CONSTRAINED SCENARIO GENERATION SYSTEM DOCUMENTATION: USER'S MANUAL AND PROGRAMMER'S GUIDE

Rajendra S. Wall

COINS Technical Report 83-02
July, 1982

Department of Computer and Information Science
University of Massachusetts, Amherst
Amherst, Massachusetts 01003

Research supported by National Science Foundation Grant IST-8017343.

1.0 Introduction

This document is the maintenance manual for the Conflict Simulation Gaming Constrained Scenario Generation System (CSGCSG or CSG). It is intended for the Systems Programmer familiar with Conflict Simulation Games, the Constrained Scenario Generation paradigm and who is an expert LISP user. It covers numerous system details intended to allow one to maintain the CSG system as well as interface to a new domain.

Additional references:

CSG User's Manual

CSG LISP Code Listing

CSG Cross Reference Listing

FMS User's Manual

CEG User's Manual

CEG Programmer's Guide

CLISP Manual

Introduction to Conflict Simulation Games

VAX Command Language Manual

1.1 Accessing the System

The system lives in the directory users1:[cegrw.thesis.project] (all files referenced in this document will be assumed to be in this directory unless otherwise specified) in the files: exec, scenario, scensubs, interface, graphics, slots, and data.

There are procedure files available to aid accessing the system:

- users1:[cegrw]proc csg - run the compiled version of the system.
- users1:[cegrw]proc csgsup - run the compiled version and invoke the supervisor.
- csgsys - loads the system level files but not the parts of the CEG system needed to actually run CSG.
- cegcsg - loads everything needed to run the system.

- csgsup - loads everything and invokes the CSG supervisor (normal user entry point).

- route - prints all functions (but not the data and slot files) to the line printer. This is best run as a submit job to the quick batch queue as it takes a long time to do.

- routed - prints all functions, slots and data to the line printer. This should be done as a submit job to the quick batch queue to run late at night because not only does it take a long time to do, but the listing takes a long time to print out.

The documentation for the system is in users1:[cegrw.document].

2.0 System Outline

Once you have loaded the system you can use it as is to maintain the data base (building new frames, modifying old ones, etc), generate scenarios, or display scenarios. You can also, of course, modify the functions that make up the system.

2.1 Entry Points

If you are going to run the system you can do so through the following entry points:

- supervisor - invokes the user supervisor (see CSG User's Manual) from which all system functions can be executed.

- copy-symbol, create-symbol, delete-symbol, modify-symbol, route-symbol, and show-symbol will all work on the frames of the system as in the FMS and CSG User's manuals.

- display-symbol - invokes the graphics package to display items on the GMR-27 (the Grinnell).

- generate-scenario - invokes the scenario generation mechanism on the supplied goal and situation. Monitors CEG search.

From the supervisor one can access the other entry points merely by entering the first word of the name (e. g. copy for copy-symbol).

2.2 Block Diagram

This section outlines the workings of the display-symbol routine when displaying a scenario and the generate-scenario routine.

2.3 Generating a Scenario

If the generate command or the generate-scenario function is given the following processing is performed:

1. The scenario generation routine is entered [(build-scenario task situation name)].

2. A set of problems which will be used to interrogate the data base through CEG is created [(build-constraints goal situation)] by analysis of the goal and situation [(situation-analysis goal situation)], [(tactical-analysis behavior fr-str en-str)], [(enemy-behavioral-tactics en-str fr-str)].

3. Each problem is presented to the CEG system which searches the data base for similar experiences [(ceg-search cl-name)] and the results are collected.

4. Each remembered experience is modified to fit the current goal/situation exactly [(instantiate-scenario scen fc-list task situation *fr-str *en-str)].

2.3.1 Example Call History

The following is a partial call history of an invocation of (generate-scenario).

I. build-scenario

  A. build-constraints

    i. situation-analysis

    ii. enemy-behavioral-tactics

    iii. tactical-analysis

  B. bind-experience-base-to-example

  C. choose-concept-space

  D. set-up-search

  E. ceg-search

    i. ceg-exec-front

      a. enemy-position-judge

(i) compare-position

(a) comp-pos-en

(b) comp-pos-fr

(c) test-strength

b. goal-judge

(i) goal-shape

(ii) goal-force

F. instantiate-scenario

2.3.2 Displaying a Scenario

Assuming the user is at the Grinnell, the system will perform the following:

1. Break up the scenario object into its component parts (goal, situation, segment list, failure list and problem list).
2. Run the scenario synopsis routine [(scenario-synopsis s-list)].
3. Put the mapsheet in the background planes [(draw-map $$map)].
4. Enter the scenario display command loop.

When told to display a segment the system will:

1. Display information about the segment in the upper left corner of the overlay planes [(display-status seg)].
2. Calculate the screen coordinates of the force and goal locations as well as the front unit direction vectors.
3. Display the course of action ply by ply [(display-coa set)] asking the user to hit the enter key on the cursor control box after each ply is displayed [(draw-coa wcoa frpos enpos goal frfr enfr gfr frstr enstr)].

