,
        write('Mission start time: '), write(ST), nl,
        write('Number of Planes Assigned : '), write(NP), nl, nl,nl,
        suggest( M ),
        level(complete,Comp,X),
        write('Completion level         ',X), nl, nl.

showMission(M) :- write('No record of mission',M), fail.

listmissions :-
```

```
        mission(X,_,Y,_,_,_),
        print_out(X,Y),
        fail.

listmissions :- nl.

print_out( X, Y ) :- !,
        write(X,'          ',Y),nl.

process3(4).        /* quit status check */

process(4, 0) :-            /* make changes to databases */
        write('Change database'), nl,
        write('Enter password: '),
        read(Password),
        password(Password),
        lisp(Selection,menu4),
        process4(Selection).

process(4, 0) :- write('Invalid password').

password(secret).

changePassword :-
        ask('Enter old password',OP),nl,
        password(OP),
        ask('Enter NEW password',NP),nl,
        replace(password(OP),password(NP)).

changePassword :-
        write('Password not changed... error on entry').

process4(0) :-
        changePassword.

process4(1) :-              /* Change plane amounts */
        lisp(Pick,menu41),
        process41(Pick).

process41(ua) :-
        uaplanes(X),
        write('There were ',X),write(' planes unassigned.'),nl,
        ask('Input the number of planes that will now be unassigned:',N),
        replace(uaplanes(_),uaplanes(N)).

process41(sm) :-
        smplanes(X),
        write('There were ',X),write(' planes on scheduled maintenance.'),nl,
        ask('Input the number of planes that will now be on scheduled maintenance:',N),
        replace(smplanes(_),smplanes(N)).

process41(bd) :-
        bdplanes(X),
        write('There were ',X),write(' planes with battle damage.'),nl,
        ask('Input the number of planes that will now have battle damage:',N),
        replace(bdplanes(_),bdplanes(N)).

process41(mf) :-
        mfplanes(X),
        write('There were ',X),write(' planes malfunctioning.'),nl,
        ask('Input the number of planes that will now be malfunctioning:',N),
        replace(mfplanes(_),mfplanes(N)).

process41(none) :-
        lisp(Selection,menu4),
        process4(Selection).

process4(2) :-
        lisp(Sel,loadtype),
        process42(Sel).

addchange(Type,add) :-
        weapon(Type,X),
        ask('Input number of units to add: ',Amt),nl,
        Y is X + Amt,
        replace(weapon(Type,X),weapon(Type,Y)).

addchange(Type,change) :-
        ask('Input number of units: ',Amt),nl,
        replace(weapon(Type,_),weapon(Type,Amt)).

addchange(Type,neither) :- write('No change made.'),nl.

process42(mm20) :-      /* change weapon amounts */
        weapon(mm20,X),
        write('There are ',X),write('rounds of 20mm.'),
        lisp(Choice,addorchange),
        addchange(mm20,Choice).

process42(cb) :-
        weapon(cb,X),
        write('There are ',X),write(' Cluster-Bombs.'),
        lisp(Choice,addorchange),
        addchange(cb,Choice).

process42(hd500) :-
        weapon(hd500,X),
        write('There are ',X),write(' 500 HD'),
        lisp(Choice,addorchange),
        addchange(hd500,Choice).

process42(ld500) :-
        weapon(ld500,X),
        write('There are ',X),write(' 500 LD.'),
        lisp(Choice,addorchange),
        addchange(ld500,Choice).
```

```
/*   Format of file:

                    Plane Assignment Rules
                    ----------------------
                          assn{.60}( Mission_type, MenuChoice,PlanesNeeded).
                          assn{not .60}( Mission_type, MenuChoice.PlanesNeeded) :-
                                         assn{.60}( Mission_type, MenuChoice,Needed),
                                         PlanesNeeded is Needed * Factor.

                                 where   Factor is 3.0 for 90+%
                                         Factor is 1.5 for 75+%
                                         Factor is 0.8 for 45+%
                                         Factor is 0.4 for 20+%
                                                                                  */

 assn{.20}(on24HrAlert,_,_).
 assn{.45}(on24HrAlert,_,_).
 assn{.60}(on24HrAlert,_,_).
 assn{.75}(on24HrAlert,_,_).
 assn{.90}(on24HrAlert,_,_).


assn{.20}(areaprep,M,NP) :- assn{.60}(areaprep,M,N), NP is N * 0.4.
assn{.45}(areaprep,M,NP) :- assn{.60}(areaprep,M,N), NP is N * 0.8.

