VARIABLE GLOSSARY

FOR THE

SCENARIST™

AUTOMATED SCENARIO GENERATION SYSTEM

PROJECT TITLE: RESEARCH IN ARTIFICIAL INTELLIGENCE FOR NON-COMMUNICATIONS ELECTRONIC WARFARE SYSTEMS

Contract No. DAAB07-89-C-P017

August 31, 1991

Prepared for:

US ARMY COMMUNICATIONS-ELECTRONICS COMMAND
Fort Monmouth, New Jersey

Prepared by:

VISTA RESEARCH CORPORATION
3826 Snead Drive
Sierra Vista, Arizona 85635
(602)378-2130 (602)790-0500

Contents

Foreword

This report was prepared by the staff of Vista Research Corporation, under Contract No. DAAB07-89-C-P017 to the US Army Communications-Electronics Command. Project staff included Dr. J. George Caldwell, Principal Investigator, Mr. William N. Goodhue, Ms. Sharon K. Hoting, Dr. William O. Rasmussen, Mr. Eric Weiss, and Mr. Fletcher Aleong. Government monitoring of the project was provided by Dr. Frank Elmer, Head of the Advanced Concepts Division of the Center for Electronic Warfare/Reconnaissance, Surveillance and Target Acquisition (EW/RSTA). The CECOM Project Manager is Dr. Frank Elmer.

I. Introduction

This report is a glossary of all of the variables used in the program code of the Scenarist automated scenario generation system. In the standard approach to program documentation, program variable definitions would ordinarily be included at the beginning of each program module. In this project, we departed from this standard program documentation procedure and prepared a separate variable glossary for two reasons. First, the contract Statement of Work required a complete glossary of all program variables. Second, in the initial development of the Scenarist, an early version of the Microsoft QuickC program development environment was used. That system placed a severe constraint on the total number of characters allowed to be included in a program file. For small header files and functions, the limitation on program file size presented no problem. For large headers and large functions, however, it was not possible to include all of the desired documentation (text) in the program file. As a "workaround" to this problem, we began the practice of placing descriptive information about module variables in "documentation" files, rather than in the modules themselves.

(Note: Some definitions are included here for the reader who is not familiar with the structure of C-language programs. A "function file" (or "source file" or "source module" or "module") is an MS-DOS file including a number of variable declarations, variable declarations, function declarations, and function definitions. A "header file" is a file containing a number of variable declarations, variable initializations, and function declarations, but no function definitions. The term "program file" is used to refer to either function files or header files. The file names of header files contain the suffix ".h", and the file names of function files contain the suffix ".c". Every variable in a function file must be declared in that file, either as a global variable at the beginning of the file ("outside-function" declaration, "top-level" declaration) or as a local variable within a function ("within-function" declaration). Header files are used to "include" a lot of variable and function declarations in several different function files, without the need for retyping all of the declarations in each file.

(The grouping of variable and function declarations into headers is essentially an art. It is done in a way that enhances program understandability. In the current version of the Scenarist, there are six header files. Each header file contains a group of related variable and function declarations that are "needed" by several functions.

(Related functions are combined into a single function file. Each function could be included in a separate file, but this makes the program harder to understand than if related functions are grouped together. It is not advisable to include a very large number of large functions in the same file, because all of the functions of a file must be recompiled whenever a change is made to any one of them.)

In the later versions of the Microsoft C program development environments (Quick C Version 2.51 and the Programmer's Workbench, both of which are based on Microsoft C version 6.0 compiler), the limitation on file size was no longer present. At that time, we could have proceeded to include the variable descriptions in the program files. Since the contract Statement of Work required a glossary, however, this would have resulted in two sets of variable descriptions -- one in the program file and one in the glossary. It was considered very desirable to avoid having two sets of variable descriptions, since significant effort would be required to maintain both of them and to insure that they contained identical (or at least consistent) definitions. For this reason, no variable definitions were included at the beginnings of the program files or functions. Although the program files and functions do not begin with a section on variable definitions, some definitions are nevertheless present in the bodies of functions, namely, definitions that are imbedded in the program code to assist program maintenance.

The organization of this glossary corresponds to the organization of the Scenarist program, i.e., the variables definitions are arranged according to file (header files or function files) and function definition within function file. The rationale for this organization is as follows. First, the purpose of the glossary is to assist a programmer who is modifying the Scenarist in understanding how it does what it does. In program maintenance, it is desirable to have a list of definitions for all variables used by each function. It is not as helpful to have one overall variable list, such as a single alphabetical variable list for the entire Scenarist program (i.e., for all program files combined). Second, the same variable name may be used in different ways in different functions (for example, the index "i" for an iteration loop). Such variables might be declared as "global" variables, but this is not a good practice. Such variables might be given "generic" definitions, but it is more helpful to indicate the specific role of each such variable in each function in which it is used. It is far more helpful to the program maintainer to present the list of variable definitions function by function, rather than to list each variable name and then list all of the functions in which it occurs and all of the various definitions in these functions. (The only exception to this is the case in which a variable name is used consistently throughout many functions in exactly the same way. In this case, the same definition is repeated over and over again in every function, and the glossary becomes duplicative. In this glossary, we shall present identical definitions only a few times. After a few identical occurrences, we shall not repeat the definition in later functions.)

The organization of the glossary is as follows. The variable definitions will be presented file by file and function by function. All header files (i.e., all files containing the suffix ".h") will be addressed first, followed by all function files (i.e., all files containing the suffix ".c"). The header files contain declarations for "global" variables. Global variables are variables that may be used by any function in the file in which they are declared. Global variables are distinguished from "local" variables, whose existence is recognized only within the function in which

they are defined. Function files contain both global and local variables. The global variables are declared at the beginning of the file outside of the body of any function. Local variables are declared inside the bodies of functions.

For each function in a file, the glossary format is as follows. The format is slightly different for header files ("headers") than for functions files and functions, since headers only declare variables whereas functions accept variable input (calling parameters, or arguments) and change variable values. For headers, all of the variables of the header will be listed and defined. For function files, a statement of the nature of the functions in the file will be given, along with a list of all functions defined in the file. Then, all (global) variables declared or initialized at the beginning of the file will be defined.

For functions (within function files), the format is as follows. First, the function declaration will be specified. That is, the function name and return-value type will be given, along with the names and types of all of the input parameters (calling parameters, arguments). Next, an explanation of the purpose and (mathematical) function of the (C-language) function will be presented. Definitions will be given for each of the input parameters and function return value (if any). A list will be given of all of the global variables whose values are changed by the function, and all of these variables will be defined. All files modified by the function will then be identified. Finally, all remaining (local) variables used in the function (e.g., iteration loop indices and other temporary variables used by the function) will be listed and defined.

In summary, the glossary format is as follows:

For Header Files:

1. Purpose of file
2. Definition of each variable in the file

For Function Files:

1. Purpose of file
2. List of functions in file
3. Definition of each variable declared or initialized at the beginning of the file
4. For each function, the following information:
    a. Declaration
    b. Purpose
    c. Definitions of input parameters
    d. Definition of function return value (if any)
    e. Global variables modified
    f. Files modified
    g. Variable definitions

A note is in order concerning the topic, "Global variables modified." Under this topic, we will specify the important global variables (or data structures) modified by the function, viz., those global variables whose

values are purposely changed by the function for use by other functions. We will not list "working," or "local-use" global variables used by the function in the role of local variables. These "local-use" global variables are global variables that are used for "local" computations, without any intention of using their values as computed in a particular function in other functions (which is the primary functional purpose of global variables). They were defined as global variables simply because they occurred in identical roles in many functions, and it seemed a little "cleaner" to declare them once in a header file as global variables rather over and over again in every function in which they occurred.

In retrospect, this local use of global variables was not a good idea. The drawback is that there are more data "flows" between functions than are necessary. This situation makes it somewhat more difficult for a programmer to maintain the Scenarist, since it is not clear whether a global variable is global because it is intended to transfer data from one function to another (through the global variable) or whether the global variable is used only as a local working variable. The most significant instance in which global variables were used simply as local variables occurs is the case of the data structures of the "unit" data structure type (unit0, unit1, genericunit[], and specificunit[]). The data structure unitblank is used globally (it contains a set of intitializing values for a unit data structure). The global variable code[] is used as a global variable; it contains the code of the last unit accessed, in whatever function it was accessed. Most other variables defined as global variables were used as global variables (i.e., their values, after being computed in one function, are used in another function).

An easy way of avoiding the problem of the unnecessary increase in the number of data flows among modules would be to convert the Scenarist to the C++ object-oriented language. In this language, the functions that process data and the data are "encapsulated" into a single object. Other reasons for considering a switch to C++ are discussed in the Final Report.

The C programming language allows for the grouping of related variables into collections called "data structures." The variables (or components, or members, or fields) of the data structure may be of different data types (e.g., character, integer, floating point, or even other data structures). Once a data structure type has been declared in a program, a number of data structures may be declared as being of this data structure type. Although the individual components of a data structure are often accessed individually, the complete data structure may be referred to using a single identifier. Data structures are often manipulated as individual entities, i.e., written to a file, read from a file, or passed to a function. Since data structures are operated on by functions, they can themselves be considered as variables, in addition to the simpler variables that comprise them.

It is helpful for a programmer maintaining a program to have not only a definition for each component of a data structure declaration, but also to have a definition for every data structure (of the various data structure

types). For this reason, this glossary contains definitions for both the components of every data structure type declaration as well as definitions of every data structure. As this glossary proceeds through the various program files, all program variables (simple data types or data structures) are defined in approximately the order in which they occur. Whenever a data structure type declaration is encountered, all of the variables in the data structure type will be defined.

With respect to the specific order in which variables are defined in this glossary, we shall define all variables in header files generally in the order in which they are declared. In function files and functions, we shall also define them generally in the order in which they are declared at the beginning of the function file or function. In functions, we will define all input parameters prior to defining all of the other ("formal") parameters of the function.

In a few cases, a variable may be declared at the beginning of a function, but never used in the function. This happens when code is modified, a variable is no longer used, and the programmer forgets to delete its declaration. The Microsoft C compiler was executed from time to time with a special option to identify such "null" variables, and they were eliminated. To save compilation time, however, this special option was not used every time the code was recompiled, and a few such unused variables exist in the Scenarist code. Whenever they are encountered, they will be denoted as "null." After null variables are discovered, they are removed from the program prior to publishing the next listing. (A few nulls remain in this glossary because it was produced after the final Scenarist program listing.)

This glossary presents definitions only for variables in software developed by Vista under the contract. Variables in software not developed by Vista under the contract are not included in the glossary. This exclusion includes variables in Microsoft software (e.g., compiler, libraries, software development systems), in CLIPS software, or in software routines derived from other sources, such as graphics, printer, mouse, or window routines obtained from textbooks or other published sources. In cases in which published routines are used, a reference for the software source is provided in the code.

This glossary presents simply definitions, not detailed descriptions or explanations. For the major variables (such as map and unit data structures), detailed descriptions are presented in the Final Report.

II. Listing of All Scenarist Program Files

The following is a listing of all of the Scenarist files, both header files and function files. This list is presented here since it specifies the (file and function) order in which the variable definitions will be

presented. The list is taken from the file s03adoca.doc, which contains, in addition to this list, a detailed description of the "unit" data structure. (A listing of the file s03adoca.doc is included in the Scenarist Programmer's Manual.)

In the list that follows, the file name (header file name or function file name) is specified on the left. To the right of the file name is a brief description (English title) of the file. For function files, the file description is followed by a list of all of the functions in the file.

Scenarist Automated Scenario Generation Program
Table of Contents
File Name: s03adoca.doc Date: August 31, 1991

Scenarist Program Files

File Name	Contents
s03adoca.doc	Documentation -- Description of Unit Data Structure
s03binca.h	Principal Header File
s03cincb.h	Header File for Map Drawing Programs -- Declarations
s03dintb.h	Header File for Map Drawing Programs -- Initialization
s03eclip.h	CLIPS Header File
s03fintf.h	Window, Menu, and Mouse Header File
s03gdemo.h	EGA, DEMO, and CLIPS Conditional Compilation Header File
s03gmain.c	Main Program: main, usrfuncs, CLIPSval_0101, CLIPSval_0201
s03ipuu.c	Reposition Unit by User Functions: _repositionunitbyuser, _repositionprogeny, _repositionfeba
s03jpsu.c	Reposition Subunits by User Function: _repositionsubunitsbyuser
s03kpsr.c	Reposition Subunits by Rule -- File 1 (General Rule Repositioning Functions Plus Functions Specific to Problem 0101 -- Field Artillery and Air Defense Radars): _repositionsubunitsbyrule, _spiralsearch, _preprocessing0101, _action0101, _suitability0101, _terrain, _elevation, _distancetofeba, _horizonangle, _mapcellsizestdcoords, _clipssuitability0101
s03kpsr2.c	Reposition Subunits by Rule -- File 2 (Functions for Problem 0201 -- TRAILBLAZER): _preprocessing0201, _createaccessibilitymap, _accessibility, _lostotarget, _LOS, _road, _slopetorearcell, _lostootherunits, _lostoheadquarters, _disttootherunits, _distancetofront, _inforwardarea, _action0201, _suitability0201, _clipssuitability0201
s03ldefu.c	Define Units Function: _defineunit

s03mcopy.c Copy, Delete, and Display Units Functions: _editunit, _copyunit, _getunitbycode, _getsubunitbycode, _getunitbyidno, _getunitbyindex, _getsubunitbyidno, _deleteunit, _displayunit, _setcoordsforzoommap, _outputunits

s03nmap.c Map Drawing Functions: _drawmap, _label, _legend, _setcoordsforzoommapb

s03pdraw.c Functions for Drawing Units on Map: _drawunit, _drawline, _transfstdtoreal

s03qgtfl.c Get Files Function: _getfilenames, _readmapheader, _readmapdata, _resetmaplocpoint

s03rsymb.c Function for Drawing Symbols: _symbol

s03sio.c Basic Input-Output Functions: _printscreen, _clearfoot, LJ_Graphic, format, Grey_Scale, Print_Pause, PromptLine, writString, writChar, put_out, status, pralel, stat, PrtInit, printch, printst, PrtPutC, getMode, getxy, gotoxy, getPage

s03twind.c Windows Functions: setWindow, _windowa, _windowb, _windowbblack, _windowbenter, _windowb1, _windowb1enter, _windowb2, _windowb3, _windowc, _windowcenter, _windowscreen, _windowscreenblue, _windowd, _windowe, _windoweenter

s03vmous.c Mouse Functions: m_reset, m_show, m_hide, m_pos, m_moveto, m_pressed, m_released, m_xlimit, m_ylimit, m_text_cursor, m_motion, m_lightpen, m_move_ratio, m_conceal, m_speed, m_graphic_cursor, initialize_cursor, Set_Graphic_Cursor, _setup_button, _clickbutton, _click_accept, _click_cancel, _call_button, _call_buttonb, _call_continueb, _call_continue, _Hardware_Setup, _setup_screen_windows

s03wgrap.c Graphics Interface Functions -- Main Menu: _display_intro_screens, _display_screen_windows, _project_selection, _main_menu_selection, _display_windowd, _About, _Units,_Rules, _Map, _Scenario_Generation, _Quit

s03ygra2.c Graphics Interface Functions -- Map- and Unit-Related: _map_menu_selection, _unit_menu_selection, _Draw_Terrain_Map, _Draw_Elevation_Map, _Draw_Road_Map, _Add_Vector_Map_with_Labels, _Add_Vector_Map_without_Labels, _Draw_Vector_Map_without_Labels, _Place_Unit_on_Map, _Zoom_Map, _Change_Map_Location, _Change_Map_Files, _Print_Map, _Define_Unit, _Copy_Unit, _Delete_Unit, _Reposition_Unit_by_User, _Reposition_Subunits_by_User, _Reposition_Subunits_by_Rules, _Reposition_FEBA, _Display_Unit,

s03xcomp.c Map Compression Program: main, _readmapheadernongraphic, _writemapheader

s03zdemo.c Run Demo Script Function: _run_demo_script

listings.c Listing Program

win7.for Program for Building a Scenarist Cellular Map File from Mapping Data in the GRASS Main Data Base

win7l.for Program for Building a Scenarist Vector Map File from Mapping Data in the GRASS Line Data Base

Scenarist Data Files

File Name	Contents
filexxxx.fil	Names of all other Scenarist data files to be used in a run, plus the names of the suitability, preprocessing, and action functions to be used in the run
intro.fil	Scenarist introduction screen
titlxxxx.fil	Map parameters and map title
geodxxxx.fil	Terrain-type cellular map file
geocxxxx.fil	Elevation cellular map
georxxxx.fil	Road availability cellular map file
geoaxxxx.fil	Map area objects (e.g., towns, bodies of water), for vector map
geolxxxx.fil	Map linear objects (e.g., roads, rivers), for vector map
geopxxxx.fil	Map point objects, for vector map
genuxxxx.fil	Generic unit file (binary file -- no listing)
specxxxx.fil	Specific unit file (binary file -- no listing)
scraxxxx.fil	Scratch file (binary file -- no listing)
symbxxxx.fil	Military unit symbols, labels
platxxxx.fil	Platform file (descriptions of platforms)
eqptxxxx.fil	Equipment file (descriptions of equipment)
febaxxxx.fil	FEBA file (locations of up to five points on FEBA)
clipxxxx.fil	CLIPS rule file
highlits.fil	Summary descriptive information about the Scenarist

III. Variable Definitions

A. Header File s03binca.h: Principal Header File

Purpose: Of the six header files in the Scenarist, five of them are very special-purpose (i.e., they contain variable and function declarations for only a few function files) and one of them is quite general (i.e., it contains variable and function declarations that are used by many function files). The header file s03binca.h is the general header. It contains the variable and function declarations related to processing of unit and map data. The other header files contain declarations related to the CLIPS

expert system, to drawing maps, to graphics interfaces (windows, menus, and use of the mouse), and to a code that specifies whether the program is to be compiled in an operational or demonstration mode.

The following are the variables declared in s03binca.h. In every case in which a variable is defined, its declaration will be specified. The declaration specifies the data type of a variable; if it is an array, the declaration also specifies the dimensions.

Note that in addition to variable declarations the header also contains a large number of function declarations. For functions, the declaration specifies the type of value returned by the function and the number and types of its calling parameters. Definitions for functions are presented in the section of the glossary dealing with function files, not in this header-file section.

Variable Definitions:

struct videoconfig vc: a variable that contains information about the hardware system video parameters. (Note: For those readers unfamiliar with the C programming language, the expression "struct videoconfig vc" is a declaration of the variable "vc", indicating that it is a data structure of data structure type "videoconfig".) The "videoconfig" data structure type is declared in a "library" provided with the Microsoft compiler, and so its components will not be defined in this glossary.

char *modnam: variable used to store name of video mode.

int nxpch: number of x pixels per character.

int nypch: number of y pixels per character.

int borderbottom: the bottom of the screen display, in pixel coordinates.

char nameofpreprocessingfn[22]: variable used to store name of preprocessing function (read from project file).

char nameofactionfn[22]: variable used to store name of actionfunction (read from project file).

char filenamesfiln[13]: the name of the "project file." The project file contains the names of 20 other files used by the Scenarist. These files specify the units and maps to be used by the Scenarist in the current run.

char introfiln[13]: the name of a file containing the text of the introductory material displayed on the screen at the beginning of a run.

char titlefiln[13]: the name of a file that contains a map title; not used in the current version.

char geodfiln[13]: the name of a file containing data for a terrain-type cellular map.

char geocfiln[13]: the name of a file containing data for an elevation cellular map.

char georfiln[13]: the name of a file containing data for a road cellular map.

char geoafiln[13]: the name of a file containing area-object data for a vector map.

char geolfiln[13]: the name of a file containing linear-object data for a vector map.

char geopfiln[13]: the name of a file containing point-object data for a vector map.

char genunitfiln[13]: the name of a file containing generic-unit data.

char specunitfiln[13]: the name of a file containing specific-unit data.

char scratchfiln[13]: the name of a file used for temporary storage during Scenarist processing.

char symbolfiln[13]: the name of a file containing the symbol numbers and labels for units.

char platformfiln[13]: the name of a file containing platform data.

char eqptfiln[13]: the name of a file containing equipment data.

char febafiln[13]: the name of a file containing FEBA data.

char clipfiln[13]: the name of a file containing CLIPS rules.

FILE *filenamesfilp: a pointer to the file having name filenamesfiln (i.e., a pointer variable containing the address of the starting storage location for that file).

FILE *introfilp: a pointer to the file having name introfiln.

FILE *titlefilp: a pointer to the file having name titlefiln.

