Package 'SpaDES'

October 7, 2016

Type Package

Title Develop and Run Spatially Explicit Discrete Event Simulation Models

Description Implement a variety of event-based models, with a focus on spatially explicit models. These include raster-based, event-based, and agent-based models. The core simulation components are built upon a discrete event simulation (DES) framework that facilitates modularity, and easily enables the user to include additional functionality by running user-built simulation modules. Included are numerous tools to visualize raster and other maps. The suggested package 'fastshp' can be installed with `install.packages(``fastshp", repos = ``http://rforge.net", type = ``source")`.

URL http://spades.predictiveecology.org, https://github.com/PredictiveEcology/ SpaDES

Version 1.3.1

Date 2016-10-07

Additional_repositories http://rforge.net

Depends R (>= 3.2.2)

Suggests covr, fastshp, knitr, Matrix, png, rgdal, rgenoud, rmarkdown, snow, tcltk, testthat (>= 1.0.2), tkrplot

Imports archivist (>= 2.1), chron, CircStats, data.table (>= 1.9.6), DEoptim (>= 2.2-3), DiagrammeR (>= 0.8.2), digest, dplyr (>= 0.5.0), DT, ff, ffbase, fpCompare, ggplot2, graphics, grDevices, grid, gridBase, httr, igraph (>= 1.0.1), lazyeval, lubridate (>= 1.3.3), methods, miniUI (>= 0.1.1), parallel, R.utils, RandomFields (>= 3.1.24), raster, RColorBrewer, rstudioapi (>= 0.5), shiny (>= 0.13), stats, sp, stringi, stringr, utils

License GPL-3

VignetteBuilder knitr, rmarkdown

BugReports https://github.com/PredictiveEcology/SpaDES/issues

ByteCompile yes

Collate 'priority.R' 'environment.R' 'misc-methods.R'
'module-dependencies-class.R' 'helpers.R' 'simList-class.R'
'POM.R' 'agent.R' 'SELES.R' 'cache.R' 'check.R' 'checkpoint.R'
'copy.R' 'experiment.R' 'initialize.R' 'simulation.R' 'load.R'
'mapReduce.R' 'mergeRaster.R' 'module-dependencies-methods.R'
'moduleMetadata.R' 'module-repository.R' 'module-template.R'
'moduleCoverage.R' 'movement.R' 'neighbourhood.R'
'numerical-comparisons.R' 'plotting-classes.R'
'plotting-colours.R' 'times.R' 'simList-accessors.R'
'plotting-diagrams.R' 'plotting-helpers.R' 'plotting-other.R'
'plotting.R' 'probability.R' 'progress.R' 'rstudio-addins.R'
'save.R' 'shine.R' 'spades-classes.R' 'spades-package.R'
'splitRaster.R' 'spread-process.R' 'zzz.R'

RoxygenNote 5.0.1

NeedsCompilation no

Author Alex M Chubaty [aut, cre],
Eliot J B McIntire [aut],
Yong Luo [ctb],
Steve Cumming [ctb],
Her Majesty the Queen in Right of Canada, as represented by the
Minister of Natural Resources Canada [cph]

Maintainer Alex M Chubaty <alexander.chubaty@canada.ca>

Repository CRAN

Date/Publication 2016-10-07 18:42:54

R topics documented:

SpaDES-package *Categorized overview of the* SpaDES *package*

Description

This package allows implementation a variety of simulation-type models, with a focus on spatially explicit models. The core simulation components are built upon a discrete event simulation framework that facilitates modularity, and easily enables the user to include additional functionality by running user-built simulation modules. Included are numerous tools to visualize various spatial data formats, as well as non-spatial data.

Bug reports: https://github.com/PredictiveEcology/SpaDES/issues

Module repository: https://github.com/PredictiveEcology/SpaDES-modules

Wiki: https://github.com/PredictiveEcology/SpaDES/wiki

1 Spatial discrete event simulation (SpaDES)

A collection of top-level functions for doing spatial discrete event simulation.

