Package 'GMSE'

August 22, 2018

Type Package

Title Generalised Management Strategy Evaluation Simulator

Version 0.4.0.7

Imports grDevices, graphics, stats, shiny, shinydashboard, shinyjs

Maintainer A. Bradley Duthie <brad.duthie@gmail.com>

Description Integrates game theory and ecological theory to construct social-ecological models that simulate the management of populations and stakeholder actions. These models build off of a previously developed management strategy evaluation (MSE) framework to simulate all aspects of management: population dynamics, manager observation of populations, manager decision making, and stakeholder responses to management decisions. The newly developed generalised management strategy evaluation (GMSE) framework uses genetic algorithms to mimic the decision-making process of managers and stakeholders under conditions of change, uncertainty, and conflict. Simulations can be run using gmse(), gmse_apply(), and gmse_gui() functions.

URL https://confoobio.github.io/gmse/

BugReports https://github.com/confoobio/gmse/issues

Depends R (>= 3.4.0)

License GPL (>= 2)

LazyData TRUE

VignetteBuilder R.rsp

Suggests knitr, rmarkdown, testthat, R.rsp

RoxygenNote 6.1.0

NeedsCompilation yes

Author A. Bradley Duthie [aut, cre] (<https://orcid.org/0000-0001-8343-4995>),
Nils Bunnefeld [ctb, fnd] (<https://orcid.org/0000-0002-1349-4463>),
Jeremy Cusack [ctb] (<https://orcid.org/0000-0003-3004-1586>),
Isabel Jones [ctb] (<https://orcid.org/0000-0002-8361-1370>),
Jeroen Minderman [ctb] (<https://orcid.org/0000-0002-8451-5540>),
Erlend Nilsen [ctb] (<https://orcid.org/0000-0002-5119-8331>),

Rocio Pozo [ctb] (<https://orcid.org/0000-0002-7546-8076>),
Sarobidy Rakotonarivo [ctb] (<https://orcid.org/0000-0002-8032-1431>),
Bram Van Moorter [ctb] (<https://orcid.org/0000-0002-3196-1993>)

Repository CRAN

Date/Publication 2018-08-22 04:30:03 UTC

R topics documented:

age_land *Age landscape*

Description

Determines how the landscape will change over the course of one time step. For now, simply reverts a specified layer back to its original values In other words, e.g., crops are annual and regrow undamaged each year.

Usage

```
age_land(LAND, landscape_ini, layer)
```

Arguments

LAND	The name of the landscape being changed
landscape_ini	The name of the original landscape replacing
layer	The layer that is being affected on the landscape

Value

the_land with one layer reset to its original cell values

Examples

```
## Not run:
LANDSCAPE_r <- age_land(LAND = LANDSCAPE_r, landscape_ini = LANDSCAPE_INI,
layer = 2);

## End(Not run)
```

anecdotal *Anecdotal model*

Description

A simulation of how many resources of a particular type are in the vicinity of each agent – this produces a kind of anecdotal evidence for each agent around their circle of view. It also potentially moves the agents during a time step.

Usage

```
anecdotal(RESOURCES = NULL, LAND = NULL, PARAS = NULL,
  AGENTS = NULL, res_type = 1, samp_age = 1, agent_type = 0,
  type_cat = 1, move_agents = FALSE, model = "IBM")
```

Arguments

RESOURCES	The resources array produced by the resource function within GMSE
LAND	The landscape array on which interactions between resources and agents occur
PARAS	The vector of parameters that hold global and dynamic parameter values used by GMSE
AGENTS	The array of agents produced in the main gmse() function
res_type	The type of resources being observed (default = 1)
samp_age	Minimum age of the resource being sampled (default = 1)
agent_type	The type of agent doing the observing (default = 0)
type_cat	The category of agent type (first 4 columns) doing observing; this will almost always be 1, so type 0 agents (managers, of which there is always one by default) will be affected
move_agents	Whether or not agents are moved during the run of anecodtal
model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)

Value

The anecdotal function outputs an R list that includes two separate arrays, including (1) a new AGENTS array and (3) a new PARAS array, each of which might be affected by the anecdotal function. The new arrays can then be read back into the broader GMSE function, thereby affecting the input into the management, user, resource, and observation models.

Examples

```
## Not run:
AGENTS_NEW <- anecdotal(RESOURCES = RESOURCES, LAND = LANDSCAPE_r,
PARAS = paras, AGENTS = AGENTS, res_type = 1, samp_age = rma, agent_type = -1,
type_cat = 1, move_agents = mva);

## End(Not run)
```

---

be_hunter *Become a hunter on the landscape*

Description

This function allows the user of the GMSE software to insert themselves as a hunter in the simulation, allowing them to cull some number of resources in a time step as observed by the agent whose ID is 2.

Usage

```
be_hunter(OBSERVATION, AGENT, RESOURCES, LAND, PARAS, view, times)
```

Arguments

OBSERVATION	The observation array produced by the observation function within GMSE
AGENT	The array of agents produced in the main gmse() function
RESOURCES	The resources array produced by the resource function within GMSE
LAND	The landscape array on which interactions between resources and agents occur
PARAS	The vector of parameters that hold global and dynamic parameter values used by GMSE
view	The distance within which agents are able to observe resources on the landscape
times	The number of times that resources are observed in the observation model of GMSE

Value

the_land A cols by rows landscape with randomly distributed cell types

Examples

```
## Not run:
HUNT_OUTCOME <- be_hunter(OBSERVATION_r, AGENTS, RESOURCES, LANDSCAPE_r,
paras, agent_view, times_observe);

## End(Not run)
```

---

case01plot *Plot results for density-based or mark-recapture sampling*

Description

Produce six panels on a plot showing resource distribution, owned land, resource dynamics and estimates, stake-holder yield, and action costs and actions made. This plot is run internally within the gmse function, and should not be used to plot results stored after running the gmse function (for this, use plot_gmse_results).

Usage

```
case01plot(res, obs, land1, land2, land3, agents, paras, ACTION, COST,
  view = NULL, times = 1)
```

Arguments

res	The resources array produced by the resource function within GMSE
obs	The array of resource observations from the observation model, used to estimate abundance of resources
land1	The first layer of the 3D landscape array, which indicates values of terrain for plotting (as of now, terrain values have no effect on the simulation and only exist for display purposes)

land2	The full list showing all layers of the landscape in each time step of GMSE
land3	The third layer of the 3D landscape array, which indicates agent ownership of the land
agents	The array of agents produced in the main gmse() function
paras	The vector of parameters that hold global and dynamic parameter values used by GMSE
ACTION	A three dimensional array of agent (manager and stakeholder) actions
COST	A three dimensional array of cost values for agent (manager and stakeholder) actions
view	The distance that an agent can see on a landscape
times	The number of times that resources are sampled per time step

Value

This function plots the dynamics of GMSE resource, observation, managemer, and user models in six separate sub-panels. (1) Upper left panel: Shows the locations of resources on the landscape (black dots); landscape terrain is also shown in brown, but at the moment, this is only cosmetic and does not reflect anything occurring in the model. (2) Upper right panel: Shows ownership of land by agents; land is divided proportional based on parameters set in gmse() and colours correspond with other subplots. If agent utilities and actions are restricted to land (‘land_ownership‘ in the gmse() function), then this gives some idea of where actions are being performed and where resources are affecting the landscape. (3) Middle left panel: Shows the actual population abundance (black solid line) and the population abundance estimated by the manager (blue solid line; shading indicates 95 percent confidence intervals) over time. The dotted red line shows the resource carrying capacity (death-based) and the dotted blue line shows the target for resource abundance as set in the gmse() function; the orange line shows the total percent yield of the landscape (i.e., 100 percent means that resources have not decreased yield at all, 0 percent means that resources have completely destroyed all yield). (4) Middle right panel: Shows the raw landscape yield for each stakeholder (can be ignored if ‘land_ownership‘ is FALSE) over time; colours correspond to land ownership shown in the upper right panel. (5) Lower left panel: The cost of stakeholders performing actions over time, as set by the manager. (6) Lower right panel: The total number of actions performed by all stakeholders over time.

