Default GMSE data structures

GMSE: an R package for generalised management strategy evaluation (Supporting Information 7)

A. Bradley Duthie[13], Jeremy J. Cusack[1], Isabel L. Jones[1], Jeroen Minderman[1], Erlend B. Nilsen[2], Rocío A. Pozo[1], O. Sarobidy Rakotonarivo[1], Bram Van Moorter[2], and Nils Bunnefeld[1]

[1] Biological and Environmental Sciences, University of Stirling, Stirling, UK [2] Norwegian Institute for Nature Research, Trondheim, Norway [3] alexander.duthie@stir.ac.uk

GMSE arguments and output

Simulations using the default GMSE sub-models described above are run using the `gmse` function, which offers a range of options for setting parameter values (see Table 1 for some select examples). Output of `gmse` is an exhaustive list that includes all resources and observations, all stakeholder decisions and actions, and all landscape properties in each time step of the simulation. Results are most easily interpreted visually, so a summary of simulation dynamics is plotted by default (the plot can also be called using the `plot_gmse_results` function, and summaries of results can be obtained using `gmse_summary` and `gmse_table`). An example below shows how simulations are set and interpreted.

Argument	Default	Description
time_max	100	Maximum time steps in simulation
land_dim_1	100	Width of the landscape (horizontal cells)
land_dim_2	100	Height of the landscape (vertical cells)
res_movement	20	Distance (cells) a resource can move in any direction (for movement rules, see res_move_type)
remove_pr	0	Density-independent probability of resource mortality during a time step
lambda	0.3	Poisson rate parameter for resource offspring number produced during a time step
agent_view	10	How far managers can see on the landscape for resource counting when observe_type = 0
res_birth_K	100000	Carrying capacity applied to the number of resources added during a time step
res_death_K	2000	Carrying capacity applied to the number of resources removed during a time step
res_move_type	1	Type of resource movement (default is up to res_movement cells in any direction)
observe_type	0	Type of resource observation (default is density-based; i.e, counting a subset on the landscape)
fixed_mark	50	For mark-recapture observation (observe_type = 1), number of marked resources
fixed_recapt	150	For mark-recapture observation (observe_type = 1), number of recaptured resources
times_observe	1	For density-based observation (observe_type = 0), landscape subsets viewed during observation
res_consume	0.5	Pr. of a landscape cell's value reduced by the presence of a resource in a time step
max_ages	5	The maximum number of time steps a resource can persist before it is removed
minimum_cost	10	The minimum cost of a user performing any action
user_budget	1000	A user's budget per time step for performing any number of actions
manager_budget	1000	A manager's budget per time step for setting policy
manage_target	1000	The manager's target resource abundance
RESOURCE_ini	1000	The initial abundance of resources
scaring	FALSE	Resource scaring (moves a resource to a random landscape cell) is a policy option
culling	TRUE	Resource culling (removes a resource entirely) is a policy option
castration	FALSE	Resource castration (sets a resource's lambda to zero) is a policy option
feeding	FALSE	Resource feeding (increases a resource's lambda) is a policy option
help_offspring	FALSE	Resource helping (increases a resource's offspring number) is a policy option

Argument	Default	Description
tend_crops	FALSE	Users can increase landscape cell values
tend_crop_yld	0.2	Proportional increase per landscape cell from tend_crops action
kill_crops	FALSE	Users can decrease landscape cell values to zero
stakeholders	4	Number of users in the simulation
land_ownership	FALSE	Users own land and increase utility indirectly from landscape instead of resource use
manage_freq	1	Frequency (in time steps) with which managers revise and enact policy
public_land	0	Proportion of land that is public (un-owned by users) if land_ownership = TRUE

Table 1: Select parameter values for initialising generalised management strategy evaluation simulations

The most important (default) GMSE data structures

The default sub-models of GMSE (`resource`, `observation`, `manager`, and `user`) use a small number of default data structures to hold the information needed in simulations. While these default sub-models do not necessarily need to be used in every run in GMSE (see use of `gmse_apply`), they will be used in any run of the `gmse` function, and in any call of the `gmse_apply` function that does not run with entirely custom sub-models. Simulation and model inference do not require an understanding of the default data structures, but such an understanding can be especially useful when running `gmse_apply` if there is a need to extract uncommonly used information, change key simulated values (e.g., landscape properties, agent budgets, or resource movement rules, as in SI4), or build custom individual-based sub-models. Here we provide a brief explanation of the following key data structures (each name below is listed as it is named in the output `gmse_apply` when `get_res = "Full"`).