4. Display the moves and combats described in the course of action [(draw-move fn pts)], [(draw-combat frcen encen)], [(draw-combat-result dscr frcen encen frfr enfr frstr enstr frinv eninv)].

5. Keep track of the position and status of the units and goals involved.

2.3.3 Example Call History

The following is a partial call history of an invocation of (display-symbol) on a scenario.

I. display-scenario

  A. scenario-synopsis

  B. draw-map

  C. display-segment

    i display-status

    ii display-coa

      a. display-status

      b. draw-goal

      c. draw-force

      d. draw-coa

        (i) draw-move

        (ii) move-result

        (iii) draw-force

        (iv) draw-combat

        (v) draw-combat-result

        (vi) combat-result

II. grflush

3.0 Function Detail

This section discusses the major level functions of each subsystem (Scenario Generation, CEG Interface, and Graphics) in detail: what the functions and variables are, what the major data structures look like, what the system is supposed to be doing and why, and hints on changing and debugging.

3.1 The Scenario Generation Subsystem

This subsystem consists of the functions generate-scenario, build-scenario, build-constraints, situation-analysis, enemy-behavioral-tactics, tactical-analysis, and instantiate-scenario.

3.1.1 Generate-scenario

This routine is the entry point to the scenario generation subsystem. It gets the name the scenario is to have, the goal to be examined and the situation in which it is to be examined. It calls build-scenario and inserts the built scenario in the symbol table and the data file if so desired. The only data structures involved are the frames that make up the goal and the situation (lisp arrays). The scenario name is bound to the list of the goal, the situation, the list of instantiated experiences, the list of constraints of problems that were not resolved, and the constraints of problems that were resolved; thus what is saved in the data file is the makeset of the unroll of the cddr of the value of the name (skipping the goal and situation, which should already be saved).

3.1.2 Build-scenario

This routine controls the top level scenario generation processing. This consists of first building the constraints by analysing the goal and situation. A series of problems is created which will then be presented to the data base through the CEG system. Theoretically, solving these problems is sufficient to recover the complete range of the future posibilities. Whether solution to all of these problems is necessary to cover the future is not known.

When all the problems have been tried, the experiences that were recovered, and the problems they are examples of solutions for, are given to the instantiate-scenario routine. The scenario name is then bound to the list of the goal, the situation, the list of experiences, the list of unsolved problems and the list of solved problems.

The only domain specific section of this routine is at the beginning of the repeat search through the problems, when the list of constraints is divided into the those that are simple binary contraints, a la a relational data base query (e. g. tactic:

frontal-attack), and those that will require the full CEG judgement mechanisms (the goal and positional analysis constraints).

The list of problems returned by the constraint generation routine (build-constraints) is in the form of a list of names, each bound to a list of constraint names.

3.1.3 Build-constraints

This routine builds a list of problem names each bound to a list of constraint names. Each constraint is an expression that details a value to be matched to a similar value of an experience in the data base. The constraints are the semantic description of the forces involved in the situation, the goal expression to be achieved (from the specifics slot of the goal frame), the enemy behavior mode, the enemy tactic given that behavior, the random effects mode and the friendly tactic.

The semantic description of the involved forces is generated by the situation-analysis routine. It returns two lists of force descriptors, one for the friendly and one for the enemy forces. Each descriptor is a triple: <type name strength>, where type is the semantic tag, name is the name of the unit/group, and strength is the current strength of the unit/group.

The goal constraint is taken directly from the goal frame.

The enemy behavior and enemy tactic constraints are generated by the routine enemy-behavioral-tactics, which analyses the situation and decides what the possible enemy behavior modes are and what tactics the enemycould use in this situation.

The random effects mode constraint is one of {helpful, neutral or detrimental}.

The friendly tactic constraint comes from the tactical-analysis routine. This routine generates all possible ways of achieving the goal given the forces available and the enemy forces in opposition.

The list of problems then consists of the situation analysis and goal constraints, which are the same for each problem, and all combinations of enemy behaviors and tactics, random effect modes and friendly tactics.

This routine is domain independent as long as you can fit your constraints into the above categories (in which case you may need new versions of the situation-analysis, enemy-behavioral-tactics, and tactical-analysis routines), otherwise you would have to create your own.

3.1.4 Situation-analysis

This routine builds the list of the lists of enemy and friendly force descriptors. This is done by calculating where on the map each force and goal is, and then checking what positional relationship there is among them. The enemy forces are analyzed by describe-enemy-position and the friendly forces by describe-friendly-position.

3.1.4.1 Describe-enemy-position - For each enemy force there are four possible descriptor types: on, between, near or outside. This routine tests an enemy force by first checking if it is on any of the locations of the goals to be achieved. If one of the goals is to destroy this enemy force then obviously it is on a goal location. The descriptor is then on or the list (on n) if there is more than one goal and the force is on goal n.