Examples

```
## Not run:
case01plot(res = RESOURCE_REC, obs = OBSERVATION_REC,
land1 = LANDSCAPE_r[,,1], land2  = LANDSCAPE_REC, land3  = LANDSCAPE_r[,,3],
agents = AGENT_REC, paras = paras, ACTION = ACTION_REC, COST = COST_REC,
view = agent_view, times = times_observe);

## End(Not run)
```

case23plot *Plot results for transect-based sampling*

Description

Produce six panels on a plot showing resource distribution, owned land, resource dynamics and estimates, stake-holder yield, and action costs and actions made. This plot is run internally within the gmse function, and should not be used to plot results stored after running the gmse function (for this, use plot_gmse_results).

Usage

```
case23plot(res, obs, land1, land2, land3, agents, paras, COST, ACTION)
```

Arguments

res	The resources array produced by the resource function within GMSE
obs	The array of resource observations from the observation model, used to estimate abundance of resources
land1	The first layer of the 3D landscape array, which indicates values of terrain for plotting (as of now, terrain values have no effect on the simulation and only exist for display purposes)
land2	The full list showing all layers of the landscape in each time step of GMSE
land3	The third layer of the 3D landscape array, which indicates agent ownership of the land
agents	The array of agents produced in the main gmse() function
paras	The vector of parameters that hold global and dynamic parameter values used by GMSE
COST	A three dimensional array of cost values for agent (manager and stakeholder) actions
ACTION	A three dimensional array of agent (manager and stakeholder) actions

Value

This function plots the dynamics of GMSE resource, observation, managemer, and user models in six separate sub-panels. (1) Upper left panel: Shows the locations of resources on the landscape (black dots); landscape terrain is also shown in brown, but at the moment, this is only cosmetic and does not reflect anything occurring in the model. (2) Upper right panel: Shows ownership of land by agents; land is divided proportional based on parameters set in gmse() and colours correspond with other subplots. If agent utilities and actions are restricted to land (‘land_ownership‘ in the gmse() function), then this gives some idea of where actions are being performed and where resources are affecting the landscape. (3) Middle left panel: Shows the actual population abundance (black solid line) and the population abundance estimated by the manager (blue solid line) over time. The dotted red line shows the resource carrying capacity (death-based) and the dotted blue line shows the target for resource abundance as set in the gmse() function; the orange line shows the total percent yield

of the landscape (i.e., 100 percent means that resources have not decreased yield at all, 0 percent means that resources have completely destroyed all yield). (4) Middle right panel: Shows the raw landscape yield for each stakeholder (can be ignored if 'land_ownership' is FALSE) over time; colours correspond to land ownership shown in the upper right panel. (5) Lower left panel: The cost of stakeholders performing actions over time, as set by the manager. (6) Lower right panel: The total number of actions performed by all stakeholders over time.

Examples

```
## Not run:
case23plot(res = RESOURCE_REC, obs = OBSERVATION_REC,
land1 = LANDSCAPE_r[,,1], land2 = LANDSCAPE_REC, land3  = LANDSCAPE_r[,,3],
agents = AGENT_REC, COST = COST_REC, ACTION = ACTION_REC, paras  = paras);

## End(Not run)
```

---

chapman_est *Chapman estimator of mark-recapture*

Description

Estimates population size using simulated mark-recapture data produced by the observation model of GMSE.

Usage

```
chapman_est(observation, paras)
```

Arguments

observation	The array of resource observations from the observation model, used to estimate abundance of resources
paras	The vector of parameters that hold global and dynamic parameter values used by GMSE

Value

The Chapman estimator (which is also performed GMSE in the manager function) returns a list that includes resource population size estimates along with 95

Examples

```
## Not run:
analysis <- chapman_est(observation=obs_t, paras = paras);

## End(Not run)
```

dens_est *Density estimator of resource abundance*

Description

Estimates population size using simulated data produced by the observation model of GMSE – it assumes that the density of resources observed on the subset of the landscape sampled equals the density on the whole landscape.

Usage

```
dens_est(observation, paras, view = view, land = land)
```

Arguments

observation	The array of resource observations from the observation model, used to estimate abundance of resources
paras	The vector of parameters that hold global and dynamic parameter values used by GMSE
view	This parameter determines the distance around an agent's location within which it can observe resources.
land	The landscape array on which interactions between resources and agents occur

Value

The density estimator (which is also performed GMSE in the manager function) returns a list that includes resource population size estimates along with 95

Examples

```
## Not run:
analysis <- dens_est(observation = obs_t, paras = paras, view = view,
land = land1);

## End(Not run)
```

gmse *GMSE simulation*

Description

The gmse function is the the primary function to call to run a simulation. It calls other functions that run resource, observation, management, and user models in each time step. Hence while individual models can be used on their own, gmse() is really all that is needed to run a simulation.

Usage

```
gmse(time_max = 100, land_dim_1 = 100, land_dim_2 = 100,
  res_movement = 20, remove_pr = 0, lambda = 0.3, agent_view = 10,
  agent_move = 50, res_birth_K = 1e+05, res_death_K = 2000,
  edge_effect = 1, res_move_type = 1, res_birth_type = 2,
  res_death_type = 2, observe_type = 0, fixed_mark = 100,
  fixed_recapt = 500, times_observe = 1, obs_move_type = 1,
  res_min_age = 0, res_move_obs = TRUE, Euclidean_dist = FALSE,
  plotting = TRUE, hunt = FALSE, start_hunting = 95,
  res_consume = 0.5, ga_popsize = 100, ga_mingen = 40,
  ga_seedrep = 20, ga_sampleK = 20, ga_chooseK = 2,
  ga_mutation = 0.1, ga_crossover = 0.1, move_agents = TRUE,
  max_ages = 5, minimum_cost = 10, user_budget = 1000,
  manager_budget = 1000, manage_target = 1000, RESOURCE_ini = 1000,
  scaring = FALSE, culling = TRUE, castration = FALSE,
  feeding = FALSE, help_offspring = FALSE, tend_crops = FALSE,
  tend_crop_yld = 0.2, kill_crops = FALSE, stakeholders = 4,
  manage_caution = 1, land_ownership = FALSE, manage_freq = 1,
  converge_crit = 0.1, manager_sense = 0.9, public_land = 0,
  group_think = FALSE)
```

Arguments

time_max	This value sets the maximum number of time steps for a simulation. There are no constraints for length of time that a simulation can run. The default is 100 time steps.
land_dim_1	This value sets the number of cells on the x dimension of the landscape (i.e., the number of columns in the landscape array; this can also be thought of as the x-axis when the landscape image is plotted). There is no maximum, but the minimum dimension of a landscape is 2 cells. The default is 100 cells.
land_dim_2	This value sets the number of cells on the y dimension of the landscape (i.e., the number of columns in the landscape array; this can also be thought of as the y-axis when the landscape image is plotted). There is no maximum, but the minimum dimension of a landscape is 2 cells. The default is 100 cells.
res_movement	This value determines how far resources move during a time step. Exact movement is probabilistic and partly affected by 'res_move_type' settings. Under default settings, during each time step, resources move from zero to res_movement cells away from their starting cell in any direction. Hence res_movement is the maximum distance away from a resources starting cell that it can move in a time step; other types of resource movement, however, interpret res_movement differently to get the raw distance moved (see res_move_type). The default value is 20.
remove_pr	This value is the density-independent and user-independent probability of a resource being removed (e.g., dying) during a time step in the resource model. Under default settings, this value is set to zero, with resource removal being determined entirely by carrying capacity on resource survival, and by user actions.

`lambda` This value is the baseline population growth rate of resources. Each resource in the simulation produces Poisson(lambda) offspring in one time step within the resource model. The value of lambda might be increased or decreased by user actions, and juvenile survival can potentially be decreased by a carrying capacity placed on birth. The default value is 0.3, meaning that the average resource produces one offspring every three time steps.

`agent_view` This value determines how far agents (managers and stakeholders) can see on the landscape. At the moment, this affects only the sampling ability of managers in the observation model for density-based and transect-based estimates of resource abundance. In these types of estimates, when managers have a higher agent_view, they are capable of observing a larger area of landscape and therefore of getting a larger (in the case of density-based estimation) or more efficient (in the case of transect-based estimation) sample of resources from which to estimate total resource abundance. The default value of agent_view is 10, so agents can see 10 cells away from their current cell in any direction.

