USGS
science for a changing world

The Lake Tahoe Basin Land Use Simulation Model

By William M. Forney and I. Benson Oldham

Open-File Report 2011-1275

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
KEN SALAZAR, Secretary

U.S. Geological Survey
Marcia K. McNutt, Director

U.S. Geological Survey, Reston, Virginia: 2011

For product and ordering information:
World Wide Web: http://www.usgs.gov/pubprod
Telephone: 1-888-ASK-USGS

For more information on the USGS—the Federal source for science about the Earth,
its natural and living resources, natural hazards, and the environment:
World Wide Web: http://www.usgs.gov
Telephone: 1-888-ASK-USGS

Suggested citation:
Forney, W.M., and Oldham, I.B., 2011, The Lake Tahoe Basin Land Use Simulation Model: U.S. Geological Survey Open-File Report 2011-1275, 63 p., available at http://pubs.usgs.gov/of/2011/1275/.

Table of Contents

Acronyms

ALLOC_DG	Parcel allocated to Douglas County
ALLOC_COMM	Parcel allocated to Commercial square footage
ALLOC_EL	Parcel allocated to El Dorado County
ALLOC_MFH	Parcel allocated to multiple family housing
ALLOC_PL	Parcel allocated to Placer County
ALLOC_PL_SENSITIVE	Parcel allocated to the sensitive lots of Placer County
ALLOC_SLT	Parcel allocated to the city of South Lake Tahoe
ALLOC_TOUR	Parcel allocated to tourist accommodation units
ALLOC_WA	Parcel allocated to Washoe County
APN	Assessor's parcel number
CTC	California Tahoe Conservancy
GUI	Graphical user interface
IPES	Individual Parcel Evaluation System
OFR	U.S. Geological Survey Open-File Report
LUSM	Lake Tahoe Basin Land Use Simulation Model
LU	Land Use
MFD	Multiple (or multi-) family dwelling (or housing)
NDSL	Nevada Division of State Lands
PAS	Plan Area Statement
RETIRE_CTC_PRIORITY	Sensitive parcel allocated to California Tahoe Conservancy
RETIRE_CTC_SECONDARY	Non-sensitive parcel allocated to California Tahoe Conservancy
RETIRE_NEV_PRIORITY	Sensitive parcel allocated to Nevada State Lands
RETIRE_NEV_SECONDARY	Non-sensitive parcel allocated to Nevada State Lands
RETIRE_TRPA_PRIORITY	Sensitive parcel allocated to Tahoe Regional Planning Agency
RETIRE_TRPA_SECONDARY	Non-sensitive parcel allocated to Tahoe Regional Planning Agency
RETIRE_USFS_PRIORITY	Sensitive parcel allocated to U.S. Forest Service
RETIRE_USFS_SECONDARY	Non-sensitive parcel allocated to U.S. Forest Service
SFD	Single family dwelling
TAU	Tourist accommodation unit
TIIMS	Tahoe Integrated Information Management System
TRPA	Tahoe Regional Planning Agency
USFS	U.S. Forest Service
USGS	U.S. Geological Survey

The Lake Tahoe Basin Land Use Simulation Model

By William M. Forney and I. Benson Oldham

Abstract

This U.S. Geological Survey Open-File Report describes the final modeling product for the Tahoe Decision Support System project for the Lake Tahoe Basin funded by the Southern Nevada Public Land Management Act and the U.S. Geological Survey's Geographic Analysis and Monitoring Program. This research was conducted by the U.S. Geological Survey Western Geographic Science Center. The purpose of this report is to describe the basic elements of the novel Lake Tahoe Basin Land Use Simulation Model, publish samples of the data inputs, basic outputs of the model, and the details of the Python code. The results of this report include a basic description of the Land Use Simulation Model, descriptions and summary statistics of model inputs, two figures showing the graphical user interface from the web-based tool, samples of the two input files, seven tables of basic output results from the web-based tool and descriptions of their parameters, and the fully functional Python code.

Introduction

The Lake Tahoe Basin Land Use Simulation Model (LUSM), which simulates parcel-based, land-use transitions in a stochastic, spatially constrained, agent-based model, was built to inform land management decisions in a complex regulatory environment in the Lake Tahoe Basin (fig. 1). Building on the experience, knowledge and research of the authors and the other U.S. Geological Survey (USGS) Western Geographic Science Center staff, the LUSM was developed in close collaboration with the Tahoe Regional Planning Agency (TRPA). The LUSM provides to TRPA, Basin resource managers and stakeholders a web-based, 20-year-scenario generation tool to simulate the possible allocation of retired (that is, maintained as open space) and developed parcels, and their subsequent transition to another land use fate. Due to the model's capability to project up to 20 years into the future, a random or stochastic element is included to account for the inherent uncertainty of the future. To give the LUSM more site-specific relevance, the model is constrained by 1) neighborhood characteristics and requirements that are defined by Plan Area Statements[1] (PAS) and 2) TRPA's land capability systems (that is, the Individual Parcel Evaluation System (IPES) and the Bailey Capability System[2]). Although the definition of an agent can vary, the agent of change in this agent-based model is the parcel's potential land owner. Using standard software packages, coding languages, and readily available datasets, the final LUSM results include reasonable tabular results that can be translated into map-based outputs with minimal additional effort by knowledgeable geographic information system users.

The LUSM is designed as a constrained, stochastic simulation model. This is based on requests of TRPA, data availability, and the consideration of land-use modeling theory when deterministic

[1] Includes community plans, both of which are defined by planners at the TRPA. Available at: *http://www.trpa.org/default.aspx?tabid=204*

[2] TRPA Code of Ordinances, Chapter 37 and Chapter 20, respectively.

outcomes can be misleading, and future scenarios are important to characterize with an adequate balance of realism and uncertainty. The model is constrained by PAS neighborhood allowances, the initial conditions set by the user, and the physical characteristics of a given parcel. The PAS must allow for the parcel's land use type or real estate commodity—otherwise, no commodities of that type will be allocated to the parcels of that PAS. The stochasticity is created by randomizing certain aspects of the selection process, the order of parcels being considered for transition, and modeling specifications and functionality. Also, an essential component—uncertainty—is generated by performing multiple runs or iterations to obtain a set of outcomes. The multiple iterations provide a count of the number of times a specific parcel transitions to a particular land use type or fate, thereby allowing for a probabilistic estimate of the likelihood of that transition.

Using the Model

The model and its database are maintained by the Tahoe Integrated Information Management System (TIIMS) and TRPA, and can be fully executed by way of the internet. The model works on a Servoy platform and Servoy's internet-based Graphical User Interface (GUI), and calls the Postgres databases and Python scripting. A description of the project can be found here: *http://www.tiims.org/Science-Research/Environmental-Modeling/TDSS.aspx*. The LUSM, with permission obtained from TRPA, can be accessed from this webpage.

Instead of running multiple iterations of allocations in a single year to create the stochastic simulation results—then moving onto the next year until 20 years have been reached—the model runs a single iteration of allocations in a given year, removes the parcels that have been allocated from the vacant pool, and moves on to the next year until 20 years have been reached. Upon reaching the twentieth year, this is considered one, full iteration of the model, then—if multiple iterations are specified—it returns to the start time and begins again. This allows for quicker model runs, and arbitrary stochasticity on an annual basis is reduced as allocation targets are maintained from year to year.

All data used for the model were either obtained directly from the TRPA, or were derived from TRPA and other public documents and datasets. The parcel land use codes of TRPA dictate the level of planning detail and functionality of the LUSM. As detailed in appendix 1, only certain TRPA land use codes are important to the LUSM: vacant (private) land (code 1); single family dwelling (SFD, code 1011); multiple family dwelling (MFD, codes 1005, 1006, and 1007); a representative class (3000) for commercial uses including retail, entertainment, services, light industrial, and wholesale/storage; a representative Tourist Accommodation Unit (TAU)—specifically the Hotel unit (code 2002); and the Resource Management category open space land (code 6401). Please note that multiple family

dwellings can also be called multi-family dwellings and housing and the acronym MFD represents them both, and that open space land is land that has been taken off the real estate market for development, or retired. Appendix 2 summarizes the 2008 assessor's parcel map dataset that is currently used in the LUSM, and characterizes the land uses in the Lake Tahoe Basin. As of 2008, sixty-eight percent of the land use is residential, less than 1 percent is tourist, over 2 percent is commercial, and over sixteen percent is open space. It is notable that of 60,376 parcels, only 6,326 parcels remain as vacant; these 6,326 parcels that potentially can transition to a different land use are used in the execution of the LUSM. In terms of area, the Tahoe Basin has over 86,000 hectares designated with some type zoning, of which over 2,750 hectares are vacant and included in the execution of the LUSM.

Inputs and Results

The LUSM has two primary inputs: the parcel database and Plan Area Statement (PAS) database (fig. 1). Samples of these inputs and the fields are provided in tables 1 and 2 of appendix 3. Table 1 provides a list and description of the variables that are available to the user, and figure 2 shows what the graphical user interface (GUI) looks like on the internet. Table 2 provides a list of the model's parameters, which are only available to be changed by certain users with administrative privileges, and figure 3 shows what that GUI looks like on the internet.

The user of the model can execute different initial conditions to generate different future land use patterns. Note that for an individual parcel there are multiple fates, which assist in creating the probability of transitioning. Each commodity type has certain variable values and decision rules that govern its behavior. The results of the LUSM are shown in tables 3 through 9, which represent examples of the primary, basic, tabular outputs of the LUSM. The examples were run with the default values of the model, using two iterations. The default values were derived from existing datasets and communications with TRPA. The seven comma-delimited .txt files produced by the web-based model were brought into Microsoft Excel for formatting and display purposes. Each table is titled the way it is provided when downloaded in a .zip file from the internet-based model for the sake of continuity and to facilitate the following documentation.

In table 3, "APN" is assessor's parcel number; "Landuse" is the TRPA Land Use Code (appendix 1); "UsePct" is the percent of the time the model designates this particular land use; "Units" are the density-restricted number of units that can be built on a parcel relevant to SFD, MFD, and TAU; and "CommSF" is the amount of commercial square footage allocated in a particular commercial parcel. In table 4, new variables are: "Category", a particular combination of land use fates for a given parcel;

"LU_XXXX", a different formulation of the TRPA land use codes and the values are the percent of the time the model designates this particular Land Use; and "*MFHUnits*," "*TourUnits*", and "*CommSF*," static variables that do not change from one run to another and thus they should be disregarded as useful output.

In table 5, new variables are: "ALLOC_DG," the likelihood that a parcel is allocated to Douglas County for SFD; "ALLOC_EL," the likelihood that a parcel is allocated to Eldorado County for SFD; "ALLOC_PL," the likelihood that a parcel is allocated to Placer County for SFD; "ALLOC_PL_SENSITIVE," the likelihood that a parcel is allocated to the sensitive lots of Placer County for SFD as related to the IPES threshold and the Vacant Lot Equation[3]; "ALLOC_WA," the likelihood that a parcel is allocated to Washoe County for SFD; "ALLOC_SLT," the likelihood that a parcel is allocated to the city of South Lake Tahoe for SFD; "ALLOC_MFH," the likelihood that a parcel is allocated to multiple family housing; "ALLOC_TOUR," the likelihood that a parcel is allocated to TAU; "ALLOC_COMM," the likelihood that a parcel is allocated to Commercial square footage; "RETIRE_CTC_PRIORITY," the likelihood that a sensitive parcel (as deemed IPES scores and Bailey scores) is allocated to the retirement program of the California Tahoe Conservancy; "RETIRE_CTC_SECONDARY," the likelihood that a non-sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; "RETIRE_NEV_PRIORITY," the likelihood that a sensitive parcel is allocated to the retirement program of the Nevada State Lands; "RETIRE_NEV_SECONDARY," the likelihood that a non-sensitive parcel is allocated to the retirement program of the Nevada State Lands; "RETIRE_TRPA_PRIORITY," the likelihood that a sensitive parcel is allocated to the retirement program of the TRPA; "RETIRE_TRPA_SECONDARY," the likelihood that a non-sensitive parcel is allocated to the retirement program of the TRPA; "RETIRE_USFS_PRIORITY," the likelihood that a sensitive parcel is allocated to the retirement

[3] TRPA Code of Ordinances, Chapter 37.8

program of the U.S. Forest Service; "RETIRE_USFS_SECONDARY," the likelihood that a non-sensitive parcel is allocated to the retirement program of the U.S. Forest Service. Retirement programs are land acquisition efforts in the Lake Tahoe Basin. Please note that the jurisdictions of the counties and retirement programs are maintained.

Table 6 represents the number of parcels that transition into a particular land use type in a given year, tracked for the duration of the model run. The column headings Y1 through Y20 designate the year of the transition. Table 7 represents the number of parcels that transition into a particular allocation type in a given year, tracked for the years of the model run. Table 8 details the number of parcels that remain available for transitioning to a particular Allocation type in a given year, tracked for the years of the model run. Table 9 represents the frequency of transition fates for the vacant parcels in the Lake Tahoe Basin. New, previously undefined variables in table 9 include: "Count," representing the frequency with which that combination occurs. Note that Category 1 is the most likely combination that results 1,435 times and represents the outcome that a vacant parcel transitions 100 percent of the time to retired/open space (that is, Land Use Code 6401).

The Python source code for the LUSM can be found in appendix 4.

Acknowledgments

This research has been supported by the USGS Geographic Analysis and Monitoring Program and funded by the Southern Nevada Public Land Management Act. The research has been conducted in cooperation with the TRPA. The authors would like to thank Peter Ng, Richard Champion, Mara Tongue, and Susan Benjamin for their thorough reviews, and the USGS Western Geographic Science Center staff who laid the groundwork of this effort, especially Richard Bernknopf, David Halsing, Mark Hessenflow, Sean Devlin, Elizabeth Duffy, Peter Ng, and Michael Gould. We appreciate the collaboration with and support of Neil Crescenti, Leif Larson, and Marie Bledsoe of TRPA.