If the force is not on a goal, then it is tested to see if it is between a friendly force and a goal, where between is defined to mean within a circle drawn through the goal and friendly force with an origin at the midpoint between the goal and the friendly force. If so, then the descriptor type between is returned or the list (between gn fn) if there is more than one goal (gn) or friendly force (fn) involved.

The next test is to see if the enemy force is within 7 hexes of a goal. If so, then the descriptor type is near or the list (near n) if there is more than one goal and the force is near goal n.

If the force fails all of the above tests then it is described as outside.

3.1.4.2 Describe-friendly-position - For each friendly force there are four possible descriptor types: on, next, near or outside. This routine first checks to see if the unit is on a goal location. If the force is to be protected then it is obviously on a goal. The descriptor type is then on or the list (on n) if there is more than one goal and the force is on goal n.

If the force is not on a goal, then it is tested to see if it is adjacent to an enemy force, since by the rules such a force is commited to immediate combat. If so, the descriptor type is next or the list (next en) if there is more than one enemy force and the friendly force is next to enemy force en.

The next test is to see if the friendly force is within 7 hexes of a goal. If so, the descriptor type is near or the list (near n) if there is more than one goal and the friendly force is nearest goal n.

Finally, if the friendly force is not on, next or near it is automatically classified as outside.

3.1.5 Enemy-behavioral-tactics

This routine examines each possible enemy behavior mode and within each calls the routine tactical-analysis to generate the possible tactics the enemy could perform given the forces involved.

It returns a list of the enemy behavior/enemy tactic constraint pairs.

3.1.6 Tactical-analysis

This routine takes a behavior mode a list of friendly forces and a list of enemy forces (who is enemy and who is friendly is left to the calling routine) and returns a list of all possible tactics that the friendly forces could perform.

If the behavior mode is aggressive then the possible tactics are determined by the number of friendly and enemy forces involved. If there are no enemy forces, then the only tactic is column-advance. If there are no friendly forces then the only tactic is sit. Otherwise, the tactic frontal-attack must be considered. If there is more than one friendly force, then encirclement is also considered. Finally, if there is more than one enemy force involved the tactic flank-attack is considered.

If the behavior mode is passive then the possible tactics are also determined by the number of friendly and enemy forces involved. If there are no enemy forces, then again the tactic column-advance is the only one suggested. If there are no friendly forces then sit is the only tactic. Otherwise, the tactic fluid-defense is suggested. Finally, if there is more than one friendly force the tactic line-defense is also suggested.

3.1.7 Instantiate-scenario

This routine calls recursively instant-scen and inst-scen which do the following work to modify remembered experiences to fit the current situation exactly (instantiation).

A copy of the experience to be instantiated is made. In the copy, the values of the slots tactic, random-effects, enemy-tactic and enemy-bahavior are replaced with the values specified by the constraint. The reason there may be a difference is if the experience is a model experience in which case it may have more than one value for the given slot.

The value in the name slot is replaced with the name of the remembered experience. The friendly and enemy strength slots are replaced with the descriptor lists from situation-analysis. The friendly and enemy names are replaced with those from the situation frame as are the friendly and enemy location slots.

The goal slot is replaced with the value from the specifics slot of the goal frame, and the destination slot with the goal-locations of the goal expression.

Finally, the coa (course of action) slot in the remembered experience is examined. If the first element of the value is prog then the coa slot in the copy is set to the eval of the remembered experience coa. Otherwise, it is left as it was remembered.

3.2 The CEG Interface Subsystem

This subsystem consists of the functions bind-experience-base-to-example, choose-concept-space, set-up-search, ceg-search, ceg-exec-front, enemy-position-judge, compare-position, comp-pos-en, comp-pos-fr, test-strength, goal-judge, goal-shape and goal-force. It also uses the Constrained Example Generation system stored in users1:[ceg.newsys] (documented elsewhere).

3.2.1 Bind-experience-base-to-example

The normal form of the experiences is as a simple array. The Constrained Example Generation system, however, expects the examples in the database it is searching to be arrays with one of the slots being the value of the example. This array of an array is too much to carry around and store all the time, so we defer actually building the example database that CEG will use until as actual generation is in progress, and then, only bind those experiences that have been chosen by choose concept space (See).

Choose-concept-space returns the list of potentially applicable experiences. Bind-experience-base-to-example takes each member of this list and creates a dummy shell array that looks like the array expected by CEG as an example - the slots have the correct values. It binds each of those arrays to an atom that is the name of the experience with "x:" concatenated on the front.

3.2.2 Choose-concept-space

There are currently 6 constraints used to select experiences from the database. Two of these, goal and enemy-position, are sent to CEG which does a complete analysis and judgement. The other four, tactic, enemy-behavior, enemy-tactic and random-effects are not semantic and are used as simple atomic labels, e. g. tactic = frontal-attack, or

random-effects = detrimental.