1.1 Simulations

There are two workhorse functions that initialize and run a simulation, and third function for doing multiple spades runs:

simInit	Initialize a new simulation
spades	Run a discrete event simulation
experiment	Run multiple spades calls

1.2 Events

Within a module, important simulation functions include:

`scheduleEvent`	Schedule a simulation event
`removeEvent`	Remove an event from the simulation queue (not yet implemented)

##

1.2 `simList` methods

Collections of commonly used functions to retrieve or set slots (and their elements) of a `simList` object are summarized further below.

1.2.1 Simulation parameters

`globals`	List of global simulation parameters.
`params`	Nested list of all simulation parameter.
`P`	Namespaced version of `params` (i.e., do not have to specify module name).

1.2.2 loading from disk, saving to disk

`inputs`	List of loaded objects used in simulation. (advanced)
`outputs`	List of objects to save during simulation. (advanced)

1.2.3 objects in simList

`ls`, `objects`	Names of objects referenced by the simulation environment.
`ls.str`	List the structure of the simList objects.
`objs`	List of objects referenced by the simulation environment.

1.2.4 Simulation paths

Accessor functions for the `paths` slot and its elements.

`cachePath`	Global simulation cache path.
`modulePath`	Global simulation module path.

inputPath	Global simulation input path.
outputPath	Global simulation output path.
paths	Global simulation paths (cache, modules, inputs, outputs).

1.2.5 Simulation times

Accessor functions for the simtimes slot and its elements.

time	Current simulation time, in units of longest module.
start	Simulation start time, in units of longest module.
end	Simulation end time, in units of longest module.
times	List of all simulation times (current, start, end), in units of longest module..

1.2.6 Simulation event queues

Accessor functions for the events and completed slots. By default, the event lists are shown when the simList object is printed, thus most users will not require direct use of these methods.

events	Scheduled simulation events (the event queue). (advanced)
current	Currently executing event. (advanced)
completed	Completed simulation events. (advanced)

1.2.7 Modules and dependencies

Accessor functions for the depends, modules, and .loadOrder slots. These are included for advanced users.

depends	List of simulation module dependencies. (advanced)
modules	List of simulation modules to be loaded. (advanced)

1.2.8 simList environment

The simList has a slot called .envir which is an environment. All objects in the simList are actually in this environment, i.e., the simList is not entirely just a list. In R, environments use pass-by-reference semantics, which means that copying an simList object using normal R assignment operation, e.g., sim2 <- sim1, will not copy the objects contained within the .envir slot. The two objects sim1 and sim2 will shared identical objects within that slot. Sometimes, this not desired, and a true copy is required.

envir	Access the environment of the simList directly (advanced)
copy	Deep copy of a simList. (advanced)

1.2.9 Checkpointing

Accessor method	Module	Description
checkpointFile	.checkpoint	Name of the checkpoint file. (advanced)
checkpointInterval	.checkpoint	The simulation checkpoint interval. (advanced)

1.2.10 Progress Bar

progressType	.progress	Type of graphical progress bar used. (advanced)
progressInterval	.progress	Interval for the progress bar. (advanced)

##

1.3 Module operations

Modules are the basic unit of SpaDES. These are generally created and stored locally, or are downloaded from remote repositories, including our SpaDES-modules repository on GitHub.

checksums	Verify (and optionally write) checksums for a module's data files.
downloadModule	Open all modules nested within a base directory
getModuleVersion	Get the latest module version # from module repository
newModule	Create new module from template
newModuleDocumentation	Create empty documentation for a new module
openModules	Open all modules nested within a base directory
moduleMetadata	Shows the module metadata
zipModule	Zip a module and its associated files

1.4 Module metadata

Each module requires several items to be defined. These comprise the metadata for that module (including default parameter specifications, inputs and outputs), and are currently written at the top of the module's .R file.

defineModule	Define the module metadata
defineParameter	Specify a parameter's name, value and set a default
expectsInput	Specify an input object's name, class, description, sourceURL and other specifications
createsOutput	Specify an output object's name, class, description and other specifications

1.5 Module dependencies

Once a set of modules have been chosen, the dependency information is automatically calculated once simInit is run. There are several functions to assist with dependency information:

depsEdgeList	Build edge list for module dependency graph

depsGraph	Build a module dependency graph using igraph

1.6 Exported SpaDES object classes

simList	The 'simList' class

1.7 Caching

Caching can be done in a variety of ways, most of which are up to the module developer. However, the one most common usage would be to cache a simulation run. This might be useful if a simulation is very long, has been run once, and the goal is just to retrieve final results. This would be an alternative to manually saving the outputs.