FILE *geodfilp: a pointer to the file having name geodfiln.

FILE *geocfilp: a pointer to the file having name geocfiln.

FILE *georfilp: a pointer to the file having name georfilp.

FILE *geoafilp: a pointer to the file having name geoafilp.

FILE *geolfilp: a pointer to the file having name geolfilp.

FILE *geopfilp: a pointer to the file having name geopfilp.

FILE *genunitfilp: a pointer to the file having name genunitfiln.

FILE *specunitfilp: a pointer to the file having name specunifiln.

FILE *scratchfilp: a pointer to the file having name scratchfiln.

FILE *symbolfilp: a pointer to the file having name symbolfiln.

FILE *platformfilp: a pointer to the file having name platformfiln.

FILE *eqptfilp: a pointer to the file having name eqptfiln.

FILE *febafilp: a pointer to the file having name febafiln.

FILE *clipfilp: a pointer to the file having name clipfiln.

struct titleline: a data structure type containing three variables that specify a map title. These variables are:

char line1[81]: the name of a variable containing the first line of the map title. (Note: for simplicity, we shall from this point on drop the expression "the name of a variable containing" from the variable definitions, and simply state what data the variable contains. In this case, the definition for line1[81] becomes simply "the first line of the map title." This abbreviated definition is not conceptually correct (since the variable in fact contains (or denotes, in a mathematical description) the map title, but is not the map title), but it is common usage. Furthermore, we are here defining the data structure type, not a particular data structure. In the data structure type declaration, the symbol "line1[81]" is just a variable name, not a storage location. In a particular data structure of that type (i.e., an instantiation of the data structure type), the symbol "line1[81]" denotes a storage location. While these distinctions are evident to a programmer, they are of little concern to someone simply trying to understand the role of the variables in the Scenarist model.)

char line2[81]: the second line of the map title

char line3[81]: the third line of the map title

struct titleline projecttitle: a data structure (of data structure type titleline) containing a title for the current project. This variable is not used in the present version of the Scenarist. (It was dropped to allow more room on the map for geographic features. Its use may be reinstituted in a later version of the Scenarist, on a screen having more pixels than a VGA monitor. For this reason it has been left in the program code.)

struct mapinfo: a data structure containing 26 variables that describe a cellular map. These variables are:

char filename[13]: the name of a map file (specifically, the name of the file containing all of the values of the variables of a data structure of type mapinfo, including this map file name).
char mapname[80]: the name of the map.
char datatopic[80]: the kind of data in terms of which the map is defined (e.g., terrain type, elevation).
char datasource[80]: the source of the map data.
char creationdate[80]: the date on which the map file was created.
char changedate[80]: the date on which the map file was last modified.

int ncats: If the map data are discrete (categorical, e.g., terrain type), this variable is the number of categories. If the map data are continuous (e.g., elevation), this variable is zero.
char codename[20][80]: the names of the map data categories (for discrete-variable maps).
char unit[80]: the name of the unit of measurement (for continuous-variable maps, e.g., meters).
int utmzonenumber: the UTM zone number.
int utmrowletter: the UTM band letter.
float xmin, ymax: the (map) coordinates of the top-left corner of the map.
float: xmax, ymin: the (map) coordinates of the bottom-right corner of the map.
float cellwidth: with width of a map cell (in meters).
char format[80]: the format of the map data, in FORTRAN notation. This format is for information only; it is not used by the Scenarist.
float xminwindow, ymaxwindow: the (map) coordinates of the top-left corner of the map "window" extracted from the map file and stored in memory.
float xmaxwindow, yminwindow: the (map) coordinates of the bottom-left corner of the map "window" extracted from the map file and stored in memory.
int rowswindow, colswindow: the number of rows and number of columns of the map "window" extracted from the map file and stored in memory.

struct mapinfo mapinf[2][3]: a data structure (of data structure type mapinfo) containing the cellular maps stored in memory. Symbolically, mapinf[maptype][mapindex], where "maptype" refers to map type (0 = discrete, 1 = continuous) and "mapindex" is an index. mapinf[0][0] contains the map data for the terrain-type map. mapinf[0][1] contains the map data for the road availability map. mapinf[0][2] contains the map data for the TRAILBLAZER accessibility map. mapinf[1][0] contains the map data for the elevation map.

int p1,p4,q1,q4: (p1,q1) and (p4,q4) are the pixel coordinates of the bounding rectangle of the viewport (location on the screen) used to draw map graphics. These coordinates are used to set the viewport for the map (with the Microsoft command _setviewport(p1,q1,p4,q4)).

int r1,r4,c1,c4: (r1,c1) and (r4,c4) are the character coordinates of the bounding rectangle of the viewport used for placing text on maps (i.e., in the map viewport, with the Microsoft command _settextwindow(r1,c1,r4,c4)).

double xminview, xmaxview, yminview, ymaxview: (xminview, yminview) and (xmaxview, ymaxview) are the map ("real") coordinates of the lower-left and upper-right corners of the "current" map. The "current" map is a map to be drawn (or just drawn) on the screen. These coordinates may refer either to a full, "unzoomed" map (i.e., the original map extracted from a map file and stored in memory) or to a zoomed map. These variables are used to define the coordinate system to be used in the map viewport (with the Microsoft command, _setwindow(1, xminview, yminview, xmaxview, ymaxview)).

double xminvieworiginal, xmaxvieworiginal, yminvieworiginal, ymaxvieworiginal: (xminvieworiginal,yminvieworiginal) and (xmaxvieworiginal,ymaxvieworiginal) are the map coordinates of the full, "unzoomed" map from the which the current map is derived. If the current map is not a zoomed map, these values are the same as xminview, xmaxview, yminview, ymaxview. These variables are used to reset the values of xminview, xmaxview, yminview, and ymaxview to their original values for the full, unzoomed map, if the user wishes to "unzoom" a zoomed map.

struct coord: a data structure type containing variables that identify the location of a point in a two-dimensional cartesian coordinate system. There are no data structures of this data structure type; this data structure type is used solely as a component of the "areaobject," "linearobject," and "pointobject" data structure types (defined below). The variables of the coord data structure type are:

float x,y: the coordinates of a point.

struct coord3d: a data structure type containing variables that identify the location of a point in a three-dimensional cartesian coordinate system. This data structure type is not presently used. The variables of the data structure are:

float x,y,z: the coordinates of the point.

struct areaobject: a data structure type containing variables that identify an area object. The variables of this data structure are:

char name[17]: the name of the object

int printind: an indicator variable that specifies whether the object's name is to be printed on the screen (0 = no print, 1 = print).

int areaobjtype: the area object type (e.g., body of water).

int index: the order of the object in the area object file. For information only; not used by Scenarist.

struct coord br1,br2: the coordinates of two diagonally opposite corners of a bounding rectangle for the area object. Before attempting to draw an area object, a check is made to see whether the bounding rectangle is outside of the window. If it is, no further processing of the object is done (for drawing).

struct coord intpoint: an interior point of the area object. Used by the "floodfill" function.

struct coord namepoint: the coordinates of the location where the name is to be printed (if it is to be printed at all).
float area, magnitude, value, meanel1, meanel2: five attributes of an area object. These names are arbitrary; the user may specify any five attributes. These variables are not used in the current version of the Scenarist.
int nopts: the number of points defining the boundary of the polygon representing the area object.
struct coord vertex[20]: the coordinates of the locations of the points defining the boundary of the polygon representing the area object.

int areavertexdim: the dimension range (or bound) for the array vertex[] in the data structure type areaobject.

struct areaobject areaobject[20]: a data structure (of data structure type areaobject) containing all of the data defining an area object. In the present version of the Scenarist, only the first element, areaobject[0], of the array areaobject[] is used. (To draw a vector map, the data structures stored in the area-object, linear-object, and point-object files are read from the file into memory one at a time and drawn on the screen. The only object in memory is the one currently being processed. The reason for the dimension of 20 for the array areaobject[] is that it is planned in a future extension of the Scenarist to store up to 20 area objects in memory. In this way, if the number of area objects in the file does not exceed 20, the area object file will not have to be accessed again, if it is desired to draw the vector map again.)

struct linobject: a data structure type containing variables that identify a linear object. The variables of this data structure are:
char name[17]: the name of the object.
int printind: a print indicator variable (i.e., a variable that specifies whether the object name is to be printed on the screen). Same definition as above.
int linobjtype: the type of linear object (e.g., road).
int index: the order of the object in the linear object file. For information only; not used by Scenarist.
struct coord br1,br2: the coordinates of two diagonally opposite corners of a bounding rectangle for the linear object. Before attempting to draw a linear object, a check is made to see whether the bounding rectangle is outside of the window. If it is, no further processing of the object is done (for drawing).
struct coord namepoint: the coordinates of the location where the name is to be printed (if it is to be printed at all).

char origobj[5],destobj[5]: labels for the origin (first point) and destination (last point) of the linear object. Not used in current version of Scenarist.

float length, width, capacity, flowrate, flowdir, value: six attributes of a linear object. These names are arbitrary; the user may specify any six attributes. These variables are not used in the current version of the Scenarist.
int nopts: the number of points defining the linear object. The linear object is drawn as a sequence of straight lines connecting the points.
struct coord vertex[85]: the coordinates of the locations of the points defining the linear object.

int linvertexdim: the dimension range (or bound) for the array vertex[] in the data structure type linobject.

struct linobject linobject[20]: a data structure (of data structure type linobject) containing all of the data defining a linear object. In the present version of the Scenarist, only the first element, linobject[0], of the array linobject[] is used.

struct pointoject: a data structure type containing variables that identify a point object. The variables of this data structure are:
char name[17]: the name of the object.
int printind: a print indicator variable (i.e., a variable that specifies whether the object name is to be printed on the screen). Same definition as above.
int ptobjtype: the type of point object (e.g., mountain peak).
int index: the order of the object in the point object file. For information only; not used by Scenarist.
struct coord namepoint: the coordinates of the location where the name is to be printed (if it is to be printed at all).
struct coord location: the coordinates of the locations of the point object.
float magnitude, value: two attributes of a point object. These names are arbitrary; the user may specify any two attributes. These variables are not used in the current version of the Scenarist.

struct pointobject ptobject[10]: a data structure (of data structure type pointobject) containing all of the data defining a point object. In the present version of the Scenarist, only the first element, ptobject[0], of the array ptobject[] is used.

int geodisc[3][32][32]: array used to store discrete cellular map data (e.g., terrain type). Symbolically, geodisc[mapindex][row][column], where mapindex is 0 (terrain type) or 1 (roads).

int geodiscdim1, geodiscdim2, geodiscdim3: the bounds for the three dimensions of geodisc.

int geocont[1][32][32]: array used to store continuous cellular map data (e.g., elevation). Symbolically, geocont[mapindex][row][column], where mapindex is 0 (elevation). Second dimension is included (even though

dimension bound is 1) to allow for easy extension of program to allow for more than a single continuous variable.

int geocontdim1,geocontdim2,geocontdim3: the bounds for the three dimensions of geocont.

float geocontmaxmin[1],geocontminmax[1],geocontquint[1][6]: symbolically, geocontmaxmin[mapindex], geocontminmax[mapindex], geocontquint[mapindex][attributecategory]. Variables used to compute the quintiles of a continuous-variable cellular maps. The quintiles are used in the display of continuous-variable maps, to represent five different levels of the variable. In the current program, no use is made of the "mapindex" dimension; it could be dropped. Furthermore, these variables do not have to be global variables; they could be declared as local variables in the map-drawing program, s03nmap.c. (These variables were made global and the index was included with an eye to saving the quintile data for several continuous-data maps (with the dimension bound increased from 1), so that they did not have to be recomputed every time a continuous map was redrawn. This feature was never implemented, owing to time/resource constraints.) The variable geocontminmax[] contains the minimum value of the map variable, for the entire map. The variable geocontmaxmin[] contains the maximum value of the map variable, for the entire map. The variable geocontquint[][i] contains the category boundaries for the quintiles (i=1,2,...6).

float elevmapcellsizestd: elevation map cellsize, in standard coordinates.

float terrainmapcellsizestd: terrain map cellsize, in standard coordinates.

float roadmapcellsizestd: road availability map cellsize, in standard coordinates.

float xgridsize, ygridsize: the average width and depth of the the unit grid cells, in standard units (this value is determined from the values of xxxxmapcellsizestd, e.g., it is one of them or the min of them)

struct unitid: a data structure type containing variables that identify a military item (unit, platform, or equipment). There are no data structures of this data structure type; this data structure type is used solely as a component of the "unit" data structure type (defined below). The variables of the unitid data structure type are:

short side: the item's side (BLUE = 1, RED = 2, GRAY = 3).
short echelon: the item's echelon (side = 1, army = 2, corps = 3, division = 4, brigade = 5, regiment = 6, battalion = 7, company = 8, platoon = 9, section = 10, squad = 11, platform = 12, equipment = 13).
short type: the item's type. "type" is a descriptor used to distinguish units of the same echelon, e.g., to distinguish a tank battalion from a military intelligence battalion.

short number: the item's number. "number" is a description used to distinguish units of the same echelon and type, e.g., to distinguish two tank battalions.
short idno: an identification number assigned by the user, to assist his retrieval of units from the specific unit file. Not used by the Scenarist for unit identification. Use 0 for generic units. If two units are assigned the same idno and the user chooses to retrieve by this identifier, the first such unit will be selected from the file.
short parentidno: the idno of a unit's parent unit.
short code[13]: a 13-component vector containing the item's side, army, corps, division, brigade, regiment, battalion, company, platoon, section, squad, platform, and equipment numbers. code[] is the identifier used by the Scenarist to store and retrieve items.
short parentcode[13]: the code[] of the unit's parent unit.

struct subid: a data structure type containing variables that identify a subordinate item of a military item (unit, platform, or equipment). There are no data structures of this data structure type; this data structure type is used solely as a component of the "unit" data structure type (defined below). The variables of the subid data structure type are:

short echelon: the subordinate item's echelon (same definition as given above, for an item).
short type: the subordinate item's type (same definition as given above, for an item).
short number: the subordinate item's number (same definition as given above, for a item).
short idno: the subordinate item's idno (same definition as given above, for an item).

struct unitatt: a data structure type containing variables that are attributes of an area subordinate item ("subitem"), i.e., a subitem of geographic type ("geogtype") 1-6. There are no data structures of this data structure type; this data structure type is used solely as a component of the "unit" data structure type (defined below). The variables of the unitatt data structure type are:

struct subid id: the subitem's identification ("id").
float corners[4][2]: For specific units, the map coordinates of the four corners of the subitem, starting at the item's right front corner (looking forward) and proceeding counterclockwise. For generic units, these are (0,0), (1,0), (1,1), (0,1).
loc[2]: the nominal location of a subitem (used for symbol location; may also be used for analysis).
float rad: the radius of an item (in meters).
short geogtype: the geographic type ("geogtype") of an item (1-7). See definition of the seven geographic type codes in the program listing (or the Final Report).
short placementcode: the placement code; a code denoting the method of placement of an item (e.g., user-suggested, rule-based). See definition of the placement code in the program listing or the Final Report.

struct itematt: a data structure type containing variables that are attributes of a point subordinate item ("subitem"), i.e., a subitem of geographic type ("geogtype") 7 -- platforms or equipment. There are no data structures of this data structure type; this data structure type is used solely as a component of the "unit" data structure type (defined below). The variables of the itematt data structure type are:

struct subid id: the subitem's id.
float loc[2]: the nominal location of a subitem (same definition as given above for area subitems).
short placementcode: the subitem's placement code (same definition as given above for area subitems).

struct unit: a data structure type containing variables that are attributes of a military item (unit, platform, or equipment). In the present version of the Scenarist, this data structure is used only for units; it is likely that a different data structure will be defined for platforms and equipment, when an application involving them is undertaken. Since the structure is currently used only for units, the term "unit" will be used below, rather than the more general term "item." The variables of the unit data structure are as follows (detailed descriptions are included in the Final Report):

struct unitid id: the unit's id.
char name[17]: the unit's name.

float corners[4][2]: For specific units, the map coordinates of the four corners of the unit, starting at the item's right front corner (looking forward) and proceeding counterclockwise. For generic units, these are (0,0), (1,0), (1,1), (0,1).
loc[2]: the nominal location of a unit (used for symbol location; may also be used for analysis).
float rad: the radius of a unit (in meters).
float flankp[2]: the relative positions of the intersections of the imaginary line separating the unit's front and rear with the unit's flanks (sides). The relative position is the proportional distance along the side, measuring from the front to the rear.
short nfront: the number of on-line (front) subunits.
struct subid frontid[5]: the id's of the front subunits.
float frontp[4][2]: the relative positions of the intersections of the imaginary lines separating the front subunits with the front and with the line defined by the variable flankp[]. The first relative position is the proportional distance along the front, measuring from the unit's first corner to the unit's second corner. The second relative position is the proportional distance along the imaginary line defined by the variable flankp[], measured from the unit's right side (facing forward) to the unit's left side.
short nrear: the number of reserve (rear) subunits.
struct subid rearid[5]: the id's of the rear subunits.
float rearp[4][2]: the relative positions of the intersections of the imaginary lines separating the rear subunits with the rear and with the line defined by the variable flankp[]. The first relative position is the proportional distance along the imaginary line defined by the variable flankp[], measuring from the unit's right side to the unit's left side. The second relative position is the

proportional distance along the rear, measured from the unit's right side to the unit's left side.
short nallarea: the number of all-area subunits.
struct subid allareaid[5]: the id's of the all-area subunits.
float allareasymloc[5][2]: the symbol location points for the all-area subunits, in relative coordinates. (All internal positions of a unit are specified in terms of "relative" or "standardized" or "unit" coordinates. This coordinate system corresponds to the linear transformation of the unit's real (map) coordinates of its four corners to the points (0,0), (1,0), (1,1), (0,1).)
short nmajorsubarea: the number of major subarea subunits.
struct subid majorsubareaid[5]: the id's of the major subarea subunits.
float majorsubareacorners[5][4][2]: the (two) coordinates (in relative coordinates) of the four corners of each of the major subarea subunits.

short nminorsubarearel: the number of large minor subarea subunits.
struct subid minorsubarearelid[5]: the id's of the large minor subarea subunits.
float minorsubarearelloc[5][2]: the symbol location points (in relative coordinates) for the large minor subarea subunits.
float minorsubarearelrad[5]: the radii of the large minor subarea subunits, measured relative to the length of the unit's smallest side.
short nminorsubareaabs: the number of small minor subarea subunits.
struct subid minorsubareaabsid[5]: the id's of the small minor subarea subunits.
float minorsubareaabsloc[5][2]: the symbol location points (in relative coordinates) for the small minor subarea subunits.
float minorsubareaabsrad[5]: the radii of the small minor subarea subunits, measured in meters.
short npoint: the number of point (nonarea) subitems.
struct subid pointid[30]: the id's of the point subitems.
float pointloc[30][2]: the location points (in relative coordinates) of the point subitems.
short nsubunits: the total number of subunits in the unit.
struct unitatt subunitatt[30]: the attributes of the unit's subunits (area subitems).
short npointsubitems: the total number of point subitems in the unit.
struct itematt subitematt[30]: the attributes of the unit's point subitems.
int objind: objective indicator variable; specifies whether the unit has a defined objective; 0 = no, 1 = yes.
int objtype: the objective type (type codes defined by user).
int objmissiontype: the mission type (mission type codes defined by user).
float objboundrect[2][2]: coordinates of two opposite corners of bounding rectangle around objective.
int navenue: number of points along avenue of approach from center of unit's front to center of bounding rectangle around objective.
float avloc[5][2]: coordinates of points along avenue of approach.

short reserved[230]: extra space in the unit data structure, so that it may not be necessary to rebuild the generic and specific unit data files if minor changes are made to the unit data structure.

struct unit genericunit[2]: an array of data structures of data structure type unit, used to store generic units.

struct unit specificunit[2]: an array of data structures of data structure type unit, used to store specific units.

struct unit unitblank, unit0, unit1: data structures of data structure type unit, used to store units (either generic units or specific units, for various purposes).

struct unit *unitpoint, *unit0point, *unit1point, *specunitpoint[2]: pointers to data structures of data structure type unit.

short symbols[3][13][10]: symbolically, symbols[side][echelon][type]. Array used to store symbol numbers for items.

char labels[3][13][10][9]; symbolically, labels[side][echelon][type][label]. Array used to store labels for items.

int nsides, nechelons, ntypes: dimension bounds for "side," "echelon," and "type" dimensions of symbols[][][] and labels[][][][].

int code[13]: array used to store an item's code.

int parentcode[13]: array used to store the code of an item's parent unit.

int codedefined: a variable that indicates whether the user has previously input a unit code. If so, that code is saved, as the "current" code. Whenever the program requires a unit code, the user is queried as to whether he wishes to use the current code or input a new code.

long fileposition: a variable that indicates the position of a record in a file.

int nfebapts: the number of points defining the FEBA.

float feba[5][2]: the coordinates of the points defining the FEBA.

