RECEIVED
OCT 13 1999
OSTI

ANL/ESD/TM-146

Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS):

Programmer's Manual

Argonne National Laboratory

Operated by The University of Chicago,
under Contract W-31-109-Eng-38, for the

United States Department of Energy

**U.S. Army Corps of Engineers
Construction Engineering Research
Laboratories (USACERL)
USACERL-98/127**

**US Army Corps
of Engineers**
Construction Engineering
Research Laboratories

Argonne National Laboratory

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States Government, and operated by the University of Chicago under the provisions of a contract with the Department of Energy.

This technical memo is a product of Argonne's Energy Systems (ES) Division. For information on the Division's scientific and engineering activities, contact:

Director, Energy Systems Division
Argonne National Laboratory
Argonne, Illinois 60439-4815
Telephone (630) 252-3724

Disclaimer

This report was prepared by Argonne National Laboratory, operated by the University of Chicago on behalf of the U.S. Department of Energy (DOE), as an account of work sponsored by the Strategic Environmental Research and Development Program (SERDP). Neither members of the University of Chicago, DOE, the U.S. Government, nor any person acting on their behalf:

- a. Makes any warranty or representation, express or implied, with respect to the use of any information, apparatus, method, or process disclosed in this report or that such use may not infringe privately owned rights; or
- b. Assumes any liabilities with respect to the use of, or damages resulting from the use of, any information, apparatus, method, or process disclosed in this report.

Reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (423) 576-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS):

Programmer's Manual

by

*Z. Li, *P.J. Sydelko, M.C. Vogt, C.M. Klaus, *K.A. Majerus, and R.C. Sundell

Center for Environmental Restoration Systems, Energy Systems Division,
Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439

June 1998

Work sponsored by Strategic Environmental Research and
Development Program (SERDP)
Arlington, Virginia

*Li and Sydelko are affiliated with Argonne's Decision and Information Sciences Division; Majerus is affiliated with the U.S. Army Corps of Engineers, Construction Engineering Research Laboratories (USACERL), Champaign, Illinois.

About IDLAMS

As funding for conservation continues to decrease, many land managers are facing a challenge: to balance their goals of providing multiple land uses, while complying with natural resource regulations and minimizing negative environmental impacts. Actions to alleviate one problem often exacerbate others. A more integrated approach to land use planning and natural resource management is needed. To meet this need, Argonne National Laboratory and the U.S. Army Construction Engineering Research Laboratories have developed the computer-based Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS).

An integrated, dynamic modeling and decision-support system has been developed to provide a long-term strategic approach for land resource management.

IDLAMS supports the multiple objectives of sustaining natural resources, facilitating appropriate land use, and complying with regulations. This integrated, dynamic modeling approach is a promising tool that can help federal, state, and private organizations fulfill their land stewardship requirements while balancing multiple management objectives and supporting their primary missions.

All too often, decision-makers face critical land management decisions without sufficient information, such as thorough environmental data; information about other, competing objectives; and knowledge of land use requirements. IDLAMS, as developed for military use, consists of ecological, erosion, and training subroutines, along with advanced decision-support techniques, all linked with a

core vegetation dynamics model that uses geographic information systems, remote sensing, and field inventory data. A user-friendly computer interface allows the land manager to operate this predictive, decision-support tool without the need for substantial computer or environmental modeling expertise. The key benefit of IDLAMS is that it can help land managers in three important ways: (1) strive toward multiple land use objectives using trade-off analysis, (2) evaluate the cost and economics

of viable alternatives for managing land use, and (3) incorporate "what if" scenarios into decision-making. IDLAMS can also speed up responses to land-use management issues, improve environmental compliance, and help balance diverse land use needs.

Organizations interested in learning more about IDLAMS should contact one of the principal investigators listed on the following page.

A user-friendly computer interface allows the land manager to operate this predictive, decision-support tool without the need for substantial computer or environmental modeling expertise.

POINTS OF CONTACT

Ms. Sydelko, and Ms. Majerus serve as points of contact for this project. For further information, please contact:

Ms. Pamela J. Sydelko
Argonne National Laboratory
Advanced Computer Applications Group
Decision and Information Sciences Division, Building 900
9700 South Cass Avenue
Argonne, Illinois 60439-4832

Phone: (630) 252-6727
E-mail: sydelkop@smtplink.dis.anl.gov

Ms. Kimberly A. Majerus
U.S. Army Corps of Engineers
Construction Engineering Research Laboratories
Natural Resource Assessment and Management Division
Land Management Laboratory
USACERL CECER LL-N
P.O. Box 9005
Champaign, Illinois 61826-9005

Phone: (217) 352-6511
E-mail: k-majerus@cecer.army.mil

Dr. Sundell, the IDLAMS Project Manager, may be contacted at:

Dr. Ronald C. Sundell
Director of Environmental Sciences Program
Department of Geography and Biology
Northern Michigan University
1401 Presque Isle
Marquette, MI 49855-5342

Phone: (906) 227-1359
E-mail: rsundell@nmu.edu

PREFACE

The Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS) is a prototype, integrated land management technology developed through a joint effort between Argonne National Laboratory (ANL) and the U.S. Army Corps of Engineers Construction Engineering Research Laboratories (USACERL). Dr. Ronald C. Sundell, Ms. Pamela J. Sydelko, and Ms. Kimberly A. Majerus were the principal investigators (PIs) for this project. Dr. Zhian Li was the primary software developer. Dr. Jeffrey M. Keisler, Mr. Christopher M. Klaus, and Mr. Michael C. Vogt developed the decision analysis component of this project. It was developed with funding support from the Strategic Environmental Research and Development Program (SERDP), a land/environmental stewardship research program with participation from the U.S. Department of Defense (DoD), the U.S. Department of Energy (DOE), and the U.S. Environmental Protection Agency (EPA).

IDLAMS predicts land conditions (e.g., vegetation, wildlife habitats, and erosion status) by simulating changes in military land ecosystems for given training intensities and land management practices. It can be used by military land managers to help predict the future ecological condition for a given land use based on land management scenarios of various levels of training intensity. It also can be used as a tool to help land managers compare different land management practices and further determine a set of land management activities and prescriptions that best suit the needs of a specific military installation.

The IDLAMS project documentation consists of four reports:

1. IDLAMS Installation Manual,
2. IDLAMS Programmer's Manual*,
3. IDLAMS User's Manual, and
4. IDLAMS Final Report.

*This programmer's manual describes the system structure, its components, and the relationship between components. A detailed explanation is also furnished for each component of the system to provide useful information for changing system structure, adding new components, or modifying the existing models. More detailed information about IDLAMS development, methodology, or use can be found in the other three reports.

