Demand: In this class final demand is informed. Demand for each use is the total extension of it in each year.

Class	Description
demand = PreComputedValues { attributes = {a,b,c} anualDemand = { {1895.75, 487.75, 435.5}, {1544.875, 456.906,455.84375},	Extension for each land use for each year. Each line in this table presents the demand for one year.

Potential: These classes calculate potential of change for each cell to be used in the allocation component.

Class	Description

potential = LinearRegression
{
landUseDrivers = {...},
regressionData =
{
{
const =,
err=,
betas = {...},
attributes ={...},
},
{
const =,
err =,
betas = {...},
attributes ={...},
},
... One table for each land use
},
}
	Calculates potential of change for each cell, based on linear regression coefficients . Can be used when land use data are continuous.
	Parameters:
	landUseDrivers: the land use drivers fields in database.
	regressionData: a table with the regression parameters for each attribute:
	const: linear regression constant
	error: linear regression estimate error
	betas: linear regression betas for land use drivers
	attributes: index of landUseDrivers to be used by the regression.
	Example:
	landUseDrivers =
	{slope, forest_reserves, connection_ports }
	betas = { 1.03, 0.42}
	attributes = {1,3}
	Means: slope beta = 1.03, connection_ports = 0.42

potential = LogisticRegression { landUseDrivers = {}, regressionData = { { { const =, betas = {...}, attributes ={...}, elasticity =,	Calculates potential of change for each cell, based on logistic regression coefficients . Can be used when land use data are discrete. Parameters: **landUseDrivers:** the land use drivers fields in database. **regressionData:** a table with the regression parameters for each attribute in each region. Therefore it is a two levels table. The first level represents the region while the inner level represent the parameters for each

}, {
const =,
betas = {...},
attributes ={...},
elasticity =,
}, ... One set for each land use
}, ... One set for each region
},
}
	attribute . Regions should be defined in a external model otherwise is considered only one region.
	const: logistic regression constant
	betas: logistic regression betas for land use drivers
	attributes: index of landUseDrivers to be used by the regression.
	elasticity: varies from 0 to 1**.** Elasticity closer to 1 is more difficulty to transition for other land uses.
	Example:
	landUseDrivers =
	{slope, forest_reserves, connection_ports }
	betas = { 1.03, 0.42}
	attributes = {1,3}
	Means: slope beta = 1.03, connection_ports = 0.42

potential =NeighAttractionLogisticRegression
{
filename = "C:\\MyGAL.gal",
landUseDrivers = {},
regressionData =
{
{
{
const =,
betas = {...},
attributes ={...},
elasticity =,
percNeighborsUse =
},
{
const =,
betas = {...},
attributes ={...},
elasticity =,
percNeighborsUse =
}, ... One set for each land use
},
},
}
	Calculates potential of change for each cell, based on logistic regression coefficients, considering cells neighborhood. Can be used when land use data are discrete.
	Parameters:
	filename: GAL file that contains the neighborhood. If no file were defined a Moore neighborhood will be created.
	landUseDrivers: the land use drivers fields in database.
	regressionData: a table with the regression parameters for each attribute in each region. Therefore it is a two levels table. The first level represents the region while the inner level represent the parameters for each attribute . Regions should be defined in a external model otherwise is considered only one region.
	const: logistic regression constant
	betas: logistic regression betas for land use drivers
	attributes: index of landUseDrivers to be used by the regression.
	elasticity: varies from 0 to 1**.** Elasticity closer to 1 is more difficulty to transition for other land uses.
	percNeighborsUse: percent of neighbors with the same use
	Example:
	landUseDrivers =
	{slope, forest_reserves, connection_ports }
	betas = { 1.03, 0.42}

	attributes = {1,3} Means: slope beta = 1.03, connection_ports = 0.42

potential = SpatialLagRegression
{
filename = "C:\\MyGAL.gal",
landUseDrivers = {...},
regressionData =
{
{
ro =,
err=,
betas = {...},
attributes ={...},
},
{
ro =,
err =,
betas = {...},
attributes ={...},
},
... One table for each land use
},
}
	Calculates potential of change for each cell, based on spatial lag regression coefficients . Can be used when land use data are continuous.
	filename: GAL file that contains the neighborhood. If no file were defined a Moore neighborhood will be created.
	landUseDrivers: the land use drivers fields in database.
	regressionData: a table with the regression parameters for each attribute:
	ro: auto regressive coefficient
	error: linear regression estimate error
	betas: linear regression betas for land use drivers
	attributes: index of landUseDrivers to be used by the regression.
	Example:
	landUseDrivers =
	{slope, forest_reserves, connection_ports }
	betas = { 1.03, 0.42}
	attributes = {1,3}
	Means: slope beta = 1.03, connection_ports = 0.42

Allocation: These classes allocate the land uses based on potential calculated for each cell.

Class	Description
allocation = AllocationClueLike { isHierarchicallyCoupled = true, maxDifference =, maxIteration = , limForest =, maxChange =, minElasticity = allocationData = { { isLog = false, static = -1 }, { isLog = true, static = -1 },	This class can be used when land use data are continuous. Parameters: **isCoupled:** if true, print model status during its execution **isLog:** inform whether the model is part of a coupling model **isHierarchicallyCoupled**: whether the model considers more than one scale. **maxDifference**: maximum difference between informed demand and demand allocated by the model. **maxIteration**: limit of interactions trying to allocate the demand. **limForest**: minimum of forest in each cell **maxChange**: limit of change in each cell **minElasticity**: initial value of the parameter which controls the allocation interaction factor

... One set for each use }, }	**allocationData:** a table with two allocatoin parameters for each land use: **isLog**: whether the land use values were log transformed. **static:** direction of land use change**:** -1: unidirectional changes 0: static, no changes are allowed 1: bidirectional changes

allocation = AllocationCluesLike
{
maxIteration =,
factorIteration =,
maxDifference =,
transitionMatrix =
{
{
{0/1, 0/1, 0/1},
{0/1, 0/1, 0/1},
{0/1, 0/1, 0/1},
}
},
}
	This class can be used when land use data are discrete.
	Parameters:
	isCoupled: if true, print model status during its execution
	isLog: inform whether the model is part of a coupling model
	maxIteration: limit of interactions trying to allocate the demand.
	factorIteration: initial value of the parameter which controls the allocation interaction factor
	maxDifference: maximum difference between informed demand and demand allocated by the model.
	transitionMatrix: a table with transition matrix for each region. Each position of matrix indicates the allowable (1) and not allowable (0) transition in a landuse x landuse matrix.

Framework: class that put the components together, executing the whole model.

Model = TopDownLuccModel
{
name = "modelName"
Starttime =,
Endtime =,
cs = CellularSpace {...}
landUseTypes = {...}
demand = ScenariosDemand {...}
potential = LogisticRegressionModel{}
allocation = AllocationCluesLike {...}
save =
{
outputTheme = "OutputTheme",
yearly = false,
saveYears = {...},
saveAttrs = {...},
}
}
	startTime: initial year considered in the model.
	endTime: last year of simulation
	cs: terraME cellular space
	landUseTypes: land use fields name in database
	Save:
	yearly: save the results after each year simulation.
	saveYears: specify which years simulation will be saved.
	If **yearly** is true all years will be saved, even if specific years have been selected in **saveYears.**
	saveAttrs: attributes to be saved.
	If attribute is log transformed it can be saved the transformed variable (name loaded from database) or the variable without transformation (name loaded from database followed by the suffix "_area")