See example in spades, achieved by using cache = TRUE argument.

cache	Caches a spades call, but often used as arg in spades
showCache	Shows information about the objects in the cache
clearCache	Removes objects from the cache
clearStubArtifacts	Removes some errors in cached files

A module developer can build caching into their module by creating cached versions of their functions.

---

2 Module functions

A collection of functions that help with making modules, in addition to all the other R packages and code.

2.1 Spatial spreading/distances methods

Spatial contagion is a key phenomenon for spatially explicit simulation models. Contagion can be modelled using discrete approaches or continuous approaches. Several SpaDES functions assist with these:

spread	Contagious cellular automata
rings	Identify rings around focal cells (e.g., buffers and donuts)
adj	An optimized (i.e., faster) version of adjacent
cir	Identify pixels in a circle around a SpatialPoints* object
distanceFromEachPoint	Fast calculation of distance surfaces

2.2 Spatial agent methods

Agents have several methods and functions specific to them:

crw	Simple correlated random walk function
heading	Determines the heading between SpatialPoints*
makeLines	Makes SpatialLines object for, e.g., drawing arrows
move	A meta function that can currently only take "crw"
specificNumPerPatch	Initiate a specific number of agents per patch

2.3 GIS operations

In addition to the vast amount of GIS operations available in R (mostly from contributed packages such as sp, raster, maps, maptools and many others), we provide the following GIS-related functions:

equalExtent	Assess whether a list of extents are all equal

2.4 Map-reduce - type operations

These functions convert between reduced and mapped representations of the same data. This allows compact representation of, e.g., rasters that have many individual pixels that share identical information.

rasterizeReduced	Convert reduced representation to full raster

2.5 Colors in Raster* objects

We likely will not want the default colours for every map. Here are several helper functions to add to, set and get colors of Raster* objects:

setColors	Set colours for plotting Raster* objects
getColors	Get colours in a Raster* objects
divergentColors	Create a color palette with diverging colors around a middle

2.6 Random Map Generation

It is often useful to build dummy maps with which to build simulation models before all data are available. These dummy maps can later be replaced with actual data maps.

gaussMap	Creates a random map using gaussian random fields
randomPolygons	Creates a random polygon with specified number of classes

2.7 Checking for the existence of objects

`SpaDES` modules will often require the existence of objects in the `simList`. These are helpers for assessing this:

`checkObject`	Check for a existence of an object within a `simList`
`checkPath`	Checks the specified filepath for formatting consistencies

2.8 SELES-type approach to simulation

These functions are essentially skeletons and are not fully implemented. They are intended to make translations from SELES. You must know how to use SELES for these to be useful:

`agentLocation`	Agent location
`initiateAgents`	Initiate agents into a SpatialPointsDataFrame
`numAgents`	Number of agents
`probInit`	Probability of intiating an agent or event
`transitions`	Transition probability

2.9 Miscellaneous

Functions that may be useful within a `SpaDES` context

`inRange`	Test whether a number lies within range [a,b]
`layerNames`	Get layer names for numerous object classes
`loadPackages`	Simple wrapper for loading packages
`numLayers`	Return number of layers
`paddedFloatToChar`	Wrapper for padding (e.g., zeros) floating numbers to character
`updateList`	Update values in a named list

---

3 Plotting

There are several user-accessible plotting functions that are optimized for modularity and speed of plotting:

Commonly used:

`Plot`	The workhorse plotting function

Simulation diagrams:

`eventDiagram`	Gantt chart representing the events in a completed simulation.
`moduleDiagram`	Network diagram of simplified module (object) dependencies.
`objectDiagram`	Sequence diagram of detailed object dependencies.