CONTENTS

<i>About IDLAMS</i>	<i>iii</i>
POINTS OF CONTACT.....	<i>iv</i>
PREFACE	<i>v</i>
ACKNOWLEDGMENTS	<i>x</i>
ACRONYMS.....	<i>xi</i>
ABSTRACT	1
PROGRAMMER'S MANUAL	1
1 INTRODUCTION.....	1
2 SYSTEM COMPONENTS.....	4
2.1 Graphical User Interface - Tcl/Tk.....	5
2.2 GRASS GIS.....	7
2.3 IDLAMS Environment	15
2.4 Data Directories and Management.....	16
2.5 GNUPLOT	17
3 VEGETATION DYNAMICS MODEL.....	18
3.1 Input File.....	18
3.2 Model Execution	19
4. WILDLIFE HABITAT SUBMODELS	28
5 EROSION SUBMODEL.....	29
6 SCENARIO EVALUATION MODEL	30
6.1 IDLAMS Succession Models.....	30
6.2 Decision-Making Strategy	32
6.3 Algorithm and Data Representation.....	33
6.4 Algorithm Implementation	34
6.5 User Input Required.....	34
6.6 Model Output Produced.....	34
7 REFERENCES	35

FIGURES

Figure 1. IDLAMS System Diagram	3
Figure 2 IDLAMS Directory Hierarchical Diagram	4
Figure 3. Sample Input for the Vegetation Dynamics Model	18
Figure 4. Flow Diagram of Vegetation Dynamics Model Main Function	19
Figure 5. Flow Diagram of Prescribed Burning Function	20
Figure 6. Flow Diagram of Vegetation Succession Function.....	21
Figure 7. Flow Diagram of Training Function	22
Figure 8. Flow Diagram of Planting Function.....	23
Figure 9 Successional Transitions in the Life of a Landcover Map Cell.....	31
Figure 10 Landcover and Training Areas Map	32
Figure 11 Typical Single-Attribute Utility Functions Graph.....	33
Figure 12 Typical Evaluation Model Statistics Report.....	34

TABLES

Table 1 Global Files and their Corresponding Models	6
Table 2 Northern Bobwhite Habitat Model Function Indexes	6
Table 3 Variables Defined in .idlamsrc File	16
Table 4 Variables Defined by Function grass.....	16
Table 5 Vegetation Categories Defined in Header File veg.h	26
Table 6 Definitions of Variables for Map Segmentation Functions	27

ACKNOWLEDGMENTS

This report was funded by the Strategic Environmental Research and Development Program (SERDP). Special thanks go to Dr. Femi Ayorinde, SERDP and Conservation Program Manager, for his assistance and support of this project. In addition, special appreciation is also expressed to Craig Phillips, Dave Jones, Malcom Ponte, Herb Abel, and the Natural Resources staff at Ft. Riley, Kansas, for their help and cooperation in the development and testing of IDLAMS.

NOTATION

AEC	Army Environmental Center
ANL	Argonne National Laboratory
ASA (ILE)	Assistant Secretary of the Army, Installations Logistics and Environment
ASCII	American Standard Code for Information Interchange
ATTACC	Army Training and Testing Area Carrying Capacity
DoD	U.S. Department of Defense
DOE	U.S. Department of Energy
DTMs	Digital terrain maps
EPA	U.S. Environmental Protection Agency
ES	Energy Systems
GIS	Geographic information system
GRASS	Geographical Resources Analysis Support System
GUI	Graphical user interface
IDLAMS	Integrated Dynamic Landscape Analysis and Modeling System
ITAM	Integrated Training Area Management
LCTA	Land Condition Trend Analysis
MAUF	Multiple-attribute utility function
MIMs	Maneuver Impact Mile
ODCSOPS	Office of the Deputy Chief of Staff for Operations and Plans
PC	Personal computer
PI	Principal investigator
RUSLE	Revised Universal Soil Loss Equation
SAUF	Single-attribute utility function
SERDP	Strategic Environmental Research and Development Program
Tcl/Tk	Tool command language/tool kits
USACERL	U.S. Army Corps of Engineers, Construction Engineering Research Laboratories
USACOE	U.S. Army Corps of Engineers

INTEGRATED DYNAMIC LANDSCAPE ANALYSIS AND MODELING SYSTEM (IDLAMS):

PROGRAMMER'S MANUAL

by

Z. Li, P.J. Sydelko, M.C. Vogt, C.M. Klaus, K.A. Majerus, and R.C. Sundell

ABSTRACT

This manual introduces readers to the Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS) from the programming perspective. This manual uses the Fort Riley, Kansas, installation of IDLAMS as an example. The system structure, its components, and the relationship between components are explained. A detailed explanation is also furnished for each component of the system to provide useful information for changing system structure, adding new components, or modifying the existing models. The intended users of this manual are those responsible for installing, maintaining, and/or enhancing the IDLAMS system. It is assumed that users possess a good working knowledge of C programming language(s), the UNIX operating system, UNIX shell script languages, the GRASS GIS, and the Tcl/Tk GUI programming language. Copies of the GRASS Programmer's Manual and GRASS User's Reference Manual are essential and should be very helpful if modifications of the models are desired.

1 INTRODUCTION

The Integrated Dynamic Landscape Analysis and Modeling System (IDLAMS) operates under the UNIX operating system and is built upon the Geographical Resources Analysis Support System (GRASS) geographic information system (GIS) (USACERL 1993b). The vegetation dynamics model is the core of IDLAMS; its parts are either written in the C language or built with existing GRASS functions. External models or applications can be written in other languages or can exist in other GISs, provided that they accept ASCII array landcover maps as input and can export ASCII arrays back into IDLAMS. The entire system is integrated with a graphical user interface (GUI) that uses the Tcl/Tk programming language. To use this system, a UNIX workstation running with a SunOS 4.1.3 operating system or an IBM-compatible personal computer (PC) running with a UNIX operating system is required. Acceptable performance was achieved on a Sun SPARC Classic with 32 MB RAM, 2 GB disk space, and a 19-in., 8-bit video display.

IDLAMS has modularity because the GRASS GIS system it builds on is a highly portable, modularized system. This means a user can add new model(s) to the system to enhance its functionality. IDLAMS is designed to be a core system that requires customization when installed

at a new site. To modify the vegetation dynamics model or add a new one, developers can use standard GRASS command format or build them with existing GRASS functions. The structure used for the Fort Riley, Kansas, vegetation dynamics model can be used and modified to fit a new site's ecosystem and natural resource management practices. Because C is the programming language for the GRASS system, all new vegetation dynamics function(s) must be implemented in the C language. A standard C compiler, normally available with a UNIX operating system, is necessary to compile the GRASS functions and the associated library. The interactions between models are GIS map layers. Additional models can be interfaced with IDLAMS simply by writing them to allow the landcover input to come from an ASCII array (exported by IDLAMS using GRASS function `r.out.ascii`). If desired, the external model can also pass the final results to IDLAMS as an ASCII array for viewing or passing to additional models. The Tcl/Tk system, versions Tcl 7.3 and Tk 3.6, used to build the GUI is in the public domain (Ousterhout 1995). The Tcl/Tk package is included in the IDLAMS delivery package. IDLAMS requires both the "wish" and "expect" components of Tcl/Tk. Developers who wish to add external models to IDLAMS can use Tcl/Tk to add new GUIs to the IDLAMS interface to provide for user input and execution for external models.

IDLAMS currently consists of four major components:

1. The vegetation dynamics model,
2. Wildlife habitat submodels,
3. The erosion submodel, and
4. A scenario evaluation model.

The vegetation dynamics model is the core of the entire IDLAMS system. The wildlife habitat submodels and the erosion submodel all use the resultant vegetation cover map from the vegetation dynamics model as their landcover condition input to determine predicted wildlife habitat condition and soil erosion status. The scenario evaluation model evaluates the effectiveness of the selected land management practices on the basis of results from the vegetation dynamics model, wildlife submodels, and erosion submodel. The evaluations are made by using a multiple-attribute utility function (MAUF). The MAUF is constructed by a weighted sum of an array of single-attribute utility functions (SAUFs). Each SAUF is a measure of the land stewardship in one aspect (for example, notices of violation).

