Package 'villager'

October 12, 2022

Title A Framework for Designing and Running Agent Based Models

Version 1.1.1

Description This is a package for creating and running Agent Based Models (ABM). It provides a set of base classes with core functionality to allow bootstrapped models. For more intensive modeling, the supplied classes can be extended to fit researcher needs.

License MIT + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

Imports readr, R6, uuid

Suggests covr, dplyr, knitr, leaflet, plotly, remotes, rmarkdown, testthat,

URL https://github.com/zizroc/villager/

BugReports https://github.com/zizroc/villager/issues/

VignetteBuilder knitr

NeedsCompilation no

Author Thomas Thelen [aut, cre], Gerardo Aldana [aut], Marcus Thomson [aut], Toni Gonzalez [aut]

Maintainer Thomas Thelen <thelen@nceas.ucsb.edu>

Repository CRAN

Date/Publication 2022-04-15 10:20:02 UTC

R topics documented:

data_writer 2
model_data 3

resource 4
resource_manager 5
simulation 7
village 8
village_state 10
winik 11
winik_manager 14

Index **18**

`data_writer` *Data Writer*

Description

A class responsible for the simulation data to disk.

Details

This class can be subclasses to provide advanced data writing to other data sources. This should also be subclassed if the winik and resource classes are subclasses, to write any addional fields to the data source.

Methods

`write()` Writes the winik and resources to disk.

Create a new data writer.

Public fields

`results_directory` The folder where the simulation resulst are written to

`winik_filename` The location where the winiks are written to

`resource_filename` The location where the resources are written to

Methods

Public methods:

- `data_writer$new()`
- `data_writer$write()`
- `data_writer$clone()`

Method `new()`**:** Creates a new data writer object that has optional paths for data files.

Usage:

```
data_writer$new(
  results_directory = "results",
  winik_filename = "winiks.csv",
  resource_filename = "resources.csv"
)
```

Arguments:

`results_directory` The directory where the results file is written to

`winik_filename` The name of the file for the winik data

`resource_filename` The name of the file for the resource data

Returns: A new winik object Writes a village's state to disk.

Method `write()`**:** Takes a state an the name of a village and writes the winiks and resources to disk

Usage:

```
data_writer$write(state, village_name)
```

Arguments:

`state` The village's village_state that's being written

`village_name` The name of the village. This is used to create the data directory

Returns: None

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
data_writer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`model_data` *R6 Class representing data that's external from resources and winiks*

Description

R6 Class representing data that's external from resources and winiks

R6 Class representing data that's external from resources and winiks

Details

It contains a single variable, 'events' for when the data holds a list of events

Public fields

`events` Any events that need to be tracked

Methods

Public methods:

- `model_data$new()`
- `model_data$clone()`

Method `new()`**:** Creates a new model_data object

Usage:

```
model_data$new()
```

Returns: A new model data object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
model_data$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`resource` *resource*

Description

This is an object that represents a single resource.

Methods

`initialize()` Create a new resource

`as_table()` Represents the current state of the resource as a tibble

Creates a new resource.

Public fields

`name` The name of the resource

`quantity` The quantity of the resource that exists

Methods

Public methods:

- `resource$new()`
- `resource$as_table()`
- `resource$clone()`

Method `new()`**:** Creates a new resource object

Usage:

```
resource$new(name = NA, quantity = 0)
```

Arguments:

`name` The name of the resource

`quantity` The quantity present Returns a data.frame representation of the resource

Method `as_table()`**:**

Usage:

```
resource$as_table()
```

Returns: A data.frame of resources

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
resource$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`resource_manager` *Resource Manager*

Description

This object manages all of the resources in a village.

Methods

`initialize()` Creates a new manager

`get_resources()` Gets all of the resources that the manager has

`get_resource()` Retrieves a resource from the manager

`add_resource()` Adds a resource to the manager

`remove_resource()` Removes a resource from the manager

`get_resource_index()` Retrieves the index of the resource

`get_states()` Returns a list of states

`load()` Loads a csv file of resources and adds them to the manager.

Public fields

`resources` A list of resource objects

`resource_class` The class used to represent resources Creates a new , empty, resource manager for a village.