1. `AGENTS`
2. `resource_array` (or `RESOURCES`)
3. `observation_array` (or `OBSERVATION`)
4. `manager_array` (or `COST`)
5. `user_array` (or `ACTION`)
6. `LAND`

Note that these are not the only data structures used in GMSE, but they are the only ones that can be easily modified in GMSE v0.4.0.3 (see, e.g., SI4), so they are the ones that we focus on here. Additionally, any custom subfunction that returns an array rather than a single value should adhere to the same structure as these defaults if any default GMSE functions are to be used in `gmse_apply`. We can investigate each data structure by running a single simulation of `gmse_apply`.

```
sim <- gmse_apply(get_res = "Full");
```

The full list output of `sim` holds each structure by name (in the case where two names are used, e.g., `resource_array` and `RESOURCES`, both are identical, but the lower case `resource_array` takes precedence in case of a change). Each data structure can be examined, changed, and incorporated into a new simulation (e.g., `new_sim <- gmse_apply(old_list = sim)`).

1. AGENTS

The `AGENTS` data structure is a two dimensional array with a fixed number of 17 columns and a number of rows that is always equal to the sum of the number of manager and users (i.e., each row is an individual agent).

```
print(sim$AGENTS);
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    0    0    0   69   93   50    0   10     0    49     0     0
## [2,]    2    1    0    0   86   91   50    0   10     0     0     0     0
## [3,]    3    1    0    0   65   39   50    0   10     0     0     0     0
## [4,]    4    1    0    0   26   48   50    0   10     0     0     0     0
## [5,]    5    1    0    0   61   23   50    0   10     0     0     0     0
##      [,14] [,15]    [,16] [,17]
## [1,]     0     0 9541.156  1000
## [2,]     0     0    0.000  1000
## [3,]     0     0    0.000  1000
## [4,]     0     0    0.000  1000
## [5,]     0     0    0.000  1000
```

In the default case above, there are five agents (one manager and four users), each represented by a unique row. Columns in the array represent the agent traits listed below.

1. ID (each agent gets a unique number)
2. Type 1 (0 indicates the manager; 1 indicates users)
3. Type 2 (currently unused)
4. Type 3 (currently unused)
5. x-location on the landscape (typically ignored)
6. y-location on the landscape (typically ignored)
7. Movement distance (typically ignored)
8. Time parameter (typically ignored)
9. Distance of vision (currently used only for managers)
10. Error parameter (currently unused)
11. Resource marking parameter (currently used only for managers)
12. Resource tally parameter (currently used only for managers)
13. Unused column 1
14. Unused column 2
15. Unused column 3
16. Yield from owned land (zero for users when default `land_ownership = FALSE`)
17. Budget

It is obvious from the above list that most columns represent traits that are either typically ignored or currently not in use. This is intended to allow for easier future development of default model options and potential customisation of sub-models in `gmse_apply`. We anticipate that future versions of GMSE will contain multiple user types with unique traits and among-user interactions.

2. resource_array

The `resource_array` (also accessible as `RESOURCES`) is a two dimensional array with a fixed number of 20 columns and a number of rows that is always equal to the total number of resources (each row is an individual resource). In the above simulation, `sim$resource_array` includes 1067 rows, so we only print out the first eight for illustration.

```
print(sim$resource_array[1:8,]);
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    1    0    0    3   57   20    1    0   0.3     0     5     0
## [2,]    2    1    0    0   92   84   20    1    0   0.3     0     3     0
## [3,]    3    1    0    0   33   59   20    1    0   0.3     0     3     0
## [4,]    4    1    0    0   81   25   20    1    0   0.3     0     4     0
## [5,]    5    1    0    0    7   27   20    1    0   0.3     0     2     0
## [6,]    6    1    0    0   18   61   20    1    0   0.3     0     4     0
```

```
## [7,]    8    1    0    0   93   38   20    1    0   0.3     0     3     0
## [8,]    9    1    0    0   95   41   20    1    0   0.3     0     3     0
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     0   0.5     0     0     0     0     0
## [2,]     0   0.5     0     0     0     0     0
## [3,]     0   0.5     0     0     0     0     0
## [4,]     0   0.5     0     0     0     0     0
## [5,]     0   0.5     0     0     0     0     0
## [6,]     0   0.5     0     0     0     0     0
## [7,]     0   0.5     0     0     0     0     0
## [8,]     0   0.5     0     0     0     0     0
```

Columns in the resource array represent the individual resource traits listed below.