The entire system is integrated by using X-Windows and GUIs (written in the Tcl/Tk programming language) to provide a user-friendly environment. Figure 1 presents a system diagram of the IDLAMS system.

Figure 1 IDLAMS System Diagram

2 SYSTEM COMPONENTS

The IDLAMS technology consists of links between various components, including software, data sets, files, and shell scripts. An overall representation of the major components of IDLAMS is provided in Figure 2.

The IDLAMS system consists of two major components, the GRASS GIS system and the IDLAMS models. Additional supporting packages include the Tcl/Tk ("expect," including "wish") language and GNUPLOT graphic utility. All of these packages are distributed together with IDLAMS for the user's convenience. However, users might not have to install these supporting packages if they are already up and running on the host machine. Details of the installation can be found in the IDLAMS Installation Manual.

Figure 2 IDLAMS Directory Hierarchical Diagram

2.1 Graphical User Interface - Tcl/Tk

IDLAMS is integrated by means of a graphical user interface (GUI) written in the Tcl/Tk programming language. The file that serves as the front driver of the system is named `idlams`. This driver pops up initial IDLAMS Tcl/Tk GUIs and turns control over to different subscreens upon the user's selection of submodel. The shell scripts for the wildlife and erosion submodels, `bw.habitat.sh`, `hs.habitat.sh`, `pc.habitat.sh`, `pmc.habitat.sh`, `rf.habitat.sh`, `ero.sh`, and the habitat area clump function `clump2class` all reside in this directory. If sites are developing new models or using other applications (e.g., Arc/Info) to generate data on wildlife, erosion, etc., the developer can add new button(s) to the Tcl/Tk graphical user interface and make the callback directly connected to new model(s). All other Tcl/Tk code is stored in the `tcl_lib` subdirectory. The system hierarchical block diagram is shown in Figure 2.

There are thirteen subdirectories in the `tcl_lib` subdirectory. They are (1) `bwtcl`, (2) `erotcl`, (3) `grass`, (4) `hstcl`, (5) `idlams`, (6) `lib`, (7) `opttcl`, (8) `pctcl`, (9) `pmctcl`, (10) `prjlib`, (11) `reglib`, (12) `rftcl`, and (13) `vegtcl`. Each subdirectory contains functions used to create a set of screens to prompt (navigate) the user to enter necessary input parameters for a specific model.

The `bwtcl` directory contains Tcl/Tk script files for the northern bobwhite submodel input interface. Similarly, subdirectories `erotcl`, `hstcl`, `pctcl`, `pmctcl`, `rftcl`, `opttcl`, and `vegtcl` contain Tcl/Tk script files used to construct Tcl/Tk interfaces for the soil erosion submodel, the Henslow's sparrow habitat submodel, the greater prairie chicken habitat submodel, the prairie mole cricket habitat submodel, the regal fritillary habitat submodel, the scenario evaluation model, and the vegetation succession model, respectively.

The subdirectory `/grass` contains scripts that set up the grass environment variables and makes them available to the UNIX operating environment. For scripts in all other subdirectories, the scripts in `reglib` provide a means for the user to reset the GRASS region (USACERL 1993a,b).

The Tcl/Tk scripts in `prjtcl` are designed to open and save simulations that were previously developed. Finally, `lib` and `idlams` collect all functions shared by more than one model. Programmers can develop new public utility functions as necessary and put them in the `lib` directory.

Several very important issues must be addressed in writing Tcl/Tk code. The first is that in Tcl/Tk, all variables are defined by default as local. To make a variable visible to other functions or even to another part of the same function, programmers must declare variables explicitly as "global" so they can be accessed outside the scope in which the variables are defined. This declaration can either be made at the beginning of the function or made outside of the function and then included. To keep code clean, global variables are declared in `*.global` files, where `*` denotes the name of a model. The current version has eight models. Hence, there are nine global variable declaration files, one for each model and the file `idlams.global` for the whole system. The names of these global files and their corresponding models are provided in Table 1.

Table 1 Global Files and their Corresponding Models

Name of Global File	IDLAMS Model
Bw.global	Northern Bobwhite habitat submodel
Ero.global	Erosion model
Hs.global	Henslow's sparrow habitat submodel
Idlams.global	IDLAMS system variables
Opt.global	Optimization/decision support
Pc.global	Greater prairie chicken habitat submodel
Pmc.global	Prairie mole cricket habitat submodel
Rf.global	Regal fritillary habitat submodel
Veg.global	Vegetation succession model

The second issue in writing Tcl/Tk is the scope of functions. To make a function callable by other function(s), a cross reference or index of this function must be created. Table 2 shows the indices for the northern bobwhite habitat model.

Table 2 Northern Bobwhite Habitat Submodel Function Indexes

```
# Tcl autoload index file, version 2.0
# This file is generated by the "auto_mkindex" command
# and sourced to set up indexing information for one or
# more commands. Typically each line is a command that
# sets an element in the auto_index array, where the
# element name is the name of a command and the value is
# a script that loads the command.

set auto_index(SaveBWProject) "source $dir/project.tcl"
set auto_index(ReadBWProject) "source $dir/project.tcl"
set auto_index(NewBwProject) "source $dir/project.tcl"
set auto_index(BwMakeHabitatMap) "source
$dir/makehabmap.tcl"
set auto_index(NhszDlg) "source $dir/nbrhdsz.tcl"
set auto_index(GetNHSz) "source $dir/nbrhdsz.tcl"
set auto_index(BwDIDlg) "source $dir/displayinp.tcl"
set auto_index(BwDisplayInputs) "source
$dir/displayinp.tcl"
set auto_index(BwGetCurMap) "source $dir/display.tcl"
set auto_index(BwGetMapList) "source $dir/display.tcl"
set auto_index(CreateBwMenu) "source $dir/Bwproc.tcl"
set auto_index(CreateBwMainWindow) "source $dir/Bwproc.tcl"
set auto_index(EnableBwItems) "source $dir/Bwproc.tcl"
set auto_index(BwTcl) "source $dir/Bwproc.tcl"
set auto_index(SetupBwArgs) "source $dir/args.tcl"
```

2.2 GRASS GIS

GRASS is a highly portable and modularized GIS system. It operates on a workstation or PC running a UNIX operating system. GRASS consists of two parts, a system driver and a collection of modularized GIS functions. The driver is responsible for setting up the GRASS environment. Each individual GRASS function accomplishes, depending on the input data, a specific task each time it is called. A set of GRASS library functions provides basic tools for GIS map manipulations and spatial analyses. These GRASS functions and library functions are all accessible to programmers for developing new functions. The only requirement for accessing these functions is to follow the GRASS data type convention. Programmers and users can find detailed information in GRASS programmer's and user's manuals (USACERL 1993a,b).

2.2.1 GRASS 4.1.3

The version of GRASS used in IDLAMS is GRASS 4.1.3. It is a floating-point version of GRASS. The category values stored in each map later can take on any real *floating-point* value, whereas with past GRASS versions could only take on *integer values*. The following are the definitions of the major structures and data types used in GRASS.