`agent_move` This value determines how far agents can move. At the moment, this does not affect much in the simulation because agent movement does not affect agent actions (interactions with resources can be limited to stakeholder's owned land, but do not currently depend on where an agent is on the landscape – effectively assuming that agents are mobile enough to do what they want to do to resources). The one exception is for density-based estimation, which can be biased by low values of agent_move by causing the manager to sample the same (or nearby) landscape cells to estimate total resource abundance; if resources are spatially autocorrelated, then managers might over or under-estimate total abundance. Therefore, as a default, this value is set to 50 so that managers can move to any cell on a (torus) landscape in a time step, removing any bias for density sampling.

`res_birth_K` This value is the carrying capacity on new resources added per time step (e.g., birth). If more offspring are born in a time step than res_birth_K, then offspring are randomly removed from the population until offspring born equals res_birth_K. By default, carrying capacity is effectively applied to death instead of birth, so the default value of res_birth_K is set to 100000 (and hence not enacted because the number of births is never this high).

`res_death_K` This value is the carrying capacity on resources in the population. Carrying capacity is realised by an increase in mortality probability as resource abundance approaches res_death_K. In each time step, realised mortality probability equals the number of resources over carrying capacity divided by the number of resources (i.e., [resource count - carrying capacity] / resource count). Hence, as the resource abundance increases above carrying capcity, mortality probability also increases in proportion, generating some stochasticity in resource survival. Note that carrying capacity is independent of user actions; if a user culls a resource this culling is applied after mortality probability due to carrying capacity has already been calculated. The default value for res_death_K is 2000.

`edge_effect` This determines what happens at the edge of the landscape. Currently there is only one option (value 1), which causes the landscape to wrap around as a torus (effectively removing the edge); resources that leave off of one side of the landscape will reappear on the other side of the landscape.

`res_move_type` This determines the type of movement that resources do. There are four different movement options: (0) No movement – resources are sessile, (1) Uniform movement in any direction up to 'res_movement' cells away during a time step. Movement direction is random and the cell distance moved is randomly selected from zero to 'res_movement'. (2) Poisson selected movement in the x and y dimensions where distance in each direction is determined by Poisson(res_movement) and direction (e.g., left versus right) is randomly selected for each dimension. This type of movement tends to look a bit odd with low 'res_movement' values because it results in very little diagonal movement. It also is not especially biologically realistic, so should probably not be used without a good reason. (3) Uniform movement in any direction up to 'res_movement' cells away during a a time step 'res_movement' times. In other words, the 'res_movement' variable of each resource is acting to determine the times that a resource moves in a time step and the maximum distance it travels each time it moves. This type of movement has been simulated in ecological models, particularly plant-pollinator systems. The default movement type is (1).

`res_birth_type` The type of resource addition (birth) that occurs. Currently, the only value allowed is 2, which causes all resources to produce Poisson(lambda) offspring each time step, where 'lambda' is the population growth rate also set as an argument in gmse simulations.

`res_death_type` The type of resource removal (death) that occurs. A value of (1) causes death to be entirely density-independent and with a probability of 'removal_pr' for each resource (which may be further affected by agent actions or interactions with landscape cells). A value of (2) causes death to be entirely density-dependent (though potentially independently affected by agents and landscape), with mortality probability calculated based on the carrying capacity 'res_death_K' set in as an argument in gmse simulations. A value of (3) allows for both density-dependent (affected by 'res_death_K') and density-independent (affected by 'removal_pr') effects on resource removal. The default 'res_death_type' is (2); values of (1) must be used carefully because it can result in exponential growth that leads to massive population sizes that slow down simulations.

`observe_type` The type of observation sampling of resources being done by managers in the observation model. There are currently four options for sampling. (0) Density-based sampling, in which managers sample all resources within some subset of the landscape; the size of this subset is all of the resources within a distance of 'agent_view' from the cell of the manager. Managers sample 'times_observe' subsets, where 'times_observe' is a parameter value set in the gmse simulation. Managers then extrapolate the density of resources in the subset to estimate the total number of resources on a landscape. (1) Mark-recapture estimate of the popluation, in which managers randomly sample 'fixed_mark' resources (without replacement) in the population without any spatial bias (if there are fewer than 'fixed_mark' resources, managers sample all resources). The manager then randomly samples 'fixed_recapt' resources (without replacement), again without any spatial bias. A Chapman estimate is then used in the manager model to estimate population size from these mark-recapture data. (2) Transect-based sampling (linear), in which a manager samples an entire row of the landscape and counts the resources on the row, then moves onto the next row of the landscape until the entire landscape has been covered. The number of cells in each

row (i.e., the height) equals ‘agent_view‘, so fewer transects are needed if agents can see farther. If ‘res_move_obs == TRUE‘, then resources can move on the landscape between each transect sampling, potentially causing observation error if some resources are double counted or not counted at all due to movement. If ‘res_move_obs == FALSE‘, then this type of observation should produce no error, and resource estimation will be exact. (3) Transect-based sampling (block), in which a manager samples a block of the landscape and counts the resources in the block, then moves on to the next (equally sized) block until the entire landscape has been covered. Blocks are square, with the length of each side equaling ‘agent_view‘, so fewer blocks are needed if agents can see farther. If ‘res_move_obs == TRUE‘, then resources can move on the landscape between each block sampling, potentially causing observation error if some resources are double counted or not counted at all due to movement. If ‘res_move_obs == FALSE‘, then this type of observation should produce no error, and resource estimation will be exact. The default observation type is 0 for density-based sampling.

`fixed_mark` This parameter affects mark-recapture observation (i.e., applies only when observe_type == 1). Its value defines how many resources will be marked in each time step as part of a mark-recapture population size estimate.

`fixed_recapt` This parameter affects mark-recapture observation (i.e., applies only when observe_type == 1). Its value defines how many resources will be (re)captured in each time step as part of a mark-recapture population size estimate.

`times_observe` This parameter defines how many times a manager will make observations within the observation model; it applies only to density-based sampling (‘observe_type = 0‘) and mark-recapture sampling (‘observe_type = 1‘). In the former case, the value determines how many times the manager goes out to sample resources from a subset of the landscape. In the latter case, the value determines how many times the manager goes out to attempt to find new resources to mark or recapture (hence its value must be greater than ‘fixed_observe‘).

`obs_move_type` This determines the type of movement that agents do. The four different movement types of agents are identical to those of resources: : (0) No movement – agents are sessile, (1) Uniform movement in any direction up to ‘agent_move‘ cells away during a time step. Movement direction is random and the cell distance moved is randomly selected from zero to ‘agent_move‘. (2) Poisson selected movement in the x and y dimensions where distance in each direction is determined by Poisson(agent_move) and direction (e.g., left versus right) is randomly selected for each dimension. This type of movement tends to look a bit odd with low ‘agent_move‘ values because it results in very little diagonal movement. It also is not especially realistic, so should probably not be used without a good reason. (3) Uniform movement in any direction up to ‘agent_move‘ cells away during a a time step ‘agent_move‘ times. In other words, the ‘agent_move‘ variable of each agent is acting to determine the times that an agent moves in a time step and the maximum distance it travels each time it moves. This type of movement has been simulated in ecological models, particularly plant-pollinator systems. The default movement type is (1).

`res_min_age` This value defines the minimum age at which resources are recorded and acted upon by agents; below this age, resources are ignored. The default value of this

parameter is 0. Note that the population might appear to go over carrying capacity regularly because carrying capacity is not realised until the next resource model if it applies to the death of resource (this is not a problem for the simulation itself, it just needs to be noted). If the value is set to 1, then offspring just produced during a time step (age = 0) are not observed or acted upon by agents.

`res_move_obs` This is a TRUE or FALSE value that defines whether or not resources are to move between 'times_observe' times being observed. The default value is TRUE, but if the option is set to FALSE then it shuts down all resource movement during sampling (making 'observe_type = 2' and 'observe_type = 3' error free).

`Euclidean_dist` This is a TRUE or FALSE value that defines whether distance in the simulation should be judged as number of cells away or the actual Euclidean distance between points (e.g., if the landscape were interpreted as a map). The default is set to FALSE, and until GMSE is capable of reading in real-world maps, I don't think there is any good reason to set it to TRUE.