Thus a structure has been created that facilitates the relational database nature of these constraints. (The structure was created using create-concept-space, which takes a list of experience names and adds them and their attributes to the current structure.) The function choose-concept-space takes a list of attribute-name/value pairs and uses them to select the list of potentially applicable experiences.

The structure $$concept-space is an attribute list of attribute lists of ordered sets. The top level attribute list is the list of constraints (random-effects, tactic, etc), each with a value that is in turn an attribute list of the possible values that that constraint can have (e. g., for random-effects, helpful, neutral and detrimental). Finally, for the value of each of the names of these low level attributes we have the ordered set of the names of all the experiences that have that value of that constraint (e. g. the list of all experiences that show random-effects = detrimental).

The routine choose-concept-space then selects each correct attriubte value experience list as specified by its given list of pairs and does an ordered set intersection on them to get the list of experiences that have all the specificed values of the constraints. A working version of $$example-list (the list used by CEG to see which example should be examined next) is then created.

This pre-CEG selection process speeds up the normal CEG search immesurably.

3.2.3 Set-up-search

This routine merely sets two of the Constrained Example Generation System policy values, the ground level policy and the modification threshold. Two parameters are passed to it, the policy and the threshold to set. Normally, the policy is a single epistemelogical type: su, ref, or model, as defined by CEG.

3.2.4 Ceg-search

This routine calls the constrained example generation repeatedly until all possible relevant experiences have been retrieved. It does this by manipulation of the constrained example generation policy parameters.

3.2.5 Ceg-exec-front

This routine, part of the constrained example generation system itself, is the entry point through which the constrained scenario generation system calls constrained example generation.

3.2.6 Enemy-position-judge

This routine is called by the constrained example generation system to determine if a potential experience is relevant. It calls compare-position which does the actual comparison of force descriptors.

3.2.6.1 Compare-position -

This routine does first a simple count to see if the current situation and the retrieved experience have the same number of involved forces. It then in turn calls comp-pos-en and comp-pos-fr which do the semantic comparisons.

3.2.7 Goal-judge

This routine does the comparison between the problem goal statement and the experience problem statement. It calls goal-shape, which tests the AND/OR keyword structure of the two goal statements, and goal-force, which tests the forces involved to see which forces have been assigned to the same task.

3.3 The Graphics Subsystem

The main routines of the Graphics subsystem are display-scenario, scenario-synopsis, display-segment, display-coa and draw-coa.

3.3.1 Display-scenario

This is the top level entry point to the graphical display of the assembled scenario set. It checks to see if the GMR-27 is allocated and displays the game board if not already present.

3.3.2 Scenario-synopsis

This routine shows the user at the terminal the ways in which the scenario segments can be organized.

4.0 The Data Base

The data base is in two files: 'slots' and 'data'. 'Slots' consists of all the indicies, etc needed by the array access functions (from record-schema). 'Data' is the actual knowledge base. It contains all the goals, situations, groups and units that have been defined, all the experiences that have been "remembered", and all the scenarios that have been generated. These items should only be changed thru the routines in the file 'exec'.

4.1 The Frames

The reader is refered to the Conflict Simulation Game Constrained Scenario Generation System User's Manual for a discussion of the frames, their slots and values.

4.2 Adding To The Data Base

If you are adding anything but experiences, there is nothing special you have to do. If you are adding experiences, however, there is other information that must be provided to the system, and there are other options open to the slot values.

4.2.1 Getting The System To Recognize Your New Experience

Experiences are accessed by first checking the $$concept-space attribute list (see above). Thus only the experiences which have been entered on the $$concept-space list will be recognized.

Once you have built some new frames, make a list of their names and supply it as an argument to create-concept-space. You must then replace the value of $$concept-space in 'data'.

4.2.2 Special Slot Forms For Model Experiences.

The following forms can be used as the slot values of model experiences so that a single experience can serve many constraints. In each case crete-concept-space will note the experience under all the possibilities, and the judge routines will check all variations.

Tactic: a list of tactics this experience covers. E. g., '(frontal-attack encirclement).

Random-effects: a list of the random effects this experience covers.

Friendly-strengths: a list of the choices of force descriptor lists that are deemed similar enough to be described in this one experience, headed by the atom 'or'. E. g., '(or(((near 2)(force 1) 11)((near 1)(force 2)11)) (((near 2)(force 1)11)(outside(force 2)11))). The position judge will try to match the situation to each choice of descriptor set.

Enemy-strengths: similar to friendly-strengths.

Enemy-behavior: a list of depicted behaviors.

Enemy-tactic: a list of depicted tactics.

Coa: a prog expression that returns a normal course of action list. If you execute the function bind-locals--in-model-expansion it will assign to the atoms tac, etac, ebeh and rand the index of the tactic, enemy-tactic, enemy-bahavior and random-effect that this instantiation is to demonstrate from the list of that particular slot. E. g., if you had the list '(helpful neutral) in the slot random-effects, and this expansion was for the random-effect neutral then bind-locals-in-model-expansion would assign 2 to rand.