Figures

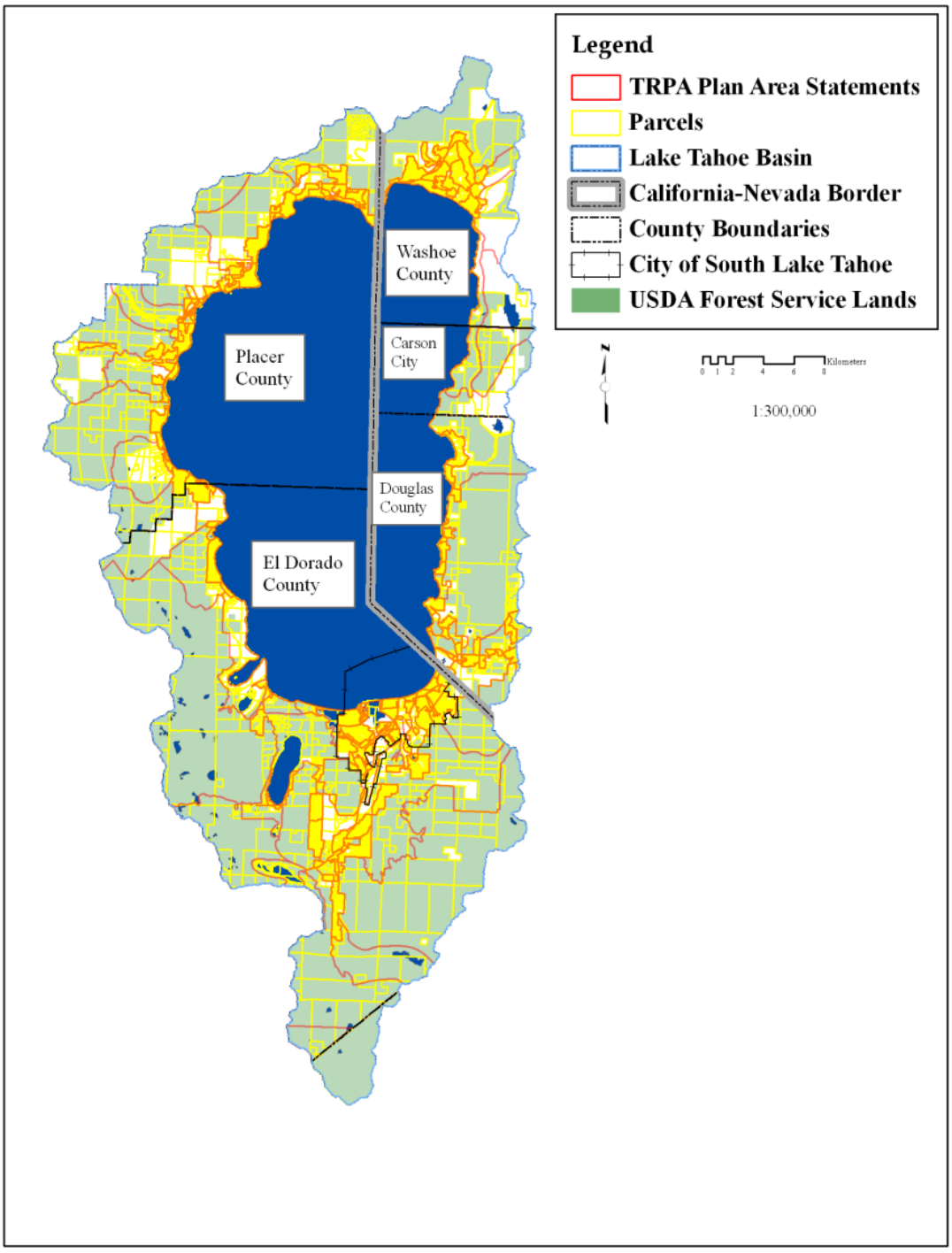

Figure 1. Overview of Lake Tahoe Basin, LUSM inputs, and relevant jurisdictions

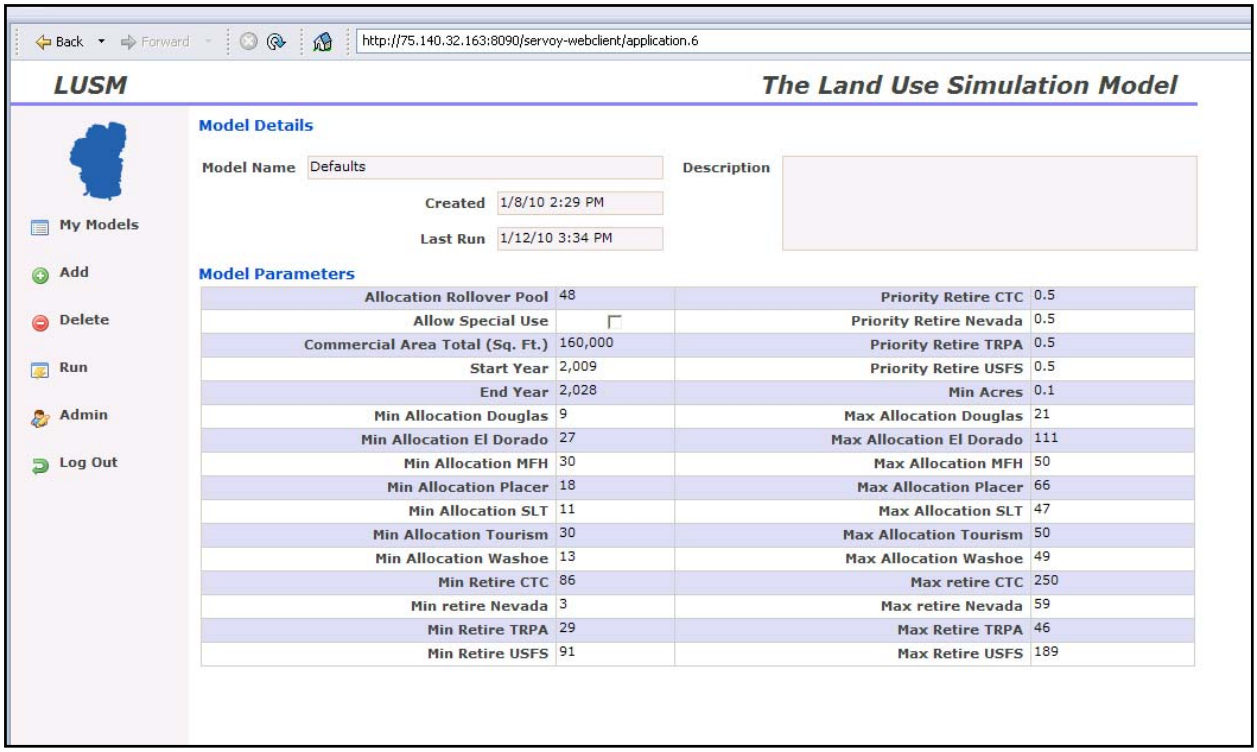

Figure 2. Screen shot of the internet GUI for the LUSM showing the list of user variables and values.

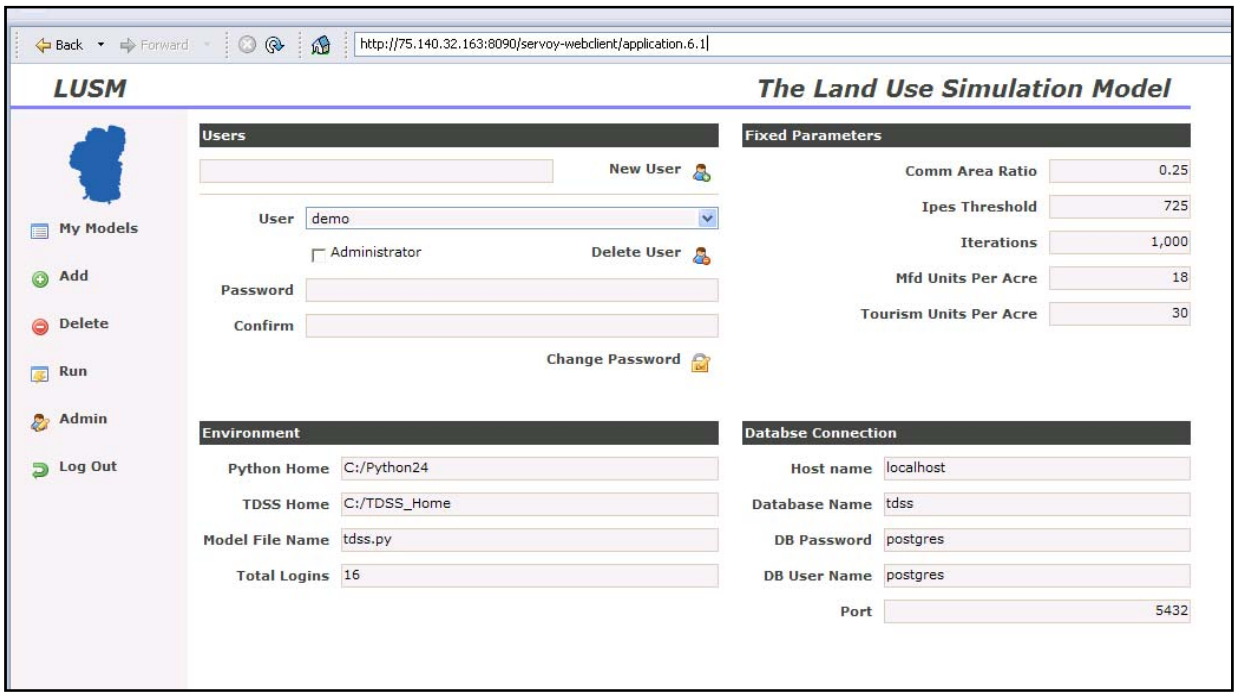

Figure 3. Screen shot of the internet GUI for the LUSM showing the Model parameter values for LUSM administrators.

The *Model File Name* shows where updates and changes to the Python Code can be made; the *Database Name* shows where the database can be updated. Also, this is where additional password-protected users and their administrator privileges can be established.

Tables

Table 1. User parameters, commodities and their descriptions in the LUSM.

[Single family dwelling (SFD); multi or multiple family dwelling (MFD); tourist accommodation unit (TAU); California Tahoe Conservancy (CTC); Nevada Division of State Lands (NDSL); Tahoe Regional Planning Agency (TRPA); U.S. Forest Service (USFS); plan area statement (PAS)]

PARAMETER OR COMMODITY NAME	DESCRIPTION OR NOTES
Development	
Douglas County—Minimum	Spatial Allocation by County of SFD
Douglas County—Maximum	Spatial Allocation by County of SFD
Washoe County—Minimum	Spatial Allocation by County of SFD
Washoe County—Maximum	Spatial Allocation by County of SFD
El Dorado County—Minimum	Spatial Allocation by County of SFD
El Dorado County—Maximum	Spatial Allocation by County of SFD
Placer County—Minimum	Spatial Allocation by County of SFD
Placer County—Maximum	Spatial Allocation by County of SFD
South Lake Tahoe—Minimum	Spatial Allocation by County of SFD
South Lake Tahoe—Maximum	Spatial Allocation by County of SFD
Multi-Family Dwelling—Minimum	Spatial Allocation for Basin of MFD
Multi-Family Dwelling—Maximum	Spatial Allocation for Basin of MFD
Tourist Accommodation Unit—Minimum	Spatial Allocation for Basin of TAU
Tourist Accommodation Unit—Maximum	Spatial Allocation for Basin of TAU
Commercial Area—Total sq. ft.	Spatial Allocation for Basin of Commercial for the entire model run
Retirement	
CTC—Minimum	Spatial Allocation within California
CTC—Maximum	Spatial Allocation within California
CTC—Priority	Weighting of preferentially sensitive lands to retire
NDSL—Minimum	Spatial Allocation within Nevada
NDSL—Maximum	Spatial Allocation within Nevada
NDSL—Priority	Weighting of preferentially sensitive lands to retire
TRPA—Minimum	Spatial Allocation for entire Tahoe Basin
TRPA—Maximum	Spatial Allocation for entire Tahoe Basin
TRPA—Priority	Weighting of preferentially sensitive lands to retire
USFS—Minimum	Spatial Allocation for entire Tahoe Basin
USFS—Maximum	Spatial Allocation for entire Tahoe Basin
USFS—Priority	Weighting of preferentially sensitive lands to retire
Other	
Allocation Rollover Pool[5]	Initial amount in the pool
Minimum Size	Minimum size threshold (ac)
Special Use Allowance	Inclusion of special uses as designated by individual PASes
Starting Year	Beginning year of the model run
Ending Year	Final year of the model run

[5] TRPA Code of Ordinances, 33.2.A.

Table 2. Model parameters and their descriptions for the LUSM.

[Multi- or multiple family dwelling (MFD); and Plan Area Statement (PAS)]

PARAMETER	DESCRIPTION OR NOTES
Commercial Area Ratio	Ratio of parcel size to commercial footprint area
IPES threshold	Value at and below which parcels are unable to be developed for Placer County until the Vacant Lot Equation is satisfied
Iterations	Number of iterations run by the model
MFD Units Per Acre	Density metric for MFD, varies by PAS
Tourism Units Per Acre	Density metric for Tourist accommodations, varies by PAS

Table 3. TahoeLanduse.txt.

[This is only a sample of the total parcel-based output. "APN" is assessor's parcel number; "Landuse" is the TRPA Land Use Code; "UsePct" is the percent of the time the model designates this particular land use; "Units" are the density-restricted number of units that can be built on a parcel relevant to single family dwelling, multiple family dwelling, and tourist accommodations; and "CommSF" is the amount of commercial square footage allocated in a particular commercial parcel]

APN	Landuse	UsePct	Units	CommSF
093-130-026	LU_1011	0.5	1	0
093-130-026	LU_6401	0.5	0	0
093-130-028	LU_1011	0.5	1	0
093-130-028	LU_6401	0.5	0	0
111-120-036	LU_1011	1	1	0
1318-23-810-017	LU_1011	0.5	1	0
1318-23-810-017	LU_6401	0.5	0	0
111-120-034	LU_1011	0.5	1	0
111-120-034	LU_6401	0.5	0	0
1418-03-811-025	LU_1011	0.5	1	0
1418-03-811-025	LU_6401	0.5	0	0
023-501-13	LU_6401	1	0	0
023-103-14	LU_6401	1	0	0
023-103-15	LU_1011	0.5	1	0
023-103-15	LU_6401	0.5	0	0
023-103-16	LU_1011	0.5	1	0
023-103-16	LU_6401	0.5	0	0
023-103-11	LU_1011	0.5	1	0
023-103-11	LU_6401	0.5	0	0
023-512-06	LU_6401	1	0	0
023-512-09	LU_6401	1	0	0
025-271-04	LU_1011	1	1	0
116-110-015	LU_1011	0.5	1	0
116-110-015	LU_6401	0.5	0	0
083-300-015	LU_1011	1	1	0
090-041-009	LU_6401	1	0	0
031-081-10	LU_6401	1	0	0
016-202-20	LU_6401	1	0	0
026-096-05	LU_1011	0.5	1	0

Table 4. TahoeCondensedLanduse.txt.

[This is only a sample of the total parcel-based output. "APN" is assessor's parcel number; "Category", a particular combination of fates for a given parcel; "LU_XXXX", a different formulation of the TRPA land use codes and the values are the percent of the time the model designates this particular land use; and "*MFHUnits*," "*TourUnits*", and "*CommSF*," static variables that do not change from one run to another and thus they should be disregarded as useful output]

APN	Category	LU_0001	LU_1005	LU_1006	LU_1007	LU_1011	LU_2002	LU_3000	LU_6401	*MFHUnits*	*TourUnits*	*CommSF*
093-130-026	19	0	0	0	0	0.5	0	0	0.5	*13*	*0*	*8264*
093-130-028	19	0	0	0	0	0.5	0	0	0.5	*11*	*0*	*6736*
111-120-036	6	0	0	0	0	1	0	0	0	*0*	*0*	*0*
1318-23-810-017	8	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
111-120-034	6	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
1418-03-811-025	8	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
023-501-13	2	0	0	0	0	0	0	0	1	*0*	*0*	*0*
023-103-14	2	0	0	0	0	0	0	0	1	*0*	*0*	*0*
023-103-15	2	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
023-103-16	2	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
023-103-11	2	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
023-512-06	2	0	0	0	0	0	0	0	1	*0*	*0*	*0*
023-512-09	2	0	0	0	0	0	0	0	1	*0*	*0*	*0*
025-271-04	4	0	0	0	0	1	0	0	0	*0*	*0*	*0*
116-110-015	6	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
083-300-015	6	0	0	0	0	1	0	0	0	*0*	*0*	*0*
090-041-009	32	0	0	0	0	0	0	0	1	*0*	*0*	*767*
031-081-10	27	0	0	0	0	0	0	0	1	*0*	*15*	*5635*
016-202-20	5	0	0	0	0	0	0	0	1	*0*	*0*	*0*
026-096-05	2	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
026-096-04	2	0	0	0	0	0	0	0	1	*0*	*0*	*0*
112-090-005	6	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
025-451-18	4	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
025-451-16	4	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
111-190-024	6	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
1418-10-611-001	1	0	0	0	0	0	0	0	1	*0*	*0*	*0*
022-171-68	1	0	0	0	0	0	0	0	1	*0*	*0*	*0*
022-171-60	3	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*
032-323-16	5	0	0	0	0	0.5	0	0	0.5	*0*	*0*	*0*

Table 5. TahoeCondensedTransitions.txt.

[This is only a sample of the total parcel-based output. “APN” is assessor’s parcel number; “ALLOC_DG,” a parcel is allocated to Douglas County; “ALLOC_EL,” a parcel is allocated to Eldorado County; “ALLOC_PL,” a parcel is allocated to Placer County; “ALLOC_PL_SENSITIVE,” a parcel is allocated to the sensitive lots of Placer County; “ALLOC_WA,” a parcel is allocated to Washoe County; “ALLOC_SLT,” a parcel is allocated to the city of South Lake Tahoe; “ALLOC_MFH,” a parcel is allocated to multiple family housing; “ALLOC_TOUR,” a parcel is allocated to Tourist accommodations; “ALLOC_COMM,” a parcel is allocated to commercial square footage; “RETIRE_CTC_PRIORITY,” a sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; “RETIRE_CTC_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; “RETIRE_NEV_PRIORITY,” a sensitive parcel is allocated to the retirement program of the Nevada State Lands; “RETIRE_NEV_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the Nevada State Lands; “RETIRE_TRPA_PRIORITY,” a sensitive parcel is allocated to the retirement program of the TRPA; “RETIRE_TRPA_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the TRPA; “RETIRE_USFS_PRIORITY,” a sensitive parcel is allocated to the retirement program of the U.S. Forest Service; “RETIRE_USFS_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the U.S. Forest Service]

APN	ALLOC_DG	ALLOC_EL	ALLOC_PL	ALLOC_PL_SENSITIVE	ALLOC_WA	ALLOC_SLT	ALLOC_MFH	ALLOC_TOUR	ALLOC_COMM	RETIRE_CTC_PRIORITY	RETIRE_CTC_SECONDARY	RETIRE_NEV_PRIORITY	RETIRE_NEV_SECONDARY	RETIRE_TRPA_PRIORITY	RETIRE_TRPA_SECONDARY	RETIRE_USFS_PRIORITY	RETIRE_USFS_SECONDARY
093-130-026	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
093-130-028	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
111-120-036	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1318-23-810-017	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
111-120-034	0	0	0.5	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0
1418-03-811-025	0.5	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0
023-501-13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0.5
023-103-14	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
023-103-15	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0.5
023-103-16	0	0	0	0	0	0.5	0	0	0	0	0.5	0	0	0	0	0	0
023-103-11	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0.5
023-512-06	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0.5
023-512-09	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0.5	0	0
025-271-04	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
116-110-015	0	0	0.5	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0
083-300-015	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
090-041-009	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
031-081-10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
016-202-20	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0.5	0
026-096-05	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0.5
026-096-04	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
112-090-005	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0
025-451-18	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0
025-451-16	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
111-190-024	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
1418-10-611-001	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
022-171-68	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
022-171-60	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0.5	0
032-323-16	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0

Table 6. TahoeLanduseByYear.txt.