`plotting` This is a TRUE or FALSE value that determines whether or not the simulation results will be plotted. The default is TRUE. If plotted, then a function is called to show the dynamics of resources and agent actinos over time. The plotted function plots the dynamics of GMSE resource, observation, managemer, and user models in six separate sub-panels. (1) Upper left panel: Shows the locations of resources on the landscape (black dots); landscape terrain is also shown in brown, but at the moment, this is only cosmetic and does not reflect anything occurring in the model. (2) Upper right panel: Shows ownership of land by agents; land is divided proportional based on parameters set in gmse() and colours correspond with other subplots. If agent utilities and actions are restricted to land ('land_ownership' in the gmse() function), then this gives some idea of where actions are being performed and where resources are affecting the landscape. (3) Middle left panel: Shows the actual population abundance (black solid line) and the population abundance estimated by the manager (blue solid line) over time. The dotted red line shows the resource carrying capacity (death-based) and the dotted blue line shows the target for resource abundance as set in the gmse() function; the orange line shows the total percent yield of the landscape (i.e., 100 percent means that resources have not decreased yield at all, 0 percent means that resources have completely destroyed all yield). (4) Middle right panel: Shows the raw landscape yield for each stakeholder (can be ignored if 'land_ownership' is FALSE) over time; colours correspond to land ownership shown in the upper right panel. (5) Lower left panel: The cost of stakeholders performing actions over time, as set by the manager. (6) Lower right panel: The total number of actions performed by all stakeholders over time.

`hunt` This is a TRUE or FALSE value that determines whether the simulation will be halted each time step after 'start_hunting' time steps to ask the user how many resources they want to hunt (some management information is given to help make this choice). This feature will be expanded upon in later versions. Right now, the human is playing the role of agent number 2, the first stake-holder in the simulation. By default, this value is set to FALSE.

`start_hunting` The time step in which the human (*not* the simulated agent) is allowed to start hunting if 'hunt = TRUE'. The default value is 95.

`res_consume` The fraction of remaining biomass (e.g. crop production) that a resource consumes while occupying a landscape cell. The default value is 0.5, so if one resource occupies the cell, then landscape production is halved, if two resources occupy the cell, then landscape production drops to 0.25; if three, then production drops to 0.125, etc.

`ga_popsize` The size of populations of agents in the genetic algorithm (not resources in the simulation). The actions of each agent in the simulation are duplicated 'ga_popsize' times, and this population of individual agent actions undergoes a process of natural selection to find an adaptive strategy. Selection is naturally stronger in larger populations, but a default population size of 100 is more than sufficient to find adaptive strategies.

`ga_mingen` The minimum number of generations in the genetic algorithms of the simulation (*not* the number of time steps in the simulation itself). The actions of each agent in the simulation are duplicated 'ga_popsize' times, and this population of individual agent actions undergoes a process of natural selection at least 'ga_mingen' times to find an adaptive strategy. If convergence criteria 'converge_crit' is set to a default value of 100, then the genetic algorithm will almost always continue for exactly 'ga_mingen' generations. The default value is 40, which is usually plenty for finding adaptive agent strategies – the objective is not to find optimal strategies, but strategies that are strongly in line with agent interests.

`ga_seedrep` At the start of each genetic algorithm, 'ga_popsize' replicate agents are produced; 'ga_seedrep' of these replicates are *exact* replicates, while the rest have random actions to introduce variation into the population. Because adaptive agent strategies are not likely to change wildly from one generation to the next, it is highly recommended to use some value of 'ga_seedrep' greater than zero; the default value is 20, which does a good job of finding adaptive strategies.

`ga_sampleK` In the genetic algorithm, fitnesses are assigned to different agent strategies and compete in a tournament to be selected into the next generation. The tournament samples 'ga_sampleK' strategies at random and with replacement from the population of 'ga_popsize' to be included in the tournament. The default value is 20.

`ga_chooseK` In the genetic algorithm, fitnesses are assigned to different agent strategies and compete in a tournament to be selected into the next generation. The tournament samples 'ga_sampleK' strategies at random and with replacement from the population of 'ga_popsize' to be included in the tournament, and from these randomly selected strategies, the top 'ga_chooseK' strategies are selected. The default value is 2, so the top 10 percent of the random sample in a tournament makes it into the next generation (note that multiple tournaments are run until 'ga_popsize' strategies are selected for the next generation).

`ga_mutation` In the genetic algorithm, this is the mutation rate of any action within an agent's strategy. When a mutation occurs, the action is either increased or decreased by a value of 1. If the action drops below zero, then the value after mutation is multiplied by -1.

`ga_crossover` In the genetic algorithm, this is the crossover rate of any action within an agent's strategy with a randomly selected different strategy in the population of size

'ga_popsize'.

`move_agents` This is a TRUE or FALSE value that defines whether or not agents should move at the end of each time step. The default value is TRUE.

`max_ages` This is the maximum age of resources. If resources reach this age, then they are removed in the resource model with a probability of 1. The default 'max_ages' is 5.

`minimum_cost` This is the mimimum cost of any action in the manager and user models. Higher values allow managers to have greater precision when setting policy. For example, managers believe (typically correctly) that they will double culling number by setting the cost of culling at 1 instead of 2. If actions always cost at least some minium value, then some increment just above that value is always available to more precisely affect user actions. Hence it is generally better to simply give everyone a bigger budget and set a minimum cost, giving more precision to managers to fine tune policy. The default value of minimum_cost is therefore set to 10.

`user_budget` This is the total budget of each stakeholder for performing actions. The cost of performing an action is determined by the 'miminimum_cost' of actions, and the policy set by the manager. The default 'user_budget' is 1000. The maximum budget is 100000.

`manager_budget` This is the total budget for the manager when setting policy. Higher budgets make it easier to restrict the actions of stakeholders; lower budgets make it more difficult for managers to limit the actions of stakeholders by setting policy. The default 'manager_budget' is 1000. The maximum budget is 10000.

`manage_target` This is the target resource abundance that the manager attempts to keep the population at; the default value is 1000.

`RESOURCE_ini` This is the initial abundance of resources at the start of the simulation; the default is 1000.

`scaring` This is a TRUE or FALSE value determining whether or not scaring is an option for managers and stakeholders. If so, then stakeholders that scare cause resources to be moved from their current landscape cell to a random cell on the landscape (note, it is possible that the resource could be scared back onto the stakeholder's own land again). The default value of this is FALSE.

`culling` This is a TRUE or FALSE value determining whether or not culling is an option for managers and stakeholders. If so, then stakeholders that cull cause the resource to be removed from the simulation permanently (i.e., killing the resource). The default value of this is TRUE.

`castration` This is a TRUE or FALSE value determining whether or not castration is an option for managers and stakeholders. If so, then stakeholders that castrate do not remove the resource from the simulation, but prohibit the resource from reproducing by setting its 'lambda' value to zero. The default value of this is FALSE.

`feeding` This is a TRUE or FALSE value determining whether or not feeding is an option for managers and stakeholders. If so, then stakeholders that feed increase a resource's growth rate (lambda) for one time step by 100 percent. The default value of this is FALSE.

help_offspring This is a TRUE or FALSE value determining whether or not feeding is an option for managers and stakeholders. If so, then stakeholders that help_offspring increase a resource's offspring production for one time step by one (i.e., one more offspring is produced). The default value of this is FALSE.

tend_crops This is a TRUE or FALSE value determining whether or not tending crops on the landscape is allowed for stakeholders. If so, then stakeholders can increase one cells yield by 50 percent for each action to 'tend_crops'. Actions on the landscape cannot be regulated by managers, so the cost of this action is always 'minimum_cost'. The default value of this is FALSE.

tend_crop_yld The per landscape cell proportional increase in crop yield when stakeholders take one action to increase yield on their landscape. The default value is set to 0.5 (i.e., a 50 percent increase in yield on a cell).

kill_crops This is a TRUE or FALSE value determining whether or not killing crops on the landscape is allowed for stakeholders. If so, then stakeholders can remove the crop yield on a cell completely for each action to 'kill_crops'. Actions on the landscape cannot be regulated by managers, so the cost of this action is always 'minimum_cost'.