B. Header File s03cincb.h: Header File for Map Drawing Programs -- Declarations

Purpose: This header file includes variable type declarations for variables related to the drawing of maps. These variables include codes for fill patterns and colors to be used on maps, and the labels to be used for six

different classes of map objects. These six map-object classes are: (1) the cells of terrain-type maps; (2) the cells of elevation maps; (3) area geographic objects; (4) linear geographic objects; (5) point geographic objects; and (6) the cells of a cellular road map. These fill patterns and colors are used to color and shade the cells of cellular maps and the objects of vector maps. The fill patterns, colors, and labels for map-object classes (1), (2) and (6) are used in the legends for terrain-type, elevation, and road cellular maps, respectively.

Note that the labels are specified here independently of the labels specified in the vector map files (area object file, linear object file, point object file) and cellular map files (terrain-type file, elevation file, road file). This feature has advantages and disadvantages. On the one hand, it leads to consistent maps -- the same fill types and colors are used across all maps for the same type of object. On the other hand, it could lead to an inconsistency and error -- the user might define a map variable category differently in the map file from the way it is specified in this header. In this case, the header definition would override the definition in the map file. Some additional consideration of how to handle the map-object filling, coloring, and labelling would be desirable.

This header file (s03cincb.h) includes variable type declarations, and the next one discussed (s03dintb.h) includes initializations for these same variables. The reason for setting up two similar header files -- one for declarations and one for initializations of essentially the same variables -- is that it was desired to use the declaration header file as an "include" file more than once. In this case, it cannot contain any initializations (a variable may be initialized only once). In most other cases in which global variables were initialized, the initializations either were included in a header file because it was "included" only once (i.e., in only one function file), or the initializations were included at the front of a function file (i.e., not using an "include" statement).

The contents of the file s03dintb.h could have been written at the beginning of a function file, except for the fact that it is very large. At the time when it was first defined, the Microsoft compiler did not allow very large function files. The declarations were "split out" of the function file, to reduce the file size.

Variable Definitions:

The following are the variables declared in s03cincb.h.

unsigned char maskmat[41][8]: array containing masking codes for 41 different fill patterns. This wide range of fill patterns was defined to facilitate the selection of fill patterns for maximum map understandability.

int maskmax: the number of different fill patterns (i.e., 41).

unsigned char mask[8]: a variable, used by the Microsoft floodfill function, containing a fill masking code.

unsigned char masknull[8]: a variable containing the masking codes (eight zeros) for a "null" fill.

char attributelabels[6][7][10]: symbolically, attributelabels[attributeno][attributecategory][label index], where the variable "attributeno" refers to map-object class and the variable "attributecategory" refers to map-object type within a map-object class. An array containing the labels for the six different classes of map objects defined above. The current version of the Scenarist allows for six different map-object classes (i.e., attributeno takes values from 0 to 5), seven different categories for each attribute (i.e., attributecategory takes values from 0 to 6) and labels of length up to nine characters.

int fillpattern[6][7]: symbolically, fillpattern[attributeno][attributecategory]. This array specifies the fill pattern to be used for each map-object type of each map-object class (i.e., for each attributecategory of each attributeno). The fill patterns are selected from the 41 fill patterns defined above. That is, this array contains numbers between 0 and 40.

int fillcolor[6][7]: symbolically, fillcolor[attributeno][attributecategory]. This array specifies the color to be used for each map-object type of each map-object class. There are sixteen different colors (defined in the program listing), and so this array contains numbers between 0 and 15.

int ncategories[6]: symbolically, ncategories[attributeno]. This array indicates the number of different map-object types for each of the six map-object classes (i.e., the number of different categories for each of the map-object classes, where map-object class is indexed using the variable "attributeno").

C. Header File s03dintb.h: Header File for Map Drawing Programs -- Initialization

Purpose: This header file includes initializations for the variables whose data types were declared in header file s03cincb.h. In addition, two additional variables are initialized.

Variable Definitions:

float geocontmaxmin[0]: this variable, used in the computation of continuous-variable cellular map quintiles, contains the maximum allowable value for the variable. The dimension was included, as discussed above, to allow for the future expansion of more than one continuous-variable cellular map (in addition to elevation). The index value 0 (of the dimension index) refers to elevation, in which case the value is 50,000. (As discussed above, this variable was defined as an arrayed global variable to enable storing and retrieval of the quintile data for several

continuous-variable cellular maps, but this feature has not yet been implemented.)

float geocontminmax[0]: this variable, used in the computation of continuous-variable map quintiles, contains the minimum allowable value for the variable. The index value 0 refers to elevation, in which case the value is -1000.

D. Header File s03eclip.h: CLIPS Header File

Purpose: This header file contains function declarations and variable declarations and initializations required by the CLIPS knowledge-based system.

Variable Definitions:

int CLIPS_suitable: a variable returned from the CLIPS system, that indicates whether a location (of a particular subitem) is suitable, according to the rules stored in the CLIPS knowledge base; 0 = not suitable, 1 = suitable.

The next ten constants are criterion levels used in the "Beqaa Valley" test case (project file proj0101.fil). Every application will involve different factors, and the rules will involve various criterion levels for those factors. Some of the criteria impose restrictions on the type of unit to which a rule applies, and some of the criteria relate to determination of the suitability of the location (of a unit satisfying certain criteria).

The constants defined below are examples of what are called "manifest constants." They are variables whose values are set at the beginning of a program run and are never changed. In effect, manifest constants are simply labels for numerical constants. They are used to increase the understandability of program code, since the code involves descriptive names rather than numbers.

#define R_GTYPE 6: In the Beqaa Valley test case, only units whose geographic type is 6 (i.e., small minor subarea subunits) are processed by the rules. The constant R_GTYPE is used to refer to this geographic type. (Note: In general, rules are applied only to subitems of geographic types 6 (small minor subarea subunits) or 7 (point subitems -- platforms or equipments) In the Beqaa Valley test case, no platforms or equipments were defined, so the rules applied only to subitems of geographic type 6. Rather than "hardwire" the code with the value "6," the "manifest constant" R_GTYPE is defined and permanently assigned the value 6 by means of the #define statement. In the rules, whenever it is desired to make a reference to geographic code 6, reference is made instead to the constant R_GTYPE. As mentioned above, this practice of defining and using manifest constants instead of numerical constants makes the rule easier to understand and the program code easier to maintain. It is probably easier for a programmer who is reading the Scenarist for the first time to remember that only units

of geogtype R_GTYPE are to be processed, than to remember that only units of geogtype 6 are to be processed. More importantly, if it is ever desired to modify this rule, the process of finding all references to this geogtype is greatly facilitated. All the programmer has to do is "search" the code for the variable R_GTYPE. Without the definition of this manifest constant, he would have to search the code for all occurrences of the number 6. Many of these may have nothing to do with geographic type.)

#define R_SIDE_B 1: In the Beqaa Valley test case, only units whose side is 1 (BLUE) are processed by the rules. The manifest constant R_SIDE_B, having value 1, denotes the BLUE side.

#define R_ECH_SEC 10: In the Beqaa Valley test case, the rules apply only to radar sections, which are of echelon 10. The constant R_ECH_SEC, having value 10, denotes the "section" echelon.

#define R_TYP_TPQ36 2: In the process of defining the units of the Beqaa Valley test case, TPQ36 radar sections were defined as sections of type 2. Any section whose type is 2 is "processed" by the rules. The constant R_TYP_TPQ36, having value 2, denotes section type 2 (i.e., denotes TPQ36 radar sections).

#define R_TYP_TPQ37 3: TPQ 37 radar sections are sections of type 3. The manifest constant R_TYP_TPQ37 refers to a TPQ37 radar section.

#define R_TYP_FAAR 5: FAAR radar sections are sections of type 5. (From this point on, we will drop detailed explanations of the rationale for the manifest constants. In every case, they were introduced to avoid making references to numbers in rules. We will simply specify what variable is being referred to (here, a section type), and the value of that variable (here, section type 5). We will not provide detailed descriptions of the rules involving these variables or criterion levels; detailed rule descriptions are presented in the Test Report.)

#define R_TYP_TER_WATR 3: The radars were not allowed to be in terrain types 3 (mountains), 4 (urban) or 5 (water). The rule specifies as "unsuitable" any terrain type of code exceeding the value "3". (The manifest constant label is misleading.)

#define R_DST_FEBA 7000.0: Radar sections are not allowed within 7000 meters of the FEBA. (That is, the variable is "distance to FEBA," and the reference value for that variable is 7000 meters.)

#define R_HORZ_LO -0.20: A radar's horizon angle (i.e., the angle from the radar to the horizon, looking toward the FEBA) cannot be less than -0.20 radians.

#define R_HORZ_HI 0.20: A radar's horizon angle cannot exceed .20 radians.

The following manifest constants are referred to in rules used in the TRAILBLAZER application (project file proj0201.fil). As above, we will not provide detailed descriptions of the rules here.

#define TB_GTYPE 6: Only subitems of geographic type 6 are processed by the rules.

#define TB_SIDE 1: Only subitems of side 1 (BLUE) are processed by the rules.

#define TB_ECHELON 11: TRAILBLAZER units are squads, whose echelon is 11.

#define TB_TYPE 1: TRAILBLAZER squads are squads of type 1.

#define TB_TERRAIN_TYPE 3: TRAILBLAZER squads are not allowed in terrain types of codes greater than 3 (mountains, urban, water).

#define TB_ACCESS 1: TRAILBLAZER squads must be located in accessible terrain (accessibility value "1".)

#define BT_LOS_TARGET 1: A line-of-sight condition must exist between a TRAILBLAZER squad and its objective (target). The LOS indicator has value "1" for LOS, "0" for no LOS.

#define TB_DIST_FRONT 2000.: The distance from a TRAILBLAZER squad to the division front may not be less than 2000 meters.

#define TB_IN_FORW_AREA 0: A TRAILBLAZER squad must be in the forward area of a division (code 0).

#define TB__LOS_OTHER 1: A TRAILBLAZER squad must have LOS with the four other TRAILBLAZER squads in the section.

#define TB_LOS_HQ 1: An LOS condition must exist from at least two TRAILBLAZER squads to the SIGINT Processing Platoon.

#define TB_DIST_OTHER 5000.: TRAILBLAZER squads may not be within 5000 meters of each other.

E. Header File s03fintf.h: Window, Menu, and Mouse Header File

Purpose: The header file s03fintf.h contains function and variable declarations for functions and variables used in functions concerned with the operation of windows, menus, and the mouse.

Variable Definitions:

#define call_mouse int86(0x33,&inreg,&outreg): the return from a call to the ROM-BIOS interrupt function int86(0x33,&inreg,&outreg) will be denoted by call_mouse.

#define EVENTMASK 0x7F: constant used in mouse calls.

#define SOFTWARE 0: constant used in mouse calls.

#define HARDWARE 1: constant used in mouse calls.

#define OFF 0: constant used in mouse calls.

#define ON 1: constant used in mouse calls.

char dir_array[80][13]: array for storage of directory names.

char inline[90]: character string to receive character test input.

int m[12], mcontrol: menu indicator variables (up to 12 boxes in window d). (Window "d" is the window to the upper-right of the screen, which contains the basic control functions of the Scenarist.)

int nboxes: the number of boxes in the current window d.

int boxvertsize: the number of pixels along the vertical axis of a menu box in the "d" window.

struct boxlabelstruct: a data structure type for storing the labels of a menu (specifically, the menu in window d). This data structure type contains the following variables:

char data[12][30]: each of the 12 rows of this array contains a menu label.

struct boxlabelstruct boxlabelmat[3]: a data structure array whose three elements contain the menu labels for the main Scenarist control menu, the "map functions" menu, and the "unit functions" menu.

struct boxlabelstruct boxlabel: a data structure containing the menu labels for the menu currently in window d.

unsigned char *face[6]: pointers to six fonts.

The remaining variables in the header file s03fintf.h are used to control the mouse. These variables were contained in code extracted from the graphics programming text(s) cited in the program listing. (These texts are listed in the References section at the end of this glossary.) Since this code was not developed under the contract, the variables will not be defined in this glossary.

F. Header File s03gdemo.h: EGA, DEMO, and CLIPS Conditional Compilation Header File

Purpose: The s03gdemo.h header file includes variable declarations and initializations to enable "conditional compilation" of the Scenarist. The Scenarist code is compiled differently, depending on whether it is to be used in a demonstration or operational mode, for an EGA or VGA monitor,

or with or without the CLIPS system. The variables of s03gdemo.h are the following.

Variable Definitions:

#define DEMO_TURNED_ON 1: This constant specifies whether the Scenarist is to be compiled in a demonstration mode or an operational mode. Define the constant to be 1 to compile a demonstration version of the Scenarist. Define the constant to be 0 for the operational version.

#define TEST_EGA_MODE 0: This constant specifies whether the program is to be compiled for an EGA or a VGA monitor (1 for EGA, 0 for VGA).

#define CLIPS_TURNED_ON 1: This constant specifies whether the program is to be compiled for use with CLIPS, or for use with C-language rule functions (1 for CLIPS, 0 for C-language rule functions).

G. Function File s03gmain.c: Main Program

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: The file contains the "main" function for the Scenarist program, and three functions required by the CLIPS system. The "main" program is the program where processing begins.

Functions Included in File:

main
usrfuncs
CLIPSval_0101
CLIPSval_0201

Definitions of Global Variables:

The following are definitions of the global (top-level, outside-function) variables in the s03gmain function file.

char filenamesfiln[13]: a variable used to store a file name.

The variables geodiscdim1, geodiscdim2, geodiscdim3, geocontdim1, geocontdim2, geocontdim3, nsides, nechelons, ntypes, areavertexdim, linvertexdim, and codedefined were defined in the principal header file, s03binca.h. They are initialized here (i.e., their values are set).

2. Function main

Declaration: void main(void)

Purpose: The function "main" is the main program. It registers fonts, calls the Scenarist introduction screen, calls the function to allow the user to select a project file, sets up the CLIPS system (if requested, by

setting CLIPS_TURNED_ON equal to 1), and either calls the demonstration code or passes control to the main menu (depending on whether the program is being used in a demonstration or operational mode).

Input Parameters: None.

Return Value: None.

Note: In the C programming language, a function either returns no value (in which case its type is "void"), or it returns a single value of its declared type. In addition, however, it may, during the course of its execution, cause changes to be made in the values of global variables. Changes are made to data structures and arrays by passing the address of the data structure or array to the function, using a pointer variable. In cases in which the function modifies the values of global variables, a list of the modified variables will be provided.

Global Variables Modified: unit0point, unit1point.

Files Modified: None.

Variable Definitions:

The variables used in main are as follows.

int number_of_fonts: the number of fonts to be registered.

int iret: a variable that indicates whether the "dribble.txt" file used by CLIPS was successfully opened.

3. Function void CLIPSval_0101

Declaration: void CLIPSval_0101(void)

Purpose: The function CLIPSval_0101 provides user feedback from the CLIPS system for the project defined by project file proj0101.fil.

Input Parameters: None.

Return Value: None.

Global Variables Modified: CLIPS_suitable.

Files Modified: None.

Variable Definitions:

char outline[40]: string variable used to store comments.

4. Function void CLIPSval_0201

Declaration: void CLIPSval_0201(void)

Purpose: The function CLIPSval_0201 provides user feedback from the CLIPS system for the project defined by project file proj0201.fil.

Input Parameters: None.

Return Value: None.

Global Variables Modified: CLIPS_suitable.

Files Modified: None.

Variable Definitions:

char outline[40]: string variable used to store comments.

H. Function File s03ipuu.c: Reposition Unit by User

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose of File: This file contains functions that enable the user to reposition a unit on the battlefield, and functions related to this function.

Functions Included in File:
_repositionunitbyuser
_repositionprogeny
_repositionfeba

The function _repositionunitbyuser repositions the unit. The function _repositionprogeny repositions all subordinate units of a repositioned unit. The function _repositionfeba repositions the Forward Edge of the Battle Area (FEBA).

Definitions of Global Variables:

None.

2. Function _repositionunitbyuser.

Declaration: void _repositionunitbyuser(void)

Purpose: This function enables a user to reposition a military unit.

Input Parameters: None.

Return Value: None.

Global Functions Modified: None.

Files Modified: None.

Variable Definitions:

char inputch: a variable used to store a character variable input by the user from the keyboard.

int i,j,ii,jj: Throughout the Scenarist code, the variables i, j, k, ii, jj, kk, iii, jjj, kkk are used as indices for iteration loops. For small iteration loops using these indices, no comment may be included in the code. For large iteration loops using these indices, comments are provided at the beginning and end of the loop to provide the programmer with a clear indication of the extent of the loop. In addition to comments, the Scenarist code is indented further each time a nested iteration loop begins. Since the use of i, j, k, ii, etc., is so extensive in the Scenarist, and since the use of these names is so common in programming, we shall, having defined these variables this first time, refrain from defining them again in other functions. Since they are local variables, they are in fact different variables in different functions. Their function (the index that controls an iteration loop) is, however, identical.

float x1: a floating-point variable input by the user over the keyboard.

int navenue: number of points on the avenue of approach.

int obj: indicator variable that indicates whether a unit has an objective. (Note: Throughout the Scenarist, much use is made of "indicator variables." An example that occurred earlier was the use of a "print" indicator variable. In every case, the value "0" indicates that no action is to be taken, and the value "1" indicates that an action is to be taken. Because this usage is consistent, we shall not repeat this definition of the indicator values every time an indicator variable occurs.

int defineobjective: indicator variable that indicates whether an objective is to be defined.

3. Function _repositionprogeny.

Declaration: void _repositionprogeny(int code[])

Purpose: This function repositions all of the subordinate units of a unit, whenever the unit is moved. It then repositions all of the subordinate units of those subordinate units, and so on, until all units whose positions were affected by the unit move are repositioned.

Input Parameters:
code: code of unit (stored in an integer array).

Return Value: None. (Note: Whenever the type of a function is "void," it does not return a value.)

Global Variables Modified: unit0point, unit1point. (Note: in every function involving processing on units, the unit data structures unit0 and unit1 are generally used, and the pointers unit0point and unit1point to these data structures are reset. As was discussed in the Introduction, these global variables are used as local variables. From this point on in this Glossary, we shall refrain from listing global variables that are used simply as local variables under the heading "Global Variables Modified," since it is not the objective to use these values outside of the function. Such variables were defined as global variables (as were unit0 and unit1) simply because they are used so often and it seemed to be more efficient (or at least, "cleaner") than redeclaring them as local variables over and over again in a variety of functions.)

Files Modified: the specific unit file being used in the run (i.e., the one named in the project file). The positions of all units subordinate to the moved unit are modified.

Variable Definitions:

float x, y: the location of a subunit corner, in standardized coordinates.

float xp, yp: the location of a subunit corner, in real (map) coordinates.

float *xppoint, *yppoint: pointers to xp and yp.

float x1p, x2p, x3p, x4p, y1p, y2p, y3p, y4p: (xip,yip) denotes the coordinates of the i-th corner of a unit, in map coordinates.

Note: the "corner" notation x, y, xp, yp, x1p, y1p,... recurs many times in other functions. The definition will not be repeated.

short stack[31][11]: a stack that includes the codes for all subunits identified so far as needing to be repositioned.

short stacksize: the number of items (codes) in the stack.

int imatch: an indicator variable that indicates whether the parentcode of a unit read from the specific unit file matches the code of the unit on the top of the stack.

int icode: an index used in an iteration loop to run through the values of a code[] array. (Note: this notation occurs in many other functions, and the definition will not be repeated.)

int index: the index (rank order) of a unit in the specific unit file.

short code0[13], code1[13], code2[13]: arrays used to store unit codes. (Note: this notation occurs in other functions, and will not be repeated.)

int unitexists: an indicator variable that indicates whether a unit of a specified code was located in a unit file (in this case, in the specific unit file).

4. Function _repositionfeba.

Declaration: void _repositionfeba(void)

Purpose: This function enables the user to reposition the FEBA.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: the "FEBA" file named in the project file being used in this run.

I. Function File s03jpsu.c: Reposition Subunits by User

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains the function that enables the user to reposition certain subunits of a unit.