[Y1 through Y20 designate the year of the transition; "LU_XXXX", a different formulation of the TRPA land use codes and the values are the percent of the time the model designates this particular Land Use; and "*MFHUnits*," "*TourUnits*", and "*CommSF*," static variables that do not change from one run to another and thus they should be disregarded as useful output]

Desc	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17	Y18	Y19	Y20
LU_0001	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LU_1005	13	9	12	9	16	9	8	6	8	12	10	4	-	-	-	-	-	-	-	-
LU_1006	1	2	2	1	2	2	2	2	2	1	3	1	-	-	-	-	-	-	-	-
LU_1007	1	1	-	1	-	1	1	1	1	1	-	1	-	-	-	-	-	-	-	-
LU_1011	187	203	173	173	128	155	123	166	163	143	149	89	-	-	-	-	-	-	-	-
LU_2002	2	2	3	2	3	4	7	3	3	2	3	1	-	-	-	-	-	-	-	-
LU_3000	3	4	2	2	-	-	-	10	3	3	2	1	-	-	-	-	-	-	-	-
LU_6401	357	368	407	379	335	389	375	347	362	357	308	254	-	-	-	-	-	-	-	-
MFHUnits	42	40	38	40	47	45	42	34	49	45	39	18	-	-	-	-	-	-	-	-
TourUnits	35	12	54	28	35	55	46	25	49	43	45	19	-	-	-	-	-	-	-	-
CommSF	6,976	4,412	3,231	4,396	-	-	-	28,036	21,558	7,323	11,024	4,531	-	-	-	-	-	-	-	-

Table 7. TahoeTransitionsByYear.txt.

[Y1 through Y20 designate the year of the transition; “ALLOC_DG,” a parcel is allocated to Douglas County; “ALLOC_EL,” a parcel is allocated to Eldorado County; “ALLOC_PL,” a parcel is allocated to Placer County; “ALLOC_PL_SENSITIVE,” a parcel is allocated to the sensitive lots of Placer County; “ALLOC_WA,” a parcel is allocated to Washoe County; “ALLOC_SLT,” a parcel is allocated to the city of South Lake Tahoe; “ALLOC_MFH,” a parcel is allocated to multiple family housing; “ALLOC_TOUR,” a parcel is allocated to tourist accommodation units; “ALLOC_COMM,” a parcel is allocated to commercial square footage; “RETIRE_CTC_PRIORITY,” a sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; “RETIRE_CTC_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; “RETIRE_NEV_PRIORITY,” a sensitive parcel is allocated to the retirement program of the Nevada State Lands; “RETIRE_NEV_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the Nevada State Lands; “RETIRE_TRPA_PRIORITY,” a sensitive parcel is allocated to the retirement program of the TRPA; “RETIRE_TRPA_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the TRPA; “RETIRE_USFS_PRIORITY,” a sensitive parcel is allocated to the retirement program of the U.S. Forest Service; “RETIRE_USFS_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the U.S. Forest Service]

Desc	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17	Y18	Y19	Y20
ALLOC_DG	12	17	14	14	15	12	2	-	-	-	-	-	-	-	-	-	-	-	-	-
ALLOC_EL	74	67	58	67	45	70	64	87	78	71	77	32	-	-	-	-	-	-	-	-
ALLOC_PL	34	51	46	45	38	44	36	50	53	46	44	30	-	-	-	-	-	-	-	-
ALLOC_PL_SENSITIVE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALLOC_WA	31	41	24	18	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALLOC_SLT	36	29	32	29	30	31	21	29	32	27	29	28	-	-	-	-	-	-	-	-
ALLOC_MFH	14	11	13	11	18	12	11	8	11	14	12	5	-	-	-	-	-	-	-	-
ALLOC_TOUR	2	2	3	2	3	4	7	3	3	2	3	1	-	-	-	-	-	-	-	-
ALLOC_COMM	3	4	2	2	-	-	-	10	3	3	2	1	-	-	-	-	-	-	-	-
RETIRE_CTC_PRIORITY	59	86	99	73	59	93	88	55	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_CTC_SECONDARY	79	84	101	79	61	81	93	120	189	171	140	124	-	-	-	-	-	-	-	-
RETIRE_NEV_PRIORITY	17	16	14	14	18	13	12	4	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_NEV_SECONDARY	22	17	11	17	18	16	14	3	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_TRPA_PRIORITY	17	21	17	20	17	18	17	11	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_TRPA_SECONDARY	18	16	21	17	22	17	21	27	41	38	37	29	-	-	-	-	-	-	-	-
RETIRE_USFS_PRIORITY	73	62	73	83	74	81	57	42	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_USFS_SECONDARY	73	69	73	77	69	71	75	87	132	149	132	102	-	-	-	-	-	-	-	-

Table 8. TahoeTransCandidatesByYear.txt.

[Y1 through Y20 designate the year of the transition; “ALLOC_DG,” a parcel is allocated to Douglas County; “ALLOC_EL,” a parcel is allocated to Eldorado County; “ALLOC_PL,” a parcel is allocated to Placer County; “ALLOC_PL_SENSITIVE,” a parcel is allocated to the sensitive lots of Placer County; “ALLOC_WA,” a parcel is allocated to Washoe County; “ALLOC_SLT,” a parcel is allocated to the city of South Lake Tahoe; “ALLOC_MFH,” a parcel is allocated to multiple family housing; “ALLOC_TOUR,” a parcel is allocated to tourist accommodation units; “ALLOC_COMM,” a parcel is allocated to commercial square footage; “RETIRE_CTC_PRIORITY,” a sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; “RETIRE_CTC_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the California Tahoe Conservancy; “RETIRE_NEV_PRIORITY,” a sensitive parcel is allocated to the retirement program of the Nevada State Lands; “RETIRE_NEV_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the Nevada State Lands; “RETIRE_TRPA_PRIORITY,” a sensitive parcel is allocated to the retirement program of the TRPA; “RETIRE_TRPA_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the TRPA; “RETIRE_USFS_PRIORITY,” a sensitive parcel is allocated to the retirement program of the U.S. Forest Service; “RETIRE_USFS_SECONDARY,” a non-sensitive parcel is allocated to the retirement program of the U.S. Forest Service]

Desc	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17	Y18	Y19	Y20
ALLOC_DG	150	123	89	63	39	16	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALLOC_EL	1,823	1,649	1,475	1,305	1,147	1,019	849	697	533	341	180	45	-	-	-	-	-	-	-	-	-
ALLOC_PL	1,130	1,109	1,014	913	813	731	625	535	421	295	168	66	-	-	-	-	-	-	-	-	-
ALLOC_PL_SENSITIVE	60	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALLOC_WA	155	105	52	22	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALLOC_SLT	1,268	1,177	1,089	991	888	801	707	600	487	362	234	107	-	-	-	-	-	-	-	-	-
ALLOC_MFH	797	716	638	561	498	431	358	300	236	164	86	29	-	-	-	-	-	-	-	-	-
ALLOC_TOUR	98	91	83	70	62	55	47	34	26	18	12	4	-	-	-	-	-	-	-	-	-
ALLOC_COMM	286	263	241	222	203	184	166	146	121	91	61	31	-	-	-	-	-	-	-	-	-
RETIRE_CTC_PRIORITY	1,299	1,145	978	791	616	459	266	106	-	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_CTC_SECONDARY	4,463	4,155	3,846	3,521	3,209	2,939	2,621	2,299	1,880	1,340	822	349	-	-	-	-	-	-	-	-	-
RETIRE_NEV_PRIORITY	260	212	154	110	75	47	24	6	-	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_NEV_SECONDARY	259	208	155	115	70	43	18	3	-	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_TRPA_PRIORITY	1,559	1,357	1,131	901	691	505	290	111	-	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_TRPA_SECONDARY	4,722	4,362	4,001	3,635	3,279	2,982	2,639	2,302	1,880	1,340	822	349	-	-	-	-	-	-	-	-	-
RETIRE_USFS_PRIORITY	1,559	1,357	1,131	901	691	505	290	111	-	-	-	-	-	-	-	-	-	-	-	-	-
RETIRE_USFS_SECONDARY	4,722	4,362	4,001	3,635	3,279	2,982	2,639	2,302	1,880	1,340	822	349	-	-	-	-	-	-	-	-	-

Table 9. TahoeCategoryTotals.txt.

["Category", a particular combination of fates for a given parcel; "Count", number of times that category and combination of fates occurs; "LU_XXXX", a different formulation of the TRPA land use codes and the values are the percent of the time the model designates this particular land use]

Category	Count	LU_0001	LU_MFH	LU_1011	LU_2002	LU_3000	LU_6401
1	1435	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%
4	1228	0.0%	0.0%	50.3%	0.0%	0.0%	49.7%
6	949	0.0%	0.0%	46.5%	0.0%	0.0%	53.5%
2	855	0.0%	0.0%	31.8%	0.0%	0.0%	68.2%
5	320	0.0%	0.0%	16.4%	0.0%	0.0%	83.6%
12	260	0.0%	23.8%	23.3%	0.0%	0.0%	52.9%
14	252	0.0%	16.5%	44.4%	0.0%	0.0%	39.1%
16	149	0.0%	15.1%	36.9%	0.0%	0.0%	48.0%
3	118	0.0%	0.0%	12.7%	0.0%	0.0%	87.3%
8	94	0.0%	0.0%	58.5%	0.0%	0.0%	41.5%
32	74	0.0%	0.0%	0.0%	0.0%	15.5%	84.5%
10	64	0.0%	0.0%	82.0%	0.0%	0.0%	18.0%
7	59	0.0%	0.0%	14.4%	0.0%	0.0%	85.6%
9	53	0.0%	0.0%	51.9%	0.0%	0.0%	48.1%
11	49	0.0%	0.0%	72.4%	0.0%	0.0%	27.6%
30	41	0.0%	0.0%	0.0%	0.0%	13.4%	86.6%
31	31	0.0%	0.0%	0.0%	0.0%	14.5%	85.5%
13	28	0.0%	16.1%	5.4%	0.0%	0.0%	78.6%
0	25	0.0%	2.0%	14.0%	14.0%	0.0%	70.0%
27	25	0.0%	0.0%	0.0%	40.0%	10.0%	50.0%
15	23	0.0%	4.3%	13.0%	0.0%	0.0%	82.6%
18	22	0.0%	4.5%	61.4%	0.0%	0.0%	34.1%
28	22	0.0%	0.0%	0.0%	43.2%	6.8%	50.0%
20	20	0.0%	0.0%	25.0%	25.0%	10.0%	40.0%
17	14	0.0%	3.6%	64.3%	0.0%	0.0%	32.1%
19	12	0.0%	8.3%	33.3%	0.0%	0.0%	58.3%
24	12	0.0%	12.5%	0.0%	0.0%	8.3%	79.2%
22	9	0.0%	11.1%	0.0%	0.0%	0.0%	88.9%
26	7	0.0%	0.0%	0.0%	35.7%	0.0%	64.3%
29	7	0.0%	0.0%	0.0%	7.1%	0.0%	92.9%
33	7	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%
21	6	0.0%	0.0%	66.7%	0.0%	0.0%	33.3%
25	4	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%
34	4	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%
23	3	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%

Appendixes

Appendix 1: Tahoe Regional Planning Agency's Parcel Land Use Codes

Bolded codes are used in the LUSM.

(Updated 10-8-98)

0 – 999	**MISCELLANEOUS**
0	Unknown
1	**Vacant (private)**
99	Streets/Highways
1000	**RESIDENTIAL**
1001	Domestic animal raising
1002	Employee housing
1003	Mobile home dwelling
1004	* Mobile Home Park
1005	**Multiple family dwelling (2-4 units)**
1006	**Multiple family dwelling (5-10 units)**
1007	**Multiple family dwelling (10+ units)**
1008	Multi-person dwelling
1009	Nursing and personal care
1010	Residential care
1011	**Single family dwelling (Existing)**
1012	Single family dwelling (New)
1013	Summer home
1014	Condominium
1015	Condominium Common Area
1016	Accessory use to a Single Family Dwelling
1017	Abandoned Residential Structure
2000	**TOURIST ACCOMMODATION**
2001	Bed and breakfast facilities
2002	**Hotel, motel, and other transient dwelling units**
2003	Time sharing (hotel/motel design)
2004	Time sharing (residential design)
3000	**COMMERCIAL**
3100	A. Retail
3101	Auto, mobile home and vehicle dealers
3102	Building materials and hardware
3103	General merchandise stores
3104	Mail order and vending
3105	Nursery
3106	Outdoor retail sales
3107	Eating and drinking places
3108	Food and beverage retail sales
3109	Furniture, home furnishings and equipment
3110	Service Stations
3200	B. Entertainment
3201	Amusements and recreation services
3202	Gaming-nonrestricted (Nevada only)
3203	Privately owned assembly and entertainment
3204	Outdoor amusements
3300	C. Services

3301	Animal husbandry services
3302	Auto repair and service
3303	Broadcasting studios
3304	Business support services
3305	Contract construction services
3306	Financial services
3307	Health care services
3308	Laundries and dry cleaning plant
3309	Personal services
3310	Professional offices
3311	Repair services
3312	Sales lots
3313	Schools – business and vocational
3314	Secondary storage

3400 D. Light Industrial

3401	Batch plants
3402	Food and kindred products
3403	Fuel and ice dealers
3404	Industrial services
3405	Printing and publishing
3406	Recycling and scrap
3407	Small scale manufacturing

3500 E. Wholesale/Storage

3501	Storage yards
3502	Vehicle and freight terminals
3503	Vehicle storage and parking
3504	Warehousing
3505	Wholesale and distribution
3601	* Abandoned Commercial Structure

4000 PUBLIC SERVICE

4100 A. General

4101	Airfields, landing strips, heliports (New non-emergency sites prohibited)
4102	Cemeteries
4103	Churches
4104	Collection stations
4105	Cultural facilities
4106	Day care centers/pre-schools
4107	Government offices
4108	Hospitals
4109	Local assembly and entertainment
4110	Local post office
4111	Local public health and safety facilities
4112	Power generating
4113	Public owned assembly and entertainment
4114	Public utility centers
4115	Regional public health and safety facilities
4116	Schools – college
4117	Schools – Kindergarten through secondary
4118	Social service organizations

4200 B. Linear Public Facilities

4201	Pipelines and power transmission
4202	Transit stations and terminals
4203	Transportation routes
4204	Transmission and receiving facilities

5000 RECREATION

5001	Beach recreation
5002	Boat launching facilities
5003	Cross country ski courses
5004	Day use areas
5005	Developed campgrounds
5006	Downhill ski facilities
5007	Golf courses
5008	Group facilities
5009	Marinas
5010	Off road vehicle courses
5011	Outdoor recreation concessions
5012	Participant sports facilities
5013	Recreation centers
5014	Recreations vehicle parks
5015	Riding and hiking trails
5016	Rural Sports
5017	Snowmobile courses
5018	Sport Assembly
5019	Undeveloped campgrounds
5020	Visitor information centers

6000 RESOURCE MANAGEMENT

6100	A. Timber Management
6101	Reforestation
6102	Regeneration harvest
6103	Sanitation salvage cut
6104	Selection cut
6105	Special cut
6106	Thinning
6107	Timber stand improvement
6108	Tree farms
6200	B. Wildlife and Fishes
6201	Early successional vegetation management
6202	Non-structural fish habitat management
6203	Non-structural wildlife habitat management
6204	Structural fish habitat management
6205	Structural wildlife habitat management
6300	C. Range
6301	Farm/Ranch structures
6302	Grazing
6303	Range pasture management
6304	Range improvement
6400	D. Open Space
6401	**Allowed in all areas of the region**
6500	E. Vegetation Protection
6501	Fire detection and suppression
6502	Fuels treatment/management
6503	Insect and disease suppression
6504	Prescribed fire/burning management
6505	Sensitive plant management
6506	Uncommon plant community management
6600	F. Watershed Improvements
6601	Erosion control
6602	Runoff control
6603	Stream environment zone restoration