1. ID (each resource gets a unique number)
2. Type 1 (currently all resources are of type 1)
3. Type 2 (currently unused)
4. Type 3 (currently unused)
5. x-location on the landscape
6. y-location on the landscape
7. Movement distance
8. Time parameter (typically ignored)
9. Density-independent removal (i.e., death) probability
10. Growth (i.e., birth) probability
11. Offspring produced
12. Age (initial resources are given a random age between 1 and the maximum age sampled from a uniform distribution; offspring always start at age zero in their time step of birth)
13. Marking indicator (used in the observation function)
14. Tallying indicator (used in the observation function)
15. Proportion of a landscape cell the resource consumes in a time step
16. Has the resource been scared by an agent?
17. Has the resource been culled by an agent?
18. Has the resource been castrated by an agent?
19. Has the resource's growth rate been increased by an agent?
20. Has the resource's offspring production been increased by an agent?

In the case of columns 16-20, the value is either zero (if no action has occurred), or some positive integer that matches the ID of the agent that has performed the act (e.g., if column 17 equals 3, then that means that the agent with ID = 3 culled the resource in the corresponding row; where more than one agent's action is possible per time step – as in scaring – the integer reflects the most recently acting agent). We anticipate that future versions of gmse will contain multiple resource types, and might add columns to include additional resource traits.

3. observation_array

The `observation_array` (also accessible as `OBSERVATION`) is a two dimensional array, the number of rows and columns of which depend on the type of observation being made (i.e., `observe_type`, which can take integer values from 0-3; see the GMSE reference manual for more information about built-in observation types that are available in GMSE). The first 20 columns of `observation_array` contain the same individual resource traits as in `resource_array`, while any additional columns provide information about how and when a resource was observed. The number of rows in `observation_array` is always equal to or less than that of `resource_array`; each resource that is observed at least once is placed into one unique row, while unobserved resources are not included as rows in the `observation_array`. In `sim`, there are 49 rows, meaning that 1018

resources were not observed at all in this time step. Below, we print out the first eight rows of the observation array.

```
print(sim$observation_array[1:8,]);
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]   57    1    0    0   76    1   20    1    0   0.3     1     3     1
## [2,]   70    1    0    0   73    2   20    1    0   0.3     0     4     1
## [3,]  105    1    0    0   71   86   20    1    0   0.3     0     3     1
## [4,]  159    1    0    0   78   83   20    1    0   0.3     2     3     1
## [5,]  162    1    0    0   75   83   20    1    0   0.3     0     2     1
## [6,]  203    1    0    0   79   97   20    1    0   0.3     1     5     1
## [7,]  207    1    0    0   71   88   20    1    0   0.3     0     5     1
## [8,]  231    1    0    0   73   87   20    1    0   0.3     1     2     1
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]
## [1,]     0   0.5     0     0     0     0     0     0     1
## [2,]     0   0.5     0     0     0     0     0     0     1
## [3,]     0   0.5     0     0     0     0     0     0     1
## [4,]     0   0.5     0     0     0     0     0     0     1
## [5,]     0   0.5     0     0     0     0     0     0     1
## [6,]     0   0.5     0     0     0     0     0     0     1
## [7,]     0   0.5     0     0     0     0     0     0     1
## [8,]     0   0.5     0     0     0     0     0     0     1
```