5.0 Using The CSG System On Your Domain

When you use the Constrained Scenario Generation System on your domain you will need to supply the following system replacement routines in addition to those to handle your routine otherwise (such as Constrained Example Generation judgement routines, display routines, etc): build-constraints, build-scenario (only the selection of which constraints are relational and which require judgement), create-concept-space, display-scenario, display-segment, inst-scen, and the data items $$data-file and $$slot-file.

You should examine the files 'errand' and 'errdata' which are an example of using the Constrained Scenario Generation System for a different domain.

You'll probably use copy-symbol to build your database. In addition, there are some undocumented database build help functions in the file 'temp' such as gen-names and gx.

6.0 Bugs, Errors And Problems

This section is an unordered collection of caveats, hints, kludges, etc the should be known by the system manager.

6.1 The System Didn't Find My New Experience

Did you re-run create-concept-space and replace $$concept-space? Did you save the experience? Check the list of solved problems (the fifth item in the list assigned to the scenario name) and the list of unsolved problems (the fourth item). These names are bound to the constraints the system was trying to find. Look at the slot values for the constraints in your experience. There is probably a discrepancy somewhere.

6.2 Trying To Debug On The GMR-27.

Often you won't get exclusive use of the Grinnell when you're trying to debug a new experience, tactic set, course of action, etc. There is a special switch called $force-draw, which if set to 't' will always redraw the map when display-scenario is entered. Be sure and make sure no one else has something important on the screen before you write to it.

If you don't need the Grinnell, enter the supervisor and say you're not at the Grinnell. This will set $force-nodraw to 't' and redefine gr_crsr_wt to beep the terminal instead of waiting for a cursor press.

6.3 Warning: Making Changes To The Database.

Beware when making changes to the database to enter (dcl) and purge/keep2 files (in particular 'data') otherwise you'll break your disk quota. It only takes 5 updates or changes of database frames to blow things up.

6.4 Compiled Constrained Scenario Generation System

The Constrained Scenario Generation System is compiled on top of the Constrained Example Generation system, so any change in Constrained Example Generation means The Constrained Scenario Generation System will also have to be recompiled. If you don't know how to do recompiles, look at the csgcomp.* and csglink.* files for hints on how to do compiles.

INDEX

1.0 Introduction

This is the User's Manual for the Conflict Simulation Gaming Constrained Scenario Generation (CSGCSG or CSG) System. It is intended for the user familiar with Conflict Simulation Games, LISP, the Constrained Scenario Generation mechanism, and the Constrained Example Generation paradigm [RISS82]. It is not intended as a Maintenance Manual.

Additional References:

- Frame Management System (FMS) Manual,
- CSG Programmer's Guide,
- Introduction to Conflict Simulation Gaming,
- CLISP Manual,
- VAX Command Language Manual.

2.0 Getting Started

To run the system thru the top level supervisor enter the VAX command:
$ @users1:[cegrw]proc csg
And then when you get the CSG: prompt type
(supervisor)
This will run the system and use the default experience data base of users1:[cegrw.thesis.project]data that contains the experiences to handle the problems listed in section 5.0 Data Base Summary, below.

3.0 The Supervisor

Through the CSG Supervisor you can create, copy, modify, delete, display, show and route symbols. The "symbols" are the names of defined frames: goals, situations, units, groups, tasks, and experiences.

Additionally you can load data files and generate and display scenarios. You can enter the command "stop" to exit the supervisor and remain in LISP, or "exit" to return to the VAX EXEC. You can also clear the Grinnell overlay.

The supervisor begins by first asking if you at "at the Grinnell", since most of the power of the system is in its ability to display information. If you are, then the map will be displayed. This should aid in the creation, modification and examination of various objects. If you are

not at the Grinnell, you can still run the system, but nothing will be done on the screen, and various Grinnell routines are rebound to avoid complications.

The rest of the processing consists of the supervisor repeatedly asking you to enter a keyword. The system then processes that keyword plus any additional information needed. Most of these keywords perform calls to the appropriate FMS routine. The major exception is the keyword "generate", which will generate a scenario from a goal and situation. It is discussed in more detail below. All the commands and their meanings are discussed in section 3.2 Command Summary.

3.1 Generating A Scenario.

To generate a scenario enter generate in response to the keyword prompt. You will then be asked for the name you wish this scenario to have, the name of the goal frame this scenario is to pursue, and the name of the situation frame in which the examination is to take place. The goal and situation must either be already defined in the data file, or you must create them before you enter the generate command.

You can use the commands show and route to examine the frames that are already present in the data base, and copy, create or modify to make new frames. The meaning of the various slots in the frames is discussed in section 4.0 Frame Descriptions, below.