Functions Included in File:
_repositionsubunitsbyuser

Definitions of Global Variables: None.

2. Function _repositionsubunitsbyuser

Declaration: void _repositionsubunitsbyuser(void)

Purpose: This function enables the user to reposition subunits within a unit. Subunits of geographic types 6 (small minor subarea units) and 7 (point subitems -- platforms and equipments) may be repositioned.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: either the generic unit file or the specific unit file of the run (i.e., named in the project file being used in the run).

Variable Definitions:

char defgenunit: indicator variable that indicates whether the unit to be modified is a generic unit.

char defspecunit: indicator variable that indicates whether the unit to be modified is a specific unit.

short i1: temporary variable used to store integer-variable input from the keyboard.

float x1, x2, x3: temporary variables used to store floating-point input from the keyboard.

float scalex, scaley: scaling variables previously used to convert all length units to meters. No longer used, since all data input are required to be in meters.

FILE *unitfilp: pointer to file (generic unit file or specific unit file) containing record of unit to be modified.

short nsubunits, npointsubunits, nminorsubareaabs, npoint: local variables used to store variables of the same name in the unit data structure.

char unitfiln[13]: name of the file containing record of unit to be modified.

long fileposition: location (in file) of record being modified.

char inputch: temporary variable used to hold the value of a character variable input by user from keyboard.

int placementcode: a code that signifies the nature of the placement of a subitem.

J. Function File s03kpsr.c: Reposition Subunits by Rule -- File 1

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions related to the positioning of subunits by rules. In addition to general functions that would be used in all applications, this file contains a number of functions that are specific to problem 0101 (which was concerned with the positioning of field artillery and air defense radars in the Beqaa Valley).

Functions Included in File:

_repositionsubunitsbyrule
_spiralsearch
_preprocessing0101
_action0101
_suitability0101
_terrain

_elevation
_distancetofeba
_horizonangle
_mapcellsizestdcoords
_clipssuitability0101

The functions _repositionsubunitsbyrule, _spiralsearch, and _mapcellsizestdcoords are general functions that will be used in any application. The function _repositionsubunitsbyrule is the "control" function that initiates and controls all of the processing required to apply rules to reposition the subitems of a unit. The function _spiralsearch implements a search for a suitable location, if the current location of a subitem (either that specified in the "canonical" location or that specified by the user) is unsuitable. The function _mapcellsizestdcoords is called at the beginning of the function _repositionsubunitsbyrule to determine the size of the cells of the terrain-type, elevation, and road availabilitiy maps, in relative coordinates. This information is used by the spiral search algorithm, to determine the search grid size. The remaining functions are specifically used in problem 0101, but similar functions are required by any other application.

To determine the suitability of a subitem's location, the function _repositionsubunitsbyrule calls the "suitability" function specified in the project file. For problem 0101, the project file is proj0101.fil. (For project xxxx, the project file is projxxxx.fil.) The suitability function is either a C-language function containing all of the logic defining the rules (_suitability0101), or an "interface" function (_clipssuitablility0101) that accesses the CLIPS expert system and uses rules stored in a CLIPS rule file specified in the project file (proj0101.fil).

Once the suitability of a subitem's location has been determined, an "action" may be taken. If the subitem's location is suitable, no action is taken; processing passes on to the next subitem. The rule processing is accomplished in two steps. First, local rules are processed. If, based on the local rules, a subitem's location is unsatisfactory, a spiral search is initiated for a suitable location (using the function _spiralsearch). Next, the global rules are processed. If, based on the global rules, a subitem's location is unsuitable, the "action" function specified in the project file (_action0101) is executed. Since in problem 0101 there are no global rules, the action function is null. It is included simply because the function _repositionsubunitsbyrules requires an action function for all problems.

The function _preprocessing0101 "precomputes" the values of constants, to make the processing of global rules more efficient. Since problem 0101 contains no global rules, this function is null.

The remaining functions (_terrain, _elevation, _distancetofeba, and _horizonangle) compute the values of factors used in the rules developed specifically for the problem 0101 application. Since these factors used

in problem 0101 are very basic and would be expected to be used in other problems, the problem-specific suffix 0101 has been omitted from their names.

Note: In the following, the terms "map coordinates" and "real coordinates" are used interchangeably, and the terms "unit coordinates," "standard coordinates," and "relative coordinates" are used interchangeably. Map coordinates are coordinate locations on a map coordinate system. Relative coordinates are coordinates of a location internal to a unit, expressed relative to the magnitude of the unit's side-to-side dimensions (the x-coordinate) and the magnitude of the unit's front-to-rear dimensions (the y-coordinate). The relative coordinates generally arise in two ways. First, they may be obtained from the unit specification, since all positions of subordinate items in a unit are specified in the generic and specific unit files in relative coordinates. Second, they may arise when a search is being made over a rectangular grid defined over a unit. Map coordinates are obtained from relative coordinates by a linear transformation.

As discussed earlier, the transformation is the linear transformation that transforms the quadrilateral-shaped unit on the map to the unit square in standard position. Facing forward, the unit's right-front corner is transformed to the point (0,0); the unit's left-front corner is transformed to the point (1,0); the unit's left-rear corner is transformed to the point (1,1); and the unit's right-rear corner is transformed to the point (0,1). The function _transfstdtoreal performs the transformation. Its calling parameters are the map coordinates of the unit's four corners. These coordinates are consistently denoted as x1p, y1p, x2p, y2p, x3p, y3p, x4p, y4p. Because this transformation is used by many Scenarist functions, these parameters occur as the calling parameters of many functions.

Definitions of Global Variables: None.

2. Function _repositionsubunitsbyrule

Declaration: void _repositionsubunitsbyrule(void)

Purpose: This function examines the positions of the subitems of a unit and determines, using the rules stored in a knowledge base, whether to reposition them. The present version processes nonarea subitems -- subitems of geographic type 6 or 7.

See the program listing for additional descriptive information about the purpose and functions of this function.

Input Parameters: None.

Return Value: None.

Global Variables Modified: CLIPS_suitable.

Files Modified: the specific unit file of the run (i.e., named in the project file being used in the run).

Variable Definitions:

int processitemstatus: a variable that indicates the stage of processing of a subitem (see detailed definition in program listing).

int item: index used for iteration loop over subitems.

int nitems: number of subitems (used both for items of geogtype 6 and geogtype 7).

int geogtype: geographic type.

float x, y: coordinates of a subitem, in standardized coordinates.

float xsuit, ysuit: coordinates of a suitable location.

float x1p, x2p,...: defined as before.

float x1, x2, x3: temporary variables used to store coordinates.

float scalex, scaley: scaling variables previously used to convert all length units to meters. No longer used, since all data input are required to be in meters.

int suitable: indicator variable that indicates if a location is suitable.

int foundsuitablelocation: indicator variable that indicates whether a suitable location has been found during the spiral search procedure.

int nsubunits: number of subunits.

int maxcells: maximum number of cells to examine in a spiral search.

int index: index over subitems.

int localglobal: switch variable used to control whether to process local rules (0) or local rules and global rules (1).

int iter: index over iteration loop used to process global rules.

int niter: maximum number of iterations allowed in processing global rules.

int numberitemsmoved: number of subitems moved in the most recent global-rule iteration loop.

int numberitemssuitable: number of subitems in suitable locations in the most recent global-rule iteration loop.

int action: indicator variable that indicates whether an item was moved in the call to an "action" function.

int placementcode: variable that indicates the method used to place an item. (Defined in detail in the program listing).

char line[80]: variable used to store a line of text.

unsigned int time1: counter variable used to control "flashes" on screen.

float xminn, xmaxx, yminn, ymaxx, xx1, yy1, xp, yp, *xppoint, *yppoint: variables used to determine the minimum and maximum coordinates (bounding rectangle coordinates) of a unit, in the determination of whether the unit is on the map (no rule processing is done for a unit if the unit is off the map, or for a subitem if the subitem is off the map).

3. Function spiralsearch

Declaration: int _spiralsearch(struct unit *unit0, int localglobal, int geogtype, int itemno, float xinit, float yinit, float *xsuit, float *ysuit, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p, int (*_suitabilityfunction)(struct unit*, int, int, int, float, float, float, float, float, float, float, float, float, float, int), int maxcells)

Purpose: This function conducts a search of the cells in a neighborhood of a subitem of a unit, to find a suitable location. The spiral search consists of a maximum of "ncycles" spiral loops around the starting point (the cell in which the subitem is located). The search is terminated if a suitable location is found, or after a total of "maxcells" cells has been examined. The value of maxcells is passed to the function as a calling parameter (argument). The value of ncycles is set in the function (equal to 3 in the current version). The spiral search is conducted in a clockwise direction. The suitability of a location is determined using the suitability function pointed to by the pointer *_suitabilityfunction. The search is carried out over the cell of a rectangular grid over the unit. The size of the grid cells is the minimum of the cells of the elevation cell map.

Input Parameters:

unit0: pointer to the unit in which the subitem is located.
localglobal: indicator that indicates whether the suitability function determines suitability using local rules only (value 0) or local and global rules (value 1).
geogtype: the geographic type of the subitem (either 6 or 7)
itemno: the index number of the subitem, in the unit data structure.
xinit,yinit: the coordinates of the starting location of the spiral search, in standard (relative) coordinates.
xsuit,ysuit: pointers to the variables in which the coordinates of a suitable location (if found) will be stored.
x1p,y1p,...: as usual, the (map) coordinates of the unit corners.

_suitabilityfunction: pointer to the suitability function to be used to determine the location suitability.
maxcells: the maximum number of cells to examine in the search for a suitable location.

Note: the value of the global variables xgridsize and ygridsize are also used as input parameters. These values are fixed as long as the same cellular maps are used. For this reason, they are passed as a global variable rather than as a function argument.

Return Value: 1 if suitable location is found, 0 otherwise.

Global Variables Modified: xsuit, ysuit (the location of a suitable location, if one is found).

Variable Definitions:

int cellcount: a counter used to store the number of cells already examined by the function.

int direction: the current direction of the spiral search (+1 = increasing x or y; -1 = decreasing x or y).

int icycles: index for the iteration loop specifying the current cycle of the spiral search.

int ncycles: the maximum number of cycles to execute in the spiral search procedure.

int istep: index for the iteration loop that specifies whether the spiral search is currently proceeding along the y axis (1) or the x axis (2).

int ixincr: x-direction cell increment (i.e., the number of cells to move in the x-direction, in the current cycle and step). Either 0 (cell) or 1 (cell).

int iyincr: y-direction cell increment.

float x,y: relative coordinates of current search location.

float xincr: x-direction search increment, in relative units (i.e., the width of a unit grid cell specified as a proportion of the unit width).

float yincr: y-direction search increment.

int spiralsuitable: indicator variable specifying the suitability of a location (0 = unsuitable, 1 = suitable).

int processitemstatus: a variable that indicates the stage of processing of a subitem (see detailed definition in program listing).

4. Function _preprocessing0101

Declaration: void _preprocessing0101(struct unit *unit0, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: The function _preprocessingxxxx computes and stores the values of certain constants needed during the spiral search procedure. Its use reduces the total amount of processing in problems that have global constraints, since in such problems certain constants are accessed many times. Since problem 0101 involves no global constraints, no preprocessing is done. The calling parameters listed as arguments of this function are those that would typically be needed to do preprocessing in a problem that did have global constraints. They are not defined here, since this function is null (i.e., does no processing); they are defined, however, for the function _preprocessing0201.

Input Parameters: Not defined (for reason given above).

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variabale Definitions: None.

5. Function _suitability0101

Declaration: int _suitability0101(struct unit *unit0, int localglobal, int geogtype, int itemno, float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p, int processitemstatus)

Purpose: This function determines the suitability of a location, according to certain suitability criteria ("rules"). The rules are separated into two categories -- local rules and global rules (see Final Report for a discussion of the two types of rules). The input parameter "localglobal" specifies which type of rules is to be processed by the function.

The current version of the Scenarist is designed to specify the rules either in a C-language function such as this one, or in a rule file of the CLIPS knowledge-based system. The purpose of this dual capability was to verify that the CLIPS system was operating correctly in the test cases (problems 01 dealing with placement of air defense and field artillery radars in the Beqaa Valley and problem 02 dealing with the placement of TRAILBLAZER units in an area near Spearfish, SD). In general, the Scenarist user may set the system up in either way (i.e., using either the C-language rule functions or CLIPS rule files). For simple applications, it would be easier for a C programmer to use the C-language functions than CLIPS.

The input parameter "processitemstatus" is not used by this function. It is used by the CLIPS interface function _clipssuitability0101. It is included as an input parameter here simply so that the number and types of the calling parameters to _suitability0101 and _clipssuitability0101 are exactly the same. The reason for keeping them the same is that which of these functions is to be used in a run is determined by a "function pointer." In order to specify the choice of suitability function by means of a function pointer, it is necessary that the calling parameters of the possible functions to be pointed to are as specified in the declaration of the function pointer. They are hence identical for all functions pointed to by the function pointer.

The following suitability criteria are used.

Local constraints (i.e., rules for which localglobal is 0):
A location is unsuitable if:

1. For radar units (i.e., units of side 1 (BLUE), echelon 10 (section), and type 2 (TPQ36 radar), type 3 (TPQ37 radar), or type 5 (FA radar)):
   a. The terrain type is mountains, urban, or water.
   b. The distance to the FEBA is less than 7 kilometers.
   c. The horizon angle is greater than .2 radians in absolute value.

Note that other units are not constrained.

Global constraints (i.e., rules for which localglobal is 1):
None

Terrain types:
0: no data
1: plains
2: hills
3: woods
4: mountains
5: urban
6: water

Input Parameters:
unit0: pointer to the unit to which the subitem whose locational suitability is being assessed belongs.
localglobal: indicator variable that indicates whether the function is to apply only local rules or both local rules and global rules.
geogtype: the geographic type of the subitem whose locational suitability is being assessed.

itemno: the index number of the subitem in the unit data structure.
x,y: the map coordinates of the subitem.
x1p,y1p,...: the map coordinates of the unit corners.
processitemstatus: not used (but must be included as a function argument, for the reasons given above).

Return Value: 0 if location is unsuitable (according to the rules specified in the function), 1 if the location is suitable.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int terraintype: terrain type.

int csuitable: suitability indicator variable (return value).

Note: The variable name "csuitable" is used (rather than the more descriptive name "suitable") to avoid confusion with variable "suitable" in the calling function (_repositionsubunitsbyrule).

int side, echelon, type: unit attributes (defined earlier).

float distance: distance to FEBA.

float angle: horizon angle.

float xx1,yy1: map coordinates of location (used to refer to map location code reused from another function).

float xp,yp: map coordinates of location (used to refer to map location in call to _transfstdtoreal, to keep calling parameter list for _transfstdtoreal the same in all functions).

float *xppoint,*yppoint: pointer variables to xp,yp (Note: C notation is a little complicated. The pointer variables are xppoint and yppoint, not *xppoint and *yppoint).

int xi,yi: map coordinates of location in viewport (pixel) coordinates.

unsigned int time1,time2: time counters.

6. Function _action0101

Declaration: int _action0101(struct unit *unit0, int geogtype, int item, float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: An action function _actionxxxx specifies what action is to be taken if an item's location is unsuitable because of global constraints. Since problem 0101 has no global constraints, this function is null, and the arguments for _action0101 will not be defined. They are the same as the arguments for _action0201, which is described later.

Input Parameters: Not defined (for reason stated above).

Return Value: Not defined (for reason stated above).

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

7. Function _terrain

Declaration: int _terrain(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function returns the terrain type (more specifically, a terrain-type code) for the location (x,y) (in relative coordinates). This information is stored in the matrix geodisc[0][row][column].

Input Parameters:

x,y: standard coordinates of location for which terrain type is desired.
x1p,y1p,...: map coordinates of unit corners.

Return Value: Terrain-type code (integer from 0 to 5, where 0 denotes no data, 1 denotes plains, 2 denotes hills, 3 denotes woods, 4 denotes mountains, 5 denotes urban, and 6 denotes water.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int irow,icol: row and column (second and third dimensions) of the matrix geodisc[][][].

int terntype: terrain type (return value).

float xreal,yreal: map coordinates corresponding to relative coordinates x,y.

float xmin: minimum x-coordinate of map.

float ymax: maximum y-coordinate of map.

cellwidth: width of map cell (in meters).

8. Function _elevation

Declaration: float _elevation(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function returns the elevation for the location (x,y) (in relative coordinates). This information is stored in the matrix geocont[0][row][column]. To save storage space, the elevation is stored in integer representation in the matrix geocont, rather than floating-point representation. The return value is, however, floating point.

Input Parameters:
x,y: standard coordinates of location for which elevation is desired.
x1p,y1p,...: map coordinates of unit corners.

Return Value: Elevation, in meters.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int irow,icol: row and column (second and third dimensions) of the matrix geocont[][][].

int elevdisc: integral representation of elevation.

float xreal,yreal: map coordinates corresponding to relative coordinates x,y.

float elevcont: floating point representation of elevation (return value).

float xmin: minimum x-coordinate of map.

float ymax: maximum y-coordinate of map.

cellwidth: width of map cell (in meters).

9. Function distancetofeba

Declaration: float _distancetofeba(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function returns the distance (in meters) from the point (x,y) (in relative coordinates) to the FEBA.

Input Parameters:
x,y: standard coordinates of location for which the distance to FEBA is desired.
x1p,y1p,...: map coordinates of unit corners.

Return Value: Distance, in meters.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

float xfeba,yfeba: coordinates of a point on the FEBA.

float distii: distance from (x,y) to a point on the FEBA.

float dist: minimum value of the values computed for distii (return value).

float x1,y1: real coordinates corresponding to relative coordinates x,y.

float xp,yp: real coordinates corresponding to relative coordinates x,y.

Note: two sets of values (x1,y1) and (xp,yp) were used to denote the real coordinates corresponding to the relative coordinates x,y for the same reasons as discussed earlier, i.e., to permit identical arguments for all calls to _transfstdtoreal and to enable code reuse. This use of two sets of values for the real coordinates occurs a number of times, and will no longer be explained.

10. Function horizonangle

Declaration: float _horizonangle(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function returns the horizon elevation angle looking from the point (x,y) (in relative coordinates) toward the front. The computation proceeds ("looks") as far as the front.

Input Parameters:

x,y: standard coordinates of location for which the horizon angle is desired.
x1p,y1p,...: map coordinates of unit corners.

Return Value: Horizon angle, in radians.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

float stepsize: The horizon angle is computed by first computing the elevation angle from the point (x,y) to each of a sequence of points along the line toward the unit front, and taking the maximum of these angles. The stepsize is the distance between these points.

float elevangle: the elevation angle to a point on the line to the unit front.

float elevanglemax: 1.57 (= p1/2) radians.

float elevanglemin: -1.57 (= -pi/2) radians.

float yistd: the distance in relative units from the point (x,y) to a point on the line to the front. Since this line parallels the unit's sides, this distance is simply the absolute value of the difference in the y coordinates.

float elevy: elevation at the point (x,y)

float elevyi: elevation at a point along the line to the front.

float dist: the real (map) distance from the point (x,y) to a point on the line to the front.

float tan: the tangent of an elevation angle.

float horizangle: the horizon angle (return value).

int nsteps: the number of steps from the point (x,y) to the front.

int missing: missing data indicator variable.

11. Function _mapcellsizestdcoords

Declaration: float _mapcellsizestdcoords(int maptype, int mapindex, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function returns a map cellsize, in relative (unit) coordinates. It converts a map cellsize from meters to relative (standard, unit) units. It does this by transforming the unit to map coordinates, calculating the length of each side, and dividing the map cell size by the maximum side length.

Input Parameters:

maptype, mapindex: map type (0 = discrete, 1 = continuous), map index (for discrete, 0=terrain type, 1 = road availability; for continuous, 0 = elevation)
x1p,y1p,...: coordinates of the unit's corners.

Return Value: The "cellsize" of a map, in relative (unit, standard) coordinates (i.e., expressed as a proportion of the length of the unit's minimum side).

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

float unitsize: distance along a unit side (in meters)

float mapcellsize: elevation map cell size.

float nmapcellsonside: the number of map cells along a unit side.

float mapcellsizeiside: map cell size in relative coordinates, for one of the unit's sides.

float mapcellsizestd: the elevation map cell size, in relative units (return value).

float x1,y1,x2,y2: the coordinates of the endpoints of a unit side, in either relative coordinates or in map coordinates.

float xp,yp: a set of map coordinates.

float *xppoint,*yppoint: pointers to xp, yp.

int iside: index for iteration loop over unit's four sides.