stakeholders This is the number of stakeholders in a simulation; there is always one manager, plus any natural number of stakeholders.

manage_caution This value moderates the caution a manager has when changing policy by assuming that at least 'manage_caution' of each possible action will always be performed by stakeholders. I manager will therefore not ignore policy for one action because no stakeholder is engaging in it; the default value of 'manage_caution' is 1.

land_ownership This value defines whether stakeholders own land and their actions are restricted to land that they own. If FALSE, then stakeholders can act on any landscape cell; if TRUE, then agents can only act on their own cells. The default of this value is FALSE.

manage_freq This is the frequency with which policy is set by managers; a value of 1 means that policy is set in the manager model every time step; a value of 2 means that poilcy is set in the manager model every other time step, etc. The default value is 1.

converge_crit This is the convergence criteria for terminating a genetic algorithm. After continuing for the minimum number of generations, 'ga_mingen', the genetic algorithm will terminate if the convergence criteria is met. Usually making this criteria low doesn't do much to improve adaptive strategies; the default value is 1, which means that the genetic algorithm will continue as long as there is greater than a 1 percent increase in strategy fitness.

manager_sense This adjusts the sensitivity that a manager assumes their actions have with respect to changes in costs (their policy). For example, given a 'manage_sense' value of 0.9, if the cost of culling resources doubles, then instead of a manager assuming the the number of culled resources per user will be cut in half, the manager will instead assume that the number of resources culled will be cut by one half times eight tenths. As a general rule, a value of ca 0.8 allows the manager to predict stake-holder responses to policy accurately; future versions of GMSE could allow managers to adjust this dynamically based on simulation history.

public_land — The proportion of the landscape that will be public, and not owned by stakeholders. The remaining proportion of the landscape will be evenly divided among stakeholders. Note that this option is only available when land_ownership == TRUE.

group_think — If TRUE, all users will have identical actions; the genetic algorithm will find actions for one user and copy them for all users. This is a useful option if a lot of users are required but variation among user decisions can be ignored.

Value

A large list is returned that includes detailed simulation histories for the resource, observation, management, and user models. This list includes eight elements, most of which are themselves complex lists of arrays: (1) A list of length 'time_max' in which each element is an array of resources as they exist at the end of each time step. Resource arrays include all resources and their attributes (e.g., locations, growth rates, offspring, how they are affected by stakeholders, etc.). (2) A list of length 'time_max' in which each element is an array of resource observations from the observation model. Observation arrays are similar to resource arrays, except that they can have a smaller number of rows if not all resources are observed, and they have additional columns that show the history of each resource being observed over the course of 'times_observe' observations in the observation model. (3) A 2D array showing parameter values at each time step (unique rows); most of these values are static but some (e.g., resource number) change over time steps. (4) A list of length 'time_max' in which each element is an array of the landscape that identifies proportion of crop production per cell. This allows for looking at where crop production is increased or decreased over time steps as a consequence of resource and stakeholder actions. (5) The total time the simulation took to run (not counting plotting time). (6) A 2D array of agents and their traits. (7) A list of length 'time_max' in which each element is a 3D array of the costs of performing each action for managers and stakeholders (each agent gets its own array layer with an identical number of rows and columns); the change in costs of particular actions can therefore be be examined over time. (8) A list of length 'time_max' in which each element is a 3D array of the actions performed by managers and stakeholders (each agent gets its own array layer with an identical number of rows and columns); the change in actions of agents can therefore be examined over time. Because the above lists cannot possibly be interpreted by eye all at once in the simulation output, it is highly recommended that the contents of a simulation be stored and interprted individually if need be; alternativley, simulations can more easily be interpreted through plots when 'plotting = TRUE'.

Examples

```
sim <- gmse(lambda = 0.3, land_ownership = TRUE, time_max = 10);
```

---

gmse_apply *GMSE apply function*

Description

The gmse_apply function is a flexible function that allows for user-defined sub-functions calling resource, observation, manager, and user models. Where such models are not specified, GMSE sub-models 'resource', 'observation', 'manager', and 'user' are run by default. Any type of sub-model

(e.g., numerical, individual-based) is permitted as long as the input and output are appropriately specified. Only one time step is simulated per call to gmse_apply, so the function must be looped for simulation over time. Where model parameters are needed but not specified, defaults from gmse are used.

Usage

```
gmse_apply(res_mod = resource, obs_mod = observation,
  man_mod = manager, use_mod = user, get_res = "basic",
  old_list = NULL, ...)
```

Arguments

res_mod	The function specifying the resource model. By default, the individual-based resource model from gmse is called with default parameter values. User-defined functions must either return an unnamed matrix or vector, or return a named list in which one list element is named either 'resource_array' or 'resource_vector', and arrays must follow the format of GMSE in terms of column number and type (if there is only one resource type, then the model can also just return a scalar value).
obs_mod	The function specifying the observation model. By default, the individual-based observation model from gmse is called with default parameter values. User-defined functions must either return an unnamed matrix or vector, or return a named list in which one list element is named either 'observation_array' or 'observation_vector', and arrays must follow the format of GMSE in terms of column number and type (if there is only one resource type, then the model can also just return a scalar value).
man_mod	The function specifying the manager model. By default, the individual-based manager model that calls the genetic algorithm from gmse is used with default parameter values. User-defined functions must either return an unnamed matrix or vector, or return a named list in which one list element is named either 'manager_array' or 'manager_vector', and arrays must follow the (3 dimensional) format of the 'COST' array in GMSE in terms of column numbers and types, with appropriate rows for interactions and layers for agents (see documentation of GMSE for constructing these, if desired). User defined manager outputs will be recognised as costs by the default user model in gmse, but can be interpreted differently (e.g., total allowable catch) if specifying a custom user model.
use_mod	The function specifying the user model. By default, the individual-based user model that calls the genetic algorithm from gmse is used with default parameter values. User-defined functions must either return an unnamed matrix or vector, or return a named list in which one list element is named either 'user_array' or 'user_vector', and arrays must follow the (3 dimensional) format of the 'ACTION' array in GMSE in terms of column numbers and types, with appropriate rows for interactions and layers for agents (see documentation of GMSE for constructing these, if desired).
get_res	How the output should be organised. The default 'basic' attempts to distill results down to their key values from submodel outputs, including resource abundances and estimates, and manager policy and actions. An option 'custom' sim-

ply returns a large list that includes the output of every submodel. Any other option (e.g. 'none') will return a large list with all of the input, output, and parameters used to run gmse_apply. This list will also include a list element named 'basic_output', which will display the default results.

`old_list` A an existing list of results from gmse_apply, produced by setting 'get_res = TRUE' to be included in the function. The parameter and data structures from the previous run will be applied to the new run of gmse_apply, thereby making it easy to loop multiple generations. Additional arguments passed to '...' will over-ride those stored in the old list, allowing global parameter values to be updated (e.g., sub-models used, management options, genetic algorithm parameters). Note that if these arguments are passed, the function will attempt to work with them even if it means removing previous list elements (e.g., if a new number of stakeholders is passed through stakeholder = new_value, then an entirely new AGENT array and user and manager arrays will need to be built).

`...` Arguments passed to user-defined functions, and passed to modify default parameter values that would otherwise be called for gmse default models. Any argument that can be passed to gmse can be specified explicitly, just as if it were an argument to gmse. Similarly, any argument taken by a user-defined function should be specified, though the function will work if the user-defined function has a default that is not specified explicitly.

Details

To integrate across different types of submodels, gmse_apply translates between vectors and arrays between each submodel. For example, because the default GMSE observation model requires a resource array with particular requirements for column identites, when a resource model subfunction returns a vector, or a list with a named element 'resource_vector', this vector is translated into an array that can be used by the observation model. Specifically, each element of the vector identifies the abundance of a resource type (and hence will usually be just a single value denoting abundance of the only focal population). If this is all the information provided, then a resource_array will be made with default GMSE parameter values with an identical number of rows to the abundance value (floored if the value is a non-integer; non-default values can also be put into this transformation from vector to array if they are specified in gmse_apply, e.g., through an argument such as lambda = 0.8). Similarly, a 'resource_array' is also translated into a vector after the default individual-based resource model is run, should the observation model require simple abundances instead of an array. The same is true of 'observation_vector' and 'observation_array' objects returned by observation models, of 'manager_vector' and 'manager_array' (i.e., COST) objects returned by manager models, and of 'user_vector' and 'user_array' (i.e., ACTION) objects returned by user models. At each step, a translation between the two is made, with necessary adjustments that can be tweaked through arguments to gmse_apply when needed.