Methods

Public methods:

- resource_manager$new()
- resource_manager$get_resources()
- resource_manager$get_resource()
- resource_manager$add_resource()
- resource_manager$remove_resource()
- resource_manager$get_resource_index()
- resource_manager$get_states()
- resource_manager$load()
- resource_manager$clone()

Method `new()`**:** Get a new instance of a resource_manager

Usage:

```
resource_manager$new(resource_class = villager::resource)
```

Arguments:

`resource_class` The class being used to describe the resources being managed Gets all of the managed resources

Method `get_resources()`**:**

Usage:

```
resource_manager$get_resources()
```

Returns: A list of resources Gets a resource given a resource name

Method `get_resource()`**:**

Usage:

```
resource_manager$get_resource(name)
```

Arguments:

`name` The name of the requested resource

Returns: A resource object Adds a resource to the manager.

Method `add_resource()`**:**

Usage:

```
resource_manager$add_resource(new_resource)
```

Arguments:

`new_resource` The resource to add

Returns: None Removes a resource from the manager

Method `remove_resource()`**:**

Usage:

```
resource_manager$remove_resource(name)
```

Arguments:

`name` The name of the resource being removed

Returns: None Returns the index of a resource in the internal resource list

Method `get_resource_index()`**:**

Usage:

```
resource_manager$get_resource_index(name)
```

Arguments:

`name` The name of the resource being located

Returns: The index in the list, or R's default return value Returns a data.frame where each row is a resource.

Method `get_states()`**:**

Usage:

```
resource_manager$get_states()
```

Details: Subclasses should not have to override this method because it takes all member variables into account

Returns: A single data.frame Loads a csv file of resources into the manager

Method `load()`**:**

Usage:

```
resource_manager$load(file_name)
```

Arguments:

`file_name` The path to the csv file

Returns: None

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
resource_manager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`simulation` *simulation*

Description

Advances one or more villages through time

Methods

`run_model()` Runs the simulation

Creates a new Simulation instance

Public fields

`length` The total number of time steps that the simulation runs for

`villages` A list of villages that the simulator will run

`writer` An instance of a data_writer class for writing village data to disk

Methods

Public methods:

- `simulation$new()`
- `simulation$run_model()`
- `simulation$clone()`

Method `new()`**:** Creates a new simulation object to control the experiment

Usage:

```
simulation$new(length, villages, writer = villager::data_writer$new())
```

Arguments:

`length` The number of steps the simulation takes

`villages` A list of villages that will be simulated

`writer` The data writer to be used with the villages Runs the simulation

Method `run_model()`**:**

Usage:

```
simulation$run_model()
```

Returns: None

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
simulation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`village` *Village*

Description

This is an object that represents the state of a village at a particular time.

Details

This class acts as a type of record that holds the values of the different village variables. This class can be subclassed to include more variables that aren't present.

Methods

`initialize()` Creates a new village

`propagate()` Advances the village a single time step

`set_initial_state()` Initializes the initial state of the village

Public fields

`name` An optional name for the village

`initial_condition` A function that sets the initial state of the village

`current_state` The village's current state

`previous_state` The village's previous state

`models` A list of functions or a single function that should be run at each timestep

`model_data` Optional data that models may need

`winik_mgr` The manager that handles all of the winiks

`resource_mgr` The manager that handles all of the resources Initializes a village

Methods

Public methods:

- `village$new()`
- `village$propagate()`
- `village$set_initial_state()`
- `village$clone()`

Method `new()`**:** This method is meant to set the variables that are needed for a village to propagate through time.

Usage:

```
village$new(
  name,
  initial_condition,
  models = list(),
  winik_class = villager::winik,
  resource_class = villager::resource
)
```

Arguments:

`name` An optional name for the village

`initial_condition` A function that gets called on the first time step

`models` A list of functions or a single function that should be run at each time step

`winik_class` The class that's being used to represent agents

`resource_class` The class being used to describe the resources Propagates the village a single time step

Method `propagate()`**:**

Usage:

```
village$propagate(current_step)
```

Arguments:

`current_step` The current time step

Details: This method is used to advance the village a single time step. It should NOT be used to set initial conditions. See the set_initial_state method.