In the case of the default parameters, the observation array has only two additional columns; the first added column 21 is currently unused, and all values in this column are zero. The second added column 22 contains a value of 1 confirming that the resource was observed. Additional options will add different numbers of columns with different values. For example, when `observe_type = 0` (managers observe all resources on a random subset of the landscape, the size of which is determined by their distance of vision) but `times_observe > 1`, managers sample more than one random subset of the landscape. A new column is added for each sampled subset, and a 1 is placed in the relevant column if the resource is observed (these collected data are then used to estimate population size). An example where `times_observe = 4` is shown below.

```
sim_t0_4 <- gmse_apply(get_res = "Full", times_observe = 4);
print(sim_t0_4$observation_array[1:8,]);
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]   16    1    0    0   93   17   20    1    0   0.3     1     2     1
## [2,]   17    1    0    0   64   12   20    1    0   0.3     0     4     1
## [3,]   20    1    0    0   76   18   20    1    0   0.3     0     3     2
## [4,]   36    1    0    0   86    2   20    1    0   0.3     0     5     1
## [5,]   43    1    0    0    3   13   20    1    0   0.3     0     3     1
## [6,]   48    1    0    0   94   13   20    1    0   0.3     2     2     1
## [7,]   57    1    0    0   75    9   20    1    0   0.3     1     4     2
## [8,]   66    1    0    0   67   12   20    1    0   0.3     0     5     1
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]     0   0.5     0     0     0     0     0     0     0     0     1
## [2,]     0   0.5     0     0     0     0     0     0     1     0     0
## [3,]     0   0.5     0     0     0     0     0     0     1     1     0
## [4,]     0   0.5     0     0     0     0     0     0     0     1     0
## [5,]     0   0.5     0     0     0     0     0     0     0     0     1
## [6,]     0   0.5     0     0     0     0     0     0     0     0     1
## [7,]     0   0.5     0     0     0     0     0     0     1     1     0
## [8,]     0   0.5     0     0     0     0     0     0     1     0     0
##      [,25]
## [1,]     0
```

```
## [2,]    0
## [3,]    0
## [4,]    0
## [5,]    0
## [6,]    0
## [7,]    0
## [8,]    0
```

This process simulates the data collection of resources (and potentially resource trait measurement) as might be performed by observers within the system. It therefore takes a virtual ecologist approach; this enables the integration of theory and empirical work and can improve the mechanistic understanding of social-ecological systems (Zurell et al., 2010).

4. manager_array

For context, it might be easier to understand `manager_array` after reading about `user_array` below. The `manager_array` (also accessible as `COST`) is a three dimensional array, each layer of which corresponds to a unique agent (rows in `AGENT` correpond to layers in `manager_array`). Hence, in the simulation output `sim$manager_array`, there are 5 layers. Each layer in `manager_array` has 13 columns, and a number of rows that varies depending on the number of agents and resource types. As of GMSE v0.4.0.3, only the first three rows are used. Two layers of `sim$manager_array` are shown below, the first being that of the manager and the second being that of the first user.

```
print(sim$manager_array[,,1:2]);
```

```
## , , 1
##
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
## [1,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [2,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [3,] 100001 100001 100001 100001 100001 100001 100001 100001     10 100001
## [4,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [5,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [6,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [7,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
##       [,11]  [,12]  [,13]
## [1,] 100001 100001     10
## [2,] 100001 100001     10
## [3,] 100001 100001     10
## [4,] 100001 100001 100001
## [5,] 100001 100001 100001
## [6,] 100001 100001 100001
## [7,] 100001 100001 100001
##
## , , 2
##
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
## [1,] 100001 100001 100001 100001 100001 100001 100001 100001     69 100001
## [2,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [3,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [4,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [5,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [6,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
## [7,] 100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
```

```
##       [,11]  [,12]  [,13]
## [1,] 100001 100001     10
## [2,] 100001 100001     10
## [3,] 100001 100001 100001
## [4,] 100001 100001 100001
## [5,] 100001 100001 100001
## [6,] 100001 100001 100001
## [7,] 100001 100001 100001
```

Each element in the array indicates the cost of performing a particular action. In the code, this is the cost of changing an element in `user_array` (which has the same dimensions as `manager_array`). The minimum value in `sim$manager_array` is therefore 10, reflecting the default `minimum_cost` value of 10. The maximum value is 100001, which is one higher than the maximum allowed manager or user budget. Where a cost is 100001, actions can therefore never be performed. An explanation of the rows and columns of `manager_array` is provided below in the description of `user_array`.