assn{.60}( areaprep, 1, 4.0 ) :- !.
assn{.60}( areaprep, 2,10.0 ) :- !.
assn{.60}( areaprep, 3,18.0 ) :- !.
assn{.60}( areaprep, 4,30.0 ) :- !.
assn{.60}( areaprep, 5,50.0 ) :- !.

assn{.75}(areaprep,M NP) :- assn{.60}(areaprep,M,N), NP is N * 1.5.
assn{.90}(areaprep,M,NP) :- assn{.60}(areaprep,M,N), NP is N * 3.0.

assn{.20}(pc,M,NP) :- assn{.60}(pc,M,N), NP is N * 0.4.
assn{.45}(pc,M,NP) :- assn{.60}(pc,M,N), NP is N * 0.8.

assn{.60}(pc,squad,1.0):- !.
assn{.60}(pc,platoon,1.0):- !.
assn{.60}(pc,company,3.0):- !.
assn{.60}(pc,battalion,8.0):- !.
assn{.60}(pc,regiment,25.0):- !.
assn{.60}(pc,other,70.0):- !.

assn{.75}(pc,M,NP) :- assn{.60}(pc,M,N), NP is N * 1.5.
assn{.90}(pc,M,NP) :- assn{.60}(pc,M,N), NP is N * 3.0.


assn{.20}(closeAirSupport,M,NP) :- assn{.60}(closeAirSupport,M,N), NP is N * 0.4.
assn{.45}(closeAirSupport,M,NP) :- assn{.60}(closeAirSupport,M,N), NP is N * 0.8.

assn{.60}(closeAirSupport,squad,1.0):- !.
assn{.60}(closeAirSupport,platoon,1.0):- !.
```

```
assn{.60}(closeAirSupport,company,3.0):- !.
assn{.60}(closeAirSupport,battalion,8.0):- !.
assn{.60}(closeAirSupport,regiment,25.0):- !.
assn{.60}(closeAirSupport,other,70.0):- !.

assn{.75}(closeAirSupport,M,NP) :- assn{.60}(closeAirSupport,M,N), NP is N * 1.5.
assn{.90}(closeAirSupport,M,NP) :- assn{.60}(closeAirSupport,M,N), NP is N * 3.0.

assn{.20}(uv,M,NP) :- assn{.60}(uv,M,N), NP is N * 0.4.
assn{.45}(uv,M,NP) :- assn{.60}(uv,M,N), NP is N * 0.8.

assn{.60}(uv,5,24.0):- !.
assn{.60}(uv,4,8.0):- !.
assn{.60}(uv,3,5.0):- !.
assn{.60}(uv,2,4.0):- !.
assn{.60}(uv,1,2.0):- !.

assn{.75}(uv,M,NP) :- assn{.60}(uv,M,N), NP is N * 1.5.
assn{.90}(uv,M,NP) :- assn{.60}(uv,M,N), NP is N * 3.0.


assn{.20}(av,M,NP) :- assn{.60}(av,M,N), NP is N * 0.4.
assn{.45}(av,M,NP) :- assn{.60}(av,M,N), NP is N * 0.8.

assn{.60}(av,0,0.0):- !.
assn{.60}(av,1,3.0):- !.
assn{.60}(av,2,6.0):- !.
assn{.60}(av,3,12.0):- !.
assn{.60}(av,4,20.0):- !.

assn{.75}(av,M,NP) :- assn{.60}(av,M,N), NP is N * 1.5.
assn{.90}(av,M,NP) :- assn{.60}(av,M,N), NP is N * 3.0.


assn{.20}(bc,M,NP) :- assn{.60}(bc,M,N), NP is N * 0.4.
assn{.45}(bc,M,NP) :- assn{.60}(bc,M,N), NP is N * 0.8.

assn{.60}(bc,1,2.0):- !.
assn{.60}(bc,2,4.0):- !.
assn{.60}(bc,3,8.0):- !.
assn{.60}(bc,4,16.0):- !.
assn{.60}(bc,5,24.0):- !.
assn{.60}(bc,6,30.0):- !.

assn{.75}(bc,M,NP) :- assn{.60}(bc,M,N), NP is N * 1.5.
assn{.90}(bc,M,NP) :- assn{.60}(bc,M,N), NP is N * 3.0.


assn{.20}(rr,M,NP) :- assn{.60}(rr,M,N), NP is N * 0.4.
assn{.45}(rr,M,NP) :- assn{.60}(rr,M,N), NP is N * 0.8.