Parameter changes are accommodated by rebuilding data structures whenever necessary. For example, if the number of stakeholders is changed (and by including an argument 'stakeholders' to gmse_apply, it is assumed that stakeholders are changing even the value is identical to what is in the old_list), then a new array of agents will be built. If landscape dimensions are changed (i.e., if the argument 'land_dim_1' or 'land_dim_2' is included), then a new landscape willl be built. For custom defined GMSE sub-functions, arguments passed to '...' will not be found or updated, so changes to arguments of custom functions should be made directly to the 'old_list', or the use of old_list should be avoided.

Examples

```
sim <- gmse_apply();
sim <- gmse_apply(stakeholders = 2);
sim <- gmse_apply(obs_mod = function(resource_vector) rnorm(1, resource_vector, 10));
```

---

gmse_gui *GMSE GUI function*

Description

The gmse_gui function will call a browser-based graphical user interface (GUI) for the gmse function. The GUI will run simulations for a limited range of parameter values and present results as plots.

Usage

```
gmse_gui()
```

Value

A browser should immediately open with the gmse graphical user interface

Examples

```
## Not run:
sim <- gmse_gui();

## End(Not run)
```

---

gmse_replicates *gmse replicate simulations*

Description

Replicates the same simulation for a set of parameter values

Usage

```
gmse_replicates(replicates, all_time = FALSE,
  hide_unused_options = TRUE, ...)
```

Arguments

replicates	The number of replicate simulations to be run
all_time	Passes to gmse_table. If TRUE, then results from all time steps of the simulation are returned; if FALSE (default), then only results from the last time step of each simulation is returned.
hide_unused_options	Passes to gmse_table. If TRUE (default), then action options (e.g., scaring, culling, etc.) that are not available are not included in the results summary. If FALSE, then they are included as 'NA'
...	Parameter values to be passed to the gmse function

Value

A simplified list that includes four elements, each of which is a table of data: 1. resources, a table showing time step in the first column, followed by resource abundance in the second column. 2. observations, a table showing time step in the first column, followed by the estimate of population size (produced by the manager) in the second column. 3. costs, a table showing time step in the first column, manager number in the second column (should always be zero), followed by the costs of each action set by the manager (policy); the far-right column indicates budget that is unused and therefore not allocated to any policy. 4. actions, a table showing time step in the first column, user number in the second column, followed by the actions of each user in the time step; additional columns indicate unused actions, crop yield on the user's land (if applicable), and the number of resources that a user successfully harvests (i.e., 'culls').

Examples

```
## Not run:
sim_replicates <- gmse_replicates(replicates = 2, time_max = 5);

## End(Not run)
```

---

gmse_summary *gmse results summary*

Description

Summarise gmse output in a more user-friendly format

Usage

```
gmse_summary(gmse_results)
```

Arguments

gmse_results	The full list as returned by the gmse function

Value

A simplified list that includes four elements, each of which is a table of data: 1. resources, a table showing time step in the first column, followed by resource abundance in the second column. 2. observations, a table showing time step in the first column, followed by the estimate of population size (produced by the manager) in the second column. 3. costs, a table showing time step in the first column, manager number in the second column (should always be zero), followed by the costs of each action set by the manager (policy); the far-right column indicates budget that is unused and therefore not allocated to any policy. 4. actions, a table showing time step in the first column, user number in the second column, followed by the actions of each user in the time step; additional columns indicate unused actions, crop yield on the user's land (if applicable), and the number of resources that a user successfully harvests (i.e., 'culls').

Examples

```
## Not run:
sim_summary <- gmse_summary(gmse_results = sim);

## End(Not run)
```

---

gmse_table *GMSE table results*

Description

The gmse_table function takes results created from simulations of the gmse and concatenates key results from a large list into a more manageable data table.

Usage

```
gmse_table(gmse_sim, hide_unused_options = TRUE, all_time = TRUE)
```

Arguments

gmse_sim	The output of a 'gmse' simulation.
hide_unused_options	Whether or not to hide results from policy options when creating the resulting table. If 'TRUE' (default), then policy and user actions that are not allowed in a simulation will not be placed as columns. If 'FALSE', then these columns will be placed with values of 'NA'.
all_time	Whether or not results from each time step from the simulation should be individually placed as a row in the resulting table ('TRUE' by default). If 'FALSE', then only the last row will be placed.

Value

A table with one or more rows of results, each of which indicates a unique 'gmse' simulation for a given time step. Columns represent key simulation including resource densities, observation estimates, policy, and user actions.

Examples

```
sim       <- gmse(time_max = 10);
sim_table <- gmse_table(gmse_sim = sim);
```

`ind_to_land` *Plot resource position on a landscape image output*

Description

Places individuals (simulated resources) on the landscape for plotting.

Usage

```
ind_to_land(inds, land)
```

Arguments

`inds`	A single time step of resources from GMSE
`land`	The landscape array on which interactions between resources and agents occur

Value

Returns a landscape in which resources are embedded for a timestep for plotting purposes

Examples

```
## Not run:
indis  <- ind_to_land(inds=res_t, land=land1);

## End(Not run)
```

`make_agents` *Agent initialisation*

Description

Initialise the agents of the G-MSE model.

Usage

```
make_agents(model = "IBM", agent_number = 2, type_counts = c(1, 1),
  move = 0, vision = 20, rows = 100, cols = 100)
```

Arguments

model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)
agent_number	This is the number of agents that are set in the model; agent number does not change during the simulation, and each agent has a unique ID
type_counts	A vector of how many agents there are of each type (element). The sum of this vector needs to equal the agent_number so that each agent can correctly be assigned a type. Currently, GMSE assumes that there are only two types of agents: managers (type 0) and stakeholders (type 1), and only one manager exists. Future versions of GMSE will allow for different options as requested.
move	This parameter affects the movement of agents each time step. There are multiple types of movement (see obs_move_type in the gmse function), but this parameter determines the distance in cells that an agent will move. Agent movement is generally less important than resource movement, and typically does not affect how agents interact with resources
vision	This parameter determines the distance around an agent's location within which it can observe resources. This is relevant for some (but not not all) types of observation in the observation model, particularly for density-based estimation (observe_type = 0 in the gmse() function).
rows	The number of rows (y-axis) on the simulated landscape; agents are randomly placed somewhere on the landscape array
cols	The number of columns (x-axis) on the simulated landscape; agents are randomly placed somewhere on the landscape array

Value

the_agents Initialised data frame of agents being modelled

Examples

```
agents <- make_agents(model = "IBM", agent_number = 2, type_counts = c(1, 1),
move = 0, vision = 20, rows = 100, cols = 100);
```

---

make_costs *COST initialisation*

Description

Initialise the cost array of the G-MSE model.

Usage

```
make_costs(AGENTS, RESOURCES, res_opts, lnd_opts, min_cost)
```

Arguments

AGENTS	The array of agents produced in the main gmse() function
RESOURCES	The resources array produced by the resource function within GMSE
res_opts	A binary vector produced by the GMSE function defining what types of stakeholder interactions with resources (scaring, culling, castration, feeding, help_offspring) are permitted
lnd_opts	A binary vector produced by the GMSE function defining what types of stakeholder interactions with the landscape (tend_crops, kill_crops) are permitted
min_cost	The minimum cost that any agent (stakeholder or manager) incurrs for performing one action. This value is also set as an option in the main gmse() function (minimum_cost). This cost is recommended to be set to a value of 10, which gives managers increased precision when adjusting costs. For example, if the mimimum cost for a stakeholder performing an action is low, then a small change in the mimimum cost could halve or double the number of actions performed from the manager's perspective, with no options in between; hence the benefit of having a high mimimum cost combined with a higher agent budget (see the main gmse() function)

Value

A three dimensional array of initialised cost values for agent (manager and stakeholder) actions of the same dimensions as the ACTION array in GMSE

Examples

```
## Not run:
COST <- make_costs( AGENTS = AGENTS, RESOURCES = starting_resources,
res_opts = user_res_opts, lnd_opts = user_lnd_opts, min_cost = minimum_cost);

## End(Not run)
```

---

make_interaction_array

Initialise array of resource and landscape-level interactions.