12. Function _clipssuitability0101

Declaration: int _clipssuitability0101(struct unit *unit0, int localglobal, int geogtype, int itemno, float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p, int processitemstatus)

Purpose: This function determines the suitability of a location, according to certain suitability criteria ("rules"). The rules are separated into two categories -- local rules and global rules (see Final Report for a discussion of the two types of rules). The input parameter "localglobal" specifies which type of rules is to be processed by the function.

As discussed in the description of the function _suitability0101, the current version of the Scenarist is designed to specify the rules either in a C-language function, or in a rule file of the CLIPS knowledge-based system. This function is the interface to the CLIPS system.

Input Parameters (the same as for _suitability0101):

unit0: pointer to the unit to which the subitem whose locational suitability is being assessed belongs.
localglobal: indicator variable that indicates whether the function is to apply only local rules or both local rules and global rules.
geogtype: the geographic type of the subitem whose locational suitability is being assessed.
itemno: the index number of the subitem in the unit data structure.
x,y: the map coordinates of the subitem.
x1p,y1p,...: the map coordinates of the unit corners.
processitemstatus: not used (but must be included as a function argument, for the reasons given above).

Return Value: 0 if location is unsuitable (according to the rules specified in the CLIPS knowledge base), 1 if the location is suitable. The return value is stored in a global variable, CLIPS_suitable, for use by the function _repositionunitsbyrule to present processing information to the user.

Global Variables Modified: CLIPS_suitable.

Files Modified: None.

Variable Definitions:

Note: the following variables are the same as those used in _suitability0101. The variable csuitable was the return value for _suitability0101. It is not used here (CLIPS_suitable is the return value for _clipssuitability0101).

int terraintype: terrain type.

int csuitable: null.

int side, echelon, type: unit attributes (defined earlier).

float distance: distance to FEBA.

float angle: horizon angle.

float xx1,yy1: map coordinates of location (used to refer to map location code reused from another function).

float xp,yp: map coordinates of location (used to refer to map location in call to _transfstdtoreal, to keep calling parameter list for _transfstdtoreal the same in all functions).

float *xppoint,*yppoint: pointer variables to xp,yp.

int xi,yi: map coordinates of location in viewport (pixel) coordinates.

K. Function File s03kpsr2.c: Reposition Subunits by Rule -- File 2

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions related to the positioning of subunits by rules, for problem 0201. Problem 0201 was concerned with the positioning of TRAILBLAZER units.

Functions Included in File:

_preprocessing0201
_createaccessibilitymap
_accessibility
_lostotarget

```
_LOS
_road
_slopetorearcell
_lostootherunits
_lostoheadquarters
_disttootherunits
_distancetofront
_inforwardarea
_action0201
_suitability0201
_clipssuitability0201
```

The functions required by _repositionsubunitsbyrules are _action0201 and either one of _suitability0201 or _clipssuitability0201. These functions are named in the project file for project 0201, proj0201.fil.

The other functions listed above are used to determine the numerical values of factors used in rules. The function _preprocessing precomputes the values of constants used in the processing of global rules. This function calls the function _createaccessibilitymap to define a map ("TRAILBLAZER accessibility map") that specifies the accessibility of a map cell by a TRAILBLAZER unit. The cellsize of this map is the minimum of the cellsizes of the terrain-type, elevation, and road availability maps. The function _createaccessibilitymap calls the function _slopetorearcell.

The remaining functions (_accessibility, _lostotarget, _LOS, _road, _lostootherunits, _lostoheadquarters, _disttootherunits, _distancetofront, _inforwardarea) are used to determine the numerical values of factors used in the rules during "on-line" processing of the rules (i.e., they do not involve precomputation). The function _road is not presently used. It was used previously when TRAILBLAZER accessibility was computed for the cells of a grid defined over a unit, instead of for the cells of a map (i.e., the TRAILBLAZER accessibility map). It has been left in the listing for potential future use.

As was the case for the functions included in file s02kprs.c, the problem-specific suffix 0201 has been left off functions that may be reusable in other applications.

Definitions of Global Variables: None.

2. Function _preprocessing0201

Declaration: void _preprocessing0201(struct unit *unit0, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function computes the values of constants that are needed for checking of the global rules of problem 0201. In this problem, it is desirable to precompute the accessibility of each cell of a map by TRAILBLAZER units, before beginning rule processing. These values are computed by the function _createaccessibilitymap.

Input Parameters:

unit0: pointer to unit for which the accessibility will be determined for every cell of the unit grid.
x1p,y1p,...: map coordinates of unit's corners.

Return Value: None.

Global Variables Modified: the array geodisc[2][][] (which stores the TRAILBLAZER accessibility map).

Files Modified: None.

Variable Definitions: None.

3. Function _createaccessibilitymap

Declaration: void _createaccessibilitymap(void)

Purpose: This function determines the accessibility of each cell of a map by TRAILBLAZER units. The cellsize of this TRAILBLAZER accessibility map is the minimum of the cellsizes of the terrain-type, elevation, and road availability maps. The value of the element is zero if the accessibility of the cell cannot be determined (e.g., no road or terrain-type data are available), 1 if the cell is not accessible, and 2 if the cell is not accessible. A cell is accessible if:

1. It has a road (i.e., the value of the function _road is 2 for the cell location).
2. Its terrain type is "plains" (i.e., the value of _terrain is 1).
3. Its terrain type is woods or hills or mountains and the slope from a neighboring accessible cell in the next-adjacent row of cells to the rear (top of map) is between +30 degrees and -30 degrees (.5236 radians). This slope is computed by the function _slopetorearcell.

It is not accessible in any other event (e.g., if its terrain type is water).

Determination of accessibility starts at the top of the map and proceeds to the bottom of the map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: geodisc[2][][].

Files Modified: None.

Variable Definitions:

int irow,icol: row and column indices of the array geodisc[2][][].

int terraintype: the terrain-type code for a location.

int roadpresent: an indicator variable that indicates whether a road is present at a location.

float x,y: the relative coordinates of a location.

float slope: the slope between two neighboring cells of the map.

char line[20]: array used to store strings for output to the screen, to inform user of progress in computing the accessibility map.

4. Function _accessibility

Declaration: int _accessibility(float x, float y)

Purpose: This function determines the accessibility of the point (x,y) (in relative coordinates) by a TRAILBLAZER squad/team. The values x and y (between 0 and 1) are converted to column and row values by multiplying by the number of columns and rows of the matrix, nxgrid and nygrid.

Input Parameters:

x,y: location (relative coordinates) for which it is desired to know the accessibility.

Return Value: The TRAILBLAZER accessibility value, as stored in the TRAILBLAZER accessibility map stored in geodisc[2][0][0].

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int irow,icol: row and column indices.

int access: accessibility (return value).

5. Function _lostotarget

Declaration: int _lostotarget(float x, float y, struct unit *unit0, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function determines whether the point (x,y) in relative (unit) coordinates has line-of-sight (LOS) to at least two of five points along the target (objective) of a unit, unit0 (a TRAILBLAZER section, in problem 0201). The five points are equally spaced along the nearest side of the bounding rectangle defining the objective. The returned value is 0 if there is no data, 1 if this condition is not satisfied, and 2 if the condition is satisfied.

Input Parameters:

x,y: the point from which LOS condition to unit objective will be determined.
unit0: pointer to the unit data structure.
x1p,y1p,...: coordinates of unit's corners.

Return Value: LOS indicator variable (0, 1, or 2), as defined above.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int los: LOS indicator variable (return value).

int losipoint: LOS indicator variable from (x,y) to a point on the bounding rectangle of the objective.

int los0count,los1count,los2count: counts of the number of boundary points having LOS indicator values of 0, 1, or 2, respectively.

float xbr1,ybr1,xbr2,ybr2: coordinates of corners of bounding rectangle of unit objective.

float xdelta: difference in x coordinates between points on bounding rectangle for which LOS condition will be checked.

float ydelta: difference in y coordinates between points on bounding rectangle for which LOS condition will be checked.

float xobj,yobj: coordinates of a point along the near side of the bounding rectangle (to which LOS condition will be checked).

int ipoint: index for iteration loop over points for which LOS condition will be checked.

6. Function _LOS

Declaration: int _LOS(float xx1, float yy1, float xx2, float yy2, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function determines whether a line-of-sight (LOS) condition exists (i.e., whether "there is line-of-sight") between the map locations that correspond to the points (xx1,yy1) and (xx2,yy2), specified in relative coordinates.

If elevation data are available for all intermediate points (between (xx1,yy1) and (xx2,yy2)), a value of 2 is returned if LOS exist, and a value of 1 is returned if LOS does not exist. If an elevation "nodata" code is

encountered, a value of 0 (nodata) is returned if LOS existed for all intermediate points having data. Otherwise, a value of 1 (no-LOS) is returned.

Input Parameters:

xx1,yy1,xx2,yy2: coordinates (relative coordinates) of the two points between which LOS will be determined.
x1p,y1p,...: map coordinates of unit corners.

Return Value: LOS indicator variable, as defined above (return value).

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int los: LOS indicator value.

int nsteps: number of elevation map cell steps along the line from (xx1,yy1) to (xx2,yy2), rounded up.

int missing: missing value indicator variable.

float stepsize: elevation map cell size.

float diststd: distance from (xx1,yy1) to (xx2,yy2), in relative coordinate system.

float elev1: elevation at first input point (xx1,yy1).

float elev2: elevation at second input point (xx2,yy2).

float elevi: elevation at intermediate point (xi,yi).

float xdelta: The LOS condition is checked at a number of points along the line between (xx1,yy1) and (xx2,yy2). xdelta is the difference in the x coordinates of two adjacent points on this line.

float ydelta: Same as xdelta, but for the y coordinate.

float xi,yi: coordinates of an intermediate point on the line between (xx1,yy1) and (xx2,yy2).

7. Function _road

Declaration: int _road(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function returns the road availability of unit cell located at relative coordinates x,y. The road availability is stored in geodisc[1][][]. This function is not presently used.

Input Parameters:

x,y: the (relative) coordinates of the location for which road availability is desired.

x1p,y1p,...: the map coordinates of the unit's corners.

Return Value: road availability (value stored in geodisc[1][][].

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int icol,irow: indices of the second and third dimensions of the matrix geodisc[][][].

int roadav: road availability (return value).

float xreal,yreal: the map coordinates corresponding to x,y.

float xmin,ymax: the minimum x-coordinate and maximum y-coordinate of the road (availability) map.

float cellwidth: the cell width of the road map.

8. Function slopetorearcell

Declaration: float _slopetorearcell(int row, int col)

Purpose: This function computes the elevation angle (slope) from a cell of the road map to the cell one cell toward the rear (top of map). This data item is needed by the function _createaccessibilitymap.

Input Parameters:

row,col: row and column indices of the map cell for which the slope to the rear cell is desired.

Return Value: Slope to rear cell. The value 1.58 is used to indicate that the slope cannot be computed because of missing data.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

float angle: the slope from the point (x,y) to the point one cell width to the rear of this point.

float anglemax: 1.57 (= pi/2) radians.

float anglemin: -1.57 (=-pi/2) radians.

float elev1: elevation of point (x,y).

float elev2: elevation of point one cell width to rear of (x,y).

dist: (real) distance between the point (x,y) and the point one cell width to the rear.

tan: the tangent of the slope from (x,y) to the point one cell width to the rear.

float x1,x1: map coordinates corresponding to (x,y).

float x2,y2: map coordinates corresponding to (x,y).

float y22: variable used to store value of y2.

float xp,yp: map coordinates of a point (input to function _transfstdtoreal).

float *xppoint,*yppoint: pointers to xp,yp.

8. Function _lostootherunits

Declaration: int _lostootherunits(float x, float y, struct unit *unit0, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function determines whether there is line-of-sight from the point (x,y) (the location of a TRAILBLAZER squad/team of the TRAILBLAZER section unit0) to the locations of at least two of the (four other) TRAILBLAZER squads of the same TRAILBLAZER section.

Procedure: Scan all of the subitems of the unit. If a TRAILBLAZER squad/team is found, check for LOS from the point (x,y). If the number of points having LOS is three or more, return a value of 2 (one of the points is the unit itself).

Input Parameters:

x,y: relative coordinates of the location from which LOS determination is desired.
unit0: pointer to unit data structure.
x1p,y1p,...: map coordinates of unit's corners.

Return Value: LOS indicator variable (0 = no data, 1 = no LOS to 3 or more units, 2 = LOS to at least two other units).

int item: index of iteration loop over subitems (of geogtype 6, the geogtype of the TRAILBLAZER squad/teams) of unit.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int nitems: number of items of geogtype 6 in the unit.

int los: LOS indicator value (return value).

int los0count, los1count, los2count: counters that count the number of times los has the values 0, 1, and 2, respectively.

int echelon, type: the echelon and type of a subitem of geogtype 6. (TRAILBLAZER squad/teams are of echelon 11, type 1; only units of this echelon and type are considered.)

float xother,yother: relative coordinates of location of another TRAILBLAZER squad/team.

9. Function _lostoheadquarters

Declaration: int _lostoheadquarters(struct unit *unit0, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function determines whether there is line-of-sight from the locations of at least two TRAILBLAZER squad/teams to the symbol location point of the TRAILBLAZER section (unit0) containing the units (the symbol location point is identified in the SIGINT Processing Platoon (SPP) containing the TRAILBLAZER section).

Procedure: Denote the symbol location point as (x,y). Scan all of the subitems of the unit. If a TRAILBLAZER squad/team is found, check for LOS from the point (x,y). If the number of points having LOS is two or more, return a value of 2.

Input Parameters:

unit0: pointer to the unit data structure.
x1p,y1p,...: map coordinates of unit's corners.

Return Value: LOS indicator variable (0 = no data, 1 = no LOS to 3 or more units, 2 = LOS to at least two other units).

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int item: index of iteration loop over subitems (of geogtype 6, the geogtype of the TRAILBLAZER squad/teams) of unit.

int nitems: number of items of geogtype 6 in the unit.

int los: LOS indicator value (return value).

int los0count, los1count, los2count: counters that count the number of times los has the values 0, 1, and 2, respectively.

int echelon, type: the echelon and type of a subitem of geogtype 6. (TRAILBLAZER squad/teams are of echelon 11, type 1; only units of this echelon and type are considered.)

int parentunitexists: indicator variable that indicates whether unit's parentunit was found in the specific unit file.

float x,y: relative coordinates of the symbol location point of the TRAILBLAZER section.

float xother,yother: relative coordinates of location of another TRAILBLAZER squad/team.

10. Function disttootherunits

Declaration: float _disttootherunits(int subitemnumber, struct unit *unit0, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function computes the minimum distance from the TRAILBLAZER squad/team number "subitemnumber" of the TRAILBLAZER section unit0 to the locations of the other (four) TRAILBLAZER squads of the same TRAILBLAZER section.

Procedure: Denote the location point of the subunit number "subitemnumber" as (x,y). Scan all of the subitems of the unit. If a TRAILBLAZER squad/team is found, compute the distance from the point (x,y). Return the minimum of these distances.

Input Parameters:

subitemnumber: the number of the TRAILBLAZER squad/team for which the minimum distance to the other squad/teams will be computed.
unit0: pointer to the unit data structure.
x1p,y1p,...: coordinates of the unit's corners.

Return Value: the minimum distance, as defined in "Purpose."

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int item: index for the iteration loop over subitems of geogtype 6.

int nitems: the number of subitems of geogtype 6.

int echelon,number,type: the echelon, number, and type of a subitem of geogtype 6.

float x,y: the relative coordinates of the location of the TRAILBLAZER team numbered subitemnumber.

float xitem,yitem: the relative coordinates of the location of another TRAILBLAZER team.

float distitem: the distance between the TRAILBLAZER team numbered subitemnumber and another TRAILBLAZER team.

float dist: the minimum value of all of the "distitem" values (return value).

int founditemno: indicator variable that indicates if the subitem numbered subitemnumber was found in the unit.

int firstimethrough: indicator variable that indicates if the current pass through the "item" loop is the first one, in which case the value of dist will be set equal to the value of distitem.

float x1,y1: map coordinates of the subitem numbered subitemnumber.

float x2,y2: map coordinates of another subitem.

float xp,yp: map coordinates of a location determined by a call to _transfstdtoreal.

float *xppoint,*yppoint: pointers to xp,yp.

11. Function _distancetofront

Declaration: float _distancetofront(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function computes distance (in meters) from point (x,y) (in relative coordinates) to the unit front.

Input Parameters:

x,y: relative coordinates of a location (specifically, the location of a TRAILBLAZER team).

x1p,y1p,...: map coordinates of a unit's corners.

Return Value: the distance to the front.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

float x1,y1: the map coordinates corresponding to x,y.

float x2,y2: the map coordinates of the point on the front corresponding to the relative coordinates x,0.

float distance: the real distance between the points whose relative coordinates are (x,y) and (x,0).

float xp,yp: the map coordinates returned by a call to the function _transfstdtoreal

float *xppoint,*yppoint: pointers to xp,yp.

12. Function _inforwardarea

Declaration: int _inforwardarea(float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function determines whether the point (x,y) (in relative coordinates) is in the front half of the unit. Return a 0 if no, a 1 if yes.

Input Parameters:

x,y: the relative coordinates of a point (specifically, the location of a TRAILBLAZER team).
x1p,y1p,...: map coordinates of the unit's corners.

Return Value: 0 if the location is not in the front half of the unit, 1 if it is.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int infront: indicator value that specifies whether a location is in the front half of a unit (return value).

13. Function _action0201

Declaration: int _action0201(struct unit *unit0, int geogtype, int item, float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p)

Purpose: This function attempts to move a subitem to a neighboring cell that is of higher elevation than the current cell.

Procedure: The procedure is as follows. The eight neighboring cells are examined, and checked for elevation. Of the neighboring cells that have equal or higher elevation, the one having the highest elevation is selected. If there is no cell having equal or higher elevation, a value of 0 is returned. If a cell of equal or higher elevation is found, the subitem is relocated to that cell, and a value of 1 is returned. If several cells have the same highest elevation the last such one encountered is selected.

Input Parameters:

unit0: pointer to the unit data structure.
geogtype: the geogtype of the subitem.
item: the subitem's item number
x,y: the relative coordinates of the subitem's location.
x1p,y1p,...: the map coordinates of the unit's corners.

Return Value: 1 if subitem is relocated (i.e., action is taken), 0 otherwise.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int action: indicator value that indicates whether the subitem is relocated (return value).

int irow,icol: row and column indices for the set of cells surrounding the cell in which the subitem is located.

int foundhigher: indicator variable that indicates whether a neighboring cell was found with elevation at least as high as the cell in which the subitem is located.

int index: subitem index in the unit data structure.

int nsubunits: number of subitems in the unit.

int nitems: number of subitems of the same geogtype as the subitem.

float stepsize: the cell size of the elevation cell map.

float elevxy: the elevation of the point (x,y).

float elevi: the elevation of a cell neighboring the cell in which the point (x,y) is located.

float elevmax: the maximum elevation of all cells examined so far.

float xnew,ynew: the relative coordinates of a point in a cell neighboring the cell in which the point (x,y) is located.

float xsav,ysav: the relative coordinates of the point in the neighboring cell having highest elevation of all neighboring cells whose elevation is at least as high as the point (x,y).

float xsuit,ysuit: variables in which the values of xsav and ysav are put, in order to reuse earlier code.

float x1,x2: abbreviated variable names for xsuit,ysuit.

float x3,scalex,scaley: not used (in code that was left over from an earlier version).

14. Function _suitability0201

Declaration: int _suitability0201(struct unit *unit0, int localglobal, int geogtype, int itemno, float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p, int processitemstatus)

Purpose: This function determines the suitability of a location.

Procedure: A location is assumed suitable unless otherwise determined. The following suitability criteria are used.

Local rules (i.e., rules involving local constraints, for which localglobal is 0):
A location is unsuitable if:
1. For any unit:
   a. The terrain type is water, mountain, or urban
2. For TRAILBLAZER squad/teams (i.e., units of side 1 (BLUE), echelon 11 (TRAILBLAZER squad/teams) and type 1 (TBMCS):
   a. The location is not accessible
   b. The location does not have LOS to the objective defined for the parentunit (i.e., the TRAILBLAZER section containing the squads).
   c. The distance from the location to the unit front is less than 2000 m.
   d. The location is not in the forward area of the unit.

Otherwise, the location is suitable.

Global rules (i.e., rules involving global constraints, for which localglobal is 1):
A location is unsuitable if:
1. For TRAILBLAZER squad/teams:
   a. The squad/team has LOS to fewer than two other

TRAILBLAZER squad/teams in the TRAILBLAZER section containing the squad/teams.
b. Fewer than two squad/teams of the TRAILBLAZER section containing this subunit have LOS to the symbol location point of the TRAILBLAZER section containing the squad/teams.
c. The minimum distance between TRAILBLAZER squad/teams is less than 5000m.