5. user_array

The `user_array` (also accessible as `ACTION`) is a three dimensional array, each layer of which corresponds to a unique agent. When considering the three dimensional `user_array`, it is helpful to keep in mind that each layer corresponds to the actions of a particular agent, that each column corresponds to a particular type of action, and that each row corresponds to a particular resource, agent, or group that the action will affect. The cost of performing any action in this array is held in `manager_array`, wherein an action's cost in `manager_array` is held in the same array element as the action itself in `user_array`. Recall from the manager array that the first layer of `user_array` corresponds to the manager actions, and that remaining layers correspond to user actions; there are therefore as many layers in `user_array` as there are agents in the model, and each row of `AGENTS` corresponds to equivalent layer of `user_array` (e.g., the manager agent, ID = 1, is in the first row of `AGENTS` and the first layer of `user_array`). The first two layers of `user_array` are shown below.

```
print(sim$user_array[,,1:2]);
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]      [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## [1,]   -2    1    0    0 1000.0000    0    0    0    0     0     0     0
## [2,]   -1    1    0    0    0.0000    0    0    0    0     0     0     0
## [3,]    1    1    0    0 -111.1111    0    0   10   69    10    10    10
## [4,]    2    1    0    0    0.0000    0    0    0    0     0     0     0
## [5,]    3    1    0    0    0.0000    0    0    0    0     0     0     0
## [6,]    4    1    0    0    0.0000    0    0    0    0     0     0     0
## [7,]    5    1    0    0    0.0000    0    0    0    0     0     0     0
##      [,13]
## [1,]     0
## [2,]     0
## [3,]    51
## [4,]     0
## [5,]     0
## [6,]     0
## [7,]     0
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
```

```
## [1,]   -2    1    0    0   -1    0    0    0   14    0    0    0    1
## [2,]   -1    1    0    0    0    0    0    0    0    0    0    0    1
## [3,]    1    1    0    0    0    0    0    0    0    0    0    0    0
## [4,]    2    1    0    0    0    0    0    0    0    0    0    0    0
## [5,]    3    1    0    0    0    0    0    0    0    0    0    0    0
## [6,]    4    1    0    0    0    0    0    0    0    0    0    0    0
## [7,]    5    1    0    0    0    0    0    0    0    0    0    0    0
```

Note that there are more columns in this array than there are possible actions in GMSE. This is because there are several columns that do not map to actions per se, but properties of agents. As of GMSE v0.4.0.3, these properties cannot be changed by other agents. Column of `user_array` are as follows.

1. The type of agent or resource being affected by an action. A value of -2 indicates that actions have a direct effect on a resource (e.g., scaring, culling, etc.). A value of -1 indicates that actions have a direct effect on a landscape layer. Positive integer values indicate actions that affect other agents, where each integer corresponds to the agents' IDs. Where the integer value is identical with the agent's own ID (e.g., row 3 in layer 1 where the element `sim$user_array[3, 1, 1] = 1`), actions affect all other agents in the model. As of GMSE v0.4.0.3, all rows except 1-3 are unused because agents do not affect one anothers actions individually; they either affect all other agents' actions indiscriminately (in the case of the manager setting policy) or do not (directly) affect other agents' actions at all (in the case of users). This data structure, however, is designed so that future versions of GMSE will allow users to affect one another directly (representing, e.g., different groups of agents lobbying for different interests, among-user conflict, etc.).
2. Type 1 of the agent or resource of interest (in practice, this is currently unused).
3. Type 2 of the agent or resource of interest (currently unused).
4. Type 3 of the agent or resource of interest (currently unused).
5. Utility associated with the recipient of the action. For example, in the case of the resource (row 1), positive values indicate that the agent wants more of these resources, while negative values indicate that the agent wants fewer. In the case of the manager (layer 1), the value in the first row equals `manage_target`, while the value in the third row is the change in resource number needed to achieve the target value (i.e., `manage_target =` 1000, and the manager's estimate is `sim$observation_vector =` 1111.1111111. The former minus the latter is -111.11111).
6. Whether or not the utility associated with the recipient of the action is dependent upon that recipient being on land owned by the actor (e.g., if users only care about resources on landscape cells that they own, then this value is 1 instead of 0).
7. Whether or not actions on the recipient are possible if the recipient is not on land owned by the actor (e.g., if users cannot cull resources that are not on their own land, then this value is 1 instead of 0).
8. The number of actions performed for scaring, which in row 3 of the manager layer 1 is interpreted as the scaring cost set by the manager for users.
9. The number of actions performed for culling, which in row 3 of the manager layer 1 is interpreted as the culling cost set by the manager for users.
10. The number of actions performed for castration, which in row 3 of the manager's layer 1 is interpreted as the castration cost set by the manager for users. Further, in row 2 for users (where column 1 equals -1), this value is instead the number of `tend_crop` actions (the number of cells on which crops are tended by users, which always is perfomed on users' own land, cannot be affected by the manager, and always equals `minimum_cost`).
11. The number of actions performed for feeding resources (increasing their growth rate, `lambda`), which in row 3 of the manager's layer 1 is interpreted as the feeding cost set by the manager for users. Further, in row 2 for users (where column 1 equals -1), this value is instead the number of `kill_crop` actions (the number of cells on which crops are destroyed by users, which always is perfomed on users' own land, cannot be affected by the manager, and always equals `minimum_cost`)
12. The number of actions performed for helping resource offspring (directly increasing offspring production), which in row 3 of the manager's layer 1 is interpreted as the helping offspring cost set by the manager for users.
13. The number of actions unspent by the user or manager; any actions allocated to this row do nothing.