7000	**SHOREZONE**
7010	Safety and navigational facilities
7020	Salvage operations
7030	Seaplane operations
7040	Tour boat operations
7050	Water borne transit
7060	Water intake lines
7070	Beach recreation
7080	Boat launching facilities
7090	Construction equipment storage
7100	Marinas
7110	Water oriented outdoor recreation concessions
7120	Commercial boating
7130	Parasailing
7140	Fish Habitat Restoration
7150	Scientific Study

Shorezone Accessories (4th digit in project code)

1	Boat ramp
2	Breakwaters or jetties
3	Buoys
4	Fences
5	Floating docks and platforms
6	Piers
7	Shoreline protective structures
8	Water intake lines
9	Beach raking

EIP PROJECT CODES

9000 EIP GENERAL

9010	Multi-purpose project (general)
9011	Redevelopment Project
9012	Community Plan Project
9013	Public Agency Project
9014	Private Funded Project
9080	Multi-Purpose Programs
9081	Research and Evaluation (TRPA)
9090	Multi-Purpose Regulations
9091	Streamlining

9100 WATER QUALITY

9110	Research/Evaluation Project
9120	SEZ Restoration Project
9130	SEZ Creation Project
9140	Erosion Control/WQ Treatment
9150	Soil Conservation Project
9180	Water Quality Programs
9181	Research and Evaluation (TRPA)
9182	SEZ Research
9190	Water Quality Regulations

9200 AQ/TRANSPORTATION

9210	Transit Project
9220	Highway Project
9230	Airport Project
9240	Bikeway Project
9250	Pedestrian Project
9260	Multi-Purpose Project
9270	Parking Project
9280	AQ/Transportation Program
9281	Research and Evaluation (TRPA)
9290	AQ/Transportation Regulation

9300 NOISE

9310	Noise Reduction Project
9380	Noise Reduction Program
9381	Research and Evaluation (TRPA)
9390	Noise Reduction Regulations

9400 RECREATION

9410	PAOT Projects
9411	Overnight
9412	Day Use
9413	Winter
9415	Marinas/Ramps
9420	Non PAOT Projects
9430	Recreation Land Acquisition
9480	Recreation Program
9481	Research and Evaluation (TRPA)
9491	Recreation Regulation

9500 FISHERY

9510	Stream
9520	Lake
9580	Fishery Program
9581	Research and Evaluation – In Stream
9582	Research and Evaluation – In Lake
9590	Fishery Regulation
9591	In-Stream Fisheries
9592	In-Lake Fisheries

9600 WILDLIFE

9610	Wildlife Enhancement Projects
9680	Wildlife Programs
9681	Research and Evaluation (TRPA)
9690	Wildlife Regulation

9700 SCENIC

9710	Scenic Restoration Project
9720	Undergrounding
9780	Scenic Programs
9781	Research and Evaluation (TRPA)
9790	Scenic Regulation

9800 VEGETATION

9810	Forest Project
9880	Vegetation Program

9881	Research and Evaluation (TRPA)
9890	Vegetation Regulation

9900 OTHER

9910	Health and Safety

Appendix 2: Assessor's 2008 Parcel Map Data Summary

[Level I of the table represents the first major category of land uses as defined in appendix 1 (e.g. 1000), level II represents the second category (e.g. 3100), and level III represents the most specific hierarchical level of the land use categories (e.g. 3105)]

Level I		Level II		Level III			Area in Acres				Level II Total	Level III Total
Description	**Count**	**Description**	**Count**	**Description**	**Count**	**Landuse**	**Min Area**	**Med Area**	**Max Area**	**Total Area**		
Unknown (null)	210		210	Unknown (or null)	210	0	0.0	0.1	4,182	6,707	6,707	6,707
Vacant	6,326		6,326	Vacant	6,326	1	0.0	0.2	468	6,801	6,801	6,801
Residential	41,219		41,219	Domestic Animal Raising	1	1001	24.3	24.3	24	24	13,790	13,790
				Mobile Home Dwelling	747	1003	0.0	0.0	1	30		
				Mobile Home Park	34	1004	0.1	0.9	23	82		
				Multiple Family Dwelling (2-4 units)	1,697	1005	0.0	0.1	41	592		
				Multiple Family Dwelling (5-10 units)	196	1006	0.1	0.2	20	79		
				Multiple Family Dwelling (10+ units)	82	1007	0.0	0.5	7	86		
				Nursing and Personal Care	1	1009	1.5	1.5	1	1		
				Residential Care	1	1010	1.3	1.3	1	1		
				Single Family Dwelling (existing)	28,927	1011	0.0	0.2	596	10,825		
				Summer Home	518	1013	0.1	0.3	134	378		
				Condominium	7,988	1014	0.0	0.0	9	218		
				Condominium Common Area	725	1015	0.0	0.5	87	1,372		
				Accessory use to a Single Family Dwelling	302	1016	0.0	0.2	4	100		
Tourist	427		427	Bed and Breakfast Facilities	3	2001	0.3	0.3	1	2	389	389
				Hotel, Motel, and other Transient Dwelling Units	256	2002	0.1	0.5	22	320		
				Time Sharing (Hotel/Motel Design)	20	2003	0.0	0.6	6	25		
				Time Sharing (Residential Design)	148	2004	0.0	0.0	24	42		
Commercial	1,279	Retail	430	n/a	1	3100	0.2	0.2	0	0	340	971
				Auto, Mobile Home and Vehicle Dealers	8	3101	0.2	0.5	3	8		
				Building Materials and Hardware	17	3102	0.2	0.5	4	18		
				General Merchandise Stores	197	3103	0.0	0.3	18	160		
				Mail Order and Vending	6	3105	0.3	0.4	1	3		
				Nursery	7	3106	0.1	0.3	1	2		
				Outdoor Retail Sales	112	3107	0.0	0.4	5	73		
				Eating and Drinking Places	25	3108	0.0	0.5	14	33		
				Food and Beverage Retail Sales	9	3109	0.1	0.3	1	3		
				Furniture, Home Furnishings and Equipment	35	3110	0.1	0.5	1	17		
				Service Stations	13	3111	0.1	0.8	8	21		

		Entertainment	14	Amusements and Recreation Services	7	3201	0.1	1.1	4	11	20	
				Gaming-nonrestricted (Nevada only)	5	3202	0.2	0.9	4	8		
				Privately Owned Assembly and Entertainment	2	3203	0.2	0.3	0	1		
		Services	385	Animal Husbandry Services	5	3301	0.1	0.4	1	3	197	
				Auto Repair and Service	55	3302	0.1	0.3	2	28		
				Broadcasting Studios	1	3303	1.1	1.1	1	1		
				Business Support Services	3	3304	0.2	0.3	2	3		
				Contract Construction Services	27	3305	0.1	0.3	2	11		
				Financial Services	16	3306	0.1	0.7	2	12		
				Health Care Services	37	3307	0.1	0.5	2	21		
				Laundries and Dry Cleaning Plant	2	3308	0.1	0.3	0	1		
				Personal Services	30	3309	0.0	0.2	2	11		
				Professional Offices	198	3310	0.0	0.3	6	103		
				Repair Services	7	3311	0.0	0.3	1	2		
				Sales Lots	1	3312	0.3	0.3	0	0		
				Schools - Business and Vocational	1	3313	0.2	0.2	0	0		
				Secondary Storage	2	3314	0.3	0.4	0	1		
		Light Industrial	91	Batch Plants	5	3401	0.6	1.0	4	9	172	
				Food and Kindred Products	26	3402	0.1	1.0	45	73		
				Fuel and Ice Dealers	5	3403	0.2	1.0	2	5		
				Industrial Services	23	3404	0.0	0.5	49	64		
				Printing and Publishing	3	3405	0.2	0.3	1	1		
				Recycling and Scrap	6	3406	0.5	0.6	3	6		
				Small Scale Manufacturing	23	3407	0.1	0.3	3	14		
		Whlsle Storage	359	Storage Yards	168	3501	0.0	0.0	34	76	243	
				Vehicle and Freight Terminals	9	3502	0.1	0.5	1	4		
				Vehicle Storage and Parking	101	3503	0.0	0.3	60	116		
				Warehousing	75	3504	0.0	0.3	4	43		
				Wholesale and Distribution	6	3505	0.2	0.6	2	5		
Public Service	537	General	349	n/a	1	4100	0.0	0.0	0	0	1,724	1,817
				Airfields, Landing Strips, Heliports	11	4101	1.0	23.5	70	336		
				Cemeteries	4	4102	0.3	0.9	10	12		
				Churches	31	4103	0.0	1.6	5	58		
				Cultural Facilities	6	4105	0.2	7.9	54	108		
				Day Care Centers/Pre-Schools	10	4106	0.1	0.4	1	5		
				Government Offices	20	4107	0.0	1.2	30	76		
				Hospitals	3	4108	3.4	5.9	7	16		
				Local Assembly and Entertainment	7	4109	0.1	0.2	3	5		
				Local Post Office	11	4110	0.2	0.7	6	11		
				Local Public Health and Safety Facilities	59	4111	0.0	0.4	120	211		
				Power Generating	7	4112	0.1	0.6	3	6		

				Public Owned Assembly and Entertainment	1	4113	13.4	13.4	13	13			
				Public Utility Centers	115	4114	0.0	0.2	87	257			
				Regional public health and Safety Facilities	18	4115	0.1	0.7	33	66			
				Schools - College	7	4116	0.4	1.6	113	132			
				Schools - Kindergarten through Secondary	31	4117	0.2	5.0	80	409			
				Social Services Organizations	7	4118	0.0	0.2	2	3			
		Linear Facilities	188	Pipelines and Power Transmission	3	4201	0.0	0.3	0	1	93		
				Transit Stations and Terminals	3	4202	0.6	0.8	11	12			
				Transportation Routes	176	4203	0.0	0.2	9	76			
				Transmission and Receiving Facilities	6	4204	0.0	0.5	2	4			
Recreation	621		621	Beach Recreation	71	5001	0.0	1.2	226	389	19,693	19,693	
				Boat Launching Facilities	8	5002	0.4	2.3	11	26			
				Day Use Areas	150	5004	0.0	7.0	3,765	12,403			
				Developed Campgrounds	16	5005	5.1	36.0	307	1,172			
				Downhill Ski Facilities	91	5006	0.1	10.5	821	4,262			
				Golf Courses	44	5007	0.0	4.7	224	1,013			
				Group Facilities	9	5008	0.7	9.6	33	110			
				Marinas	177	5009	0.0	0.0	19	66			
				Outdoor Recreation Concessions	6	5011	0.2	1.0	6	11			
				Participant Sports Facilities	9	5012	0.2	3.5	30	87			
				Recreation Centers	12	5013	0.9	1.5	18	52			
				Riding and Hiking Trails	22	5015	0.1	1.9	23	96			
				Rural Sports	1	5016	0.7	0.7	1	1			
				Undeveloped Campgrounds	1	5019	4.6	4.6	5	5			
				Visitor Information Centers	4	5020	0.0	0.4	1	2			
Resource Mgmt	9,757	Open Space	9,757	Retired	9,757	6401	0.0	0.3	3,747	164,385	164,385	164,385	
	60,376		60,376		60,376					214,552	214,552	214,552	

Appendix 3: Samples of Required Databases for the Land Use Simulation Model

Below are samples of records and their attributes of the two primary datasets used in PostGreSQL.

Table 1. Parcel database with Plan Area Statement ID linkage.

Assessor's Parcel Number	Plan Area Statement ID	Jurisdiction	Area (ac)	IPES Score	Bailey Score	South Lake Tahoe
010-170-01	1520	EL	49.475	0	4	FALSE
010-170-02	1520	EL	160.659	0	4	FALSE
014-231-05	1540	EL	0.263	826	5	FALSE
014-231-09	1540	EL	0.323	874	5	FALSE
014-232-01	1540	EL	0.230	836	5	FALSE
014-232-03	1540	EL	0.203	814	5	FALSE
014-232-09	1540	EL	0.276	814	5	FALSE
014-234-02	1540	EL	0.246	846	5	FALSE
014-234-11	1540	EL	0.246	846	5	FALSE
014-235-06	1540	EL	0.276	836	1b	FALSE
014-236-05	1540	EL	0.230	826	5	FALSE
014-236-08	1540	EL	0.230	804	5	FALSE
014-236-12	1540	EL	0.230	811	5	FALSE
014-237-03	1540	EL	0.230	820	5	FALSE
014-237-04	1540	EL	0.230	820	5	FALSE
014-238-12	1540	EL	0.246	836	5	FALSE
014-241-07	1540	EL	0.230	830	1b	FALSE
014-242-06	1540	EL	0.229	0	1b	FALSE
014-243-07	1540	EL	0.230	440	5	FALSE
014-244-06	1540	EL	0.246	805	5	FALSE
014-244-11	1540	EL	0.246	783	5	FALSE
014-247-04	1540	EL	1.758	0	4	FALSE
014-262-03	1540	EL	0.232	819	5	FALSE
014-262-10	1540	EL	0.244	768	5	FALSE
014-262-12	1540	EL	0.232	807	5	FALSE

Table 2. Plan Area Statement database

Plan Area Statement ID	Plan Area Statement Name	Multiple Family Dwelling	Single Family Dwelling	Tourist Accommodation	Commercial Uses	Multiple Family Dwelling Density (units/ac)	Tourist Accommodation Density (units/ac)
001A1	TAHOE CITY	N	N	A	A	0	40
001A2	TAHOE CITY	N	N	N	A	0	0
001A3	TAHOE CITY	N	N	N	S	0	0
001A4	TAHOE CITY	A	A	A	S	15	40
001A5	TAHOE CITY	A	A	S	A	15	40
20	FAIRWAY TRACT	N	A	N	N	0	0
21	FAIRWAY TRACT SA	A	A	N	N	8	0
22	FAIRWAY TRACT SA	N	A	S	A	0	40
30	LOWER TRUCKEE	N	A	N	S	0	0
40	BURTON CREEK	N	S	N	S	0	0
50	ROCKY RIDGE	N	A	S	S	0	20
70	LAKE FOREST GLEN	A	A	N	N	15	0
71	LAKE FOREST GLEN	A	A	N	A	15	0
80	LAKE FOREST	N	A	N	N	0	0
81	LAKE FOREST	N	A	N	S	0	0
009A1	LAKE FOREST COM.	S	S	S	A	15	40
009A2	LAKE FOREST COM.	S	S	N	A	15	0
009B0	DOLLAR HILL	S	S	N	A	15	0
100	DOLLAR POINT	N	A	N	N	0	0
120	NORTH TAHOE HIGH	N	S	N	N	0	0
130	WATSON CREEK	N	N	N	N	0	0

Notes: (A) is allowed use, (S) is a special use and must be considered under the provisions of the particular plan area statement or community plan, and (N) is not allowed use. The density of single family dwellings is 1 unit per parcel, and the density of commercial lots is 25 percent of a parcel's area.