Other useful plotting functions:

clearPlot	Helpful for resolving many errors
clickValues	Extract values from a raster object at the mouse click location(s)
clickExtent	Zoom into a raster or polygon map that was plotted with Plot
clickCoordinates	Get the coordinates, in map units, under mouse click
dev	Specify which device to plot on, making a non-RStudio one as default
newPlot	Open a new default plotting device
rePlot	Replots all elements of device for refreshing or moving plot

---

4 File operations

In addition to R's file operations, we have added several here to aid in bulk loading and saving of files for simulation purposes:

getFileName	Get the name of current file
loadFiles	Load simulation objects according to a filelist
rasterToMemory	Read a raster from file to RAM
saveFiles	Save simulation objects according to outputs and params

---

5 Sample data and modules included in package

Five maps and three modules are included within the SpaDES package

5.1 Data

Several dummy data sets are included for testing of functionality

spadesMaps	Help showing included maps

5.2 Modules

Several dummy modules are included for testing of functionality. These can be found with file.path(find.package("SpaD

randomLandscapes	Imports, updates, and plots several raster map layers
caribouMovement	A simple agent-based (a.k.a., individual-based) model
fireSpread	A simple model of a spatial spread process

6 Package options

SpaDES uses the following options to configure behaviour:

- spades.cachePath: The default local directory in which to cache simulation outputs. Default is a temporary directory (typically /tmp/SpaDES/cache).
- spades.inputPath: The default local directory in which to look for simulation inputs. Default is a temporary directory (typically /tmp/SpaDES/inputs).
- spades.lowMemory: If true, some functions will use more memory efficient (but slower) algorithms. Default FALSE.
- spades.modulePath: The default local directory where modules and data will be downloaded and stored. Default is a temporary directory (typically /tmp/SpaDES/modules).
- spades.moduleRepo: The default GitHub repository to use when downloading modules. Default "PredictiveEcology/SpaDES-modules".
- spades.nCompleted: The maximum number of completed events to retain in the completed event queue. Default 1000L.
- spades.outputPath: The default local directory in which to save simulation outputs. Default is a temporary directory (typically /tmp/SpaDES/outputs).
- spades.tolerance: The default tolerance value used for floating point number comparisons. Default .Machine$double.eps^0.5.

Author(s)

Alex M. Chubaty <alexander.chubaty@canada.ca>

Eliot J. B. McIntire <eliot.mcintire@canada.ca>

.arrangement-class *The* .arrangement *class*

Description

This class contains the plotting arrangement information.

Details

These gp* parameters will specify plot parameters that are available with gpar(). gp will adjust plot parameters, gpText will adjust title and legend text, gpAxis will adjust the axes. size adjusts point size in a SpatialPoints object. These will persist with the original Plot call for each individual object. Multiple entries can be used, but they must be named list elements and they must match the ... items to plot. This is true for a RasterStack also, i.e., the list of named elements must be the same length as the number of layers being plotted. The naming convention used is: RasterStackName$layerName, i.e, landscape$DEM.

Slots

`rows` numeric. Number of rows in the arrangement.

`columns` numeric. Number of columns in the arragnement.

`actual.ratio` numeric. Ratio of columns to rows

`ds.dimensionRatio` numeric. Ratio of the device size to the ratio of the extents

`ds` numeric of length 2. The dimensions of the plotting window in inches

`objects` list of length number of spatial objects. Each list has a character vector of the layer names in each of those

`isRaster` logical vector, indicating whether each object is a Raster* object

`names` character vector. The names of the layers in the plot

`extents` list of class Extent objects. These are needed to calculate the `ds.dimensionRatio`, which is used to scale the Spatial objects correctly

`isSpatialObjects` logical indicating whether the object(s) are `spatialObjects` or not

`layout` list of length 2, with width and height measurements for layout.

`gp` a gpar object or list of named gpar objects. These names must match the names of the `...` objects. Default is NULL. See details.

`gpText` a gpar object or a list of named gpar objects. These names must match the names of the `...` objects. Default is NULL. See details.

`gpAxis` a gpar object or a list of named gpar objects. These names must match the names of the `...` objects. Default is NULL. See details.

`size` a numeric or a named list of numerics, used for SpatialPoints plots. Default is 5. See details.