It is important to note that only the variables and data types defined in the `GIS.h` file are listed below. Developers can define their own variables and data types as necessary. To access these library functions, the header file `<gis.lib>` must be included in the application function. Developers must also guarantee that new variable names are distinct from those defined in the library.

```
#ifndef __GRASS_GIS_LIB__
#define __GRASS_GIS_LIB__
static char *GRASS_copyright = "GRASS (TM) Public Domain
    Software" ;
#define MAXEDLINES  50
#define RECORD_LEN  80
#define NEWLINE      '\n'
#define RECLASS_TABLE 1
#define RECLASS_RULES 2
#define RECLASS_SCALE 3

#define METERS      1
#define FEET        2
#define DEGREES     3

typedef int CELL;
typedef double DCELL;
typedef float FCELL;
extern CELL CELL_NODATA;
```

/*

```

enum RASTER_MAP_TYPE{ CELL_TYPE = 0,
                      FCELL_TYPE = 1,
                      DCELL_TYPE = 2
                      };
doesn't compile on sun :( ?
*/

typedef int RASTER_MAP_TYPE;
#define CELL_TYPE 0
#define FCELL_TYPE 1
#define DCELL_TYPE 2

#define PROJECTION_XY 0
#define PROJECTION_UTM 1
#define PROJECTION_SP 2
#define PROJECTION_LL 3
#define PROJECTION_OTHER 99

#define PROJECTION_FILE "PROJ_INFO"
#define UNIT_FILE "PROJ_UNITS"

struct Cell_head
{
    int format ; /* max numer of bytes per cell minus 1 */
    int compressed ; /* 0 = uncompressed, 1 = compressed, -1 pre
3.0 */
    int rows, cols ; /* number of rows and columns in the data */
    int proj ; /* Projection (see #defines above) */
    int zone ; /* Projection zone */
    double ew_res ; /* East to West cell size */
    double ns_res; /* North to South cell size */
    double north; /* coordinates of layer */
    double south; double east;
    double west;
} ;

struct _Color_Rule_
{
    struct
    {
        DCELL value;
        unsigned char red,grn,blu;
    } low, high;
    struct _Color_Rule_ *next;
    struct _Color_Rule_ *prev;
};

struct _Color_Info_
{

```

```

    struct _Color_Rule_ *rules;
0    int n_rules;
    struct
    {
        unsigned char *red;
        unsigned char *grn;
        unsigned char *blu;
        unsigned char *set;
        int nalloc;
        int active;
        } lookup;
    struct
    {
        DCELL *vals;
        /* pointers to color rules corresponding to the intervals
        between vals */
        struct _Color_Rule_ **rules;
        int nalloc;
        int active;
        } fp_lookup;
        DCELL min, max;
};

struct Colors
{
    int version;      /* set by read_colors: -1=old,1=new */
    DCELL shift;
    int invert;
    int is_float;     /* defined on floating point raster data? */
    int null_set;     /* the colors for null are set? */
    unsigned char null_red, null_grn, null_blu;
    int undef_set;    /* the colors for cells not in range are set?
    */
    unsigned char undef_red, undef_grn, undef_blu;
    struct _Color_Info_ fixed, modular;
    DCELL cmin, cmax;
};

struct Reclass
{
    char name[50];    /* name of cell file being reclassified */
    char mapset[50];  /* mapset in which "name" is found */
    int type/* type of reclass */
    int num; /* size of reclass table */
    CELL min,max;    /* table range */
    CELL *table ;/* reclass table */
} ;

struct FPReclass_table
{

```

```

    DCELL dLow, dHigh;    /* domain */
    DCELL rLow, rHigh;    /* range   */

};

/* reclass structure from double to double
   used by r.recode to reclass between types:
   int to double, float to int,...
*/

struct FPreclass {

    int defaultDRuleSet /* 1 if default domain rule set */
    int defaultRRuleSet; /* 1 if default range rule set */
    int infiniteLeftSet; /* 1 if negative infinite interval rule
        exists */
    int infiniteRightSet; /* 1 if positive infinite interval rule
        exists */
    int rRangeSet; /* 1 if range (i.e. interval) is set */
    int maxNofRules;
    int nofRules;
    DCELL defaultDMin, defaultDMax;
        /*default domain minimum and maximum value */
    DCELL defaultRMin, defaultRMax;
        /* default range minimum and maximum value */
    DCELL infiniteDLeft, infiniteDRight; /* neg infinite rule */
    DCELL infiniteRLeft, infiniteRRight; /* pos infinite rule */
    DCELL dMin, dMax; /* minimum and maximum domain values in
        rules */
    DCELL rMin, rMax; /* minimum and maximum range values in rules
        */
    struct FPreclass_table *table;

};

struct Quant_table
{
    DCELL dLow, dHigh;
    CELL cLow, cHigh;
};

```

```

struct Quant
{
    int truncate_only;
    int round_only;
    int defaultDRuleSet;
    int defaultCRuleSet;
    int infiniteLeftSet;
    int infiniteRightSet;
    int cRangeSet;
    int maxNofRules;
    int nofRules;
    DCELL defaultDMin, defaultDMax;
    CELL defaultCMin, defaultCMax;
    DCELL infiniteDLeft, infiniteDRight;
    CELL infiniteCLeft, infiniteCRight;
    DCELL dMin, dMax;
    CELL cMin, cMax;
    struct Quant_table *table;
    struct
    {
        DCELL *vals;
        /* pointers to Quant rules corresponding to the intervals
        between vals */
        struct Quant_table **rules;
        int nalloc;
        int active;
        DCELL inf_dmin, inf_dmax;
        CELL inf_min, inf_max;
        /* all values smaller than inf_dmin become inf_min */
        /* all values larger than inf_dmax become inf_max */
        /* inf_min and/or inf_max can be NULL if there are no inf
        rules */
    } fp_lookup;
};

struct Categories
{
    CELL ncats; /* total number of categories */
    CELL num; /* the highest cell values. Only exists
    for backwards compatibility = (CELL)
    max_fp_values in quant rules */
    char *title; /* name of data layer */
    char *fmt; /* printf-like format to generate labels */
    float m1; /* Multiplication coefficient 1 */
    float a1; /* Addition coefficient 1 */
    float m2; /* Multiplication coefficient 2 */
    float a2; /* Addition coefficient 2 */

```



```

    struct Quant q;    /* rules mapping cell values to index in
                        list of labels */
    char **labels;    /* array of labels of size num */
    int * marks; /* was the value with this label was used? */
    int nalloc;
    int last_marked_rule ;

/* NOTE: to get a rule corresponding to cats.labels[i], use
G_get_ith_c/f/d_raster_cat (pcats, i, val1, val2)
it calls
G_quant_get_ith_rule(&cats->q, i, val1, val2, &index,
&index);
and idex ==i, because rule is added at the same time as a
label, and quant rules are never reordered. Olga apr,95 */
} ;

struct History
{
    char    mapid[RECORD_LEN];
    char    title[RECORD_LEN];
    char    mapset[RECORD_LEN];
    char    creator[RECORD_LEN];
    char    maptype[RECORD_LEN];
    char    datsrc_1[RECORD_LEN];
    char    datsrc_2[RECORD_LEN];
    char    keywrd[RECORD_LEN];
    int     edlinecnt;
    char    edhist[MAXEDLINES][RECORD_LEN];
} ;

struct Cell_stats
{
    struct Cell_stats_node
    {
        int idx;
        long *count;
        int left;
        int right;
    } *node ;    /* tree of values */
    int tlen ;    /* allocated tree size */
    int N;        /* number of actual nodes in tree */
    int curp;
    long null_data_count;
    int curoffset;
} ;

struct Histogram
{
    int num;
    struct Histogram_list