The system will then search the experience base for applicable memories, modify them to fit the current situation, and collect them into a scenairo. During this process the progress of the search being conducted by the manipulation of the CEG (Constrained Example Generation) system. This will be a series of ground level probes (the phrase "GROUND LEVEL SEARCH" will be displayed at the terminal) followed by either the original, premodification remembered experience or an explanation of what the search failed to find. If a memory was recovered, the data base is repeatedly probed to see if more information is available, until there is no more.

If a memory of the desired form is not in the data base, the generation continues with a probe for the next type of desired memory and so on.

After all the probes are completed the user is asked if the generated scenario should be saved on disk. This is a very time consuming process (10 - 20 minutes depending on the size of the scenario) and should be done when there is plenty of time available. Even if the scenario is not

saved, it can still be displayed on the Grinnell. Once the scenario is built you can use the display command to see the result on the Grinnell.

When you enter display to see the scenario you will first enter the scenario synopsis routine. This routine will categorize the relevant experiences recovered from the data base by tactic. It will then asks for a category to further subdivide the experiences. This can be any of the keywords {enemy-behavior, enemy-tactic, random-effects or tactic} or the meta-keywords {reset or stop}. When you enter one of the normal classification keywords you will be shown how the experiences are distributed over that category within the previous categories. Entering the keyword 'reset' will move back to the top level of classification and ask for a new top level category. The keyword 'stop' will leave the synopsis routine and enter the actual display routine.

The display routine also has a command/keyword loop. The commands and their meanings are:

Command	Meaning
np	next projection within segment
pp	previous projection within segment
ns	first projection of next segment (first command to enter)
ps	first projection of previous segment
fscan	search forward for first segment of context with specified attribute and value
rscan	search backwards for first segment of context with specified attribute and value
list	list of commands
help	list of commands and meanings
stop	end display

The commands nc, pc, ns, ps, fscan, and rscan all select a segment to be displayed. The particulars of this segment (the name of the experience remembered, the goal of this segment, the tactic exemplified, the enemy behavior, the enemy tactic and the class of random effects considered) are printed at the terminal. The user is then asked whether or not the segment should actually be displayed {y or n}.

If 'y' is entered then the segment is displayed, ply by ply with the user asked to press the enter key on the cursor control box at the end of each ply (the terminal will beep). At the end of each player turn if the home key followed by the enter key is pressed the segment display will be aborted.

If 'n' is entered the keyword/command loop described above is reentered.

3.2 Command Summary

The following is a list of the commands available through the supervisor, the additional parameters they will ask for, and what they will do.

3.2.1 Command Detail

Clear - Clears the Grinnell overlay planes. Used between uses of the display command to allow clean viewing of items.
No Additional Parameters.

Copy - Makes a copy of an existing frame and allows modification of the slots of the new version. See the FMS manual for complete details of how frame modification works.
Additional Parameters:

The type of frame being copied (e. g. goal, situation).

The name of the new frame.

The name of the old frame being copied.

Any modifications to the new frame.

Create - Makes a new frame. See the FMS manual for complete documentation of creation of frames.
Additional Parameters:

The type of frame to create.

The name you wish this frame to have.

The values of the slots of the frame.

Delete - Removes a frame definition.
Additional Parameters:

The type of frame to be deleted.

The name of the frame to be deleted.

Display - Show the symbol on the Grinnell, if posible. Some items can not be shown graphically (e. g. model experiences). Display is different from show or route in that rather than just showing the slots and values, the meaning of the item in terms of the map display is shown. E. g., displaying a situation will actually draw on the screen the units on each side in their specified locations.
Additional Parameters:

The type of frame whose information is to be displayed.

The name of the frame to display.

Exit - Leave LISP and return to VAX Executive monitor.
No Additional Parameters.

Generate - Generate a scenario from a goal and a situation.
Additional Parameters:

The name of the scenario to generate.

The goal whose completion to examine.

The situation in which to examine goal completion.

Help - Gives brief help information.
No Additional Parameters.

List - Prints list of commands.
No Additional Parameters.

Load - Load alternative data file. The default data file specification is then set to this file (for all later creates, copies, etc.).
Additional Parameters:

Full file name including disk and directory as a string.

Modify - Modify frame by changing values in slots. See FMS manual for full documentation on how to modify frames.
Additional Parameters:

Type of frame to modify.

Name of frame to modify.

Route - Pretty print all frames of a given type (or all frames if so indicated) to the line printer. See the FMS manual for a complete discussion of route.
Additional Parameters:

The type of frame to route.

Show - Print all names of given frame type, entire symbol table, or pretty print a single frame to the terminal. See FMS manual for a complete discussion of show.
Additional Parameters:

The type of frame to show.

[Optional] The name of the frame to show.

4.0 Frame Descriptions

This section reviews the available conflict simlation gaming domain frames, what their slots are, their default and expected values.