Author(s)

Eliot McIntire

See Also

spadesClasses

.fileEdit *Open a file for editing*

Description

Rstudio's `file.edit` behaves differently than `utils::file.edit`. The workaround is to have the user manually open the file if they are using Rstudio, as suggested in the Rstudio support ticket at https://support.rstudio.com/hc/en-us/community/posts/206011308-file-edit-vs-utils-file-edit.

Usage

```
.fileEdit(file)
```

Arguments

file	Character string giving the file path to open.

Value

Invoked for its side effect of opening a file for editing.

Author(s)

Alex Chubaty

.fileExtensions *File extensions map*

Description

How to load various types of files in R.

This function has two roles: 1) to proceed with the loading of files that are in a simList or 2) as a short cut to simInit(inputs = filelist). Generally not to be used by a user.

How to load various types of files in R.

Usage

```
.fileExtensions()

loadFiles(sim, filelist, ...)

## S4 method for signature 'simList,missing'
loadFiles(sim, filelist, ...)

## S4 method for signature 'missing,ANY'
loadFiles(sim, filelist, ...)

## S4 method for signature 'missing,missing'
loadFiles(sim, filelist, ...)

.saveFileExtensions()
```

Arguments

sim	simList object.
filelist	list or data.frame to call loadFiles directly from the filelist as described in Details
...	Additional arguments.

Author(s)

Eliot McIntire

Alex Chubaty

See Also

inputs

Examples

```
## Not run:

# Load random maps included with package
filelist <- data.frame(
    files = dir(system.file("maps", package = "SpaDES"),
            full.names = TRUE, pattern = "tif"),
    functions = "rasterToMemory", package = "SpaDES"
)
sim1 <- loadFiles(filelist = filelist)
clearPlot()
if (interactive()) Plot(sim1$DEM)

# Second, more sophisticated. All maps loaded at time = 0, and the last one is reloaded
#  at time = 10 and 20 (via "intervals").
# Also, pass the single argument as a list to all functions...
#  specifically, when add "native = TRUE" as an argument to the raster function
files = dir(system.file("maps", package = "SpaDES"),
            full.names = TRUE, pattern = "tif")
arguments = I(rep(list(native = TRUE), length(files)))
filelist = data.frame(
   files = files,
   functions = "raster::raster",
   objectName = NA,
   arguments = arguments,
   loadTime = 0,
   intervals = c(rep(NA, length(files)-1), 10)
)

sim2 <- loadFiles(filelist = filelist)

# if we extend the end time and continue running, it will load an object scheduled
#  at time = 10, and it will also schedule a new object loading at 20 because
#  interval = 10
end(sim2) <- 20
sim2 <- spades(sim2) # loads the percentPine map 2 more times, once at 10, once at 20

## End(Not run)
```

.fillInputRows *An internal function for coercing a data.frame to inputs()*

Description

An internal function for coercing a data.frame to inputs()

Usage

```
.fillInputRows(inputDF, startTime)
```

Arguments

inputDF	A data.frame with partial columns to pass to inputs( ) <-
startTime	Numeric time. The start(sim).

.fillOutputRows *An internal function for coercing a data.frame to inputs()*

Description

An internal function for coercing a data.frame to inputs()

Usage

```
.fillOutputRows(outputDF, endTime)
```

Arguments

outputDF	A data.frame with partial columns to pass to inputs( ) <-
endTime	Numeric time. The end(sim).

.first *Event priority*

Description

Preset envent priorities: 1 = first (highest); 5 = normal; 10 = last (lowest).

Usage

```
.first()

.highest()

.last()

.lowest()

.normal()
```

Value

A numeric.

Author(s)

Alex Chubaty

.makeSpadesPlot *Make a* .spadesPlot *class object*

Description

Builds a .spadesPlot object from a list of objects.