These may be used when any action would lead the agent to a less than desirable outcome, such as if only culling exists as a policy option (default), but managers do not want to increase the cost of culling because resource density is above `manage_target`.

In the genetic algorithm, values in elements of a `user_array` layer are potentially modified according to each agent's objective, as constrained by costs in `manager_array`.

6. LAND

Events in default GMSE sub-models occur on a spatially-explicit landscape `LAND`, which is stored as a three dimensional array. The size of this landscape is specified with the `land_dim_1` and `land_dim_2` arguments of GMSE, which determine the length, in cells, of the y and x dimensions of the landscape, respectively (e.g., if `land_dim_1 = 10` and `land_dim_2 = 1000`, then the landscape will be one very long horizontal transect). The total number of landscape cells on which resources and agents can interact is therefore the product of `land_dim_1` and `land_dim_2`. In addition, all landscapes have three layers, which hold three separate values of information for each x-y location. The first layer is unused in GMSE v0.4.0.3; the second layer holds crop production on a cell, and the third layer holds the owner of the cell (corresponding to the ID of an agent, where the manager's ID = 0 defines public land). An 8×8 portion of the landscape from `sim` is shown below.

```
print(sim$LAND[1:8,1:8,]);
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    1    1    1    1    1    1    1
## [2,]    1    1    1    1    1    1    1    1
## [3,]    1    1    1    1    1    1    1    1
## [4,]    1    1    1    1    1    1    1    1
## [5,]    1    1    1    1    1    1    1    1
## [6,]    1    1    1    1    1    1    1    1
## [7,]    1    1    1    1    1    1    1    1
## [8,]    1    1    1    1    1    1    1    1
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1  1.0  1.0    1    1  1.0    1    1
## [2,]    1  1.0  1.0    1    1  1.0    1    1
## [3,]    1  0.5  1.0    1    1  1.0    1    1
## [4,]    1  1.0  1.0    1    1  1.0    1    1
## [5,]    1  1.0  0.5    1    1  0.5    1    1
## [6,]    1  1.0  1.0    1    1  0.5    1    1
## [7,]    1  1.0  1.0    1    1  1.0    1    1
## [8,]    1  1.0  1.0    1    1  1.0    1    1
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    1    1    1    1    1    1    1
## [2,]    1    1    1    1    1    1    1    1
## [3,]    1    1    1    1    1    1    1    1
## [4,]    1    1    1    1    1    1    1    1
## [5,]    1    1    1    1    1    1    1    1
## [6,]    1    1    1    1    1    1    1    1
```

```
## [7,]    1    1    1    1    1    1    1    1
## [8,]    1    1    1    1    1    1    1    1
```

In the case of the above, all of the cells in this square patch of landscape are owned by agent 1 (i.e., the manager; see `sim$LAND[„3]`), and we can see that crop production on this patch of land has been decreased from 1 in several cells as a consequence of consumption by resources (see `sim$LAND[„2]`). In SI4, we show how landscape cell values can be manipulated to customise the placement of land ownership.

Conclusions

We have focused on the data structures `AGENTS`, `resource_array`, `observation_array`, `manager_array`, `user_array`, and `LAND` because these are the data structures that can be most readily manipulated to customise GMSE simulations. An example of how to do this within a loop using `gmse_apply` can be found in SI4. While other data structures exist within GMSE (e.g., see the output of `gmse_apply` when `get_res = Full`), we do not recommend manipulating these structures for custom simulations.

Many data structures contain elements that are unused in GMSE v0.4.0.3, and in all cases this is designed for ease of ongoing development of new GMSE features. Requests for new features can be made on GitHub using the GMSE Wiki or the GMSE Issues page.

References

Zurell, D., Berger, U., Cabral, J. S., Jeltsch, F., Meynard, C. N., Münkemüller, T., Nehrbass, N., Pagel, J., Reineking, B., Schröder, B., and Grimm, V. (2010). The virtual ecologist approach: Simulating data and observers. *Oikos*, 119(4):622–635.