4.1 Frame Detail

experience - This frame is the memory in the data base. Novice users should not try to manipulate the experience frames in the primary data file either by create, modify or delete. Additionally, it is possible to get into trouble if you generate a scenario, manipulate the data base, and then try to build a new scenario. Manipulation of experience frames should only be done by the System Manager or a System Programmer building their own data base.

name - a unique name for this frame. (Default: current date and time)

type - epistemological class - su, ref, model, ce. (Default: su)

tactic - the friendly tactic that this is and example of. (Default: sit)

random-effects - which random effects (helpful, neutral, detrimental) this is an example of. (Default: neutral)

friendly-units - the number of friendly units/groups involved in the remembered experience. (Default: 1)

friendly-names - the names of the units/groups involved. (Default: nil)

friendly-strengths - a list of the situation analysis descriptors for each friendly force involved. Form of descriptor: <type name strength> where type is one of (on, near, next or outside) as defined by the situation analysis function; name is the name of the unit/group frame being described; and strength is the force strength of the unit/group. (Default: nil)

disposition - the tactical arrangement of the friendly forces. (Default: col, for column)

location - the locations (hex numbers) of the friendly forces. (Default: nil)

terrain - the type of terrain in which the experience takes place. (Default: clear)

enemy-units - the number of enemy units/groups involved. (Default: nil)

enemy-names - the names of the enemy unit/group frames involved. (Default: nil)

enemy-strengths - a list of force descriptors from the situation analysis for each enemy force involved. Form: <type name strength> where type is one of (on, between, near or outside); name is the name of the unit/group frame; and strength is the strength of the force. (Default: nil)

enemy-tactic - the tactic the enemy is using. (Default: sit)

enemy-behavior - the mode of behavior the enemy forces are assumed to be in. (Default: neutral)

enemy-disposition - the tactical arrangement of enemy forces. (Default: line)

enemy-location - the hex number locations of the enemy forces. (Default: nil)

distance - the number of turns of unrestricted movement it would take for the closest friendly unit/group to reach one the goal locations. (Default: nil)

destination - the hex number location of the goals at the beginning of the course of action in the experience. (Default: nil)

engagements - a list of pairs of force numbers that engaged in combat during this experience. E. g. if friendly force 1 fought enemy force 2 then the pair (1 2) would be in the list. (Default: nil)

goal - the list of goal descriptors that were to be achieved in this experience (similar to the goal frame's specifics slot). (Default: nil)

coa - the course of action of the remembered experience as a list of ply descriptors. Each ply descriptor is a list of movement and combat descriptors depicting what happened. (Default: sit)

Example:

```
experience frame: exp-e1-front-goal:0026
name:  " 6-JAN-198200:22:38.12"
type:  su
tactic:  frontal-attack
random-effects:  helpful
friendly-units:  4
friendly-names:  ((gregg davidson) (mcnair robertson))
friendly-strengths:  ((next (gregg davidson) 7)
                      (next (mcnair robertson) 7))
disposition:  line
location:  (("1909" "1909") ("1910" "1910"))
terrain:  (clear)
enemy-units:  1
enemy-names:  ((union-8))
enemy-strengths:  (((between 1 2) (union-8) 5))
enemy-tactic:  frontal-attack
enemy-behavior:  aggressive
enemy-disposition:  line
enemy-location:  (("1809"))
distance:  1
destination:  ("1508")
engagements:  (((1 2) 1))
goal:  (take ("1508") ((force 1) (force 2)))
coa:  (((combat (1 2) 1 (enemy-retreat aac)))
       ((combat 1 1 (friendly-retreat aac)))
       ((move 1 (to (right-flank -1)))
        (move 2 (to (left-flank -1)))
       (combat (1 2)
               1
               (enemy-eliminated aac)))
       nil
       ((move 1 (to (on goal)))))
```

goal - This frame is used to describe goals to be achieved.

name - a unique name for this frame. (Default: current date and time)

type - the type of goal. One of three choices: position, for territorial goals; unit, for unit/group destruction/protection goals; and mixed, for some combination. (Default: position)

value - the numeric value of this goal. (Default: 5)

time - how far in the future in number of turns before this goal is to be achieved. (Default: 0)

specifics - a list of goal descriptors (with optional connectives) specifying what is to be achieved. The format of each descriptor is (connective <desc1> <desc2> ...) where the connectives are either of (and or). Each desc is <verb object indirect-object> where verb is one of (take, hold, destroy, or protect); object is either the location to be taken or held, or the unit/group to be destroyed or protected; and indirect-object is the friendly forces with which the goal is to be achieved. This is the most important slot of this frame. Examples:

(take "1703" gregg) - take possesion of hex "1703" with unit gregg.

(and (take "1703" gregg)(destroy union-5 (gregg forrest mcnair))) - take possesion of hex "1703" with unit gregg and remove the unit union-5 form the game.