Usage

```
.makeSpadesPlot(plotObjects, plotArgs, whichSpadesPlottables, ...)

## S4 method for signature 'list,list'
.makeSpadesPlot(plotObjects, plotArgs,
  whichSpadesPlottables, ...)

## S4 method for signature 'list,missing'
.makeSpadesPlot(plotObjects, plotArgs,
  whichSpadesPlottables, ...)
```

```
## S4 method for signature 'missing,missing'
.makeSpadesPlot(plotObjects, plotArgs,
  whichSpadesPlottables, ...)
```

Arguments

plotObjects	list. Any plot objects.
plotArgs	list. Any arguments that the the grid package can accept for the specific grob types, e.g., rasterGrob, polygonGrob, etc.
whichSpadesPlottables	Logical indicating which objects in the Plot call can be plotted by Plot.
...	additional arguments. Currently nothing.

Value

A .spadesPlot object, which has 2 slots, one for the plot arrangement (i.e., layout and dimensions) and onefor all of the spadesGrobs (stored as a spadesGrobList of lists .spadesGrob objects).

Author(s)

Eliot McIntire

.makeViewports *Make viewports*

Description

Given a set of extents, and a layout for these extents, this function will output a viewport tree to allow plotting.

Usage

```
.makeViewports(sPlot, newArr = FALSE)
```

Arguments

sPlot	An object of class .spadesPlot.
newArr	Logical indicating whether this function will create a completely new viewport. Default FALSE. # @importFrom NetLogoRClasses extent

Details

This function will either create a totally new set of viewports, or simply add some nested viewports to an existing arrangement, i.e., is there still white space availabe to plot.

Author(s)

Eliot McIntire

`.moduleDeps-class` *The* `.moduleDeps` *class*

Description

Descriptor object for specifying SpaDES module dependecies.

Slots

`name` Name of the module as a character string.

`description` Description of the module as a character string.

`keywords` Character vector containing a module's keywords.

`authors` The author(s) of the module as a `person` object.

`childModules` A character vector of child module names. Modules listed here will be loaded with this module.

`version` The module version as a `numeric_version`. Semantic versioning is assumed `http://semver.org/`.

`spatialExtent` Specifies the module's spatial extent as an `Extent` object. Default is `NA`.

`timeframe` Specifies the valid timeframe for which the module was designed to simulate. Must be a `POSIXt` object of length 2, specifying the start and end times (e.g., `as.POSIXlt(c("1990-01-01 00:00:00", "2100-1`
Can be specified as `NA` using `as.POSIXlt(c(NA, NA))`.

`timeunit` Describes the time (in seconds) corresponding to 1.0 simulation time units. Default is `NA`.

`citation` A list of citations for the module, each as character strings. Alternatively, list of filenames of `.bib` or similar files. Defaults to `NA_character_`.

`documentation` List of filenames refering to module documentation sources.

`reqdPkgs` Character vector of R package names to be loaded. Defaults to `NA_character_`.

`parameters` A `data.frame` specifying the object dependencies of the module, with columns `paramName`, `paramClass`, and `default`, whose values are of type `character`, `character`, and `ANY`, respectively. Default values may be overridden by the user by passing a list of parameters to `simInit`.

`inputObjects` A `data.frame` specifying the object dependecies of the module, with columns `objectName`, `objectClass`, and `other`. For objects that are used within the module as both an input and an output, add the object to each of these `data.frames`.

`outputObjects` A `data.frame` specifying the objects output by the module, following the format of `inputObjects`.

Author(s)

Alex Chubaty

See Also

`.simDeps`, `spadesClasses`

.simDeps-class *The* .simDeps *class*

Description

Defines all simulation dependencies for all modules within a SpaDES simulation.

Slots

`dependencies` List of `.moduleDeps` dependency objects.

Author(s)

Alex Chubaty

See Also

`.moduleDeps`, `spadesClasses`

.simList-class *The* simList *class*

Description

Contains the minimum components of a SpaDES simulation. Various slot accessor methods (i.e., get and set functions) are provided (see 'Accessor Methods' below).

Details

Based on code from chapter 7.8.3 of Matloff (2011): "Discrete event simulation". Here, we implement a discrete event simulation in a more modular fashion so it's easier to add simulation components (i.e., "simulation modules"). We use S4 classes and methods, and use `data.table` instead of `data.frame` to implement the event queue (because it is much more efficient).

Slots

`modules` List of character names specifying which modules to load.

`params` Named list of potentially other lists specifying simulation parameters.