```

```

    {
        CELL cat;
        long count;
    } *list;
} ;

struct Range
{
    CELL min;
    CELL max;
    int first_time; /* whether or not range was updated */
};

struct FPRange
{
    DCELL min;
    DCELL max;
    int first_time; /* whether or not range was updated */
};

/*
** Structure for I/O of 3dview files (view.c)
*/
struct G_3dview{
    char pgm_id[40]; /* user-provided identifier */
    float from_to[2][3]; /* eye position & lookat position */
    float fov; /* field of view */
    float twist; /* right_hand rotation about from_to */
    float exag; /* terrain elevation exaggeration */
    int mesh_freq, poly_freq; /* cells per grid line, cells
    per polygon */
    int display_type; /* 1 for mesh, 2 for poly, 3 for both */
    int lightson, dozero, colorgrid, shading, fringe, surfonly,
    doavg; /*bools*/
    char grid_col[40], bg_col[40], other_col[40]; /* colors */
    float lightpos[4]; /* easting, northing, height, 1.0 for
    local 0.0 infin */
    float lightcol[3]; /* values between 0.0 to 1.0 for red,
    grn, blu */
    float ambient, shine;
    struct Cell_head vwin;
};

struct Key_Value
{
    int nitems;
    int nalloc;
    char **key;
    char **value;
};

```

```

};

struct Option      /* Structure that stores option info      */
{
    char *key ;      /* Key word used on command line */
    int type ;        /* Option type                    */
    int required ;    /* REQUIRED or OPTIONAL           */
    int multiple ;    /* Multiple entries OK           */
    char *options     /* Approved values or range or NULL */
    char *key_desc;   /* String describing option      */
    char *answer ;    /* Option answer                 */
    char *def ;       /* Where original answer gets saved */
    char **answers ;  /* Option answers (for multiple=YES)*/
    struct Option *next_opt ; /* Pointer to next option struct */
    char *gisprompt ; /* Interactive prompt guidance   */
    int (*checker)() ; /* Routine to check answer or NULL */
    int count;
} ;

struct Flag        /* Structure that stores flag info */
{
    char key ;      /* Key char used on command line */
    char answer ;    /* Stores flag state: 0/1         */
    char *description ; /* String describing flag meaning */
    struct Flag *next_flag ; /* Pointer to next flag struct */
} ;

/* for G_parser() */
#define TYPE_INTEGER 1
#define TYPE_DOUBLE 2
#define TYPE_STRING 3
#define YES 1
#define NO 0

#ifndef FILE
#include <stdio.h>
#endif

#include "gisdefs.h"

#include "datetime.h"
#include "site.h"

struct TimeStamp {
    DateTime dt[2]; /* two datetimes */
    int count;
};

/*__GRASS_GIS_LIB__*/

```

2.2.2 Programming Library

The *GIS Library* is the primary programming library provided with the GRASS GIS system. This library contains functions needed to access the map database. More than a hundred GRASS library functions are available in the GIS library. Developers can use them anywhere in their applications as long as the rules specified in the GRASS Programmer's Manual and the User's Reference Manual are followed. A detailed explanation of each function is beyond the scope of this manual. Users and programmers are advised to see the relevant GRASS manuals and documentation (USACERL 1993a,b).

2.2.3 Segmentation Technique

Although detailed explanations on usage of all of the GRASS functions are provided in the GRASS Programmer's Manual, a few tips on the map segmentation technique might be helpful to programmers, because this technique must be used when a large map layer and multiple map rows are to be manipulated in a model. If the map segmentation technique must be used in a model, attention should be given to the following:

1. The category data must be read as character strings and then converted to corresponding integer or floating-point data;
2. The types of variables designated to receive the category values must be defined as double precision if the map to be read is a floating-point map;
3. The segment buffer must be flushed before the segment file can be reopened;
4. The segment buffer must be flushed before the termination of the function;
5. The program must implement code to remove the segment file created, the library will not remove it automatically;
6. The addresses of the receiving variables (instead of the variables themselves) must be used in reading segmentation files.

Additional information concerning the map segmentation technique is provided in later sections of this manual. For more details on the map segmentation technique, please see the GRASS Programmer's Manual (USACERL 1993a).

2.3 IDLAMS Environment

IDLAMS is built on the GRASS GIS environment. In order to run IDLAMS and GRASS GIS functions, some environment variables must be set at the start up of IDLAMS. An IDLAMS run control file `.idlamsrc` file is produced when IDLAMS is installed. All of the environmental variables necessary to run IDLAMS are defined in this file. Table 3 is an example of the `.idlamsrc` file.

Table 3 Variables Defined in .idlamsrc File

```

GISBASE: /home/grass4.1
GISDBASE: /data0
LOCATION_NAME: riley
MAPSET: DLMS
PAINTER: ppm
DIGITIZER: none
MONITOR: x1

```

Each time an IDLAMS section starts, the driver program (1) reads the parameters, (2) starts a text format input section for the user to make changes if so desired, and (3) exports them as UNIX environment variables. In addition, the user must also be prevented from starting another session, which could disturb the integrity of the GIS database. These two important problems are solved in IDLAMS by the function `grass.tcl`. The IDLAMS system calls this function when the user clicks the CONTINUE button in the first screen. It then picks up the process number, PID, sets up GRASS environment variables, and exports them to the UNIX environment. The parameters set by the function `grass` are listed in Table 4.

Table 4 Variables Defined by Function `grass`

```

PATH (GISDBASE): /data0
LOCATION_NAME: riley
MAPSET: IDLAMS
PAINTER: ppm
DIGITIZER: none
GISLOCK: pid
MONITOR: x1
XDRIVER_LEFT: 650
XDRIVER_TOP: 5

```

The last two items are optional for running GRASS. The purpose of setting these parameters is to define the position of the GRASS monitor.

2.4 Data Directories and Management

IDLAMS works directly with GRASS4.1 GIS maps in a mapset named IDLAMS. The user must set up this mapset before running IDLAMS. Programmers can find the data structure of GRASS mapsets in the GRASS4.1 Programmer's Manual (USACERL 1993a).

Users can import maps in formats other than GRASS GIS either via an ASCII file or directly, by converting the import maps into GRASS format via format converting utility functions (if such functions are available). Similar functions also exist in other GIS packages. In addition, the user can also find useful tools for map format conversion from the GRASS web page at:

<http://www.cecer.army.mil/grass/>

Many GRASS GIS system users have made contributions to this web site.

2.5 GNUPLOT

The IDLAMS technology's scenario evaluation model utilizes the GNUPLOT software (Williams and Kelley 1993). Within IDLAMS, GNUPLOT is used to produce and display graphics, specifically bar charts, as visual displays to the user. Several bar charts are available to the user as a representation of functionality.

It is important to clarify that the information displayed in the IDLAMS scenario evaluation GNUPLOT bar charts serves only as a representation of the functionality of the GNUPLOT software, linked with single-attribute and multiple-attribute utility functions that are based upon the objectives within the scope of a larger decision-making process. The inputs and outputs are not meaningful quantities (values) and currently serve only as placeholders. A programmer, system administrator, or user who is accessing and running the scenario evaluation model should observe how it operates and the functionality that it provides, rather than attempting to actually use the scenario evaluation in making any decision on the land management alternatives under consideration through IDLAMS. Within IDLAMS, GNUPLOT is utilized only for displaying bar charts for the purposes described in the previous section. In summary:

"GNUPLOT is a command-line driven interactive function plotting utility for UNIX, MSDOS, and VMS platforms. The software is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally intended as a graphical program which would allow scientists and students to visualize mathematical functions and data. GNUPLOT handles both curves (2 dimensions) and surfaces (3 dimensions). Surfaces can be plotted as a mesh fitting the specified function, floating in the 3-d coordinate space, or as a contour plot on the x-y plane. For 2-d plots, there are also many plot styles, including lines, points, lines with points, error bars, and impulses (crude bar graphs). Graphs may be labeled with arbitrary labels and arrows, axes labels, a title, date and time, and a key. The interface includes command-line editing and history on most platforms."