Returns: None Runs the user defined function that sets the initial state of the village

Method `set_initial_state()`**:** Runs the initial condition model

Usage:

```
village$set_initial_state()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
village$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`village_state` *village_state*

Description

This is an object that represents the state of a village at a particular time.

Details

This class acts as a type of record that holds the values of the different village variables. This class can be subclassed to include more variables that aren't present.

Methods

Creates a new State

Public fields

`step` The time step that the state is relevant to

`winik_states` A list of winik states

`resource_states` A list of resource states

Methods

Public methods:

- `village_state$new()`
- `village_state$clone()`

Method `new()`**:** Initializes all of the properties in the state to the ones passed in. This should be called by subclasses during initialization.

Usage:

```
village_state$new(
  step = 0,
  winik_states = vector(),
  resource_states = vector()
)
```

Arguments:

`step` The time step that the state is relevant to

`winik_states` A vector of tibbles representing the states of the winiks

`resource_states` A vector of tibbles representing the states of the resources

Details: When adding a new property, make sure to add it to the tibble representation.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
village_state$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`winik` *Winik*

Description

This is an object that represents a villager (winik).

Details

This class acts as an abstraction for handling villager-level logic. It can take a number of functions that run at each timestep. It also has an associated

Methods

`as_table()` Represents the current state of the winik as a tibble

`get_age()` Returns age in terms of years

`get_gender()`

`get_days_sincelast_birth()` Get the number of days since the winik last gave birth

`initialize()` Create a new winik

`propagate()` Runs every day

Create a new winik

Public fields

`identifier` A unique identifier that can be used to identify and find the winik

`first_name` The winik's first name

`last_name` The winik's last name

`age` The winik's age

`mother_id` The identifier of the winik's mother

`father_id` The identifier of the winik's father

`profession` The winik's profession

`partner` The identifier of the winik's partner

`gender` The winik's gender

`alive` A boolean flag that represents whether the villager is alive or dead

`children` A list of children identifiers

`health` A percentage value of the winik's current health

Methods

Public methods:

- `winik$new()`
- `winik$is_alive()`
- `winik$get_days_since_last_birth()`
- `winik$add_child()`
- `winik$as_table()`
- `winik$clone()`

Method `new()`**:** Used to created new winik objects.

Usage:

```
winik$new(
  identifier = NA,
  first_name = NA,
  last_name = NA,
  age = 0,
  mother_id = NA,
  father_id = NA,
  partner = NA,
  children = vector(mode = "character"),
  gender = NA,
  profession = NA,
  alive = TRUE,
  health = 100
)
```

Arguments:

`identifier` The winik's identifier

`first_name` The winik's first name

`last_name` The winik's last naem

`age` The age of the winik

`mother_id` The identifier of the winik's monther

`father_id` The identifier of the winik' father

`partner` The identifier of the winik's partner

`children` An ordered list of of the children from this winik

`gender` The gender of the winik

`profession` The winik's profession

`alive` Boolean whether the winik is alive or not

`health` A percentage value of the winik's current health

Returns: A new winik object A function that returns true or false whether the villager dies This is run each day

Method `is_alive()`**:**

Usage:

```
winik$is_alive()
```

Returns: A boolean whether the winik is alive (true for yes) Gets the number of days from the last birth. This is also the age of the most recently born winik

Method `get_days_since_last_birth()`**:**

Usage:

```
winik$get_days_since_last_birth()
```

Returns: The number of days since last birth Connects a child to the winik. This method ensures that the 'children' vector is ordered.

Method `add_child()`**:**

Usage:

```
winik$add_child(child)
```

Arguments:

`child` The Winik object representing the child

Returns: None Returns a data.frame representation of the winik

Method `as_table()`**:** I hope there's a more scalable way to do this in R; Adding every new attribute to this function isn't practical

Usage:

```
winik$as_table()
```

Details: The village_state holds a copy of all of the villagers at each timestep; this method is used to turn the winik properties into the object inserted in the village_state.

Returns: A data.frame representation of the winik

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
winik$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

winik_manager *Winik Manager*

Description

A class that abstracts the management of aggregations of Winik classes. Each village should have an instance of a winik_manager to interface the winiks inside.

Methods

`add_winik()` Adds a single winik to the manager.

`get_average_age()` Returns the average age, in years, of all the winiks.

`get_living_winiks()` Gets a list of all the winiks that are currently alive.

`get_states()` Returns a data.frame consisting of all of the managed winiks.

`get_winik()` Retrieves a particular winik from the manager.

`get_winik_index()` Retrieves the index of a winik.

`initialize()` Creates a new manager instance.

`load()` Loads a csv file defining a population of winiks and places them in the manager.

`remove_winik()` Removes a winik from the manager

Creates a new winik manager instance.

Public fields

`winiks` A list of winiks objects that the winik manager manages.

`winik_class` A class describing winiks. This is usually the default villager supplied 'winik' class

Methods

Public methods:

- `winik_manager$new()`
- `winik_manager$get_winik()`
- `winik_manager$get_living_winiks()`
- `winik_manager$add_winik()`
- `winik_manager$remove_winik()`
- `winik_manager$get_states()`
- `winik_manager$get_winik_index()`
- `winik_manager$connect_winiks()`
- `winik_manager$get_living_population()`
- `winik_manager$get_average_age()`
- `winik_manager$add_children()`
- `winik_manager$load()`
- `winik_manager$clone()`

Method `new()`**:**

Usage:

```
winik_manager$new(winik_class = villager::winik)
```

Arguments:

`winik_class` The class that's being used to represent agents being managed Given the identifier of a winik, sort through all of the managed winiks and return it if it exists.

Method `get_winik()`**:** Return the R6 instance of a winik with identiifier 'winik_identifier'.

Usage:

```
winik_manager$get_winik(winik_identifier)
```

Arguments:

`winik_identifier` The identifier of the requested winik.

Returns: An R6 winik object Returns a list of all the winiks that are currently alive.

Method `get_living_winiks()`**:**

Usage:

```
winik_manager$get_living_winiks()
```

Returns: A list of living winiks Adds a winik to the manager.

Method `add_winik()`**:**

Usage:

```
winik_manager$add_winik(new_winik)
```

Arguments:

`new_winik` The winik to add to the manager

Returns: None Removes a winik from the manager

Method `remove_winik()`**:**

Usage:

```
winik_manager$remove_winik(winik_identifier)
```

Arguments:

`winik_identifier` The identifier of the winik being removed

Returns: None Returns a data.frame of winiks

Method `get_states()`**:**

Usage:

```
winik_manager$get_states()
```

Details: Each row of the data.frame represents a winik object

Returns: A single data.frame of all winiks Returns the index of a winik in the internal winik list

Method `get_winik_index()`**:**

Usage:

```
winik_manager$get_winik_index(winik_identifier)
```

Arguments:

`winik_identifier` The identifier of the winik being located

Returns: The index in the list, or R's default return value Connects two winiks together as mates

Method `connect_winiks()`**:**

Usage:

```
winik_manager$connect_winiks(winik_a, winik_b)
```

Arguments:

`winik_a` A winik that will be connected to winik_b

`winik_b` A winik that will be connected to winik_a Returns the total number of winiks that are alive

Method `get_living_population()`**:**

Usage:

```
winik_manager$get_living_population()
```

Returns: The numnber of living winiks Returns the averag age, in years, of all of the winiks

Method `get_average_age()`**:**

Usage:

```
winik_manager$get_average_age()
```

Details: This is an *example* of the kind of logic that the manager might handle. In this case, the manager is performing calculations about its aggregation (winiks). Note that the 364 days needs to work better

Returns: The average age in years Takes all of the winiks in the manager and reconstructs the children

Method `add_children()`**:**

Usage:

```
winik_manager$add_children()
```

Details: This is typically called when loading winiks from disk for the first time. When children are created during the simulation, the family connections are made through the winik class and added to the manager via add_winik.

Returns: None Loads winiks from disk.

Method `load()`**:**

Usage:

```
winik_manager$load(file_name)
```

Arguments:

`file_name` The location of the file holding the winiks.

Details: Populates the winik manager with a set of winiks defined in a csv file.

Returns: None

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
winik_manager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index

data_writer, 2

model_data, 3

resource, 4
resource_manager, 5

simulation, 7

village, 8
village_state, 10

winik, 11
winik_manager, 14