assn{.60}(rr,1,2.0):- !.
assn{.60}(rr,2,2.0):- !.
```

```
assn{.60}(rr,3,4.0):- !.
assn{.60}(rr,4,6.0):- !.

assn{.75}(rr,M,NP) :- assn{.60}(rr,M,N), NP is N * 1.5.
assn{.90}(rr,M,NP) :- assn{.60}(rr,M,N), NP is N * 3.0.


assn{.20}(bridge,M,NP) :- assn{.60}(bridge,M,N), NP is N * 0.4.
assn{.45}(bridge,M,NP) :- assn{.60}(bridge,M,N), NP is N * 0.8.

assn{.60}(bridge,1,2.0):- !.
assn{.60}(bridge,2,2.0):- !.
assn{.60}(bridge,3,4.0):- !.
assn{.60}(bridge,4,6.0):- !.
assn{.60}(bridge,5,8.0):- !.

assn{.75}(bridge,M,NP) :- assn{.60}(bridge,M,N), NP is N * 1.5.
assn{.90}(bridge,M,NP) :- assn{.60}(bridge,M,N), NP is N * 3.0.


assnpc( 5, X,Y,Z ) :- assn{.90}(X,Y,Z).
assnpc( 4, X,Y,Z ) :- assn{.75}(X,Y,Z).
assnpc( 3, X,Y,Z ) :- assn{.60}(X,Y,Z).
assnpc( 2, X,Y,Z ) :- assn{.45}(X,Y,Z).
assnpc( 1, X,Y,Z ) :- assn{.20}(X,Y,Z).

/*
        assigning additional aircraft for various factors:

*/

extra_assn(anti_air,1,0).          /* Anti-aircraft guns */
extra_assn(anti_air,2,1).
extra_assn(anti_air,3,2).
extra_assn(anti_air,4,4).
extra_assn(anti_air,5,7).

extra_assn(samsites,1,0).          /* Surface to Air missile sites */
extra_assn(samsites,2,1.5).
extra_assn(samsites,3,3).
extra_assn(samsites,4,5).
extra_assn(samsites,5,10).

extra_assn(locover,1,1).           /* Low altitude cover */
extra_assn(locover,2,1.125).
extra_assn(locover,3,2.0).
extra_assn(locover,4,2.5).

extra_assn(terrain,open,1).        /* Terrain type */
extra_assn(terrain,hilly,1.3).
extra_assn(terrain,mountains,2).

extra_assn(cover,open,1).          /* Protective cover type */
extra_assn(cover,sb,1.25).
extra_assn(cover,jungle,1.5).
extra_assn(cover,urban,1.8).

extra_assn(avehics,0,0).             /*  number of armoured vehicles */
extra_assn(avehics,1,3).
extra_assn(avehics,2,6).
extra_assn(avehics,3,14).
extra_assn(avehics,4,20).

assign_avs(AV,Nplanes) :- !,extra_assn(avehics,AV,Nplanes).


assign_std_factors(AA,SS,Nplanes) :-  /*  assign values for these typical mission factors */
        !,
        extra_assn(anti_air,AA,A),
        extra_assn(samsites,SS,S),
        Nplanes is A + S.

compute_terr_factors(Terr,Cov,LC,Factor) :- /*  Multiplying factors to number of planes */
        !,
        extra_assn(terrain,Terr,T),
        extra_assn(cover,Cov,C),
        extra_assn(locover,LC,L),
        X is T * C,
        Factor is X * L.

/*    Load assignments :

        loads will be made into a database, since lists are so difficult
        to implement. The tuples added to the database will follow this
        general rule:                                      _

                load_missiontype( Mnum, &possible factors) :-
                                assert( loadpc( Mnum, Load, Fusing, %x ) ),
                                                .
                                                .
                                                .
                                assert( loadpc( Mnum, Load, Fusing, %x ) ).

        where Mnum is a mission number, possible factors include materials,
        roadtypes, defensive positions, cover etc...                           */


/*    cover_loads( Mnum, Perc, CoverLevel )                                    */

cover_loads( M, P, open ) :-
        X is P/2,
      assert( loadpc( M, napalm, impact, X ) ),
      assert( loadpc( M, cb, impact, X ) ).

cover_loads( M, P, sb ) :-
        Y is P/3,
        assert( loadpc( M, napalm, impact, Y ) ),
        assert( loadpc( M, cb, impact, Y ) ),
        assert( loadpc( M, hd500, impact, Y ) ).

cover_loads( M, P, jungle) :-
        Y is P/3,
        assert( loadpc( M, hd500, impact, Y ) ),
        assert( loadpc( M, m750, impact, Y ) ),
        assert( loadpc( M, m2000, impact, Y ) ).

cover_loads( M, P, urban ) :-
        X is P/2,
        assert( loadpc( M, m2000, impact, X ) ),
        assert( loadpc( M, atg, impact, X ) ).