(http://science.nas.nasa.gov/people/woo/gnuplot_info.html 1998)

3 VEGETATION DYNAMICS MODEL

The vegetation dynamics model is the central component of IDLAMS. It takes as input an ASCII text file containing information on the run inputs and environment variables, plus directory structure (e.g., resolution, geographic region). The output is a GRASS raster map layer.

3.1 Input File

The input file, which contains all information necessary for the model to run a simulation, can be constructed with any text editor or interface. Figure 3 shows an example of an input file for the vegetation dynamics model.

<u>Line</u>	<u>Description</u>
1. 1992 2007	starting/ending year
2. trn.dist	training map
3. landcover_input_map	landcover map
4. himgt.medtrn.500m.scen	output map
5. 0 1 3 4 5	reclass categories
6. 500000	maneuver impact miles
7. 500	erosion control planting (ac)
8. 500	native planting (ac)
9. 50000	prescribed burning (ac)
10. 2	successional change (forest)
11. 4 15	successional change (early, late)

Figure 3 Sample Input for the Vegetation Dynamics Model

The first line of the input file consists of two items. The first item is the starting year of the simulation; the second item is the ending year. The second line provides the name of the training intensity distribution map. This map contains a floating-point, normalized training intensity value in each pixel and is generated for installations by using methodology established through the Land Based Carrying Capacity Program (Guertin, Rewerts, and Dubois 1997). The third line item in the input file is the initial landcover map, followed on the fourth line by the name of the desired output map. The fifth line of the input file consists of five items, which are the categories the user selects to reclass the original categories in the input landcover map. These five new categories are as follows: (1) Forest, (2) Forest/Shrub/Grass Mixture, (3) Disturbed (training damage), (4) Early Successional, and (5) Late Successional, respectively.

The sixth input line is the Maneuver Impact Miles (MIMs), a measure of the training load calculated as part of the Army Training and Testing Area Carrying Capacity (ATTACC) program (Concepts Analysis Agency 1996). The seventh, eighth, and ninth lines of the input file are the acreages of the land management practices (i.e., erosion control planting, native planting, and prescribed burning, respectively). The tenth and eleventh input lines describe the vegetation succession parameters. Of the three parameters defining vegetation succession, the first is the

probability (in percent) that a grass pixel changes to forest (line 10). The second and third are the number of years it takes for early successional to change into late successional and for damaged to change into early successional grass categories (line 11). The command line format for calling the vegetation dynamics model is:

```
r.veg.change    input_file_name
```

The input file is built by a series of GUI screens to guide the user in entering necessary input data. Cost functions and a display screen have been implemented in the system to dynamically calculate and display the costs of the user-selected land management activities. The regional cost estimates of land management practices can be found in Gebhart and Warren (1995). These costs are held in a file named `cost.tbl` in the IDLAMS main directory. The user and/or programmer can change these numbers by using any text editor. The cost of each activity and the sum of the individual costs are displayed dynamically on the same screen simultaneously. The input of Maneuver Impact Miles (MIMs) provides a measure of the land training impact and is consistent with the trend of carrying capacity studies in the current efforts on Army Training and Testing Area Carrying Capacity (ATTACC) and Integrated Training Area Management (ITAM) research supported by the Army. For further information concerning these programs, contact the Army Environmental Center (AEC) at Aberdeen Proving Ground, Maryland, the Office of the Deputy Chief of Staff for Operations and Plans (ODCSOPS), and the Assistant Secretary of the Army Installations Logistics and Environment (ASA (ILE)) at Headquarters, Department of the Army, Washington, D.C.

3.2 Model Execution

Upon execution of the vegetation dynamics model, the initial age status of the landcover is determined by randomly assigning ages within different landcover types in the user-provided initial landcover map. This is done because the number of years each pixel has been in its current landcover class is unknown. Upon execution of the vegetation dynamics model, these pixels then proceed through succession into new landcover classes, based on what the user has selected as the time intervals between each successional stage. If no land management or training takes place in a simulation, vegetation categories simply change via succession at the rates specified by the user. However, as described below, land management and training activities affect succession in various ways. Figures 4 through 8 show flow diagrams for the vegetation dynamics model and its functional components.

The user-specified acreage (number of acres) of prescribed burning is then applied by selecting a random cell and allowing the "burning" to spread from there to surrounding areas. However, valley areas defined by a *valleydef* map are not burned by the prescribed burning. *Valleydef* is an internal IDLAMS function that identifies the valley boundaries for terrain areas with woody draws (these draws shelter the enclosed vegetation from fires). These draws are identified from processing digital terrain maps (DTMs). The prescribed burning function could be altered to take burning rotation plan maps as input if desired. The prescribed burning will alter landcover classes by (1) speeding up early successional to late successional and damaged to early successional if the current pixel in the map is early succession or damaged category, respectively, and (2) preventing late succession from succeeding into shrub or forest. The landcover category of a given pixel is then checked, and natural succession takes place if the category of the cell as well as its surrounding pixels meet succession criteria.

Figure 4 Flow Diagram of Vegetation Dynamics Model Main Function

Figure 5 Flow Diagram of Prescribed Burning Function

Figure 6 Flow Diagram of Vegetation Succession Function

Figure 7 Flow Diagram of Training Function

Next, the training impact is distributed onto the landcover map, on the basis of the Maneuver Impact Miles and the training intensity distribution map. The former is the equivalent of Abrahms M1A1 main battle tank maneuver tracking miles on an installation caused by all training activities, and the latter is a 0-1 scale normalized training intensity (Concepts Analysis Agency 1996). The number of MIMs applied to each pixel is determined by multiplying the average MIMs for the whole installation by the training intensity value of the pixel. The number of MIMs applied to each pixel is then converted to area by multiplying MIMs by the width of the footprint of the M1A1 tank. This represents the area of the pixel that receives training impact. Thus, the training impact is directly related to the training intensity.

A comparison is made between the area affected by training impact and the size of the cell. For each pixel, if the total area affected by training is less than 40% of the pixel size and the vegetation category of the pixel is early successional grasslands, the pixel remains early successional. Conversely, if the percent affected exceeds 40%, the pixel reverts to damaged grasslands. This 40% cut-off is a best estimate and can be changed if research supports a better estimate in the future. Similarly, the same cut-off rule is applied to pixels representing late successional grasslands, except that the affected area must be greater than or equal to 60% of the pixel for a later successional category to become a damaged vegetation category.

Finally, the planting function (erosion control or native species) is applied. This is done by first aggregating or "clumping" (`r.clump` in GRASS) the areas of damaged grasslands and then randomly planting only on those damaged areas that exceed 15 acres in size.