Appendix 4: Python Code for the Land Use Simulation Model

The following code and its extensive comments executes as a stand-alone version and is provided here as a professional courtesy for any developer who would like to apply the code to their own problem or location, or prove to themselves that the code is adequate and fully functional. Note that this version of the code is not an exact copy of the code behind the TIIMS-hosted decision support tool.

```
from random import *
from copy import deepcopy
from pg import DB
import datetime
import time


#Tahoe Decision Support Software
#Model coded by Ben Oldham
#Model logic prototype developed by Ben Oldham and Will Forney
#Model transferred to new Desktop by Will Forney, Peter Ng and Mike Gould. Will Updated Default Model Values, 1/25/2010.

#Notes:
#
# The model consists of a single simulation within which there are a number of runs.
# During the course of a simulation run, vacant parcels can stay vacant or they can transition into a variety of other landuses.
# These landuses are LU_0001, LU_1005, LU_1006, LU_1007, LU_1011, LU_2002, LU_3000, LU_6401.
# These transitions can occur as a result as a variety of different mechanisms.

#Generalized Model flow:
# Pass in the Model ID.
# Initialize the model object:
#  Obtain the model parameter values.
#  Get PAS db data (model.PASList) that specifies what landuses and densities are allowed for the parcels in each PAS (PAS.allowedLanduseDensities).
#  Get vacant parcel db data (model.parcelList) with attributes that help determine the various possible fates of a parcel.
```

```
#  For each parcel, identify the possible transitions and landuses that the parcel qualifies for (parcel.possibleTransitionsMasterList, parcel.landuseCounts).
#  Generate a master set of lists (model.transitionCandidatesMasterLists), one for each transition type, with each list containing the apns of the parcels that
qualify for that transition type.
#  Initialize a number of analysis statistics.
#  Identify each parcels category (parcels with the same category all share functionally similar attributes and can be expected to have similar outcome
probabilities).
# Initialize the run object.
# Perform n runs (where n = the ITERATIONS parameter):
#  At the start of each run, initialize the run (run.initRun):
#   Initialize the year and pool values, etc.
#   Refresh the working copy of the transition candidate parcel lists (thisRunTransitionCandidates) and randomize them.
#   For each parcel, copy the thisRunPossibleTransitions list from the possibleTransitionsMasterList.
#  For each year of the run:
#   Initialize the thisYearTrans list (initRunYear):
#    Determine how many transitions of each type will be scheduled for the current year based on the input parameters (run.thisYearTrans) (stochastic values).
#     For all SFH and Retirement transitions, add that number of items to the list.
#     MFH, Tourist and Commercial each have only a single item added to the list as they are handled with a different mechanism (pools).
#    Randomize the list.
#   Do some minor annual tasks (filterLargeCommLots, checkVacantLotEqPL).
#   Attempt to do the current year's scheduled transitions:
#    For each transition, determine the transition list, landuse, and transition type (there are special pool mechanisms for MFH, Tour and Comm).
#    If a candidate parcel exists (the length of the transition list is greater than 0), do the transition (if no candidates exist, then the scheduled transition is
disregarded):
#     With the first parcel in the transition list:
#      For each of the transition types in the parcel thisRunPossibleTransitions, remove that parcel from the associated transition list (in
thisRunTransitionCandidates).
#      Update various analysis statistics.
#   Increment the run year.
# Now that the simulation is done, do some minor massaging of the analysis statistics to make them ready for output (calcOutputResults).
# Print the seven analysis reports.


class TDSS(object):
 #class instantiates the Model and Run objects, runs the appropriate number of iterations, and prints the results

 def __init__(self, modelID):
#Starting the simulation.
#The model object stores data and results for the entire course of the simulation (all runs).
#The run object stores data and results for a single run of the simulation.
  print "START: ", datetime.datetime.fromtimestamp(time.time())
```

```
  model = Model(modelID)
#Initialize the Model object - get db data (paramater, pas, parcel), do initialization for the simulation.
  run = Run(model)
#Initialize the Run object.
#Now that we're initialized, we can do the simulation.
  for iteration in range(model.params["ITERATIONS"]):
   print "ITER = ", iteration
   run.initRun()                                          #Initialize start-of-run values and transition lists.
   while (run.thisYear <= model.params["TOTAL_YEARS"]):
    run.initRunYear()                                     #Determine the scheduled transitions for the year.
    run.filterLargeCommLots()                             #Unclog the commercial "development pipeline".
    run.checkVacantLotEqPL()                              #See if the PL vacant lot equation is satisfied.
    run.annualTransitions()                               #Attempt to process the scheduled transitions for the year.
    run.thisYear += 1
#Simulation is finished, now print the results
  model.calcOutputResults()                               #Massage the analysis statistics to make them ready for output.
  model.printLanduse(str(modelID))                        #Analysis report #1.
  model.printCondensedLanduse(str(modelID))               #Analysis report #2.
  model.printCondensedTransitions(str(modelID))           #Analysis report #3.
  model.printLanduseByYear(str(modelID))                  #Analysis report #4.
  model.printTransitionsByYear(str(modelID))              #Analysis report #5.
  model.printTransCandidatesByYear(str(modelID))          #Analysis report #6.
  model.printCategoryTotals(str(modelID))                 #Analysis report #7.
  print "END:", datetime.datetime.fromtimestamp(time.time())


class Model(object):

 def __init__(self, modelID):
#The model object stores data and results for the entire course of the simulation (all runs).

  self.landuseNames = ("LU_0001", "LU_1005", "LU_1006", "LU_1007", "LU_1011", "LU_2002", "LU_3000", "LU_6401", "MFHUnits", "TourUnits",
"CommSF") #really a grabbag of stuff - used for the TahoeLanduseByYear report
  self.transTypes = ("ALLOC_DG", "ALLOC_EL", "ALLOC_PL", "ALLOC_PL_SENSITIVE", "ALLOC_WA", "ALLOC_SLT", "ALLOC_MFH",
"ALLOC_TOUR", "ALLOC_COMM", "RETIRE_CTC_PRIORITY", "RETIRE_CTC_SECONDARY", "RETIRE_NEV_PRIORITY",
"RETIRE_NEV_SECONDARY", "RETIRE_TRPA_PRIORITY", "RETIRE_TRPA_SECONDARY", "RETIRE_USFS_PRIORITY",
"RETIRE_USFS_SECONDARY")
  self.dbConn = self.getDBConn()
  self.params = self.getFixedParams()                                    #Simulation parameters
  self.params["MODEL_ID"] = modelID
```

```
  self.params.update(self.getUserParams())
  self.PASList = self.getPASs()                                          #List of all PASs, key is pas_id
  self.parcelList = self.getParcels(self.params["IPES_THRESHOLD"])       #List of all vacant parcels, key is apn
  self.dbConn.close()
  self.getParcelPossibleFates()                                          #For each parcel, identify valid transition types and add them to the parcels
possibleTransitionsMasterList.
 # Also initializes the parcel landuse analysis statistic (landuseCounts) (reports #1 & 2).
  self.transitionCandidatesMasterLists = self.popTransitionCandidates() #A master set of lists that never change, one for each transition type.
 # Each sub-list contains a list of parcels (apn numbers) that are valid candidates for that transition type.
 # If an apn is in a particular model.transitionCandidatesMasterLists list, than the possibleTransitionsMasterList for that parcel will always include that transition
type.
 # There is always a one-to-one correspondence between the apns in a particular model.transitionCandidatesMasterLists list and the transitions in the member
parcel.possibleTransitionsMasterList.
  self.initParcelTransitionCounts()                                      #Initialize the parcel transition type analysis statistic (report #3).
  self.landuseByYear = self.initLanduseByYear()                          #Initialize the simulation landuse by year analysis statistic (report #4).
  self.transitionsByYear = self.initTransitionsByYear()                  #Initialize the simulation transitions by year analysis statistic (report #5).
  self.transCandidatesByYear = self.initTransCandidatesByYear()          #Initialize the simulation transitions candidates by year analysis statistic (report
#6).
  self.categoryTotals = self.initCategoryTotals()                        #Initialize the simulation category totals analysis statistic (report #7).
  self.calcParcelCatLanduseBinaryTotal()                                 #Calculate parcel landuseBinaryTotal values (used to help determine the parcel's
category).
  self.calcParcelCategory()                                              #For each parcel, identify the category for the parcel.


 def getDBConn(self):
#Opens a connection to the postgresql database.
#Java version may want to read connection parameters from some sort of ini file

  dbname = "test2"
  host = "localhost"
  port = 5432
  options = None
  tty = None
  user = "postgres2"
  password = "QAZSE45tgb1!2@"
  return DB(dbname, host, port, options, tty, user, password) # returns a connection to the database

 def getFixedParams(self):
#Getting the model parameters that aren't exposed to the users for input.
#Java version will probably read paramater values from a model_execution_parameters table using the Model ID as an identifier.
```

```
#Both fixed and user parameter values should probably be stored in the model_execution_parameters table.
#Master default values should probably be stored in a static table (perhaps fixed_params).

  params = {}
  dbParams = self.dbConn.query("SELECT * FROM fixed_params").dictresult()
#params["ITERATIONS"] = dbParams[0]["iterations"]
#The somewhat arbitrary value of 500 used in the previous version of the model is again used as a default.
  params["ITERATIONS"] = 2
  params["IPES_THRESHOLD"] = dbParams[0]["ipes_threshold"]
#At some point in the run, threshold may change to zero once the PL vacant lot equation has been satisfied.
  params["COMM_AREA_RATIO"] = dbParams[0]["comm_area_ratio"]
#Ratio used to tell the net commercial area sf based on the parcel size (default set to historical value of .25)
  params["MFD_UNITS_PER_ACRE"] = dbParams[0]["mfd_units_per_acre"]   #Empirically derived (default set to 18).
  params["TOUR_UNITS_PER_ACRE"] = dbParams[0]["tour_units_per_acre"] #Empirically derived (default set to 30).
  return params

 def getUserParams(self):
#Getting the model parameters with user-input values.
#Java version will probably read paramater values from a model_execution_parameters table using the Model ID as an identifier.
#Both fixed and user parameter values should probably be stored in the model_execution_parameters table.
#Master default values should probably be stored in a static table (perhaps default_user_params) and then displayed to the user.
#Most of the values (annual housing allocation, retirement, allocation rollover pool) were determined by Will Forney based on historical averages and review of
TRPA Code of Ordinances such as the Residential Allocation Performance Table.
#Other values were jointly arrived at by discussion Will Forney/Ben Oldham.

  params = {}
#housing allocations - all are annual numbers
  params["MIN_ALLOC_DG"] = 9          #was 10
  params["MAX_ALLOC_DG"] = 21         #was 14, then 17 for Ben's Evaluation
  params["MIN_ALLOC_EL"] = 27         #was 73
  params["MAX_ALLOC_EL"] = 111        #was 89, then 83 for Ben's Evaluation
  params["MIN_ALLOC_PL"] = 18         #was 38
  params["MAX_ALLOC_PL"] = 66         #was 46, then 50 for Ben's Evaluation
  params["MIN_ALLOC_WA"] = 13         #was 32
  params["MAX_ALLOC_WA"] = 49         #was 40, then 38 for Ben's Evaluation
  params["MIN_ALLOC_SLT"] = 11        #was 29
  params["MAX_ALLOC_SLT"] = 47        #still 35
  params["MIN_ALLOC_MFH"] = 30        #was 18
  params["MAX_ALLOC_MFH"] = 50        #was 18
  params["MIN_ALLOC_TOUR"] = 30       #was 5
```

```
  params["MAX_ALLOC_TOUR"] = 50      #was 10
#retirements - all are annual numbers
  params["MIN_RETIRE_CTC"] = 86      #was 5
  params["MAX_RETIRE_CTC"] = 250     #was 10
  params["MIN_RETIRE_NEV"] = 3       #was 0
  params["MAX_RETIRE_NEV"] = 59      #was 5
  params["MIN_RETIRE_TRPA"] = 29     #was 5
  params["MAX_RETIRE_TRPA"] = 46     #was 10
  params["MIN_RETIRE_USFS"] = 91     #was 5
  params["MAX_RETIRE_USFS"] = 189    #was 20
#retirement priorities
  params["PRIORITY_RETIRE_CTC"] = .5
  params["PRIORITY_RETIRE_NEV"] = .5
  params["PRIORITY_RETIRE_TRPA"] = .5
  params["PRIORITY_RETIRE_USFS"] = .5
#other params
  params["COMM_AREA_TOTAL_ALLOC"] = 160000      #Net commercial area sf allocation for the entire term of the model - was 40000
  params["ALLOC_ROLLOVER_POOL"] = 48            #was 300 for Ben's Evaluation
  params["MIN_ACRES"] = .1                      #Applies to SFH, MFH and tourist parcels
  params["ALLOW_SPECIAL_USE"] = False           #Will parcels in PAS's with LU "S" be included? (Default inclusion is LU "A")
  params["START_YEAR"] = 2009
  params["END_YEAR"] = 2028
  params["TOTAL_YEARS"] = params["END_YEAR"] - params["START_YEAR"] + 1
  return params

 def getPASs(self):
#Getting the PAS (TRPA Plan Area Statement) data from the db (the db contains data for all PASs).

  PASList= {}
  PASs = self.dbConn.query("SELECT * FROM pas_areas").dictresult()
  for pas in PASs:
   PASList[pas["pas_id"]] = PAS(self.params["ALLOW_SPECIAL_USE"], **pas)
  return PASList

 def getParcels(self, ipesThreshold):
#Getting the parcel data from the db (the db contains only vacant parcels - landuse code 1)

  parcelList = {}
  parcels = self.dbConn.query("SELECT * FROM parcels").dictresult()
  for parcel in parcels:
```

```
   parcelList[parcel["apn"]] = Parcel(self.PASList, ipesThreshold, **parcel)
  return parcelList

 def getParcelPossibleFates(self):
#Here, the parcels possible transitions and possible landuses are identified and stored.
#Go through all the parcels, and, for each landuse allowed by the parcel's PAS, call the appropriate function.
#The selected parcelPossibleLanduseXXXX function will examine the parcel to see if it meets the criteria for that landuse.
#If it meets the criteria, the called function adds the appropriate possible transition type to the parcels possibleTransitionsMasterList.
#Also, initialize the parcel landuse analysis statistic (reports #1 & 2).
#Gives each parcel a dictionary of valid possible landuses (landuseCounts) with each value set to 0 (dictionary does not typically include the entire set of
landuses).
#Ultimately, each value will represent the number of times that the parcel transitioned to that landuse.

  landuseFunctions = {"LU_1005": self.parcelPossibleLanduse1005, "LU_1011": self.parcelPossibleLanduse1011, "LU_2002": self.parcelPossibleLanduse2002,
"LU_3000": self.parcelPossibleLanduse3000, "LU_6401": self.parcelPossibleLanduse6401}
  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   PAS = self.PASList[parcel.pas_id]
   for landuse in PAS.allowedLanduseDdensities.keys():
    landuseFunctions.get(landuse)(parcel, PAS.allowedLanduseDdensities[landuse])
   parcel.landuseCounts["LU_0001"] = 0

 def parcelPossibleLanduse1005(self, parcel, PASdensity):
#MFH
#Parcel derivedIPES must exceed 0, it must exceed the minimum size, and the calculated units must exceed 1.
#The landuse is either 1005 (2-4 units) or 1006 (5-10 units) or 1007 (10+ units) depending on the number of units that the parcel supports.
#Calculate unitsMFH, a new Parcel attribute which is the developable MFH units based on the min of parcel size * MFH_UNITS_PER_ACRE, or the legal max
density from PASdensity

  if (parcel.derivedIPES > 0) and (parcel.acres >= self.params["MIN_ACRES"]):
   units = min(int(parcel.acres * self.params["MFD_UNITS_PER_ACRE"]), PASdensity)
   if units > 1:
    parcel.unitsMFH = units
    parcel.possibleTransitionsMasterList.append("ALLOC_MFH")
    landuse = parcel.mfhLanduse(units)
    parcel.landuseCounts[landuse] = 0

 def parcelPossibleLanduse1011(self, parcel, PASdensity):
#SFH
#Parcel derivedIPES must exceed 0, and it must exceed the minimum size.
```

```
#The transition type depends on the jurisdiction (5 jurisdictions, including SLT).
#Additionally, the transition type for jurisdiction PL has two types, depending on the derivedIPES (because of the PL vacant lot equation).

  if (parcel.derivedIPES > 0) and (parcel.acres >= self.params["MIN_ACRES"]):
   if parcel.jurisdiction == "PL":
    if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
     parcel.possibleTransitionsMasterList.append("ALLOC_PL")
    else:
     parcel.possibleTransitionsMasterList.append("ALLOC_PL_SENSITIVE")
   elif parcel.jurisdiction == "EL":
    if parcel.slt:
     parcel.possibleTransitionsMasterList.append("ALLOC_SLT")
    else:
     parcel.possibleTransitionsMasterList.append("ALLOC_EL")
   elif parcel.jurisdiction in ("DG", "WA"):
    parcel.possibleTransitionsMasterList.append("ALLOC_" + parcel.jurisdiction)
   parcel.landuseCounts["LU_1011"] = 0

 def parcelPossibleLanduse2002(self, parcel, PASdensity):