Otherwise, the location is suitable.

Terrain types:
0: no data
1: plains
2: hills
3: woods
4: mountains
5: urban
6: water

Note: The local variable csuitable is used (rather than "suitable") to avoid confusion with the variable suitable in calling function. Note also that the function _clipssuitable0201 (which parallels this one in function and structure) stores the return value in a global variable CLIPS_suitable, which is used by _repositionsubunitsbyrule. Since the value of csuitable is never used by _repositionsubunitsbyrule, it is simply a local variable, not a global one.

Input Parameters:
unit0: pointer to the unit data structure.
localglobal: indicator variable that indicates whether suitability is determined using local rules only or using both local and global rules.
geogtype: the geographic type of the subitem whose locational suitability is being determined.
itemno: the index of the subitem in the unit data structure.
x,y: the relative coordinates of the location.
x1p,y1p,...: the map coordinates of the unit's corners.

Return Value: 0 if the location is unsuitable, 1 if the location is suitable.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int terraintype: the terrain type of the location (x,y).

int csuitable: indicator variable that indicates whether the location (x,y) is suitable.

int side,echelon,number,type: the side, echelon, number, and type of the subitem numbered itemno.

float xx1,yy1: map coordinates of the point (x,y).

float xp,yp: map coordinates returned by the function _transfstdtoreal.

float *xppoint,*yppoint: pointers to xp,yp.

int xi,yi: viewport (pixel) coordinates corresponding to the point (x,y).

unsigned int time1,time2: time counters, used to control the timing of "flashes" on the screen.

int access: accessibility indicator variable (stores a return value from a call to _accessibility).

int lostgt: LOS to target (objective) indicator variable (stores a return value from a call to _lostotarget).

int losother: stores a return value from a call to _lostootherunits.

int loshq: stores a return value from a call to _lostoheadquarters.

int inforwarearea: stores a return value from a call to _inforwardarea.

float disttoother: stores a return value from a call to _distootherunits.

float disttofront: stores a return value from a call to _distancetofront.

15. Function _clipssuitability0201

Declaration: int _clipssuitability0201(struct unit *unit0, int localglobal, int geogtype, int itemno, float x, float y, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p, int processitemstatus)

Purpose: This function determines the suitability of a location, using the CLIPS knowledge-based system. This version (0201) is a test version; it contains prototypical rules for placement of TRAILBLAZER units.

Input Parameters:
The same as _suitability0201.

Return Value: The same as _suitability0201. Note that the return value is stored in the global variable CLIPS_suitable, for use by _repositionsubunitsbyrule. The value of this variable indicates what stage of rule processing has just been done.

Global Variables Modified: CLIPS_suitable.

Files Modified: None.

Variable Definitions:

The variables of _clipssuitability0201 are the same as those of _suitability0201, plus the following.

#define LOCAL 0: manifest constant denoting the value of the variable localglobal (0) used to specify that only local rules are to be processed.

#define GLOBAL 1: manifest constant denoting the value of the variable localglobal (1) used to specify that both local and global rules are to be processed.

char CLIPS_buffer[300]: character array used to store text that describes the stage of CLIPS rule processing.

L. Function File s03ldefu.c: Define Units Program

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains a single function, _defineunit. This function enables the user to "define" a unit, i.e., to specify all of the unit's characteristics in terms of the unit parameters used in the Scenarist system.

Functions Included in File:
_defineunit

Definitions of Global Variables: None.

2. Function _defineunit

Declaration: void _defineunit(void)

Purpose: This function enables the user to "define" a unit, i.e., to completely specify its attributes in terms of the unit parameters used in the Scenarist system, and store it in a unit file (for later retrieval).

Note: In the Scenarist documentation, the term "define" is used in a specific sense, to refer to the process of specifying values for unit attributes, inputting them into the Scenarist system, and storing them in a unit file (either a generic unit file or a specific unit file). The term "create" is used synonymously with "define." A unit that has been defined is also said to "exist."

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

char input: variable used to store a character input by the user from the keyboard.

char defgenunit: indicator variable used to specify whether the user wishes to define a generic unit.

char defspecunit: indicator variable used to specify whether the user wishes to define a specific unit.

char parentunitexists: indicator variable used to specify whether a unit's parent unit was found in the specific unit file.

short index: the index number in the unit record of a subitem having a specified code.

int ystart,yend: line-position variables used to set text position in window b.

float x1,x2,x3: variables used to input floating-point data from the keyboard.

float pp1,pp2: variables used to input floating-point data whose values are between 0.0 and 1.0 from the keyboard.

float x,y: variables used to specify a location (in relative coordinates).

float xrad: the radius of a unit (radius of a circular area associated with a unit, or half of the length of a square area associated with a unit).

float scalex,scaley: scaling factors to convert to meters; no longer used; left in code in case allow for input of nonmetric data in a future version (as reminders of where the code will have to be modified).

Note: Many of the variables that occur next refer to components of the unit data structure. Their names are derived from the component names, specifically, the suffix of the component name. For example, the variable "nfront" is derived from the component unit0.nfront. The components of the unit data structure were defined above (in the definitions of variables and data structure types of the header file s03binca.h). This convention for naming local variables is common in C programming. Because of the common use of this naming convention, we shall not repeat the definitions of local variables derived from unit data structure components in each function in which they occur. Local variables derived from data structures occur in many Scenarist functions, and the glossary would contain much duplication (i.e., an identical definition for every function in which a

data structure is used). More importantly, the repetition would impede, rather than assist, an experienced C programmer working on the Scenarist. With all of the duplication, the glossary would be much larger than necessary, and much larger than it is now. Furthermore, it would be unnecessarily troublesome to find definitions for specialized, function-specific local variables, if the glossary were "cluttered" with repeated definitions of local variables derived from the unit data structure.

Note: The variables x, y, xp, yp, *xppoint, *yppoint, x1p, y1p, x2p, y2p, x3p, y3p, x4p, y4p are used in many Scenarist functions. These variables have already been defined a number of times. The variables x,y are used to denote the relative coordinates of a point to be transformed to map coordinates by the function _transfstdtoreal. The variables xp,yp refer to the map coordinates. The variables *xppoint and *yppoint are pointers to xp and yp. The variables x1p, y1p,..., are the map coordinates of the unit. These variables are used in many Scenarist functions, always with the same notation and definitions. Their definitions will no longer be repeated.

char inline[80]: array used to store text for output to the screen (in "graphics text" mode).

(Reminder: the definition of name[17] was not provided, because it is a local variable name derived from the name of the corresponding component of the unit data structure, viz., unit0.name[].)

FILE *unitfilp: pointer to the file (more correctly, to the memory location containing data about the file) in which the unit being defined will be stored.

char unitfiln[12]: the name of the unit file in which the unit being defined will be stored.

int nwrite: indicator variable that specifies whether the new unit was correctly written in the file.

char inputch: variable used to contain a character input from the keyboard.

int intx1,intx2,inty1,inty2: variables used to store the parameters of "boxes" in window b, in which the mouse pointer is placed to select certain function options.

int isave,jsave,iblink: variables used to store information about menu boxes.

int page=0: In the EGA mode, it is possible to store screen data in two "pages." This is not possible in the EGA mode (one page only). Some of the graphics functions allow for a "page" parameter. This option is never used by the Scenarist, but the page parameter remains in these functions

because they were extracted from other sources. The variable "page" occurs in other functions, and will not be redefined.

char lines[2][36]: array used to store text to be output to the screen (in graphics text mode).

int highlighted_selection_number: variable used to indicate which function option was selected by the user.

M. Function File s03mcopy.c: Copy, Delete, and Display Units Program

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions for "copying" a unit, for deleting a unit from a file (either the generic unit file or the specific unit file), for displaying a unit on the screen, and for outputting a formatted description of all units in the generic unit file or the specific unit file to the printer or to a file.

Functions Included in File:

_editunit
_copyunit
_getunitbycode
_getsubunitbycode
_getunitbyidno
_getunitbyindex
_getsubunitbyidno
_deleteunit
_displayunit
_setcoordsforzoommap
_outputunits

The function _copyunit "copies" a unit. The term "copying" refers to the procedure of creating a new unit from an existing (already defined) unit. The new unit contains most of the same data as the old (existing) unit. The data that may be modified for the new unit depends on whether it is a generic unit or a specific unit. If the new unit is a generic unit, the only "new" information is the unit code and name. If the new unit is a specific unit, the user must specify the unit code, name, id no, parent unit's id no, and parent unit's code. If the parent unit exists (i.e., has been defined to the Scenarist, so that it is in the specific unit file), the map coordinates of the location of the new unit are derived from the parent's location. If the parent unit does not exist, the user must input the map coordinates of the unit's corners. If the new unit is a specific unit, the user has the option of specifying an objective, mission, and avenue of approach to the objective.

In the current version of the Scenarist, the user has a very limited ability to edit units. He may modify the relative locations of certain subitems (i.e., those of geogtypes 6 and 7). He may reposition ("move") an entire

unit (that has no parent unit defined), at which time he may respecify the unit's objective, mission, and avenue of approach. The only way a user may make other changes to a unit is to delete it from the file and redefine it. This function file contains a function (_deleteunit) for deleting a unit from the file in which it is stored (either a generic unit file or a specific unit file).

After a number of units have been defined, the user may forget the code number of a unit. The code number is necessary to identify a unit to be displayed on the screen, or to be processed by the rules. The "display" function, _displayunit, enables the user to review the index numbers, id numbers, and codes for all of the units in the specific unit file, and to display any one of them on the screen. The unit is displayed in the maximum size that fits on the map window (i.e., on the smallest-size geographic area possible).

The function _outputunits enables the user to obtain a formatted output of all of the units in the generic unit or specific unit files. This output may be directed to the printer or to a disk file of the user's choice.

In addition to the primary functions, _copyunit, _deleteunit, _displayunit, and _outputunits, the file contains a number of other functions called by these functions.

The function _editunit rewinds a "scratch" file (used in the process of deleting a unit), writes a recognizable sequence of characters (0...1...2...) to the scratch file, and reads this sequence from the scratch file into a unit data structure called unitblank. The "constant" unitblank is used to initialize unit data structures in the function _defineunit. This is done to assist the debugging process. The generic and specific unit files are "binary" files, i.e., unit data structures are written to them in binary format. It is difficult to decode binary data on the screen. To assist the decoding process during the course of debugging, the following is done. Prior to "filling" a unit data structure with data in the function _defineunit, the unit data structure is initialized with unitblank. Much of the initialized data is not changed by the process of defining a unit. When the defined unit is copied to the file, recognizable characters show up in the parts of the data structure that were not modified. The presence of these recognizable characters makes the debugging process much easier.

Calls to _editunit are made prior to each call to _defineunit and _copyunit. These calls initialize unitblank (redundantly, after the first call) and reset the scratch file to the initializing values.

The functions _getunitbycode, _getunitbyidno, and _getunitbyindex retrieve units from a unit file. These functions are used by the function _display to retrieve a unit from the specific unit file. The unit identifier used in the retrieval process may be either the unit's code, its idno, or its index (record number). The function _getunitbycode is also used by a number of other Scenarist functions to retrieve units by their codes.

The function _getsubunitbycode determines the array index, in the variable (data structure component) unit0.subunitatt[], in which the data for a subunit of specified code are located. This information is used in the process of retrieving the map coordinates of a newly copied unit from its parent unit. It is also used by the function _repositionprogeny, in the process of repositioning all of the subordinate units of a repositioned unit.

The function _getsubunitbyidno determines the array index, in the variable unit0.subunitatt[], in which the data for a subunit of specified idno are located. This function is not presently used. (It probably will never be used, since all of the Scenarist's subunit access is done by code, not by idno. This code was developed with an eye to allowing the user to retrieve either units or subunits for display, using idno's (as well as by code or index). The present Scenarist version allows the user to display units, but not subunits (unless they are defined as independent units). There are no plans to implement a capability to display undefined subunits.)

The function _setcoordsforzoommap is called by the _display function, to zoom a map so that the unit to be displayed "just fits" on the map, i.e., so that the map boundaries become a bounding rectangle for the unit. This function is also called by the function _setcoordsforzoommapb, which allows the user to specify a map zoom.

Definitions of Global Variables: None.

2. Function _editunit

Declaration: void _editunit(void)

Purpose: Initialize the global data structure unitblank. Reset (initialize and rewind) the scratch file. (Purpose described above in detail).

Input Parameters: None.

Return Value: None.

Global Variables Modified: unitblank.

Files Modified: scratch file (pointer scratchfilp, name scratchfiln).

Variable Definitions:

char blank=' ': a blank character, no longer used.

int index: a counter that counts increases in the value of the index i by hundreds.

int size: a variable that stores the size (in bytes) of a data structure.

char symbol: a variable used to store a character.

char init[]: a string filled with numbers and letters, in order.

3. Function _copyunit

Declaration: void _copyunit(void)

Purpose: Define a new unit by copying data from an already-defined unit (the "old" unit) and modifying it. (Recall the special definition of the term "define," provided above in the description of the function _defineunit.)

Input Parameters: None.

Output Parameters: None.

Global Variables Modified: none (other than the numerous "working" variables, such as "code" or "unit0," that occur in similar roles in many functions and were defined as global variables).

Files Modified: generic unit file or specific unit file, according as the user is using _copyunit to create a new generic unit or a new specific unit.

Variable Definitions:

char input: a variable used to store a character input by the user from the keyboard.

char copyfromgenunit: an indicator variable that indicates whether a unit will be copied from the generic unit file (specified in the project file, projxxxx.fil); that is, that the "old" unit is a generic unit.

char copyfromspecunit: an indicator variable that indicates whether a unit will be copied from the specific unit file; that is, that the "old" unit is a specific unit.

char copytogenunit: an indicator variable that indicates whether a unit will be copied to the generic unit file (i.e., the new unit is a generic unit).

char copytospecunit: an indicator variable that indicates whether a unit will be copied to the specific unit file (i.e., that the new unit is a specific unit).

FILE *filefromp: pointer to the file from in which the "old" unit (to be copied) is located.

FILE *filetop: pointer to the file in which the "new" unit (being created) will be stored.

char fileton[13]: name of the file in which the new unit will be stored.

int index: the index of a subitem in the array unit1.subunitatt[].

float x1: a variable in which a floating-point variable input by the user from the keyboard is stored.

int parentunitexists: an indicator variable that indicates whether the parent unit of the unit being defined exists (i.e., is stored in a unit file).

int oldunitexists: an indicator variable that indicates whether the old unit (whose code is specified by the user) exists.

int oldcode[13]: the code of the old unit (being copied from).

int newcode[13]: the code of the new unit (being created).

int y1,y2,...: variables used to control mouse and window functions (same definitions as given earlier in function _defineunit).

4. Function _getunitbycode

Declaration: int _getunitbycode(int code[], struct unit *unit0, FILE *unitfilp)

Purpose: This function retrieves a unit of a specified code from a specified file and stores it in a specified unit data structure. (The term "retrieves" means that all of the unit's data will be read from the file and stored in memory.) This function is called by many other functions.

Input Parameters:

code[]: the code of the unit to be retrieved.
unit0: pointer to the unit data structure in which the data of the specified unit are to be stored.
unitfilp: pointer to the file from which the unit is to be retrieved.

Return Value: 0 if unit not found, 1 if unit found.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int icode[13]: array used to store the code of a unit read from the file.

5. Function _getsubunitbycode

Declaration: int _getsubunitbycode(int code[], struct unit *unit0)

Purpose: This function determines the index, in the array unit0.subunitatt[], of the array component in which the data of the subunit of specified code are stored, in the specified unit. It is used to retrieve locational data stored in a unit's parent unit, when creating a unit by "copying," or when repositioning subordinate units after repositioning ("moving") a unit.

Input Parameters:

code[]: code of the subunit whose index is desired.
unit0: pointer to the unit containing the specified subunit.

Return Value: If the subunit is found, its index number (0, 1, ...) is returned. If it is not found, -1 is returned.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int n: number of subunits in the unit.

int icode[]: code of a subunit in the unit.

6. Function _getunitbyidno

Declaration: int _getunitbyidno(int idno, struct unit *unit0, FILE *unitfilp)

Purpose: This function retrieves a unit by idno (i.e., finds the data record of the unit whose id no is idno in a unit file and places it in a data structure in memory). It is called only by the function _displayunit, in which the user is given the option of retrieving a unit by idno (the Scenarist retrieves units only by code, in its internal processing).

Input Parameters:

idno: id no of the unit to be retrieved.
unit0: pointer to data structure in which the retrieved data are to be placed.
unitfilp: pointer to the file to be searched for the unit data.

Return Value: 1 if unit data are found, 0 otherwise.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int iidno: variable in which the value unit0.id.idno is placed, for a record in the file.

7. Function _getunitbyindex

Declaration: int _getunitbyindex(int index, struct unit *unit0, FILE *unitfilp)

Purpose: This function retrieves a unit by index (i.e., finds the record numbered "index" in a unit data file and places it in a data structure in memory). It is called only by the function _displayunit, in which the user is given the option of retrieving a unit by index (the Scenarist retrieves units only by code, in its internal processing).

Input Parameters:

index: index (record number) of the unit to be retrieved.

unit0: pointer to data structure in which the retrieved data are to be placed.
unitfilp: pointer to the file to be searched for the unit data.

Return Value: 1 if unit data are found, 0 otherwise.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int indexi: record number of a record in the unit file.

8. Function _getsubunitbyidno

Declaration: int _getsubunitbyidno(int idno, struct unit *unit0)

Purpose: This function determines the index, in the array unit0.subunitatt[], of the array component in which the data of the subunit of specified idno are stored, in the specified unit. This function is not presently used. It was designed (as discussed above) for potential use in selecting subunits (in addition to units) from the specificunit file, for display. The current version of the Scenarist displays only units, however, not subunits, and so this function is not used.

Input Parameters:

idno: id no of the subunit whose index is desired.
unit0: pointer to the unit containing the specified subunit.

Return Value: If the subunit is found, its index number (0, 1, ...) is returned. If it is not found, -1 is returned.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int n: number of subunits in the unit.

int iidno: id no of a subunit in the unit.

9. Function _deleteunit

Declaration: void _deleteunit(void)

Purpose: This function deletes a unit (i.e., a unit record) from a unit file (either a generic unit file or a specific unit file). It is used to delete a unit whose specification is determined to be in error, and not correctable by means of the Scenarist's current (limited) unit editing capability (to move a unit or to reposition subitems of geogtypes 6 or 7).

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: The unit file in which the unit record to be deleted is located.

Variable Definitions:

char deletefromgenunit: indicator variable that specifies whether the unit to be deleted is contained in the generic unit file (i.e., is a generic unit).

char deletefromspecunit: indicator variable that specifies whether the unit to be deleted is contained in the specific unit file.

char ideleted: indicator variable that specifies whether the unit has been deleted.

char imatch: indicator variable that indicates whether a unit of code specified by the user is in the file (specified by the user).

FILE *unitfilp: pointer to the file from which the unit is to be deleted.

int idno: id no of unit to be deleted.

char unitfiln[13]: name of file from which the unit is to be deleted.

10. Function _displayunit

Declaration: void _displayunit(void)

Purpose: This function lists all of the units contained in the specific unit file, and allows the user to select one of them (at a time) for display

on the screen. The units are listed in the order in which they are in the file. This order is called the unit's "index." The user may select the unit to be displayed by index, idno, or code.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

char inputch: variable used to store a character input by the user from the keyboard.

int index: the unit's index (record number in the unit file).

int idno: the unit's id no.

int row2,col2: parameters used to set the current window to the full screen.

11. Function _setcoordsforzoommap

Declaration: void _setcoordsforzoommap(struct unit *unit0)

Purpose: This function sets the parameters used to zoom a map so that the map boundaries form a bounding rectangle for a specified unit (i.e., the unit "just fits" on the map).

Input Parameters:
unit0: pointer to the unit for which the zoom parameters are to be determined.

Return Value: None.

Global Variables Modified:
xminview, xmaxview, yminview, ymaxview: the zoom parameters.

Files Modified: None.

Variable Definitions:

float xmn,xmx,xnew,xxmax,ymn,ymx,ynew,yymax,delta: variables used to compute the coordinates of the bounding rectangle of the input unit.