Example:

```
goal frame: goal-1
name:  "13-NOV-198115:18:21.88"
type:  position
value:  5
time:  1
specifics:  (take "2007" (conf-g1 conf-g2))
```

group/unit - This frame is used to describe forces that can be involved in various problems.

name - the unique name for this frame. (Default: the current date and time)

units - the names of the units in this group (from the names of the counters that would be in a stack on the board). (Default: nil)

type - the type of units in the group. (Default: inf, for infantry)

cur-loc - a list of the hex number locations of each unit. (Default: nil)

strengths - a list of the strengths of each of the units. (Default: nil)

total-strength - the combined strength of all the units in the group. (Default: 0)

Example:

```
group frame: conf-g1
name:  "30-JAN-198214:29:39.21"
units:  (gist wilson)
type:  inf
affiliation:  conf
cur-loc:  ("2413" "2413")
strengths:  (5 5)
total-strength:  10
```

situation - This is the frame used to describe situations in which goals will be achieved.

name - the unique name for this frame. (Default: current date and time)

date - the game turn in which this situation occurs. (Default: 1)

area-concerned - a semantic tag describing where on the board this situation occurs. E. g. Dyer's Bridge. (Default: nil)

friendly-units - a list of unit/group descriptors one for each friendly unit/group involved. This slot must be in the correct format. Format of each descriptor: <name location strength> where name is the name of a defined friendly unit/group; location is the current location (as a hex number or list of hex numbers, one for each counter) of this unit/group (this may supercede the entry in the cur-loc slot of the unit/group frame); and strength is the strength of this force. The order of descriptors does not matter as the system will rearrange the list to suit itself as the need arises. (Default: the dummy help list ((name1 loc1 strngth1)(name2 loc2 strngth2)) that is supposed to help you remember the format.)

enemy-units - a list of unit/group descriptors one for each enemy unit/group involved. This slot must be in the correct format, which is the same as that for friendly-units above. (Default: the dummy help list ((name1 loc1 strngth1)(name2 loc2 strngth2)).)

map - the map descriptor of the area involved. Not used. (Default: nil)

friendly-losses - a list of the friendly forces that have been lost up to this point in the game. Not used. (Default: nil)

enemy-losses - a list of the enemy forces that have been lost up to this point in the game. Not used. (Default: nil)

friendly-reinf - a list of descriptors similar to the slot friendly-units above showing forces that may be considered as reserves or reinforcements. (Default: nil)

enemy-reinf - a list of descriptors similar to the slot enemy-units above showing forces that may be considered as enemy reserves or reinforcements. (Default: nil)

semantic-desc - a semantic description of the situation.

Example

```
situation frame: sit-2
name:  "13-NOV-198115:13:56.26"
date:  1
area-concerned:  ("2007" "2010" "2011")
friendly-units:  ((conf-g1 ("1911" "1911") 10)
                  (conf-g2 ("2012" "2012") 9))
enemy-units:  ((union-1 ("2009") 5) (union-2 ("1809") 5))
map:  nil
friendly-losses:  nil
enemy-losses:  nil
friendly-reinf:  nil
enemy-reinf:  nil
semantic-desc:  (("Enemy" "force" "defending" "trail" "juncture"
```

5.0 Data Base Summary

The data base consists of experiences that have been designed to provide tactical advice on any of the following problems:

- Acquisition of simple territorial goals with no enemy opposition.
- Acquisition of a territorial goal with one friendly force against one defending enemy force with 3:1 odds.

- Acquisition of a territorial goal with one friendly force against one defending enemy force with 2:1 odds.
- Acquisition of a territorial goal with one friendly force against one defending enemy force with 1:1 odds.
- Acquisition of a territorial goal with two friendly forces against one defending enemy force with 3:1 odds.
- Acquisition of a territorial goal with two friendly forces against one defending enemy force with 2:1 odds.
- Acquisition of a territorial goal with two friendly forces against one defending enemy force with 1:1 odds.
- Acquisition of two territorial goals with two friendly forces against two defending enemy forces with 3:1 odds on each enemy force.
- Destruction of one enemy force with one friendly force with 3:1 odds.
- Destruction of one enemy force with two friendly forces with 3:1 odds.
- Destruction of two enemy forces with two friendly forces with 3:1 odds against each enemy.
- Holding a section of road with a single friendly unit against two enemy units who have 2:1 odds while trying to protect that unit from being destroyed.
- Holding a region of the map with two friendly units against two enemy units with 3:1 odds against each of the friendly units that are also to be protected.

There are a number of stored goals, situations, scenarios etc. in the data base. These can be seen with the Show command. There are two canned defensive scenario demonstrations: def-scen-1 and def-scen-2. There are also three canned offensive scenario demonstrations: demo-1, demo-2 and demo-3. These can all be displayed on the GMR-27 with the Display command.

6.0 References

[RISS82] Rissland, E. L., "Constrained Example Generation" COINS Technical Report, University of Massachusetts, 1982.