#Tourist
#Parcel derivedIPES must exceed 0, it must exceed the minimum size, and the calculated units must exceed 1.
#Calculate unitsTour, a new Parcel attribute which is the developable tour units based on the min of parcel size * TOUR_UNITS_PER_ACRE, or the legal max
density from PASdensity

  if (parcel.derivedIPES > 0) and (parcel.acres >= self.params["MIN_ACRES"]):
   units = min(int(parcel.acres * self.params["TOUR_UNITS_PER_ACRE"]), PASdensity)
   if units > 1:
    parcel.unitsTour = units
    parcel.possibleTransitionsMasterList.append("ALLOC_TOUR")
    parcel.landuseCounts["LU_2002"] = 0

 def parcelPossibleLanduse3000(self, parcel, PASdensity):
#Commercial
#Parcel derivedIPES must exceed 0.
#Calculate commFt, a new Parcel attribute which is the developable comm sf based on the parcel size * COMM_AREA_RATIO (no maximum)

  SF_PER_ACRE = 43560
  if (parcel.derivedIPES > 0):
   feet = int(parcel.acres * SF_PER_ACRE * self.params["COMM_AREA_RATIO"])
   if feet > 0:
```

```
    parcel.commFt = feet
    parcel.possibleTransitionsMasterList.append("ALLOC_COMM")
    parcel.landuseCounts["LU_3000"] = 0

 def parcelPossibleLanduse6401(self, parcel, PASdensity):
#Retirement
#All parcels are eligible for retirement.
#There are 4 retirement agencies and every parcel will be eligible for retirement by exactly 3 of these agencies.
#All parcels throughout the Tahoe basin qualify for USFS retirement.
#All parcels throughout the Tahoe basin qualify for TRPA retirement.
#Nevada parcels (DG and WA) qualify for NEV retirement.
#California parcels (EL/SLT and PL) qualify for CTC retirement.
#Additionally, each agency is deemed to have a priority retirement policy and a secondary retirement policy.
#All agencies are deemed to prioritize sensitive parcels (derivedIPES <= IPES_THRESHOLD).
#Therefore, each parcel qualifies for 3 of the 8 different retirement transition types.

  parcel.landuseCounts["LU_6401"] = 0
  if parcel.derivedIPES <= self.params["IPES_THRESHOLD"]:
   parcel.possibleTransitionsMasterList.append("RETIRE_USFS_PRIORITY")
   parcel.possibleTransitionsMasterList.append("RETIRE_TRPA_PRIORITY")
   if parcel.jurisdiction in ("EL", "PL"):
    parcel.possibleTransitionsMasterList.append("RETIRE_CTC_PRIORITY")
   if parcel.jurisdiction in ("DG", "WA"):
    parcel.possibleTransitionsMasterList.append("RETIRE_NEV_PRIORITY")
  else:
   parcel.possibleTransitionsMasterList.append("RETIRE_USFS_SECONDARY")
   parcel.possibleTransitionsMasterList.append("RETIRE_TRPA_SECONDARY")
   if parcel.jurisdiction in ("EL", "PL"):
    parcel.possibleTransitionsMasterList.append("RETIRE_CTC_SECONDARY")
   if parcel.jurisdiction in ("DG", "WA"):
    parcel.possibleTransitionsMasterList.append("RETIRE_NEV_SECONDARY")

 def popTransitionCandidates(self):
#Generate a master set of lists, one for each transition type, with each list containing the apns of the parcels that qualify for that transition type.

  transitionCandidates = {}
  for transType in self.transTypes:
   transitionCandidates[transType] = []
  for apn in self.parcelList:
   for transType in self.parcelList[apn].possibleTransitionsMasterList:
```

```
    transitionCandidates[transType].append(apn)
  return transitionCandidates

 def initParcelTransitionCounts(self):
#Initialize the parcel transition type analysis statistic (report #3).
#Gives each parcel a dictionary of transition types with each value set to 0.
#Ultimately, each value will represent the number of times that the parcel experienced that transition type.

  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   for transType in self.transTypes:
    parcel.transitionCounts[transType] = 0

 def initLanduseByYear(self):
#Initialize the simulation landuse by year analysis statistic (report #4).
#Creates a set of lists, one for each landuse.
#Gives landuseByYear[landuse] a list of length TOTAL_YEARS with each value set to 0.
#Ultimately, each list will contain a set of values that represent the average number of parcels that transitioned to that landuse in that year.
#landuseByYear[landuse][0] is the count for that landuse in year 1, while the last item in the list, landuseByYear[landuse][19] is the count for that landuse in
year 20.

  landuseByYear = {}
  for landuse in self.landuseNames:
   landuseByYear[landuse] = [0] * self.params["TOTAL_YEARS"]
  return landuseByYear

 def initTransitionsByYear(self):
#Initialize the simulation transitions by year analysis statistic (report #5).
#Creates a set of lists, one for each transition type.
#Gives transitionsByYear[transType] a list of length TOTAL_YEARS with each value set to 0.
#Ultimately, each list will contain a set of values that represent the average number of parcels that experienced that transition type in that year.
#transitionsByYear[transType][0] is the count for that transition in year 1, while the last item in the list, transitionsByYear[transType][19] is the count for that
transition in year 20.

  transitionsByYear = {}
  for transType in self.transTypes:
   transitionsByYear[transType] = [0] * self.params["TOTAL_YEARS"]
  return transitionsByYear

 def initTransCandidatesByYear(self):
```

```
#Initialize the simulation transitions candidates by year analysis statistic (report #6).
#Creates a set of lists, one for each transition type.
#Gives transCandidatesByYear[transType] a list of length TOTAL_YEARS + 1 with all values set to 0
#except for the year zero values which are initialized with the appropriate number of candidate parcels.
#These lists are different from the other "ByYear" lists in that there is a year zero.
#Ultimately, each list will contain a set of values that represent the average number of candidate parcels for that transition type at the end of that year.
#transCandidatesByYear[transType][0] is the count of available candidate parcels for that transition type at the end of year 0,
#transCandidatesByYear[transType][1] is the count of available candidate parcels for that transition type at the end of year 1,
#while the last item in the list, transCandidatesByYear[transType][20] is the count of available candidate parcels for that transition type at the end of year 20.

  transCandidatesByYear = {}
  for transType in self.transTypes:
   transCandidatesByYear[transType] = [0] * (self.params["TOTAL_YEARS"] + 1)
   transCandidatesByYear[transType][0] = len(self.transitionCandidatesMasterLists[transType])
  return transCandidatesByYear

 def initCategoryTotals(self):
#Initialize the simulation category totals analysis statistic (report #7).
#Creates a set of lists, one for each category (there are 35 categories).
#Each list contains seven values, each initially set to 0.
#The first list value is the count of how many parcels are associated with that category.
#The next six values in the list are associated with the six major landuses (1005/1006/1007 is consolidated into LU_MFH).
#Ultimately, each landuse value will represent the average chance that category member parcels were transitioned into that landuse (ranges from 0 to 1).

  categoryTotals = {}
  for catID in range(35):
   categoryTotals[catID] = {"Count": 0, "LU_0001": 0, "LU_MFH": 0, "LU_1011": 0, "LU_2002": 0, "LU_3000": 0, "LU_6401": 0}
  return categoryTotals

 def calcParcelCatLanduseBinaryTotal(self):
#For each parcel, calculate a value (landuseBinaryTotal) that will help determine the parcel's category.
#This value stores all the valid possible landuses in a single "binary number" which is then used by calcParcelCategory.

  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   for landuse in parcel.landuseCounts:
    if landuse == "LU_1011":                                    #SFH
     parcel.landuseBinaryTotal += 1
    if landuse in ("LU_1005", "LU_1006", "LU_1007"):            #MFH
     parcel.landuseBinaryTotal += 2
```

```
    if landuse == "LU_2002":                    #Tourist
     parcel.landuseBinaryTotal += 4
    if landuse == "LU_3000":                    #Commercial
     parcel.landuseBinaryTotal += 8

 def calcParcelCategory(self):
#A parcel category identifies a parcel in terms of its expected behavior as determined by its attributes.
#Parcels with the same category all share functionally similar attributes and can be expected to have similar outcome probabilities (which will generally display a bell curve distribution).
#Categories depend on a small set of variables (jurisdiction, landuses, IPES and size).
#Theoretically, there are some 320 categories (5 jurisdictions * 16 use combos * 2 IPES buckets * 2 size buckets (usually)). (Use combos = 0 uses (1 way) + 1 use (4 ways) + 2 uses (6 ways) + 3 uses (4 ways) + 4 uses (1 way))
#In practice, empirical examination of the model results indicated that there some 35 identifiable categories.
#Many of the categories have a relatively small number of members, with only ten of the categories having as many as 100 member parcels out of the 6,281 parcels modeled (smallest has 3 members).
#The most populated categories are:
# 1 - 1,435 parcels - various
# 4 - 1,229 parcels - jur = EL (not SLT), landuse = SFH, IPES > 725, size >= .1
# 6 - 958 parcels - jur = PL, landuse = SFH, IPES > 725, size >= .1
# 2 - 856 parcels - jur = SLT (EL), landuse = SFH, IPES > 725, size >= .1
# 5 - 319 parcels - jur = PL, landuse = SFH, IPES 1 to 725, size >= .1
# 12 - 261 parcels - jur = SLT (EL), landuse = SFH + MFH, IPES > 725, size >= .1
# 14 - 252 parcels - jur = EL (not SLT), landuse = SFH + MFH, IPES > 725, size >= .1
# 16 - 149 parcels - jur = PL, landuse = SFH + MFH, IPES > 725, size >= .1
# 3 - 117 parcels - jur = SLT (EL), landuse = SFH + MFH, IPES 1 to 725, size >= .1
# 8 - 100 parcels - jur = DG, landuse = SFH, IPES > 725, size >= .1
#Category #1 is a special category where, for a variety of reasons, no development ever occurs.
#Category #0 (23 member parcels) is also special - its parcels are so unusual that they avoid forming even a small category and so are left in the inhomogeneous category #0.

#The below mare's nest of if statements suffice only to identify parcels in categories 1 through 34.
#If a parcels should happen to belong to one of the other 286 (320-34=286) theoretical categories, the original category value of 0 is left unchanged.
#Finally, larger acreage parcels that have an allowed commercial landuse must be treated carefully because of the tendency for large commercial parcels to get "stuck in the pipeline".

  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   if parcel.landuseBinaryTotal == 0:           #no allowed uses
    parcel.category = 1
   elif parcel.landuseBinaryTotal == 1:         #SFH only
```

```
 if parcel.jurisdiction == "EL":
  if parcel.slt:
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 2
   else:
    parcel.category = 3
  else:
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 4
   else:
    parcel.category = 5
 elif parcel.jurisdiction == "PL":
  if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
   parcel.category = 6
  else:
   parcel.category = 7
 elif parcel.jurisdiction == "DG":
  if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
   parcel.category = 8
  else:
   parcel.category = 9
 elif parcel.jurisdiction == "WA":
  if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
   parcel.category = 10
  else:
   parcel.category = 11
elif parcel.landuseBinaryTotal == 2:          #MFH only
 if parcel.jurisdiction == "EL":
  if parcel.slt:
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 22
 elif parcel.jurisdiction == "WA":
  if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
   parcel.category = 23
elif parcel.landuseBinaryTotal == 3:          #SFH, MFH
 if parcel.jurisdiction == "EL":
  if parcel.slt:
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 12
   else:
```

```
    parcel.category = 13
   else:
    if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
     parcel.category = 14
    else:
     parcel.category = 15
  elif parcel.jurisdiction == "PL":
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 16
  elif parcel.jurisdiction == "WA":
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 17
   else:
    parcel.category = 18
 elif parcel.landuseBinaryTotal == 4:          #Tour only
  if parcel.jurisdiction == "PL":
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 26
 elif parcel.landuseBinaryTotal == 8:          #Comm only
  if parcel.jurisdiction == "EL":
   if parcel.slt:
    if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
     if parcel.acres < 4:
      parcel.category = 30
     elif parcel.acres < 12:
      parcel.category = 0
     else:
      parcel.category = 1
   else:
    if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
     if parcel.acres < 4:
      parcel.category = 31
     elif parcel.acres < 12:
      parcel.category = 0
     else:
      parcel.category = 1
  elif parcel.jurisdiction == "PL":
   if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
    parcel.category = 32
  elif parcel.jurisdiction == "DG":
```

```
    if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
     parcel.category = 33
    else:
     parcel.category = 34
   elif parcel.landuseBinaryTotal == 10:        #MFH, Comm
    if parcel.jurisdiction == "EL":
     if parcel.slt:
      if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
       parcel.category = 24
    elif parcel.jurisdiction == "WA":
     if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
      parcel.category = 25
   elif parcel.landuseBinaryTotal == 11:        #SFH, MFH, Comm
    if parcel.jurisdiction == "PL":
     if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
      parcel.category = 19
   elif parcel.landuseBinaryTotal == 12:        #Tour, Comm
    if parcel.jurisdiction == "EL":
     if parcel.slt:
      if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
       parcel.category = 27
    elif parcel.jurisdiction == "PL":
     if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
      parcel.category = 28
    elif parcel.jurisdiction == "WA":
     if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
      parcel.category = 29
   elif parcel.landuseBinaryTotal == 13:        #SFH, Tour, Comm
    if parcel.jurisdiction == "PL":
     if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
      parcel.category = 20
    elif parcel.jurisdiction == "WA":
     if parcel.derivedIPES > self.params["IPES_THRESHOLD"]:
      parcel.category = 21

 def calcOutputResults(self):
#Now that the simulation is finished, we need to do some massaging of the analysis statistics to make them ready for output.
#Loop through all the parcels to:
# Create a LU_0001 total by subtracting the totals of other landuse totals from the number of iterations (needed because LU_0001 is not a transition - the parcel
is simply remaining in its original state).
```

```
# Populate the categoryTotals "Count" field with the count of member parcels.
# With all parcel landuseCounts (used for reports #1 & 2):
# For each landuse:
# Convert landuseCounts values from raw simulation totals to percentages by dividing by the number of iterations.
#  Populate categoryTotals (report #7) with landuses by landuse totals (LU_1005/6/7 are consolidated into LU_MFH).
# With all parcel transitionCounts (used for report #3):
# Convert the transitionCounts from raw simulation totals to average values by dividing by the number of iterations.
# With all simulation landuseByYear (report #4):
# For each year of each landuse:
# Convert the landuseByYear value from a raw simulation total to an average value by dividing by the number of iterations.
# With all simulation transitionsByYear (report #5):
# For each year of each transition type:
#  Convert the transitionsByYear value from a raw simulation total to an average value by dividing by the number of iterations.
#With all simulation transCandidatesByYear (report #6):
# For each year of each transition type (except year 0 which is unique to this report and needs no adjustment):
# Convert the transCandidatesByYear value from a raw simulation total to an average value by dividing by the number of iterations.
#With all simulation categoryTotals (report #7):
# For each landuse of each category:
# Convert the category landuse count from a raw simulation total to a percentage by dividing by the category count.

 for apn in self.parcelList:
  parcel = self.parcelList[apn]
  parcel.landuseCounts["LU_0001"] = self.params["ITERATIONS"] - sum(landuseCt for landuseCt in parcel.landuseCounts.values())
  self.categoryTotals[parcel.category]["Count"] += 1
  for landuse in parcel.landuseCounts:
   parcel.landuseCounts[landuse] /= float(self.params["ITERATIONS"])
   if landuse in ("LU_1005", "LU_1006", "LU_1007"):
    self.categoryTotals[parcel.category]["LU_MFH"] += parcel.landuseCounts[landuse]
   else:
    self.categoryTotals[parcel.category][landuse] += parcel.landuseCounts[landuse]
  for transType in parcel.transitionCounts:
   parcel.transitionCounts[transType] /= float(self.params["ITERATIONS"])
 for landuse in self.landuseByYear:
  for year in range(self.params["TOTAL_YEARS"]):
   self.landuseByYear[landuse][year] /= float(self.params["ITERATIONS"])
 for transType in self.transitionsByYear:
  for year in range(self.params["TOTAL_YEARS"]):
   self.transitionsByYear[transType][year] /= float(self.params["ITERATIONS"])
 for transType in self.transCandidatesByYear:
  for year in range(1, (self.params["TOTAL_YEARS"]) + 1):
```

```
    self.transCandidatesByYear[transType][year] /= float(self.params["ITERATIONS"])
  for catID in self.categoryTotals:
   category = self.categoryTotals[catID]