The vegetation dynamics model is written as a standard GRASS function, named `r.veg.change`. The source code is placed, as are all other GRASS main raster functions, in the GRASS subdirectory:

```
src/raster/r.veg.change/cmd/
```

The vegetation dynamics model consists of seven functions:

```
main.c
assign_init_age.c
change_cat.c
check_change2forest.c
cseg_get.c, dseg_get.c
planting.c
prescribed_fire.c
```

Function `main.c` serves as a driver to (1) read input parameters and maps, (2) initialize map segmentation files for input and temporary maps, (3) call other functions to perform vegetation simulation calculations under given training intensity and land management activities, and (4) produce an output vegetation map at the end of the simulation. Function `assign_init_age.c` randomly assigns initial age or a representation of the number of years a cell has been in a particular landcover category in the input landcover map. Function `change_cat.c` is designated to change the age or the category of the current category if the current category meets conditions set forth for natural succession category change. Vegetation categories are all defined in the header file `veg.h`. Functions

`assign_init_age.c`, `change_cat.c`, and the header file `veg.h` must be modified if the current classification categories need to be changed. Table 5 shows the vegetation categories defined in `veg.h`.

Table 5 Vegetation Categories Defined in Header File `veg.h`

<code>#define C_NODATA</code>	0	<code>/* Nodata cell category</code>	<code>*/</code>
<code>#define C_FOREST</code>	1	<code>/* Forest Category</code>	<code>*/</code>
<code>#define C_SHRUB</code>	2	<code>/* Shrub Category</code>	<code>*/</code>
<code>#define C_DISTURBED</code>	3	<code>/* Training Damaged Category</code>	<code>*/</code>
<code>#define C_EARLY</code>	4	<code>/* Early Successional Category</code>	<code>*/</code>
<code>#define C_LATE</code>	5	<code>/* Late Successional Category</code>	<code>*/</code>
<code>#define C_TURF</code>	6	<code>/* Planted Turflike/Erosion Control Grass</code>	<code>*/</code>
<code>#define C_NATIVE</code>	7	<code>/* Planted Native Grass</code>	<code>*/</code>

A map segmentation technique has been employed in this vegetation dynamics model. With this technique, maps are partitioned into block data in a format that is readable and writeable as if the whole map were stored in computer memory, whereas in fact only a fraction of the map is actually stored in the memory. This technique allows the model to swap only part of a huge map into the computer memory at a time and perform manipulations so that the model can handle any size map at any desired level of resolution. This use of map segmentation makes IDLAMS more applicable to installations with large data sets.

The map segmentation technique has been implemented as part of the GRASS library function. Programmers can access these functions by including the `<seg.h>` header file in the application function. The call to these functions is the same as to any other library function. However, a few extra parameters must be defined. In order to make the segmentation library function compatible with the floating-point map, a modified version of `cseg_get.c` and a double-precision floating-point version of this function `dseg_get.c` have to be made. Table 6 shows the parameters defined in the header file `veg.h` for the map segmentation technique to function. A complete list and detailed explanation of the segmentation functions and their required variables can be found in the GRASS Programmer's Manual (USACERL 1993a).

Finally, the function `planting.c` assigns a given amount of grass to be planted within clumped damaged areas. Also, the function `prescribed_fire.c` burns a given area clumped around a randomly selected cell. More information about the development of the vegetation dynamics model can be found in the IDLAMS Final Report.

Table 6 Definitions of Variables for Map Segmentation Functions

```

/* Define the segmentation variables and assign them values */

#define SROWS 9      /* Number of rows in the segment */
                    /* file */

#define SCOLS 13     /* Number of columns in the segment*/
                    /* file.  SROW and SCOL together */
                    /* define the format of the segment */
                    /* file. */

#define SWAP_BLOCK 512
                    /* SWAP_BLOCK defines the size */
                    /* in byte of the portion of the map */
                    /* retained in memory space */

int trnmap_fd, lndmap_fd;
int age_fd, fireaffect_fd;
int temp_fd, valydef_fd;
    /* Define the file descriptors for the */
    /* segmentation files used to */
    /* segment the maps to be opened. */

char *lndseg_file, *trnseg_file;
char *ageseg_file, *fireaffect_file;
char *tempseg_file, *valydef_file;
    /* Define the segmentation files */

int flen, ilen;    /* The lengths of floating point */
                    /* map cell FCELL and the integer */
                    /* map cell CELL respectively */

SEGMENT trnmap_seg, lndmap_seg;
    /* Define segmentation structures */
    /* for the training intensity map */
    /* and the land cover map. */

SEGMENT fireaffect_seg, age_seg;
    /* Define segmentation structures to */
    /* store fire affect and vegetation */
    /* succession criterion */

SEGMENT valy_seg;
    /* Define segmentation structure */
    /* forest spread constraint valley */
    /* def map. */

SEGMENT temp_seg;
    /* Define segmentation structures */
    /* for a working segmentation file */

```


4 WILDLIFE HABITAT SUBMODELS

All wildlife habitat submodels are written in UNIX Bourne shell scripts using GRASS functions. If desired, sites can develop these models in other GIS modeling languages (e.g., ARC/INFO, GRID). The inputs of these models are directly obtained from the Tcl/Tk graphical user interface. A habitat suitability map is calculated for each wildlife species. The calculations are performed on the basis of the current landcover map or the output map of the vegetation succession model and other maps, such as foodplots, soil texture, and nesting area, where applicable. GRASS functions `r.mapcalc`, `r.buffer`, `r.clump`, `r.reclass`, `r.stats`, and `r.neighbors` are employed in the calculation. In addition, an internal function, `clump2classes`, has been developed in C language to provide a tool for screening out aggregated areas or "clumps" having a contiguous area less than the user-specified size.

The command line format of function `clump2class` is:

```
clump2classes p1 p2 p3 p4
```

where

p^1 = `r.clump` map statistic output (generated using `r.stats`);

p^2 = user-desired minimum contiguous nesting area size;

p^3 = 10,000, a conversion for hectares; and

p^4 = output file (this file is used to pass through `r.reclass` to reclass the `r.clump` result map into only those meeting the contiguous area criteria).

There are five different wildlife habitat simulation submodels in the current Fort Riley IDLAMS system. They are as follows: (1) northern bobwhite (*Colinus virginianus*), (2) Henslow's sparrow (*Ammodramus henslowii*), (3) greater prairie chicken (*Tympanuchus cupido*), (4) prairie mole cricket (*Gryllotalpa major*), and (5) regal fritillary (*Speyeria idalia*). Again, users can develop their own submodel(s) as appropriate for specific ecosystems or regions. More information about the development of the wildlife submodel can be found in the IDLAMS Final Report.

5 EROSION SUBMODEL

The soil erosion submodel in IDLAMS calculates the soil erosion status by dividing the result of the Revised Universal Soil Loss Equation (RUSLE) (Wischmeier and Smith 1978) by soil erosion tolerance, T, factors. The soil erosion model is also implemented in Bourne shell programming language by using the GRASS function `r.mapcalc`. User inputs for this model are (1) K factor map, (2) LS factor map, (3) T factor map, and (4) C factor map. The C factor map is the only factor that will change on the basis of different vegetation dynamics simulations. The user is provided with a table for manually reclassifying landcover to C factor values. For sites implementing the Land Condition Trend Analysis (LCTA), data can be used to reclass landcover into C factor values. Again, sites wishing to implement alternative erosion or watershed models may do so by ensuring that these models can take ASCII landcover arrays as input. More information about the development of the erosion submodel can be found in the IDLAMS Final Report.