Description

Initialise array of resource and landscape-level interactions.

Usage

```
make_interaction_array(RESOURCES, LAND)
```

Arguments

RESOURCES	The resources array produced by the resource function within GMSE
LAND	The landscape array on which interactions between resources and agents occur

Examples

```
## Not run:
Jacobian <- make_interaction_array(RESOURCES = starting_resources,
LAND = LANDSCAPE_r);

## End(Not run)
```

`make_interaction_table` *Initialise array of resource and landscape-level interactions.*

Description

Initialise array of resource and landscape-level interactions.

Usage

```
make_interaction_table(RESOURCES, LAND)
```

Arguments

RESOURCES	The resources array produced by the resource function within GMSE
LAND	The landscape array on which interactions between resources and agents occur

Examples

```
## Not run:
interaction_tabl <- make_interaction_table(starting_resources, LANDSCAPE_r);

## End(Not run)
```

`make_landscape` *Landscape initialisation*

Description

Initialise the landscape of the G-MSE model.

Usage

```
make_landscape(model, rows, cols, cell_types = 1, cell_val_mn = 1,
  cell_val_sd = 0, cell_val_max = 1, cell_val_min = 0, layers = 3,
  ownership = 0, owner_pr = NULL)
```

Arguments

model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)
rows	The dimension of the other side of the landscape (e.g., Longitude)
cols	The dimension of one side of the landscape (e.g., Latitude)
cell_types	Scalar or vector of all possible types of landscape cells
cell_val_mn	Mean cell value (e.g., defining crop output on a cell)
cell_val_sd	Standard devation of cell values on a landscape
cell_val_max	The maximum value of a cell
cell_val_min	The minimum value of a cell
layers	The number of layers in the 3D landscape (should usually be 3)
ownership	A scalar or vector of agent IDs that own land
owner_pr	The proportion of land owned by each agent with a unique ID. Note that 'owner_pr' must be of the same size as 'ownership', and the order of 'owner_pr' reflects the order of agent IDs to which owned land is being assigned

Value

the_land A cols by rows landscape with randomly distributed cell types

Examples

```
land <- make_landscape(model = "IBM", rows = 10, cols = 10, cell_types = 1,
cell_val_mn = 1, cell_val_sd =  0)
```

---

make_resource *Resource initialisation*

Description

Initialise the resources of the G-MSE model.

Usage

```
make_resource(model = "IBM", resource_quantity = 100,
  resource_types = 1, rows = 100, cols = 100, move = 1,
  rm_pr = 0, lambda = 0, consumption_rate = 0.1, max_age = 5)
```

Arguments

model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)
resource_quantity	The total number of resources being initialised (e.g., the population size of the resource at the first time step)
resource_types	The number of resource types that exist. Currently, only one resource type is recommended, but future versions of GMSE will include multiple resource types if requested
rows	The number of rows (y-axis) on the simulated landscape; resources are randomly placed somewhere on the landscape array
cols	The number of columns (x-axis) on the simulated landscape; resources are randomly placed somewhere on the landscape array
move	This parameter affects the movement of resources each time step. There are multiple types of movement (see res_move_type in the gmse function), but this parameter determines the distance in cells that a resource will move
rm_pr	This parameter sets the baseline probability of resource removal (death) per time step; this probability can be affected by user actions or carrying capacity, so a probability of zero does not ensure that resources will necessarily persist until the end of the simulation
lambda	This is the parameter for Poisson random sampling affecting birthrate; each resource gives birth to Poisson(lambda) offspring in the resource model
consumption_rate	Rate at which resource consumes crops on landscape; consumption affects the landscape by decreasing values on the landscape array (which may, e.g., be interpreted as crop production being decreased), and might also affect resource demographic parameters depending on other global options set in GMSE
max_age	Maximum age allowed for a resource to be (in time steps)

Value

the_resources Initialised data frame of resources being modelled

Examples

```
resource <- make_resource(model = "IBM", resource_quantity = 100,
resource_types = 1, rows = 100, cols = 100, move = 1, rm_pr = 0, lambda = 0,
consumption_rate = 0.5, max_age = 5);
```

make_utilities *Utility initialisation*

Description

Initialise the utilities of the G-MSE model.

Usage

```
make_utilities(AGENTS, RESOURCES)
```

Arguments

AGENTS	The array of agents produced in the main gmse() function
RESOURCES	The resources array produced by the resource function within GMSE

Value

A three dimensional ACTION array of initialised agent (manager and stakeholder) actions of the same dimensions as the COST array in GMSE

Examples

```
## Not run:
ACTION <- make_utilities(AGENTS = AGENTS, RESOURCES = starting_resources);

## End(Not run)
```

manager *Manager model*

Description

A model of manager decisions for a single time step. Managers set costs for user actions.

Usage

```
manager(RESOURCES = NULL, AGENTS = NULL, LAND = NULL, PARAS = NULL,
  COST = NULL, ACTION = NULL, INTERACT = NULL, inter_tabl = NULL,
  OBSERVATION = NULL, model = "IBM")
```

Arguments

RESOURCES	The resources array produced by the resource function within GMSE
AGENTS	The array of agents produced in the main gmse() function
LAND	The landscape array on which interactions between resources and agents occur
PARAS	The vector of parameters that hold global and dynamic parameter values used by GMSE
COST	A three dimensional array of cost values for agent (manager and stakeholder) actions
ACTION	A three dimensional array of agent (manager and stakeholder) actions
INTERACT	An interaction (Jacobian) matrix of resources & landscape layer effects
inter_tabl	Interaction table indexing types with the INTERACT matrix
OBSERVATION	The array of resource observations from the observation model, used to estimate abundance of resources
model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)
...	Other arguments to be passed to a user-defined model

Value

The manager function outputs an R list that includes five separate arrays, including (1) an new RESOURCES array, (2) a new AGENTS array, (3) a new LAND array, (4) a new ACTIONS array, and a new (5) COST array, each of which might be affected by the user function. The new arrays can then be read back into the broader GMSE function, thereby affecting the input into the user, resource, and observation models.

Examples

```
## Not run:
MANAGER_OUT <- run_manage(RESOURCE_c = RESOURCES, LANDSCAPE_c = LAND,
PARAMETERS_c = PARAS, AGENT_c = AGENTS, COST_c = COST, ACTION_c = ACTION,
JACOBIAN_c = INTERACT, INTERACT_c = inter_tabl, OBSERVATION_c = OBSERVATION);

## End(Not run)
```

---

observation *Observation model*

Description

A simulation of techniques (e.g., capture-mark-recapture) for estimating population size and properties.

Usage

```
observation(RESOURCES = NULL, LAND = NULL, PARAS = NULL,
  AGENTS = NULL, inter_tabl = NULL, fixed_mark = FALSE,
  times_observe = 1, res_min_age = 0, agent_type = 0, type_cat = 1,
  observe_type = 0, res_move_obs = FALSE, model = "IBM")
```

Arguments

RESOURCES	The resources array produced by the resource function within GMSE
LAND	The landscape array on which interactions between resources and agents occur
PARAS	The vector of parameters that hold global and dynamic parameter values used by GMSE
AGENTS	The array of agents produced in the main gmse() function
inter_tabl	Interaction table indexing types with the INTERACT matrix
fixed_mark	Fixed number of individuals marked? (A number, or FALSE)
times_observe	Number of times that the observations are made (e.g., managers go out sampling n times in an area of the landscape)
res_min_age	Minimum age of the resource being sampled (default = 1)
agent_type	The type of agent doing the observing (default = 0)
type_cat	The category of agent type (first 4 columns) doing observing; this will almost always be 1, so type 0 agents (managers, of which there is always one by default) will perform the observations
observe_type	The type of method used to do the observing. For types of observation exist: (1) Density based observation, where observers count all of the resources within a subset of the landscape (the manager function can then later estimate total resource number from this estimate). (2) Mark-recapture based observation, where observers tag a fixed number of randomly sampled resources on the landscape some number of 'times'; some of these resources marks are later interpreted as marks ('fixed_mark') while the rest are interpreted as recaptures. (3) Transect based observation, where observers sample a linear transect, observing all resources on the transect one row of landscape cells at a time, until all landscape cells are sampled; between samples, resources might move generating observation error. (4) Block based sampling, which is very similar to Transect based sampling; here observers instead sample square blocks of a landscape, counting resources one block at a time, until the whole landscape is sampled; between samples resources might move generating observation error.
res_move_obs	Defines whether or not resources move during observation (default = FALSE). Note that if this is FALSE, then observation methods (observe_type) 3 and 4 produce no observation error
model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)
...	Other arguments to be passed to a user-defined model

Value

The observation function outputs an R list that includes three separate arrays, including (1) an new OBSERVATION array that holds observed resources and their traits with additional columns indicating when the resources were observed (relevant, e.g., for mark-recapture), (2) a new AGENTS array, and (3) a new PARAS array, each of which might be affected by the user function. The new arrays can then be read back into the broader GMSE function, thereby affecting the input into the management, user, and resource models.