   for landuse in category.keys():
    if landuse != "Count":
     category[landuse] /= float(category["Count"])

 def printLanduse(self, modelID):
#Analysis report #1.
#Generates output that has one line with the probability percentage for every landuse of every parcel (where the probability is greater than 0).
#If the landuse is MFH or tourist, include the appropriate number of units.
#If the landuse is comm, include the appropriate number of developed sf.
#This report isn't particularly useful, as it's pretty difficult to import the data into ArcMap in any useful way, however, it was the original output format agreed to
with the client.
#The total of the UsePct values should equal the number of parcels in the simulation.

  self.logFile = open("TahoeLanduse" + modelID + ".txt", "w")
  self.logFile.write("APN,Landuse,UsePct,Units,CommSF\n")
  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   for landuse in parcel.landuseCounts:
    useCount = parcel.landuseCounts[landuse]
    if useCount > 0:
     commSF = 0
     units = 0
     if landuse in ("LU_1005", "LU_1006", "LU_1007"):
      units = parcel.unitsMFH
     elif landuse == "LU_2002":
      units = parcel.unitsTour
     elif landuse == "LU_1011":
      units = 1
     elif landuse == "LU_3000":
      commSF = parcel.commFt
     parcelData = (apn, landuse, str(useCount), str(units), str(commSF))
     self.logFile.write(",".join(parcelData) + "\n")
  self.logFile.close()

 def printCondensedLanduse(self, modelID):
#Analysis report #2.
#Generates one line of data for every parcel.
```

```
#Data included are the probability percentages for all landuses, all units/sf, and the category code.
#This report is similar to the LandUse report (#1), but it is in a format that makes it much easier to import into ArcMap.
#The sum total of all landuse totals should equal the number of parcels in the simulation.

  self.logFile = open("TahoeCondensedLanduse" + modelID + ".txt", "w")
  self.logFile.write("APN,Category,LU_0001,LU_1005,LU_1006,LU_1007,LU_1011,LU_2002,LU_3000,LU_6401,MFHUnits,TourUnits,CommSF\n")
  landuses = ("LU_0001", "LU_1005", "LU_1006", "LU_1007", "LU_1011", "LU_2002", "LU_3000", "LU_6401")
  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   dataRow = [apn]
   dataRow.append(str(parcel.category))
   for landuse in landuses:
    if landuse in parcel.landuseCounts:
     dataRow.append(str(parcel.landuseCounts[landuse]))
    else:
     dataRow.append("0")
   if set(["LU_1005", "LU_1006", "LU_1007"]) & set(parcel.landuseCounts):
    dataRow.append(str(parcel.unitsMFH))
   else:
    dataRow.append("0")
   if "LU_2002" in parcel.landuseCounts:
    dataRow.append(str(parcel.unitsTour))
   else:
    dataRow.append("0")
   if "LU_3000" in parcel.landuseCounts:
    dataRow.append(str(parcel.commFt))
   else:
    dataRow.append("0")
   self.logFile.write(",".join(dataRow) + "\n")
  self.logFile.close()

 def printCondensedTransitions(self, modelID):
#Analysis report #3.
#Generates one line of data for every parcel.
#Data included are the probability percentages for every transition type.
#This report is similar to the CondensedLanduse report (#2) except that it contains transition data instead of landuse data.
#The sum total of all transition type totals should equal the number of parcels in the simulation.

  self.logFile = open("TahoeCondensedTransitions" + modelID + ".txt", "w")
  fileHeader = ["APN"]
```

```
  fileHeader.extend(self.transTypes)
  self.logFile.write(",".join(fileHeader) + "\n")
  for apn in self.parcelList:
   parcel = self.parcelList[apn]
   dataRow = [apn]
   for transType in self.transTypes:
    dataRow.append(str(parcel.transitionCounts[transType]))
   self.logFile.write(",".join(dataRow) + "\n")
  self.logFile.close()

 def printLanduseByYear(self, modelID):
#Analysis report #4.
#Generates one line of data for every landuse.
#Data included are the average number of parcels transitioning to a particular landuse, broken down by year.
#This report contains summarized data that is not suitable for import into ArcMap.
#The sum total of all year totals should equal the number of parcels in the simulation, as should the sum total of all the landuses.
#The landuse totals should match the landuse totals from report #2.

  self.logFile = open("TahoeLanduseByYear" + modelID + ".txt", "w")
  fileHeader = ["Desc"]
  for year in range(self.params["TOTAL_YEARS"]):
   fileHeader.append("Y" + str(year + 1))
  self.logFile.write(",".join(fileHeader) + "\n")
  for landuseName in self.landuseNames:
   dataRow = [landuseName]
   for year in range(self.params["TOTAL_YEARS"]):
    dataRow.append(str(self.landuseByYear[landuseName][year]))
   self.logFile.write(",".join(dataRow) + "\n")
  self.logFile.close()

 def printTransitionsByYear(self, modelID):
#Analysis report #5.
#Generates one line of data for every transition type.
#Data included are the average number of parcels experiencing a particular transition type, broken down by year.
#This report contains summarized data that is not suitable for import into ArcMap.
#The sum total of all year totals should equal the number of parcels in the simulation, as should the sum total of all the transition types.
#The transition type totals should match the transition type totals from report #3.

  self.logFile = open("TahoeTransitionsByYear" + modelID + ".txt", "w")
  fileHeader = ["Desc"]
```

```
 for year in range(self.params["TOTAL_YEARS"]):
  fileHeader.append("Y" + str(year + 1))
 self.logFile.write(",".join(fileHeader) + "\n")
 for transType in self.transTypes:
  dataRow = [transType]
  for year in range(self.params["TOTAL_YEARS"]):
   dataRow.append(str(self.transitionsByYear[transType][year]))
  self.logFile.write(",".join(dataRow) + "\n")
 self.logFile.close()

 def printTransCandidatesByYear(self, modelID):
#Analysis report #6.
#Generates one line of data for every transition type.
#Data included are the number of candidate parcels for that transition type broken down by year.
#Data values represent the number of candidates at the end of that year.
#Accordingly, this report, unlike the by-year reports # 4 and #5, contains a year zero which represents the initial state of the simulation.
#This report contains summarized data that is not suitable for import into ArcMap.
#Neither the yearly totals or the transition type totals have any real meaning.

 self.logFile = open("TahoeTransCandidatesByYear" + modelID + ".txt", "w")
 fileHeader = ["Desc"]
 for year in range((self.params["TOTAL_YEARS"]) + 1):
  fileHeader.append("Y" + str(year))
 self.logFile.write(",".join(fileHeader) + "\n")
 for transType in self.transTypes:
  dataRow = [transType]
  for year in range((self.params["TOTAL_YEARS"]) + 1):
   dataRow.append(str(self.transCandidatesByYear[transType][year]))
  self.logFile.write(",".join(dataRow) + "\n")
 self.logFile.close()

 def printCategoryTotals(self, modelID):
#Analysis report #7.
#Generates one line of data for every category.
#Data included are the probability percentages for all landuses as well as the count of the member parcels for that category.
#This report contains summarized data that is not suitable for import into ArcMap (but symbolizing parcels by category within ArcMap provides excellent
feedback)..
#The sum of the counts should equal the number of parcels in the simulation.

 self.logFile = open("TahoeCategoryTotals" + modelID + ".txt", "w")
```

```
 fileHeader = ["Category"]
 catFields = ["Count", "LU_0001", "LU_MFH", "LU_1011", "LU_2002", "LU_3000", "LU_6401"]
 fileHeader.extend(catFields)
 self.logFile.write(",".join(fileHeader) + "\n")
 for catID in range(35):
  category = self.categoryTotals[catID]
  dataRow = [str(catID)]
  for field in catFields:
   dataRow.append(str(category[field]))
  self.logFile.write(",".join(dataRow) + "\n")
 self.logFile.close()


class PAS(object):

 def __init__(self, allowSpecialUse, **kwargs):
#The pas_areas table in the db has been reviewed and validated so that validation does not have to occur in the model code.
#Values in the pas_id field (primary key) are limited to valid PAS IDs.
#Values in the landuse fields (lu_1005_allowed, lu_1011_allowed, lu_2002_allowed, lu_3000_allowed) are limited to "A", "S", and "N".
#Whenever a lu_1005_allowed or lu_2002_allowed field has a "A" or a "S", the corresponding density field (lu_1005_density, lu_2002_density) must be an
integer greater than 0.

#A PAS (TRPA Plan Area Statement) is a geographic unit encompassing one or more parcels.
#Each PAS has a set of applicable permissible landuses and densities that are stored in the pas_areas table.
#These uses are defined for landuse codes 1005 (MFH), 1011 (SFH), 2002 (Tourist), 3000 (Commercial),
#with permissions of either "A" (allowed), "S" (special use) or "N" (not allowed).
#Landuse densities exist only for MFH and Tourist landuses (a density of 1 is assigned to other landuses
#for purposes of the model).
#Landuse code 1005 is really a catchall for 1005 (2-4 units), 1006 (5-10 units), 1007 (10+ units)
#as the number of units that can built on a parcel depends on its size.
#Landuse code 2002 includes hotels only - not the other three tourist types (2001/2003/2004).
#Landuse 3000 includes all commercial landuses, with the "most permissive" use prevailing (if there is a
#mix of "A", "S", and "N" in the commercial use section of the PAS document, then the PAS is set to "A").
#Every PAS is assumed to have an allowed 6401 (retired) landuse, even though this is not explicitly in the db table.

#Each PAS object has a allowedLanduseDensities dictionary. The keys are the landuse codes and the values are the
#allowed density for that landuse. A landuse code is only present in the dictionary if that landuse is permitted.

 self.pas_id = kwargs["pas_id"]
 self.allowedLanduseDensities = {}
```

```
 if self.useAllowed(allowSpecialUse, kwargs["lu_1005_allowed"].upper()):
  self.allowedLanduseDensities["LU_1005"] = kwargs["lu_1005_density"]
 if self.useAllowed(allowSpecialUse, kwargs["lu_1011_allowed"].upper()):
  self.allowedLanduseDensities["LU_1011"] = 1
 if self.useAllowed(allowSpecialUse, kwargs["lu_2002_allowed"].upper()):
  self.allowedLanduseDensities["LU_2002"] = kwargs["lu_2002_density"]
 if self.useAllowed(allowSpecialUse, kwargs["lu_3000_allowed"].upper()):
  self.allowedLanduseDensities["LU_3000"] = 1
 self.allowedLanduseDensities["LU_6401"] = 1

 def useAllowed(self, allowSpecialUse, useval):
#Decide if a particular landuse is allowed or not based on the PAS permissions and the
#user specified treatment of the "S" (allow special) permission.
  allowed = False
  if (useval == "A") or ((useval == "S") and allowSpecialUse):
   allowed = True
  return allowed


class Parcel(object):
#The parcels table in the db has been reviewed and validated so that validation does not have to occur in the model code.
#Values in the apn field (primary key) are limited to valid APNs.
#Values in the pas_id field (foreign key) are limited to valid PAS IDs.
#Values in the jurisdiction field are limited to DG, EL, PL, WA.
#Values in the acres field must be greater than 0.
#Values in the ipes_score field must be 0 or greater.
#Values in the land_cap field are limited to 1a, 1b, 1c, 2, 3, 4, 5, 6, 7.
#The slt field is a boolean field.

 @staticmethod
 def mfhLanduse(units):
#Identifies the appropriate MFH landuse code based on the number of units that the parcel supports.

  if units <= 4:
   landuse = "LU_1005"
  elif units <= 10:
   landuse = "LU_1006"
  else:
   landuse = "LU_1007"
  return landuse
```

```
 def __init__(self, PASList, ipesThreshold, **kwargs):
  self.apn = kwargs["apn"]                              #Assessors Parcel Number
#attributes that help determine the various possible fates of a parcel
  self.pas_id = kwargs["pas_id"]                        #Foreign key to PAS data
  self.jurisdiction = kwargs["jurisdiction"]            #DG, EL, PL, WA (but not SLT, which is a subset of EL)
  self.acres = kwargs["acres"]                          #Parcel size
  self.ipes_score = kwargs["ipes_score"]                #Score indicating the "developability" of a parcel - higher scores are better suited for development.
# Many of these values from the client are "0" - which was used to denote an IPES score of zero or to indicate that the IPES score was unknown (garbage data).
# Hence the need to attempt to develop an IPES alternative
# (Every one of the approximately 1800 PL parcels has an IPES score of 0, which would otherwise make the calculation of the PL vacant lot equation
problematic).
# This score is not directly used in the model (derivedIPES is).
  self.land_cap = kwargs["land_cap"]
#IPES alternative generated from a GIS join of Bailey valued areas with parcel areas (1a, 1b, 1c, 2, 3, 4, 5, 6, 7).
# This score is not directly used in the model (derivedIPES is).
  self.slt = self.getBool(kwargs["slt"])
#True if the parcel is a South Lake Tahoe parcel (subset of EL) (generated from a GIS join of the SLT area with parcel areas).
  self.derivedIPES = self.getDerivedIPES(self.ipes_score, self.land_cap, ipesThreshold)
#Final IPES score that the model uses to help determine possible fates for parcel.
#initializing some simulation logic attributes - they will be populated/calculated later on
  self.possibleTransitionsMasterList = []               #The possible transitions for a parcel based on its attributes.
# These are calculated once at the beginning of the simulation and never changes.
# If a transition is in a parcel.possibleTransitionsMasterList, then that parcel's apn will be found in the appropriate model.transitionCandidatesMasterLists
transition list.
# There is always a one-to-one correspondence between the transitions in parcel.possibleTransitionsMasterList and the apns in the appropriate
model.transitionCandidatesMasterLists list.
  self.thisRunPossibleTransitions = []                  #The current possible transitions for a parcel (in the current run).
# Copied from the possibleTransitionsMasterList at the beginning of each run.
# Can sometimes change in the middle of a run as events dictate.
# Prior to a parcel being transitioned, if a transition is in a parcel.thisRunPossibleTransitions, then that parcel's apn will be found in the appropriate
run.thisRunTransitionCandidates transition list.
# Prior to a parcel being transitioned, there is always a one-to-one correspondence between the transitions in parcel.thisRunPossibleTransitions and the apns in
the appropriate run.thisRunTransitionCandidates list.
#initializing some analysis statistics attributes - they will be populated/calculated later on
  self.landuseCounts = {}                               #Analysis statistic that counts the number of landuses by landuse throughout the simulation (reports #1 &
2).
# The keys are the landuse (LU_0001, LU_1005, LU_1006, LU_1007, LU_1011, LU_2002, LU_3000, LU_6401).
# Landuses LU_0001, LU_1006, LU_1007 are newly used here as they did not exist in the PAS.allowedLanduseDensities.
  self.transitionCounts = {}                            #Analysis statistic that counts the number of transitions by transition type (report #3).
```

```
  self.category = 0                                    #A parcel category identifies a parcel in terms of its expected behavior as determined by its attributes.
# Parcels with the same category all share functionally similar attributes and can be expected to have similar outcome probabilities (which will generally display
a bell curve distribution).
  self.landuseBinaryTotal = 0                          #Value that, once calculated (calcParcelCatLanduseBinaryTotal), helps determine the parcel's category

 def getBool(self, boolval):
#Needed because of a pg/python oddity.

  if boolval == 't':
   return True
  else:
   return False

 def getDerivedIPES(self, ipes, land_cap, ipesThreshold):
#If the ipes_score is 0 (garbage data), use the land_cap value instead and translate the land_cap values into generalized IPES scores.
#Otherwise, the IPES score is greater than 0, which means that we have a good value that we can use.
#Translation values determined by Will Forney.
#The derivedIPES value is then used by the model to help determine possible fates for parcel.

  derivedIPES = ipes
  if ipes == 0:
   if land_cap in ("1a", "2", "3"):                    #less suitable for development (sensitive)
    derivedIPES = ipesThreshold
   elif land_cap in ("4", "5", "6", "7"):              #well suited for development (not sensitive)
    derivedIPES = ipesThreshold + 1
   else:#1b, 1c                                        #development prohibited
    derivedIPES = 0
  return derivedIPES


class Run(object):

 def __init__(self, model):
  self.model = model
  self.thisRunTransitionCandidates = {}
  self.thisYearTrans = []

##  self.model = model
#The Run sometimes needs access to data contained within the Model object - making the Model an attribute of the Run object gives easy access to this data.
##  self.thisRunTransitionCandidates = {}
```

```
#A mutable set of lists, one for each transition type, with each list containing the apns of the parcels that qualify for that transition type.
# As the years of the run go by, the length of these lists is shortened as transitions occur.
# Prior to a parcel being transitioned, if an apn is in a particular run.thisRunTransitionCandidates list, than the thisRunPossibleTransitions for that parcel will always include that transition type.
# Prior to a parcel being transitioned, there is always a one-to-one correspondence between the apns in a particular run.thisRunTransitionCandidates list and the transitions in the member parcel.thisRunPossibleTransitions.
##  self.thisYearTrans = []                     #A list of transitions to be executed in the current year of the current run.

 def initRun(self):