12. Function _outputunits

Declaration: void _outputunits(void)

Purpose: This function outputs a formatted list of all of the units in either a generic unit file or a specific unit file to the printer or to a file. The unit file to be output is specified by the user; it is either the generic unit file specified in the project file (projxxxx.fil) or the specific unit file specified in the project file. This function is the interface of the Scenarist with the user's application for which scenarios are desired. The user uses this function to output the units to a file after the subunits have been repositioned by the Scenarist. He then would reformat the output file for input to the model requiring a scenario. This function is also useful for creating a formatted hard copy output of all of the units in the generic unit file and specific unit file. (These file may not be printed using the DOS print command, because they are stored in binary, not ASCII (text) format.)

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

char defgenunit: indicator variable that indicates that the generic unit file is to be output. (This name is misleading; the code was reused from the "define units" function.)

char defspecunit: indicator variable that indicates that the specific unit file is to be output.

int index, indexi: indices used to number the units on the output lists.

char outputfiln[13]: name of the output file.

FILE *unitfilp: pointer to the unit file to be output.

FILE *outputfilp: pointer to the output file.

char unitfiln[13]:name of the unit file to be output.

float xx[4],yy[4]: variables used to contain the map coordinates of a subitem's corners.

int transfstdtoreal: constant that specifies that the function _transfstdtoreal is to transform relative coordinates to map coordinates (and not be used in a "null" mode).

char outputtofile: indicator variable that indicates whether the unit file is to be output to a file or to the printer.

N. Function File s03nmap.c: Map Drawing Program

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions for drawing maps. The same function is used to draw either a cellular map or a vector map.

Functions Included in File:

_drawmap
_label
_legend
_setcoordsforzoommapb

The function _drawmap draws the map. It calls the function _label to place coordinate labels on the map and the function _legend to draw a legend for the map.

The function _setcoordsforzoommapb enables the user to zoom the map in various ways. One of these ways is to set the map boundaries as the bounding rectangle of the last unit to be accessed. That function is executed by the function _setcoordsforzoommap, described above.

Definitions of Global Variables:

extern char *face[6]: fonts.

2. Function _drawmap

Declaration: void _drawmap(int cellmaptype, int mapindex, int vectormap, int printlabels)

Purpose: This function draws a map. The map is either a cellular map or a vector map, as specified by the input parameters. Which cellular map data set is to be used (if a cellular map is to be drawn) is also specified by an input parameter. Whether to print labels on vector map objects is also controlled by an input parameter.

Input Parameters:

int cellmaptype: constant that specifies the type of cellular map to be drawn (none, discrete-variable, or continuous-variable)
int mapindex: constant that specifies which map of the specified type is to be drawn.

int vectormap: constant that specifies the manner in which a vector map is to be drawn (not at all, superimposed, or drawn after erasing the map window)
int printlabels: constant that specifies whether labels are to be printed for the objects on a vector map.

Input Parameter Value Definitions:
if cellmaptype = 0, don't draw any cell map
if cellmaptype = 1, draw discrete-variable map

```
    if mapindex = 0, draw terrain-type map
    if mapindex = 1, draw road map
  if cellmaptype = 2, draw continuous-variable map
    if mapindex = 0, draw elevation map
  if vectormap = 0, don't draw or superimpose vector map
  if vectormap = 1 superimpose ("add") vector map
  if vectormap = 2 draw vector map (i.e., erase window
    first)
  if printlabels = 0 don't print labels on vector map
    objects
  if printlabels = 1 print labels on vector map objects
```

Note: discrete-variable maps (e.g., terrain-type, road availability) and continuous-variable maps (e.g., elevation) are referred to as cell maps.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int intx1,inty1: variables used to store viewport (pixel) coordinates.

int patternno: variable used to store fill pattern number.

int attributeno: index used to refer to map data type.

int attributecategory: index used to refer to a particular category of a map data type.

int color,color0,color1: variables used to specify color code numbers.

int end: end-of-file indicator variable.

int npts: number of points used to define the boundary of an area object.

float xsize,ysize: map cell dimensions (map units).

float xmin,ymin,xmax,ymax: coordinates of the lower-left and upper-right map corners.

int irow,icol: indices used to scan map array rows and columns.

int nxcells,nycells: number of horizontal cells (rows) and vertical cells (columns) of the map (32 in current version of Scenarist).

int int1,int2,int3,int4: viewport (pixel) coordinate locations of the bounding rectangle of a circle used to represent a point object.

int fillarobs: indicator variable that indicates whether area objects are to be "filled" with a pattern. (With the Microsoft C compiler, it is possible to fill either cells or area objects, but not both. The Microsoft fill function works erratically if it is attempted to fill an area that has already been filled.)

float cont, cmin, cmax, intquintrange, value: variables used to compute frequency-distribution quintiles of a continuous variable.
int contdisc, valuedisc: variables used to store map cell data.

double x1,x2,xint,y1,y2,yint: variables used to determine the relative coordinates of a map cell boundary (used in drawing cell boundaries and computing distribution quintiles).

int ncats: the number of categories (levels, values) of a discrete variable.

int maptype: cellmaptype-1, used as the first index of the mapinf array (which contains map information).

3. Function _label

Declaration: void _label(void)

Purpose: This function draws coordinate labels on a map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int col,row: column and row of text position.

float x1,y1: coordinate values.

float xincr,yincr: increments of x and y axis coordinate labels (map values).

float leginc: increments of x or y axis coordinate labels, text position coordinates.

4. Function _legend

Declaration: void _legend(int maptype, int mapindex)

Purpose: This function draws a legend for a map (in window "e").

Input Parameters:

maptype: constant that specifies the map type (0 = discrete-variable map, 1 = continuous-variable map).

mapindex: constant that specifies the map index for a particular map type (for a discrete-variable map, 0 = terrain-type map, 1 = road availability map; for a continuous-variable map, 0 = elevation map).

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int col,row: column and row of text position.

int x1,y1,x2,y2: pixel locations of legend box bounding rectangle.

int color: variable used to store fill color code numbers.

int patternno: variable used to store fill pattern number.

int ncats: number of categories of a map data type.

char line[81]: character array used to store text to be printed on the screen (graphics text format).

5. Function _setcoordsforzoommapb

Declaration: void _setcoordsforzoommapb(void)

Purpose: This function zooms a map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: xminview, xmaxview, yminview, ymaxview (map zoom parameters; coordinates of map bounding rectangle).

Files Modified: None.

Variable Definitions:

float zoommapwidth: width of zoomed map (in meters).

float xmn,xmx,ymn,ymx: variables used to compute zoom parameters.

int specunitno: array index of array data structure used to store unit data (used if the user wishes to specify the zoom parameters as the bounding rectangle of a map that is the bounding rectangle of the unit).

char inputch: variable used to store a character input by the user from the keyboard.

O. Function File s03pdraw.c: Functions for Drawing Units on Map

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions for drawing units on a map.

Functions Included in File:

_drawunit
_drawline
_transfstdtoreal

The function _drawunit draws a unit on a map. The function _drawline draws a line between two specified points (whose coordinates are specified in map coordinates). The function _transfstdtoreal transforms from the relative coordinates in terms of which a unit is defined to map coordinates.

All of the locations of the internal components of a unit are specified in terms of the relative coordinates discussed above. The map location of the unit is specified in terms of the map coordinates of the four corners of the quadrilateral representing the unit. A unit is drawn in window "b," using map coordinates. (This is enabled by making a call to the function _setwindow to define the map coordinate system as the coordinate system used by window b, and using the _moveto and _lineto functions to draw lines in this coordinate system.) In order to draw a line whose coordinates are specified in relative coordinates, it is necessary to transform those coordinates to map coordinates. This is done by means of the function _transfstdtoreal, which executes the linear transformation described earlier.

The function _drawline combines the _moveto and _lineto commands used to draw a line into a single function call.

Definitions of Global Variables:

extern char *face[6]: fonts.

2. Function _drawunit

Declaration: void _drawunit(struct unit *unit0, int transfstdtoreal, int windowno)

Purpose: This function draws a unit on a map. (As discussed below, the function could also be used to draw a generic unit in the relative coordinate system, but this capability was never implemented.)

Input Parameters:

unit0: pointer to the unit to be drawn on the map.

transfstdtoreal: indicator variable that indicates whether the function _transfstdtoreal is to execute the transformation from relative coordinates to map coordinates, or to do nothing (i.e., bypass this transformation). The option to skip the transformation was included so that the code for drawing a map could be used to draw a unit either on a map or in the relative coordinate system, without having to introduce skips around every call to _transfstdtoreal. In the current version of the Scenarist, the option to draw a map in the relative coordinate system was never implemented. It was planned to implement this option during the definition of a unit, or as a special "display" function that could draw generic units, in the relative coordinate system (the function _displayunit displays only specific units, in the map coordinate system). These capabilities were never implemented, because of time/resource limitations.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

float xx1,yy1,xx2,yy2,xx3,yy3,xx4,yy4: map coordinates of points.

int colorunit,colorfeba: color codes for the unit and the FEBA.

float x1,x2,y1,y2: coordinates of bounding rectangle of unit's objective.

char label[9]: label of a subitem.

char name[17]: unit name.

short iside, iech, itype: indices defining a subitem's side, echelon, and type (for accessing a subitem's symbol number and label.

int int1,int2: pixel locations of the unit's name.

float radius: the radius of a subitem.

int xdirection, ydirection: variables that determine the coordinates of the next point to be drawn, when drawing a subitem.

3. Function drawline

Declaration: void _drawline(float x1, float y1, float x2, float y2)

Purpose: This function draws a (straight) line from the point (x1,y1) to the point (x2,y2), where these points are specified in map coordinates.

Input Parameters:
x1,y1: coordinates of initial end of line to be drawn.
x2,y2: coordinates of terminal end of line to be drawn.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

4. Function _transfstdtoreal

Declaration: void _transfstdtoreal(float x, float y, float *xppoint, float *yppoint, float x1p, float y1p, float x2p, float y2p, float x3p, float y3p, float x4p, float y4p, int transfstdtoreal)

Purpose: This function transforms the point (x,y) in relative (standard) coordinates to the point (*xppoint,*yppoint) in map (real) coordinates. See Final Report for mathematical description of transformation.

Input Parameters:
x,y: the point to be transformed.
xppoint,yppoint: pointers to the transformed point.
x1p,y1p,...: map coordinates of the unit's corners.
transfstdtoreal: indicator variable that indicates whether the function is to perform or skip the transformation.

Return Value: None.

Global Variables Modified: the variables pointed to by xppoint, yppoint (denoted by xp, yp).

Files Modified: None.

Variable Definitions:

float px,py,qx,qy: x,y,1-x,1-y.

float c1,c2,d1,d2: map coordinates corresponding to the intersections (in the relative coordinate system) of lines drawn through the point (x,y) parallel to the unit sides.

float c,d: map coordinates of transformed point.

P. Function File s03qgtfl.c: Get Files Function

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions for opening all of the files specified in the project file selected by the user, and for inputting data from these files. These data are used to "initialize," or "set up" the Scenarist program, at the beginning of a run. The input data include a project title, the symbol number array, the label array, the FEBA specification, and the cellular map data.

Functions Included in File:

_getfilenames
_readmapheader
_readmapdata
_resetmaplocpoint

The function _getfilenames reads the names of the files to be used in the Scenarist run from the project file (selected by the user), opening those files, and reading data from them.

The function _readmapheader reads the "header" section of the terrain-type and elevation cellular map files. The user is informed of a "default" location from which a 32-cell by 32-cell map "window" will be extracted from these two map files. The function _resetmaplocpoint is called to provide the user with the option of changing the map location. The map cell data are then extracted from these two files by the function _readmapdata and stored in the arrays geodisc[][][] and geocont[][][]. The header and map cell data are then read from the road availability map file, if one has been specified in the project file.

Definitions of Global Variables: None (other than global variables defined in the header files included at the beginning of the file).

2. Function _getfilenames

Declaration: void _getfilenames(void)

Purpose: This function reads the names of the files to be used in the Scenarist run from the project file selected by the user, opens these file, and reads data from them. Prior to reading the cellular map data, the user is given the option of modifying the map location point from the default location determined by the program.

Input Parameters: None.

Return Value: None.

Global Variables Modified: arrays geodisc, geocont, symbols, labels, feba, mapinf.

Files Modified: None. (All files specified in project file are opened.)

Variable Definitions:

int end: end-of-file indicator variable.

float locx,locy: coordinates of map location point.

int nrowsw,ncolsw: number of rows and columns in map window (32 in current version).

3. Function readmapheader

Declaration: void _readmapheader(int maptype, int mapindex, FILE *datafilp)

Purpose: This function reads a map "header" from a map file (viz., the file pointed to by the file pointer datafilp), and stores it in the array mapinf[maptype][mapindex].

Input Parameters:

maptype: the type of cellular map to be read (0 = discrete-variable map (terrain type, road availability); 1 = continuous-variable map (elevation))

mapindex: the index of the cellular map to be read (for discrete-variable maps: 0 = terrain type, 1 = road availability, 2 = TRAILBLAZER accessibility; for continuous-variable maps: 0 = elevation)

datafilp: pointer to the file from which the map data are to be read. (This file is either the discrete-data cellular map file specified in the project file or the continuous-data cellular map file specified in the project file.)

Return Value: None.

Global Variables Modified: the array mapinf.

Files Modified: None.

Variable Definitions:

char chdata: variable used to store a character input by the user from the keyboard.

int ncats: the number of categories of discrete-variable data (0 if continuous-variable data are being read).

int x1,y1,x2,y2: pixel positions of "continue" box placed on bottom of screen.

char line[80]: array used to store text data to be output to the screen (in graphics text mode).

4. Function _readmapdata

Declaration: void _readmapdata(int maptype, int mapindex, FILE *datafilp, float locx, float locy)

Purpose: This function reads cellular map data from a cellular map file. The function extracts a 32-cell by 32-cell map "window" from a cellular map file (which may have more than or less than 32 rows and 32 columns), from the map location point (locx,locy).

The "map location point" is a reference point used to position a map. The user has the option of specifying the map location point as the top-left corner of the map or the center of the map, and for specifying the map coordinates of the location point. The user specifies the map location (in a call to _resetmaplocpoint) prior to reading the map data (with this function).

Input Parameters:

maptype: same as for function _readmapheader
mapindex: same as for function _readmapheader
datafilp: same as for function _readmapheader
locx,locy: coordinates of map location point.

Return Value: None.

Global Variables Modified: arrays geodisc, geocont.

Files Modified: None.

Variable Definitions:

int windowrowindex,windowcolindex: indices for the map data arrays, geodisc and geocont.

int irow,icol: indices for scanning the map cell data in the map file.

int nrows, ncols: the number of rows and columns in the map file.

int nrowsw,ncolsw: the number of rows and columns in the map window read from the map file (32 and 32 in the current version of the Scenarist).

int discdata, contdata: variables used to store a map cell value.

float xmin,ymax,xmax,ymin: minimum and maximum map coordinates (i.e., values of the map boundaries).

float cellwidth: width of a map cell.

5. Function _rsetmaplocpoint

Declaration: void _resetmaplocpoint(void)

Purpose: This function enables the user to redefine the map location point and to specify the map location for the map location point. (See additional discussion above.) This function determines the values of the map boundaries, xminview, yminview, xmaxview, ymaxview. These values are changed during the course of the run if the map is zoomed. In case it is desired to "unzoom" the map, the original values of these parameters are saved, in the variables xminvieworiginal, yminvieworiginal, xmaxvieworiginal, and ymaxvieworiginal.

Input Parameters: None.

Return Value: None.

Global Variables Modified: xminview, xmaxview, yminview, ymaxview, xminvieworiginal, xmaxvieworiginal, yminvieworiginal, ymaxvieworiginal.

Files Modified: None.

Variable Definitions:

char inputch: variable used to store a character input by the user from the keyboard.

char line[80]: array used to store a line of text, for output to the screen.

float temp1,temp2: variables used to store map coordinates input by the user from the keyboard.

int x1,y1,x2,y2: pixel coordinates of the "continue" button drawn on the bottom of the screen.

float cellwidthview: the minimum value of the cellwidths of the terrain-type and elevation maps. This value is used as the cellwidth when drawing a map (regardless of what the cellwidth of the road availability map is).

Q. Function File s03rsymb.c: Function for Drawing Symbols

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains the function _symbol for drawing symbols of subunits.

Functions Included in File:
_symbol

Definitions of Global Variables:

extern char *face[6]: fonts.

2. Function _symbol

Declaration: void _symbol(int no, char *label, float xmap, float ymap)

Purpose: This function draws symbols for subitems (units are drawn using the function _drawunit). The function currently contains the code to draw 13 symbols. If the user wishes to use a symbol other than one of these 13, he must add new code (as a new "case") to this function.

Input Parameters:
no: symbol number of the symbol to be drawn.
label: pointer to a character array containing a label to be drawn next to the symbol.
xmap,ymap: map coordinates of the location where the symbol is to be drawn.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int x,y: pixel coordinates corresponding to map coordinates xmap, ymap.

int x1,y1,x2,y2: parameters defining a rectangle, generally centered at (x,y). This rectangle is where the main body of the symbol will be located (e.g., the rectangular part of a brigade symbol).

int xx1,yy1,xx2,yy2: parameters defining a smaller rectangle, located near the preceding rectangle. This rectangle is a second part of the symbol (e.g., the echelon designator on the top of the rectangular part of a brigade symbol).

int colortemp: variable containing a color code.

extern char masknull[8]: "mask" used to contain a fill code (used by the Microsoft floodfill function).

short color: variable containing a color code.

R. Function File s03sio.c: Basic Input-Output Functions

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains a number of input-output functions related to printing a graphics screen and controlling graphics output to the screen.

Functions Included in File:

- _printscreen
- _clearfoot
- LJ_Graphic
- Grey_Scale
- Print_Pause
- PromptLine
- writString
- writChar
- put_out
- status
- pralel
- stat
- PrtInit
- printch
- printst
- PrtPutC
- getMode
- getxy
- gotoxy
- getPage

The function _printscreen prints a screen containing graphics (the "Print Screen" button on the keyboard is useful only for printing a screen containing text, not graphics).

The function _clearfoot that erases the last three lines on the screen. That section of the screen is often used in the Scenarist for a variety of messages. This function clears this "footer" area.

The function LJ_Graphic is the function that accomplishes the printing of the graphics screen. It calls the functions strcpy, format, printst, pralel, and Grey_Scale. The functions Print_Pause and PromptLine are not used. The printer-related functions writString, writChar, put_out, status, pralel, stat, PrtInit, printch, printst, and PrtPutC are alternative printer output functions that were used in the process of debugging the function LJ_Graphic and making it more efficient.

The function getMode returns the current video mode.

The function getxy gets the current cursor (text) position.

The function gotoxy moves the cursor to a specified (text) position.

The function getPage returns the active page number.

The function gotoxy is used in the Scenarist. The functions getMode, getxy, and getPage are no longer used (they were used in debugging).

All of the functions in this file except for _printscreen and _clearfoot were extracted from texts on computer graphics (References 1-5). This glossary defines only variables in functions developed under the contract. For this reason, only variables contained in _printscreen and _clearfoot are defined below.

Definitions of Global Variables: None (other than manifest constants associated with the functions extracted from graphics texts).

2. Function _printscreen

Declaration: void _printscreen(void)

Purpose: This function prints a graphics screen on a laser printer that is compatible with an HJ LaserJet printer.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

char input: variable used to store character input by the user from the keyboard.

int row2,col2: parameters used to reset the text window to the full screen.

3. Function _clearfoot

Declaration: void _clearfoot(int page)

Purpose: This function erases the last three lines on the specified page and moves the text cursor to the beginning of the third line from the bottom.

Input Parameters: the page number (always zero).

Return Value: None.

Global Functions Modified: None.

Files Modified: None.

Variable Definitions:

int line: a line number.

S. Function File s03twind.c: Windows Functions

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions used to define the various windows used on the Scenarist screen display. These windows are labeled "a," "b," "c," "d," and "e." Window "a" is at the top of the screen. Window "b" is the map window. Window "c" is a footer at the bottom of the screen. Window "d" is the menu window to the top-right of the screen. Window "e" is the legend or text window to the bottom-right of the screen.

Functions Included in File:

setWindow
_windowa
_windowb
_windowbblack
_windowbenter
_windowb1
_windowb1enter
_windowb2
_windowb3
_windowc
_windowcenter
_windowscreen
_windowscreenblue
_windowd
_windowe
_windoweenter

The function setWindow is not used. The other _window... functions perform various "windows" functions, such as defining a viewport, defining a text window, and defining a (graphics) window.

Definitions of Global Variables: None. (Recall that we are not defining system variables, such as union REGS reg, a return parameter for the ROM-BIOS int86 interrupt function.)