6 SCENARIO EVALUATION MODEL

A complete description of the decision-making process built into IDLAMS is beyond the scope of this document, and is covered in depth in the IDLAMS Final Report. The following brief description is meant only as an introduction to the workings of the scenario evaluation model and a discussion of topics of interest to programmers.

6.1 IDLAMS Succession Models

A fundamental theory of operation of the IDLAMS environment involves the simulation of natural vegetation succession over time for a given area of land. This simulation can be performed in several ways, to provide different levels of detail and approximation needed in different IDLAMS modules. The vegetation dynamics model executes a cell-by-cell succession simulation. The scenario evaluation model works directly with the result maps from that cell-by-cell simulation. It can also execute a bulk succession simulation and process the more general results for trends. The succession stages used by both the cell-by-cell model and the bulk model are illustrated in Figure 9.

Figure 9 Successional Transitions in the Life of a Landcover Map Cell

The cell-by-cell model, which predicts succession on the basis of each cell's neighbors, is very slow but provides spatial detail regarding what landcover type appears, and where it appears, in a site. Statistics gathered on the final resultant map (using *r.stats*) are used to score that scenario's usefulness. Figure 10 illustrates a typical landcover map.

Figure 10 Landcover and Training Areas Map

The bulk statistics model gathers statistics at the start of the simulation, using these values only to predict the successive Markov state probabilities. In this way, it executes many times faster but at the cost of true spatial detail associated with the resultant map. The bulk model is called quickly and uses the scenario evaluation model to score different land management practices.

6.2 Decision-Making Strategy

Scenario Evaluation

The decision-making process involves defining, examining, and comparing the effects of land management practices implemented at a given military installation. Any given collection of parameters and a time period over which they are valid and operate is referred to as a scenario. The decision-making process begins by evaluating each scenario result through a systematic process of identifying land use objectives, assigning objective measures and objective weights, and collecting these objectives, measures, and weights into single- and multiple-attribute utility functions that can be mathematically optimized.

Scenario Comparison

The parameters selected and the values used to represent them were gathered during a staged workshop, where experienced personnel were interviewed through a system of interrelated and counter-related questions to identify and compare the objectives and establish their measures and ranked weights (Keeney 1992). The objectives, measures, and weights were combined into single-attribute utility functions, yielding a multiple-attribute utility function score for each scenario. The utility score values produced are concatenated into a script command line call to GNU software's GNUPLOT (Williams and Kelley 1993) for visual display to the user (see Figure 11).

Figure 11 Typical Single-Attribute Utility Functions Graph

Each scenario has a utility (or composite functional usefulness) assigned to it, so the management practices modeled by the parameter choice can be compared.

6.3 Algorithm and Data Representation

The environmental objectives are represented by simple category values. Each objective has associated with it a series of formal measures and weights for those measures, each represented by a numeric value. These measures are defined through interviews with the user(s) and are combined into simple algebraic (in this case) utility functions of the form:

$$\begin{aligned}
 U(X) = & \\
 & k_{11} \ u_{11}(x_{111}, x_{112}, x_{113}) + \\
 & k_{12} \ u_{12}(x_{12}) + \\
 & k_{13} \ u_{13}(x_{131}, x_{132}, x_{133}) + \\
 & k_{21} \ u_{21}(x_{21}) + \\
 & k_{22} \ u_{22}(x_{221}, x_{222}, x_{223}) +
 \end{aligned}$$

```

k23 u23(x23) +
k24 u24(x24) +
k25 u25(p251 x251 + p252 x252 + p253 x253 + p254 x254)

```

The evaluation of these functions into utility scores is straightforward. The collection of scenario statistics uses only small amounts of memory, in comparison with storing entire landcover maps. The statistics are stored as dynamic arrays representing sparse matrices. In this way, memory is used efficiently and allows reasonably large problems to be addressed.

6.4 Algorithm Implementation

The decision-making process involves evaluating a series of utility functions. The utility functions were first developed and modeled in a spreadsheet environment, where they were mathematically validated. The utility functions were then coded in C and UNIX scripts and validated.

All developed code compiles and executes by using standard ANSI C compilers. Subdirectories for all the required codes are listed, and the IDLAMS Installation Manual describes how they are set up for the operation.

6.5 User Input Required

The decision making begins by prompting the user for parameter values through a Tcl/Tk “expect” interface. The “expect” calls capture the input the user is typing in widget fields and store them in local environment variables. The bulk of these values have preprogrammed defaults, but in order to execute successfully, the user must select the names of several GIS maps as well. The individual map names are captured and stored as global variables.

6.6 Model Output Produced

The output from the scenario evaluation model is a report generated and opened for review during the IDLAMS session (see Figure 12). The report output screen is produced by a Tcl/Tk script.

output					
Measure	Weight	Input Value of Measure	SAU	Contribution to Total	Normed Measure
artillery	0.023	75,500	0.975	0.022	0.709
infantry	0.026	79,750	0.158	0.004	0.163
maneuvering	0.395	90,000	0.733	0.289	0.827
durability	0.236	45,000	0.392	0.092	0.333
violations	0.202	0,030	0.265	0.053	0.600
endangered	0.051	8,494	0.255	0.013	0.415
wildlife	0.021	81,743	0.577	0.012	0.553
vegetation	0.028	66,055	0.871	0.024	0.709
hunting(pc)	0.018	8,494	0.214	0.003	0.077
hunting(bw)	0.018	81,743	0.577	0.010	0.553
The total value for our utility function is 0.525					

Figure 12 Typical Evaluation Model Statistics Report

7 REFERENCES

- Concepts Analysis Agency, 1996, Evaluation of Land Value Study (ELVS), U.S. Army Concepts Analysis Agency, Study Report CAA-SR-96-5, Bethesda, Md.
- Gebhart, D.L., and S.D. Warren, 1995, Regional Cost Estimates for Rehabilitation and Maintenance Practices on Army Training Lands, USACERL Technical Report 96/02, Oct.
- Guertin, P.J., C.C. Rewerts, and P.C. Dubois, 1997, Training Use Distribution Modeling, U.S. Army Construction Engineering Research Laboratories, Champaign, Ill., CERL Technical Note 97-XXX, Sept., (in preparation), 6 pp.
- Internet address: http://science.nas.nasa.gov/people/woo/gnuplot_info.html 1998, June 5, 1998.
- Keeney, R., 1992, Value-Focused Thinking, Harvard University Press, Cambridge, Mass.
- Ousterhout, J., 1995, Tcl/Tk Installation and Documentation Directory, University of California at Berkeley, ouster@cs.berkeley.edu.
- USACERL: See U.S. Army Corps of Engineers, Construction Engineering Research Laboratories.
- U.S. Army Corps of Engineers, Construction Engineering Research Laboratories (USACERL), 1993a, Geographic Resources Analysis Support System (GRASS), Version 4.1, Programmer's Manual, M. Shapiro et. al., Champaign, Ill.
- U.S. Army Corps of Engineers, Construction Engineering Research Laboratories (USACERL), 1993b, Geographic Resources Analysis Support System (GRASS), Version 4.1, User's Reference Manual, Champaign, Ill.
- Williams, T., and C. Kelley, 1993, GNU PLOT, An Interactive Plotting Program, Version 3.5, info-gnuplot-request@dartmouth.edu.
- Wischmeier, W.H., and D.D. Smith, 1978, *Predicting Rainfall Erosion Losses — A Guide to Conservation Planning*, U.S. Department of Agriculture, Agricultural Handbook No. 537.