  self.thisYear = 1
#The first year of a run is always year 1 (year 0 references refer to conditions prior to the start of the model).
  self.commAllocated = 0
#This is a cumulative value representing the number of commercial sq. ft. transitioned throughout the course of the run.
  self.mfhRolloverPool = 0
#Each year, the run tries to use as many MFH allocation units as it can. The value typically hovers a little bit above zero throughout the course of the run and never drops below zero.
  self.tourRolloverPool = 0
#Each year, the run tries to use as many Tourist allocation units as it can. The value typically hovers a little bit above zero throughout the course of the run and never drops below zero.
  self.vacantLotEqSatisfiedPL = False
#See http://www.trpa.org/documents/agendas/Archive/apc_agendas/2002/apc/2002-02-13.PDF, pages 41/42.
# Placer county is the only county that still needs to meet this requirement.
  self.refreshTransitionCandidates()
#Generate a randomized list of transitions scheduled for the year.

 def refreshTransitionCandidates(self):
#first, populate the run's thisRunTransitionCandidates lists from the model's transitionCandidates
  for transType in self.model.transitionCandidatesMasterLists.keys():
   self.thisRunTransitionCandidates[transType] = deepcopy(self.model.transitionCandidatesMasterLists[transType])
   shuffle(self.thisRunTransitionCandidates[transType])
#second, populate the Parcel currentPossibleTransitions from the Parcel initialPossibleTransitions
  for apn in self.model.parcelList:
   parcel = self.model.parcelList[apn]
   parcel.thisRunPossibleTransitions = deepcopy(parcel.possibleTransitionsMasterList)

## def refreshTransitionCandidates(self):
#Make a randomized set of run transition candidate parcels (thisRunTransitionCandidates) copied from the transitionCandidatesMasterLists.
#Also, for each parcel, copy the thisRunPossibleTransitions list from the possibleTransitionsMasterList.

###Refreshing the run thisRunTransitionCandidates list:
```

```
##  for transType in self.model.transitionCandidatesMasterLists.keys():
##   self.thisRuntransitionCandidates[transType] = deepcopy(self.model.transitionCandidatesMasterLists[transType])
##   shuffle(self.thisRunTransitionCandidates[transType])
###Refreshing all the parcels thisRunPossibleTransitions list:
##  for apn in self.model.parcelList:
##   parcel = self.model.parcelList[apn]
##   parcel.thisRunPossibleTransitions = deepcopy(parcel.initialPossibleTransitions)

 def OLDinitRunYear(self):
#No longer user - replaced by a gaussian version.

  self.thisYearTrans = []                           #Reset the list
  self.thisYearTrans.extend(["ALLOC_DG"] * randint(self.model.params["MIN_ALLOC_DG"], self.model.params["MAX_ALLOC_DG"]))
  self.thisYearTrans.extend(["ALLOC_EL"] * randint(self.model.params["MIN_ALLOC_EL"], self.model.params["MAX_ALLOC_EL"]))
  self.thisYearTrans.extend(["ALLOC_PL"] * randint(self.model.params["MIN_ALLOC_PL"], self.model.params["MAX_ALLOC_PL"]))
  self.thisYearTrans.extend(["ALLOC_WA"] * randint(self.model.params["MIN_ALLOC_WA"], self.model.params["MAX_ALLOC_WA"]))
  self.thisYearTrans.extend(["ALLOC_SLT"] * randint(self.model.params["MIN_ALLOC_SLT"], self.model.params["MAX_ALLOC_SLT"]))
  self.thisYearTrans.extend(["RETIRE_CTC"] * randint(self.model.params["MIN_RETIRE_CTC"], self.model.params["MAX_RETIRE_CTC"]))
  self.thisYearTrans.extend(["RETIRE_NEV"] * randint(self.model.params["MIN_RETIRE_NEV"], self.model.params["MAX_RETIRE_NEV"]))
  self.thisYearTrans.extend(["RETIRE_TRPA"] * randint(self.model.params["MIN_RETIRE_TRPA"], self.model.params["MAX_RETIRE_TRPA"]))
  self.thisYearTrans.extend(["RETIRE_USFS"] * randint(self.model.params["MIN_RETIRE_USFS"], self.model.params["MAX_RETIRE_USFS"]))
  self.thisYearTrans.extend(["ALLOC_MFH"])
  self.thisYearTrans.extend(["ALLOC_TOUR"])
  self.thisYearTrans.extend(["ALLOC_COMM"])
  shuffle(self.thisYearTrans)

  self.mfhRolloverPool += randint(self.model.params["MIN_ALLOC_MFH"], self.model.params["MAX_ALLOC_MFH"])
  self.tourRolloverPool += randint(self.model.params["MIN_ALLOC_TOUR"], self.model.params["MAX_ALLOC_TOUR"])
  self.thisYearCommTarget = int(((self.model.params["COMM_AREA_TOTAL_ALLOC"] / self.model.params["TOTAL_YEARS"]) * self.thisYear) -
self.commAllocated)

 def initRunYear(self):
#Determine how many transitions of each type will be scheduled based on the input parameters (values are stochastic).
#Stochastic values are calculated using a Gaussian (normal curve) distribution. Earlier versions of the model used a straight-line distribution, but no great
difference between the methods was noted.
#For all SFH and Retirement transitions, add that number of items to the list (thisYearTrans).
#MFH, Tourist and Commercial each have only a single item added to the list as they are handled with a different mechanism (pools).
#The list is then randomized and is ready for use.

#Some notes on transitions and model logic:
```

```
#Transitions generally have min and max annual values based on user inputs (Comm only has a single value for the entire simulation).
#These annual values do not change throughout the course of the simulation.
#SFH allocations are assumed to vary due to various deduction and incentive increments.
#Input parameter values should evenly bracket expected allocations rather than reflecting the theoretical min/max values that the TRPA code provides for
#(min annual allocation total of 78, max annual allocation total of 294, base allocation total of 150).
#All allocations are expected to be used in the year of issue (with the exception of MFH/tourist/Comm allocations which work with an allocation pool
mechanism.

  self.thisYearTrans = []                          #Reset the list
  self.thisYearTrans.extend(["ALLOC_DG"] * self.randomGauss(self.model.params["MIN_ALLOC_DG"], self.model.params["MAX_ALLOC_DG"]))
  self.thisYearTrans.extend(["ALLOC_EL"] * self.randomGauss(self.model.params["MIN_ALLOC_EL"], self.model.params["MAX_ALLOC_EL"]))
  self.thisYearTrans.extend(["ALLOC_PL"] * self.randomGauss(self.model.params["MIN_ALLOC_PL"], self.model.params["MAX_ALLOC_PL"]))
  self.thisYearTrans.extend(["ALLOC_WA"] * self.randomGauss(self.model.params["MIN_ALLOC_WA"], self.model.params["MAX_ALLOC_WA"]))
  self.thisYearTrans.extend(["ALLOC_SLT"] * self.randomGauss(self.model.params["MIN_ALLOC_SLT"], self.model.params["MAX_ALLOC_SLT"]))
  self.thisYearTrans.extend(["RETIRE_CTC"] * self.randomGauss(self.model.params["MIN_RETIRE_CTC"], self.model.params["MAX_RETIRE_CTC"]))
  self.thisYearTrans.extend(["RETIRE_NEV"] * self.randomGauss(self.model.params["MIN_RETIRE_NEV"], self.model.params["MAX_RETIRE_NEV"]))
  self.thisYearTrans.extend(["RETIRE_TRPA"] * self.randomGauss(self.model.params["MIN_RETIRE_TRPA"],
self.model.params["MAX_RETIRE_TRPA"]))
  self.thisYearTrans.extend(["RETIRE_USFS"] * self.randomGauss(self.model.params["MIN_RETIRE_USFS"], self.model.params["MAX_RETIRE_USFS"]))
  self.thisYearTrans.extend(["ALLOC_MFH"])
  self.thisYearTrans.extend(["ALLOC_TOUR"])
  self.thisYearTrans.extend(["ALLOC_COMM"])
  shuffle(self.thisYearTrans)

  self.mfhRolloverPool += self.randomGauss(self.model.params["MIN_ALLOC_MFH"], self.model.params["MAX_ALLOC_MFH"])
  self.tourRolloverPool += self.randomGauss(self.model.params["MIN_ALLOC_TOUR"], self.model.params["MAX_ALLOC_TOUR"])
  self.thisYearCommTarget = int(((self.model.params["COMM_AREA_TOTAL_ALLOC"] / self.model.params["TOTAL_YEARS"]) * self.thisYear) -
self.commAllocated)

 def randomGauss(self, minVal, maxVal):
#Calculated a gaussian (normal curve) integer value based on a min and max, which are enforced.
#The sigma (standard deviation) is deemed to be one third of the difference between the mean and the min.

  sigma = (maxVal - minVal) / 6.0
  returnVal = gauss((maxVal + minVal) / 2.0, sigma)
  if returnVal > maxVal:
   returnVal = maxVal
  if returnVal < minVal:
   returnVal = minVal
  return int(round(returnVal))
```

```
 def filterLargeCommLots(self):
#Large lots can clog up the commercial "development pipeline", causing the total developed commercial s.f. to be much less than the allocation parameter.
#Accordingly, if, during the current run year, a lot is too large to be developed, remove it from the ALLOC_COMM thisRunTransitionCandidates list
#(and the parcel thisRunPossibleTransitions list as well).

  maxSize = self.model.params["COMM_AREA_TOTAL_ALLOC"] - self.commAllocated
  for apn in self.thisRunTransitionCandidates["ALLOC_COMM"]:
   parcel = self.model.parcelList[apn]
   if parcel.commFt > maxSize:
    parcel.thisRunPossibleTransitions.remove("ALLOC_COMM")
    self.thisRunTransitionCandidates["ALLOC_COMM"].remove(apn)

 def checkVacantLotEqPL(self):
#See http://www.trpa.org/documents/agendas/Archive/apc_agendas/2002/apc/2002-02-13.PDF, (pdf pages 48-50).
#Placer county is the only county that still needs to meet this requirement.
#Once the PL vacant lot equation is satisfied, add all the ALLOC_PL_SENSITIVE parcels to the ALLOC_PL list.

  if self.vacantLotEqSatisfiedPL == False:
   if ((len(self.thisRunTransitionCandidates["ALLOC_PL_SENSITIVE"]) / 1667.0) < 0.2):
    self.vacantLotEqSatisfiedPL = True
#now all non-zero IPES PL parcels may be developed
    if (len(self.thisRunTransitionCandidates["ALLOC_PL_SENSITIVE"]) > 0):
     self.thisRunTransitionCandidates["ALLOC_PL"].extend(self.thisRunTransitionCandidates["ALLOC_PL_SENSITIVE"])
     shuffle(self.thisRunTransitionCandidates["ALLOC_PL"])

     for apn in self.thisRunTransitionCandidates["ALLOC_PL_SENSITIVE"]:
      parcel = self.model.parcelList[apn]
      parcel.thisRunPossibleTransitions.append("ALLOC_PL")
      parcel.thisRunPossibleTransitions.remove("ALLOC_PL_SENSITIVE")
     self.thisRunTransitionCandidates["ALLOC_PL_SENSITIVE"] = []

 def annualTransitions(self):
#Go through the transitions scheduled for this year and attempt to fulfill them by calling the function appropriate to each transition type.
#Each of these five called functions determines the appropriate transition list, landuse, and transition type and then calls the transitionParcel function which does
the actual transitioning of a parcel.
#The SFH and retirement functions will be called repeatedly whereas the MFH, Tour and Comm functions are only called once each year.
#When all scheduled transitions have been processed, update the transCandidatesByYear analysis statistic (report #6).

  for transType in self.thisYearTrans:
```

```
   if transType in ("ALLOC_DG", "ALLOC_EL", "ALLOC_PL", "ALLOC_WA", "ALLOC_SLT"):
    self.sfhTransition(transType)
   elif transType in ("RETIRE_CTC", "RETIRE_NEV", "RETIRE_TRPA", "RETIRE_USFS"):
    self.retireTransition(transType)
   elif transType == "ALLOC_MFH":
    self.mfhTransition(transType)
   elif transType == "ALLOC_TOUR":
    self.tourTransition(transType)
   elif transType == "ALLOC_COMM":
    self.commTransition(transType)
  for transType in self.model.transTypes:
   self.model.transCandidatesByYear[transType][(self.thisYear)] += len(self.thisRunTransitionCandidates[transType])

 def sfhTransition(self, transType):
#SFH transitions are straightforward - the transition list, landuse, and transition type are always well defined.

  self.transitionParcel(self.thisRunTransitionCandidates[transType], "LU_1011", transType)

 def retireTransition(self, transType):
#Retirement transitions are complicated by the fact that we have both priority and secondary retirements.
#Accordingly, we need to determine which list and transaction type to use, priority or secondary.
#Assuming that there are parcels in both priority and secondary lists, we randomly choose which list to use based on the user priority parameter for that
retirement type.
#If only one of the lists is populated, we will default to using that list.

  priorityList = self.thisRunTransitionCandidates[transType + "_PRIORITY"]
  secondaryList = self.thisRunTransitionCandidates[transType + "_SECONDARY"]
  if (len(priorityList) > 0) and (len(secondaryList) > 0):
   if random() < self.model.params["PRIORITY_" + transType]:
    transType += "_PRIORITY"
   else:
    transType += "_SECONDARY"
  elif (len(priorityList) > 0):
   transType += "_PRIORITY"
  else:
   transType += "_SECONDARY"
#If the secondary list has zero length, transitionParcel will do nothing.
  self.transitionParcel(self.thisRunTransitionCandidates[transType], "LU_6401", transType)

 def mfhTransition(self, transType):
```

```
#MFH transition are complicated by two things:
#1) MFH transitions use a pool approach rather than individually scheduled transitions.
#2) We don't know the landuse (which depends on the number of units a parcel supports).
#Pool approach transition types attempt to transition as many parcels as possible, one after the next, without exceeding the current pool allocation.
# Any unused pool capacity rolls forward to the next year.
# For example, say that at the beginning of the current year the mfhRolloverPool stood at 40 and the units of the first 3 parcels in the MFH list were 16, 4, 25.
#  The first and second parcels would be transitioned in the current year, but the third parcel would not be because the available pool units would be exceeded.
#  At the beginning of the next year, the parcel with 25 units would be first in the list and the starting pool allocation would be 20 (to which would be added the
new allocation for the year).
#To obtain the landuse, we look at units of the first parcel in the MFH list and calculate the landuse from that.
#Because this is a pool approach transition type, there is only one MFH transition scheduled per year.
#The landuseByYear analysis statistic for MFHUnits is also updated here.

  done = False
  transList = self.thisRunTransitionCandidates[transType]
  while ((not done) and (len(transList) > 0)):
   apn = transList[0]
   candidateParcel = self.model.parcelList[apn]
   parcelUnits = candidateParcel.unitsMFH
   if parcelUnits <= self.mfhRolloverPool:
    landuse = candidateParcel.mfhLanduse(parcelUnits)
    self.transitionParcel(transList, landuse, transType)
    self.model.landuseByYear["MFHUnits"][(self.thisYear - 1)] += parcelUnits
    self.mfhRolloverPool -= parcelUnits
   else:
    done = True

 def tourTransition(self, transType):
#Tourist transitions use a pool approach similar to the MFH transition approach described above.
#Tourist transitions are straightforward - the transition list, landuse, and transition type are always well defined.

  done = False
  transList = self.thisRunTransitionCandidates[transType]
  while ((not done) and (len(transList) > 0)):
   apn = transList[0]
   candidateParcel = self.model.parcelList[apn]
   parcelUnits = candidateParcel.unitsTour
   if parcelUnits <= self.tourRolloverPool:
    self.transitionParcel(transList, "LU_2002", transType)
    self.model.landuseByYear["TourUnits"][(self.thisYear - 1)] += parcelUnits
```

```
    self.tourRolloverPool -= parcelUnits
   else:
    done = True

 def commTransition(self, transType):
#Comm transitions use a pool approach similar to the MFH and Tour transition approachs described above.
#One difference is that the Comm "pool" is expressed as a single value representing the allowance for the entire life of the simulation, rather than as an annual value.
#The run attribute thisYearCommTarget is calculated as a surrogate for a pool value.
#Another difference is that Comm is based on developed s.f. (some 25% of the parcel s.f.) rather than on parcel units.

  done = False
  transList = self.thisRunTransitionCandidates[transType]
  while ((not done) and (len(transList) > 0)):
   apn = transList[0]
   candidateParcel = self.model.parcelList[apn]
   parcelCommSF = candidateParcel.commFt
   if parcelCommSF <= self.thisYearCommTarget:
    self.transitionParcel(transList, "LU_3000", transType)
    self.model.landuseByYear["CommSF"][(self.thisYear - 1)] += parcelCommSF
    self.thisYearCommTarget -= parcelCommSF
    self.commAllocated += parcelCommSF
   else:
    done = True

 def transitionParcel(self, transList, landuse, transType):
#Now that a scheduled transition has had the setup work done to identify the transition list, landuse, and transition type,
# we can attempt to do a transition.
#A transition can only be done if candidate parcels exist (the length of the transList is greater than 0).
#If there are no candidate parcels, then there is no transition.
#If there is a candidate parcel, then we do the transition with the first parcel in the list:
# For each of the transition types in parcel.thisRunPossibleTransitions, remove that parcel from the associated transition list (this makes the parcel unavailable for any other transitions).
# Update various analysis statistics.

  if len(transList) > 0:
   apn = transList[0]
   parcel = self.model.parcelList[apn]
   for transName in parcel.thisRunPossibleTransitions:
    self.thisRunTransitionCandidates[transName].remove(apn)
```

```
    parcel.landuseCounts[landuse] += 1
    parcel.transitionCounts[transType] += 1
    self.model.landuseByYear[landuse][(self.thisYear - 1)] += 1
    self.model.transitionsByYear[transType][(self.thisYear - 1)] += 1


def main():
  #Here we are invoking the model with a hardcoded model id.
  #The final Java version should pass this value in.
  t = TDSS("1a")

if __name__ == "__main__":
  #Program execution begins here
  main()
```