2. Function setWindow

Declaration: void _setWindow(int x1, int y1, int x2, int y2, int color)

Purpose: This function defines a window using the ROM-BIOS set window function. Not used.

Input Parameters:

x1,y1,x2,y2: coordinates of bounding rectangle of window.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None (other than system variables).

3. Function _windowa

Declaration: void _windowa(void)

Purpose: This function sets up window "a," for the screen header.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int row2,col2: number of text (character) rows and columns on the screen.

4. Function _windowb

Declaration: void _windowb(void)

Purpose: This function sets up window "b," filled in blue.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

5. Function _windowbblack

Declaration: void _windowbblack(void)

Purpose: This function sets up window "b," filled in black (i.e., with a clear screen).

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

6. Function _windowbenter

Declaration: void _windowbenter(void)

Purpose: This function sets up window "b," but with no fill (i.e., this enables a return to the map window to superimpose additional drawings on what is already there).

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

7. Function _windowb1

Declaration: void _windowb1(void)

Purpose: This function sets up window "b1," used in the function _defineunit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

8. Function _windowb1enter

Declaration: void _windowb1enter(void)

Purpose: This function sets up window "b1," with no fill or clearviewport or border. That is, it simply "returns" to window b1.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

9. Function _windowb2

Declaration: void _windowb2(void)

Purpose: This function sets up window "b2," used in _defineunit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

10. Function _windowb3

Declaration: void _windowb3(void)

Purpose: This function sets up window "b3," used in _defineunit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

11. Function _windowc

Declaration: void _windowc(void)

Purpose: This function sets up window "c," with a border, and filled in blue.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

12. Function _windowcenter

Declaration: void _windowcenter(void)

Purpose: This function sets up window "c," with no border, fill, or clearscreen (i.e., the function simply returns to window c).

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

13. Function _windowscreen

Declaration: void _windowscreen(void)

Purpose: This function sets the window to the full screen.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

14. Function _windowscreenblue

Declaration: void _windowscreenblue(void)

Purpose: This function sets the window to the full screen, and fills it with blue.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

15. Function _windowd

Declaration: void _windowd(void)

Purpose: This function sets up window "d," filled in blue.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

16. Function _windowe

Declaration: void _windowe(void)

Purpose: This function sets up window "e," filled in blue.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

17. Function _windoweenter

Declaration: void _windoweenter(void)

Purpose: This function sets up window "e," with a border, but no fill or clear.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

T. Function File s03vmous.c: Mouse Driver Functions

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains functions used to control the mouse, to check the system hardware and graphics setup, and to initialize the mouse and window parameters.

Functions Included in File:

```
m_reset
m_show
m_hide
m_pos
m_moveto
m_motion
m_lightpen
m_move_ratio
m_conceal
m_speed
m_graphic_cursor
initialize_cursor
Set_Graphic_Cursor
_setup_button
_clickbutton
_click_accept
_click_cancel
_call_button

_call_buttonb
_call_continueb
_call_continue
_Hardware_Setup
_setup_screen_windows
```

All of the functions listed above except for the last are mouse interface functions extracted from graphics texts, and so no variables of those functions will be defined.

The function _Hardware_Setup checks the hardware setup and the graphics setup, and outputs data about these setups to the screen.

The function _setup_screen_windows checks the video parameters and initializes the Scenarist mouse and graphics parameters.

Definitions of Global Variables: None.

2. Function _Hardware_Setup

Declaration: void _Hardware_Setup(void)

Purpose: This function checks the hardware setup and the graphics setup, and outputs data about these setups to the screen.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

3. Function setup screen windows

Declaration: void _setup_screen_windows(void)
Purpose: The function _setup_screen_windows checks the video parameters and initializes the Scenarist mouse and graphics parameters.

Input Parameters: None.

Return Value: None.

Global Variables Modified: p1, p4, q1, q4, c1, c4, r1, r4, modnam, nxpch, nypch, borderbottom

Files Modified: None.

Variable Definitions:

char *modnamvga="_VRESCOLOR": pointer to manifest constant denoting VGA color monitor.

char *modnamega="_ERESCOLOR": pointer to manifest constant denoting EGA color monitor.

char *modnam: variable containing the manifest constant describing the system's color monitor.

int page: video page.

int nxpix: number of x pixels on screen.

int nypix: number of y pixels on screen.

int nxch: number of x characters on screen.

int nych: number of y characters on screen.

U. Function File s03wgrap.c: Graphics Interface Programs -- Main Menu

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains graphics interface programs related to the Scenarist's "main menu." The main menu provides the user with a choice of the basic Scenarist function categories.

Functions Included in File:

```
_display_intro_screens
_display_screen_windows
_project_selection
_main_menu_selection
_display_windowd
_About
_Units
_Rules
_Map
_Scenario_Generation
_Quit
```

The function _display_intro_screens displays the introductory screens when the Scenarist run begins. The function _display_screen_windows places sets up all five of the Scenarist windows. The function _project_selection enables the user to select a project file, containing the names of the data files to be used in the current run. The function _main_menu_selection displays the basic Scenarist option choices in the "d" window. The function _display_windowd prints the main menu choices in the "d" window. The functions _About, _Units, _Rules, _Map, _Scenario_Generation, and _Quit initiate the six main menu choices.

Definitions of Global Variables:

struct boxlabelstruct boxlabelmat[3]: array containing the labels of all three menus used in controlling the Scenarist (the main menu, the map functions menu, and the unit functions menu).

unsigned char *face[6]: array containing the codes used to register fonts.

2. Function _display_intro_screens

Declaration: void _display_intro_screens(void)

Purpose: This function displays the introductory screens at the beginning of the Scenarist run.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int x1,x2,y1,y2: parameters used to set up the "continue" button at the bottom of the screen. (Note: these variables occur in a number of other functions that follow; their definitions will not be repeated.)

3. Function display screen windows

Declaration: void _display_screen_windows(void)

Purpose: This function displays all of the five major windows used by the Scenarist.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

4. Function project selection

Declaration: void _project_selection(void)

Purpose: This function enables the user to select the project file, which contains the names of all of the data files to be used in the Scenarist run.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int numfiles: number of files in the directory.

char ptemp[80]: array used to store text to be output to the screen.

int irow,icol: row and column indices of file name table.

int isave, jsave, iblink: variables used to keep track of mouse polling.

int mouse_selected_box: variable that indicates what box was selected by a mouse click.

int ifiles: file index.

char *argvin: pointer to string containing a project file name.

struct fint_t find: variable used as argument for MS-DOS system function _dos_findfirst.

long size: the size of a data structure.

5. Function _main_menu_selection

Declaration: void _main_menu_selection(void)

Purpose: This function displays Scenarist main menu, which specifies the major categories of program functions.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int ibox: index over menu boxes.

int returnvalue: the box index number of a selected highlighted box.

6. Function _display_windowd

Declaration: void _display_windowd(void)

Purpose: This function displays the main menu choices in window "d."

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.
int ibox: index over the boxes of a menu.

Variable Definitions: None.

7. Function _About

Declaration: void _About(void)

Purpose: This function provides summary information about the Scenarist.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

8. Function _Units

Declaration: void _Units(void)

Purpose: This function offers options for processing military units.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

9. Function _Rules

Declaration: void _Rules(void)

Purpose: This function offers options for editing rules (entry point for future development).

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

10. Function _Map

Declaration: void _Map(void)

Purpose: This function offers options for processing maps.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

11. Function _Scenario_Generation

Declaration: void _Scenario_Generation(void)

Purpose: This function offers options for control of large-scale rule-based positioning of items (entry point for future development).

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

12. Function _Quit

Declaration: void _Quit(void)

Purpose: This function terminates Scenarist processing.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

V. Function File s03ygra2.c: Graphics Interface Programs -- Map- and Unit-Related

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains graphics interface programs related to the Scenarist's menus that provide options for map- and unit-related functions.

Functions Included in File:

_map_menu_selection
_unit_menu_selection
_Draw_Terrain_Map
_Draw_Elevation_Map
_Draw_Road_Map
_Add_Vector_Map_with_Labels
_Add_Vector_Map_without_Labels
_Draw_Vector_Map_without_Labels
_Place_Unit_on_Map
_Zoom_Map

_Change_Map_Location
_Change_Map_Files
_Print_Map
_Define_Unit
_Copy_Unit
_Delete_Unit
_Reposition_Unit_by_User
_Reposition_Subunits_by_User
_Reposition_Subunits_by_Rules
_Reposition_FEBA
_Display_Unit
_Output_Units

The function _map_menu_selection displays the menu with the map processing options. These options are: _Draw_Terrain_Map, _Draw_Elevation_Map, _Place_Unit_on_Map, _Zoom_Map, _Change_Map_Location, _Change_Map_Files, and _Print_Map. The function _unit_menu_selection displays the menu with the unit processing options. These options are: _Reposition_Unit_by_User, _Reposition_Subunits_by_User, _Reposition_Subunits_by_Rules, _Define_Unit, _Delete_Unit, _Reposition_FEBA, _Display_Unit, and _Output_Units.

Definitions of Global Variables: None.

2. Function _map_menu_selection

Declaration: void _map_menu_selection(void)

Purpose: This function displays the menu with the map processing options.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int ibox: index over the menu boxes.

int returnvalue: the box index number of a selected highlighted box.

3. Function _unit_menu_selection

Declaration: void _unit_menu_selection(void)

Purpose: This function displays the menu with the unit processing options.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int ibox: index over the menu boxes.

int returnvalue: the box index number of a selected highlighted box.

4. Function _Draw_Terrain_Map

Declaration: void _Draw_Terrain_Map(void)

Purpose: This function draws a terrain map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

5. Function _Draw_Road_Map

Declaration: void _Draw_Road_Map(void)

Purpose: This function draws a road availability map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

6. Function _Draw_Elevation_Map

Declaration: void _Draw_Elevation_Map(void)

Purpose: This function draws an elevation map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

7. Function _Draw_Vector_Map_with_Labels

Declaration: void _Draw_Vector_Map_with_Labels(void)

Purpose: This function draws a vector map with labels on the map objects. This function is not presently used.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

8. Function _Draw_Vector_Map_without_Labels

Declaration: void _Draw_Vector_Map_without_Labels(void)

Purpose: This function draws a vector map without labels on the map objects.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

9. Function _Add_Vector_Map_with_Labels

Declaration: void _Add_Vector_Map_with_Labels(void)

Purpose: This function superimposes a vector map with object labels on the existing map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

10. Function _Add_Vector_Map_without_Labels

Declaration: void _Add_Vector_Map_without_Labels(void)

Purpose: This function superimposes a vector map without object labels on the existing map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

11. Function _Place_Unit_on_Map

Declaration: void _Place_Unit_on_Map(void)

Purpose: This function draws a unit on the map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

12. Function _Zoom_Map

Declaration: void _Zoom_Map(void)

Purpose: This function zooms a map.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

13. Function _Change_Map_Location

Declaration: void _Change_Map_Location(void)

Purpose: This function enables the user to change the map location.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

14. Function _Change_Map_Files

Declaration: void _Change_Map_Files(void)

Purpose: This function enables the user to change the map files.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

15. Function _Print_Map

Declaration: void _Print_Map(void)

Purpose: This function prints the screen.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

16. Function Reposition Unit by User

Declaration: void _Reposition_Unit_by_User(void)

Purpose: This function enables the user to reposition a unit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

17. Function Reposition Subunits by User

Declaration: void _Reposition_Subunits_by_User(void)

Purpose: This function enables the user to reposition the subitems in a unit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

18. Function Reposition Subunits by Rules

Declaration: void _Reposition_Subunits_by_Rules(void)

Purpose: This function repositions the subitems of a unit, using the rules stored in the knowledge base.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

19. Function _Define_Unit

Declaration: void _Define_Unit(void)

Purpose: This function enables a user to define a unit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

20. Function _Copy_Unit

Declaration: void _Copy_Unit(void)

Purpose: This function enables a user to define a unit by copying data from an existing unit.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

21. Function _Delete_Unit

Declaration: void _Delete_Unit(void)

Purpose: This function enables the user to delete a unit from a unit file.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

22. Function _Reposition_FEBA

Declaration: void _Reposition_FEBA(void)

Purpose: This function enables the user to reposition the FEBA.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

23. Function _Display_Unit

Declaration: void _Display_Unit(void)

Purpose: This function enables the user to obtain a list of all of the units in the specific unit file and to display one of them at a time.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

24. Function _Output_Units

Declaration: void _Output_Units(void)

Purpose: This function enables the user to output a formatted list of all of the units in either the generic unit file or the specific unit file to the printer or to a file.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions: None.

W. Function File s03zdemo.c: Run Demo Script Function

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains the function _run_demo_script, which runs a demonstration of the Scenarist. In order to run this demo, the program has to be recompiled with the "demo" option turned on in the header file s03gdemo.h (i.e., replace the statement "#define DEMO_TURNED_ON 0" with the statement "#define DEMO_TURNED_ON 1".

Functions Included in File:
_run_demo_script

Definitions of Global Variables: None.

2. Function _run demo script

Declaration: void _run_demo_script(void)

Purpose: This function runs a demonstration of the Scenarist.

Input Parameters: None.

Return Values: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

int ibox: index over the menu boxes.

int returnvalue: index of selected menu box.

X. Function File s03xcomp.c: Map Compression Program

1. Function File Purpose, List of Included Functions, and Definitions of Global Variables

Purpose: This file contains the function used to generate lower-resolution maps from higher-resolution maps.

Functions Included in File:

main

_readmapheadernongraphic
_writemapheader

The program main "compresses" a high resolution map, by "averaging" contiguous blocks of cells, or by computing an indicator variable that indicates the presence of specific attribute values. The average is either the arithmetic average or the mode. The arithmetic average should be used for continuous data (e.g., elevation), and the mode or indicator variable for discrete (categorical) data (e.g., terrain type). For averaging, the presence of any "no data" code results in a nodata code for the aggregate. For the indicator variable, a nodata code results if there is any nodata code in the block and the specified values are not present in the block.

Definitions of Global Variables:

struct mapinfo mapinf[2][3]: symbolically, mapinf[map type][map index}. Array containing map information for a discrete-data cell map (map type = 0, in which case the map index codes are 0 = terrain-type, 1 = road availability, and 2 = TRAILBLAZER accessibility) or a continuous-data cell map (map type = 1, in which case the map index code is 0 = elevation).

int xdisci[1000][10]: array for storing categorical data counts to determine mode.

int xconti[1000]: array for storing sums to determine mean.

int discdata: variable used to store discrete data.

int contdata: variable used to store continuous data.

int maximum: maximum frequency count of values in a block of cells to be aggregated (used in computation of mode).

int mode: mode of the set of values in a block of cells to be aggregated.

int mean: mean of a set of values in a block of cells to be aggregated.

int nrows, ncols: number of rows and columns in the original (high-resolution) map file.

int ncats: number of data categories for the map (discrete data).

int ncomp: number of adjacent cells to aggregate (i.e., a square block of cells of size ncomp by ncomp will be aggregated).

int nrowsnew, ncolsnew: number of rows and columns in the aggregated (lower-resolution) map file.

int irow, icol, icat, irownew, icolnew: indices over nrows, ncols, ncats, nrowsnew, ncolsnew.

char hiresfiln[13]: high-resolution file name.

char lowresfiln[13]: low-resolution file name.

FILE *hiresfilp, *lowresfilp, *geodfilp, *geocfilp: pointers to the high-resolution file, the low-resolution file, the discrete-data map file, and the continuous-data map file.

char inputch, inputchind: variables used to store characters input by the user from the keyboard.

int ival: index over the number of attribute values to check for, in the computation of an indicator variable.

int nvals: the number of attribute values to check for, in the computation of an indicator variable.

int val[100]: the attribute values to check for, in the computation of an indicator variable.

int nodata[1000]: array of indicator variables that indicate whether a "no data" value occurred in a set of cells to be aggregated by averaging.

char nodatacode: indicator variable that indicates whether the value "0" in the file indicates "no data."

2. Function main

Declaration: void main(void)

Purpose: Described above.

Input Parameters: None.

Return Value: None.

Global Variables Modified: None (all global variables used as local variables).

Files Modified: low-resolution file created and filled with low-resolution map data.

Variable Definitions: None.

3. Function _readmapheadernongraphic

Declaration: void _readmapheadernongraphic(char measurementlevel)

Purpose: This function reads a map header from a map file. This function performs essentially the same functions as the function _readmapheader. Some of the differences between these two functions are the following. This function uses text input-output instead of graphics text output. The mapinf array here has a single dimension (map type), whereas in _readmapheader it has two (map type and map index). The function _readmapheader has three input parameters (map type, map index, file pointer), whereas this function has only one.

Input Parameters:

measurementlevel: variable that indicates the data type; 0 for discrete, or categorical (terrain-type, road availability) data, 1 for continuous (elevation) data.

Return Value: None.

Global Variables Modified: None.

Files Modified: None.

Variable Definitions:

char chdata: variable used to store character data.

int ncats: number of data categories.

int itype: data type indicator; the index of the first dimension of the mapinf array; 0 for discrete data, 1 for continuous data.

char line[80]: array used to store text to be printed on the screen.

FILE *datafilp: pointer to the map file.

4. Function _writemapheader

Declaration: void _writemapheader(char measurementlevel)

Purpose: This function writes a map header to a map file.

Input Parameters:

measurementlevel: variable that indicates the data type; 0 for discrete, or categorical (terrain-type, road availability) data, 1 for continuous (elevation) data.

Return Value: None.

Global Variables Modified: None.

Files Modified: The low-resolution map file (output from main).

Variable Definitions:

int ncats: number of data categories.

int itype: data type indicator; the index of the first dimension of the mapinf array; 0 for discrete data, 1 for continuous data.

char line[80]: array used to store text to be printed on the screen.

FILE *datafilp: pointer to the map file.

References

The following five texts were the sources for many of the graphics, windows, and mouse routines. In the Scenarist listing, they are referred to as "Young," "Stevens," "Ezzell," "Johnson," and "Schildt."

1. Young, Michael J., Systems Programming in Microsoft C, Sybex, Inc., San Francisco, 1989.

2. Stevens, Roger T., Graphics Programming in C, M&T Books, Redwood City, CA, 1988.

3. Ezzell, Ben, Graphics Programming in TurboC2.0, Addison-Wesley, New York, 1989.

4. Johnson, Nelson, Advanced Graphics in C: Programming and Techniques, Osborne McGraw-Hill, Berkeley, 1987.

5. Schildt, Herbert, Teach Yourself C, Osborne McGraw-Hill, New York, 1990. (This text is also a helpful general reference for the C programming language.)

The current version of the Scenarist was programmed using Microsoft C. The following texts are available from Microsoft.

6. C for Yourself, Microsoft Press, Belleview, WA, 1990.

7. Microsoft C Advanced Programming Techniques, Microsoft Press, Belleview, WA, 1990.

8. Microsoft Quick C Compiler: Programmer's Guide, Microsoft Press, Belleview, WA, 1990.

9. Petzold, Charles, Programming Windows, Microsoft Press, 1990. (We decided against using the Microsoft Windows Software Development Kit (SDK), but this text was very helpful in getting started with the SDK.)

The following texts were helpful in using Microsoft C.

10. Feibel, Werner, Using Microsoft C, Osborne McGraw-Hill, New York, 1989.

11. Kelly-Bootle, Stan, Mastering Quick C, Sybex, Inc., San Francisco, 1989.

The following is a reference for the ROM-BIOS ("Read Only Memory -- Basic Input Output System" -- the low-level software used to control the hardware input/output on the 80x86 family of microprocessors). It is helpful in understanding the nature of the low-level system commands used to access information about system resources and to control these resources directly (i.e., without going through "high-level" C language commands, such as "putc" or "printf"). These commands (calls to the function int86) are used many times in the graphics and printer interface routines extracted from texts 1-5. A major use of these routines was in the routine used to print a Scenarist screen display on a HP LaserJet printer. The "Print Screen" button on the keyboard of 80x86 systems is set up to print a text screen on a laser printer, but it will not print a graphics screen. (It can be reset using a certain MS-DOS command to print a graphics screen on certain IBM printers, but we needed a print-screen routine that would work for any printer operating in an HP LaserJet emulation mode.) In addition to BIOS system calls, several printer interface routines use MS-DOS system calls (calls to function bdos). (These calls are not as "low level" as the BIOS calls.)

12. System BIOS for IBM PC/XT/AT Computers and Compatibles, Phoenix Technologies, Ltd., Addison-Wesley Publications, Reading, MA, 1989.