Examples

```
## Not run:
OBSERVATION_NEW <- observation(RESOURCES = RESOURCES, LAND = LANDSCAPE_r,
PARAS = paras, AGENTS = AGENTS, inter_tabl = interaction_tabl, fixed_mark = fxo,
times_observe = tmo, res_min_age = rma, agent_type = 0, type_cat   = 1, observe_type = obt,
res_move_obs   = rmo);

## End(Not run)
```

---

plot_gmse_effort *Plot the effort made by each user for each action*

Description

Produce a five panel plot in which each panel compares the permissiveness of each action (scaring, culling, etc.) from the manager with the effort put into each action by individual users.

Usage

```
plot_gmse_effort(sim_results)
```

Arguments

sim_results	Output from gmse to be plotted.

Value

This function plots the permissiveness that each manager exhibits for each user action (scaring, culling, etc.) and the effort that each individual user puts into each action over time. On the left axis, permissiveness is calculated as 100 minus the percent of the manager's budget put into increasing the cost of a particular action, so, e.g., if a manager puts all of their effort into increasing the cost of culling, then permissiveness of culling is 0; if they put none of their effort into increasing the cost of culling, then permissiveness of culling is 100. On the right axis, percentage of user action expended is the percent of a user's budget put into a particular action (note, these might not add up to 100 because users are not forced to use their entire budget). Coloured lines that are above black lines could potentially (cautiously) be interpreted as conflict between managers and users.

Examples

```
## Not run:
plot_gmse_effort(sim_results = sim);

## End(Not run)
```

`plot_gmse_results` *Plot the results of a gmse simulation*

Description

Produce six panels on a plot showing resource distribution, owned land, resource dynamics and estimates, stake-holder yield, and action costs and actions made.

Usage

```
plot_gmse_results(sim_results)
```

Arguments

`sim_results`	Output from gmse to be plotted.

Value

This function plots the dynamics of GMSE resource, observation, managemer, and user models in six separate sub-panels. (1) Upper left panel: Shows the locations of resources on the landscape (black dots); landscape terrain is also shown in brown, but at the moment, this is only cosmetic and does not reflect anything occurring in the model. (2) Upper right panel: Shows ownership of land by agents; land is divided proportional based on parameters set in gmse() and colours correspond with other subplots. If agent utilities and actions are restricted to land (‘land_ownership‘ in the gmse() function), then this gives some idea of where actions are being performed and where resources are affecting the landscape. (3) Middle left panel: Shows the actual population abundance (black solid line) and the population abundance estimated by the manager (blue solid line; shading indicates 95 percent confidence intervals) over time. The dotted red line shows the resource carrying capacity (death-based) and the dotted blue line shows the target for resource abundance as set in the gmse() function; the orange line shows the total percent yield of the landscape (i.e., 100 percent means that resources have not decreased yield at all, 0 percent means that resources have completely destroyed all yield). (4) Middle right panel: Shows the raw landscape yield for each stakeholder (can be ignored if ‘land_ownership‘ is FALSE) over time; colours correspond to land ownership shown in the upper right panel. (5) Lower left panel: The cost of stakeholders performing actions over time, as set by the manager. (6) Lower right panel: The total number of actions performed by all stakeholders over time.

Examples

```
## Not run:
plot_gmse_results(sim_results = sim);

## End(Not run)
```

rec.n *R data for recruitment used in SI4 vignette*

Description

This object contains SI4 vignette simulation output

Usage

```
rec.n
```

Format

a set of `gmse_apply` output examples

Author(s)

Brad Duthie

resource *Resource model*

Description

A population model of resource (including population) dynamics for a single time step.

Usage

```
resource(RESOURCES = NULL, LAND = NULL, PARAS = NULL,
  model = "IBM")
```

Arguments

`RESOURCES`	The resources array produced by the resource function within GMSE
`LAND`	The landscape array on which interactions between resources and agents occur
`PARAS`	The vector of parameters that hold global and dynamic parameter values used by GMSE
`model`	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)

Value

The resource function outputs an R list that includes three separate arrays, including (1) an new RESOURCES array, (2) a new LAND array, (3) a new PARAS array, each of which might be affected by the user function. The new arrays can then be read back into the broader GMSE function, thereby affecting the input into the observation, management, and user models.

Examples

```
## Not run:
RESOURCE_NEW <- resource(RESOURCES = RESOURCES, LAND = LANDSCAPE_r,
PARAS = paras, model = "IBM");

## End(Not run)
```

`ssb.n` *R data for spawning stock biomass used in SI4 vignette*

Description

This object contains SI4 vignette simulation output

Usage

```
ssb.n
```

Format

a set of `gmse_apply` output examples

Author(s)

Brad Duthie

`user` *User model*

Description

A model of user decisions for a single time step. These decisions result in stakeholder actions that can potentially affect resources and the landscape in a GMSE simulation.

Usage

```
user(RESOURCES = NULL, AGENTS = NULL, LAND = NULL, PARAS = NULL,
  COST = NULL, ACTION = NULL, INTERACT = NULL, inter_tabl = NULL,
  model = "IBM")
```

Arguments

RESOURCES	The resources array produced by the resource function within GMSE
AGENTS	The array of agents produced in the main gmse() function
LAND	The landscape array on which interactions between resources and agents occur
PARAS	The vector of parameters that hold global and dynamic parameter values used by GMSE
COST	A three dimensional array of cost values for agent (manager and stakeholder) actions
ACTION	ACTION A three dimensional array of agent (manager and stakeholder) actions
INTERACT	An interaction (Jacobian) matrix of resources & landscape layer effects
inter_tabl	Interaction table indexing types with the INTERACT matrix
model	The type of model being applied (Currently only individual-based – i.e., 'agent-based' – models are allowed)
...	Other arguments to be passed to a user-defined model

Value

The user function outputs an R list that includes five separate arrays, including (1) an new RESOURCES array, (2) a new AGENTS array, (3) a new LAND array, (4) a new ACTIONS array, and a new (5) COST array, each of which might be affected by the user function. The new arrays can then be read back into the broader GMSE function, thereby affecting the input into the resource, observation, and management models.

Examples

```
## Not run:
USERS <- user(RESOURCES = RESOURCES, AGENTS = AGENTS, LAND = LANDSCAPE_r,
PARAS = paras, COST = COST, ACTION = ACTION, INTERACT = Jacobian,
inter_tabl = interaction_tabl, model = "IBM");

## End(Not run)
```

---

utility_layer *Utility layer for initialisation.*

Description

Function to initialise a layer of the UTILITY array of the G-MSE model.

Usage

```
utility_layer(agent_IDs, agent_number, res_types)
```

Arguments

agent_IDs	Vector of agent IDs to use (including -1 and -2, which indicate direct actions to the landscape and resources, respectively)
agent_number	The number of agents to use (length of agent_IDs)
res_types	The number of unique resource types (cols 2-4 of RESOURCES); for now, this should always be 1

Value

A layer of the COST or ACTION array, as called in building either make_costs or make_utilities, respectively. This layer corresponds to the costs or actions of a single agent, with the larger array in in which it is placed including all agents

Examples

```
## Not run:
UTIL_LIST <- utility_layer(agent_IDs, agent_number, res_types);

## End(Not run)
```

Index