/*    defpos_loads( Mnum, Perc, DPLevel )                                  */

defpos_loads( M, P, none ).

defpos_loads( M, P, trenches ) :-
        X is P/2,
        assert( loadpc( M, hd500, delayed, X ) ),
        assert( loadpc( M, m750, delayed, X ) ).

defpos_loads( M, P, tunnels ) :- assert( loadpc( M, m2000, delayed, P ) ).

defpos_loads( M, P, rcb ) :-
        X is P/2,
        assert( loadpc( M, m2000, delayed, X ) ),
        assert( loadpc( M, m750, delayed, X ) ).

load_24hr( M ) :-
        assert( loadpc( M, napalm, impact, 0.40 ) ),
        assert( loadpc( M, hd500, impact, 0.40 ) ),
        assert( loadpc( M, atg, proximity, 0.20 ) ).

load_closeAir( M, P ) :-
        A is .45*P, B is .10*P,
        assert( loadpc( M, napalm, impact, A ) ),
        assert( loadpc( M, hd500, impact, A ) ),
        assert( loadpc( M, cb, proximity, B ) ).

load_ap( M, P ) :-
        Y is P/3,
        assert( loadpc( M, cb, proximity, Y ) ),
        assert( loadpc( M, m750, delayed, Y ) ),
        assert( loadpc( M, m2000, delayed, Y ) ).

/* interdiction loads (std) */

load_pers( M, P ) :-
        Z is P/4,
        assert( loadpc( M, hd500, impact, Z ) ),
        assert( loadpc( M, m750,  impact, Z ) ),
        assert( loadpc( M, m2000, impact, Z ) ),
        assert( loadpc( M, cb, proximity, Z ) ).

load_uv( M, P ) :-
        X is P/2,
        assert( loadpc( M, atg, impact, X ) ),
        assert( loadpc( M, napalm,  impact, X ) ).

load_av( M, P ) :-
        X is P/2,
        assert( loadpc( M, atg, impact, X ) ),
        assert( loadpc( M, ld500, impact, X ) ).

load_cuts( M, P, 1 ) :-
        X is P/2,
        assert( loadpc( M, m750, delayed, X ) ),
        assert( loadpc( M, m2000, delayed, X ) ).

load_cuts( M, P, 2 ) :-
        X is P/2,
        assert( loadpc( M, m750, impact, X ) ),
        assert( loadpc( M, atg, impact, X ) ).

load_cuts( M, P, 3 ) :-
        X is P/2,
        assert( loadpc( M, m750, impact, X ) ),
        assert( loadpc( M, atg, impact, X ) ).

load_buildmats( M, P, wood ) :- assert( loadpc( M, napalm, impact, P ) ).
load_buildmats( M, P, rhut ) :-
        X is P/2,
        assert( loadpc( M, hd500, proximity, X ) ),
assert( loadpc( M, m750,  proximity, X ) ).
load_buildmats( M, P, brick) :-
        X is P/2,
        assert( loadpc( M, m750,  proximity, X ) ),
        assert( loadpc( M, m2000, proximity, X ) ).
load_buildmats( M, P, rc    ) :-
        X is P/2,
        assert( loadpc( M, m750,  delayed, X ) ),
        assert( loadpc( M, m2000, delayed, X ) ).

load_bridgemats( M, P, wood) :-
        Y is P/3,
        assert( loadpc( M, hd500, impact, Y ) ),
        assert( loadpc( M, m750,  impact, Y ) ),
        assert( loadpc( M, atg,    impact,    Y ) ).

load_bridgemats( M, P, concrete) :-
        X is P/2, Z is P/4,
        assert( loadpc( M, m750,  delay, Z ) ),
        assert( loadpc( M, m2000, delay, Z ) ),
        assert( loadpc( M, atg,   impact,  X ) ).

load_bridgemats( M, P, steel) :-
        X is P/2,  Z is P/4,
        assert( loadpc( M, m750,  delay, Z ) ),
        assert( loadpc( M, m2000, delay, Z ) ),
        assert( loadpc( M, atg,   impact,  X ) ).
```