`events` The list of scheduled events (i.e., event queue), as a `data.table`. See 'Event Lists' for more information.

`current` The current event, as a `data.table`. See 'Event Lists' for more information..

`completed` The list of completed events, as a `data.table`. See 'Event Lists' for more information.

`depends` A `.simDeps` list of `.moduleDeps` objects containing module object dependency information.

`simtimes` List of numerical values describing the simulation start and end times; as well as the current simulation time.

`inputs` A list of length 2, containing: 1) a `data.frame` or `data.table` of files and metadata, and 2) a list of optional arguments to pass to an import function.

`outputs` A list of length 2 containing: 1) a `data.frame` or `data.table` of files and metadata, and 2) a list of optional arguments to pass to an export function.

`paths` Named list of `modulePath`, `inputPath`, and `outputPath` paths. Partial matching is performed.

`.envir` Environment referencing the objects used in the simulation. Several "shortcuts" to accessing objects referenced by this environment are provided, and can be used on the `simList` object directly instead of specifying the `.envir` slot: `$`, `[[`, `ls`, `ls.str`, `objs`. See examples.

Accessor Methods

Several slot (and sub-slot) accessor methods are provided for use, and categorized into separate help pages:

`simList-accessors-envir`	Simulation enviroment.
`simList-accessors-events`	Scheduled and completed events.
`simList-accessors-inout`	Passing data in to / out of simulations.
`simList-accessors-modules`	Modules loaded and used; module dependencies.
`simList-accessors-objects`	Accessing objects used in the simulation.
`simList-accessors-params`	Global and module-specific parameters.
`simList-accessors-paths`	File paths for modules, inputs, and outputs.
`simList-accessors-times`	Simulation times.

Event Lists

Event lists are sorted (keyed) first by time, second by priority. Each event is represented by a `data.table` row consisting of:

`eventTime`	The time the event is to occur.
`moduleName`	The module from which the event is taken.
`eventType`	A character string for the programmer-defined event type.
`eventPriority`	The priority given to the event.

Note

The `simList` class extends the `.simList` superclass by adding a slot `.envir` to store the simulation environment containing references to simulation objects. The `simList_` class extends the `.simList` superclass, by adding a slot `.list` containing the simulation objects. Thus, `simList` is identical to `simList_`, except that the former uses an environment for objects and the latter uses a list. The class `simList_` is only used internally.

Author(s)

Alex Chubaty and Eliot McIntire

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

---

.spadesEnv *The SpaDES environment*

Description

Environment used internally to store internal package objects and methods.

Usage

```
.spadesEnv
```

Format

An object of class `environment` of length 1.

.spadesGrob-class *The* .spadesGrob *class*

Description

This class contains the plotting .spadesGrob information.

Details

These `gp*` parameters will specify plot parameters that are available with `gpar()`. `gp` will adjust plot parameters, `gpText` will adjust title and legend text, `gpAxis` will adjust the axes. `size` adjusts point size in a `SpatialPoints` object. These will persist with the original `Plot` call for each individual object. Multiple entries can be used, but they must be named list elements and they must match the `...` items to plot. This is true for a RasterStack also, i.e., the list of named elements must be the same length as the number of layers being plotted. The naming convention used is: `RasterStackName$layerName`, i.e, `landscape$DEM`.

Slots

`plotName` character. Name of the plot frame, which is by default a concatenation of the `objName` and `layerName`

`objName` character. Name of object represented by this .spadesGrob

`envir` environment. The environment in which to find the objName

`layerName` character. Name of the layer represented by this .spadesGrob. Primarily used for RasterStacks

`objClass` character. Class of the object represented by this .spadesGrob

`isSpatialObjects` logical. TRUE if the object is one of the SpaDES recognized spatialObject classes

`plotArgs` list. Any parameters needed for plotting, set by Plot call.

Author(s)

Eliot McIntire

See Also

`spadesClasses`

`.spadesPlot-class` *The* `.spadesPlot` *class*

Description

This class contains all necessary information to build a Plot on a device, except the actual data. Thus, this class differs notably from the grid package, which keeps a copy of all data *and* information in a hidden location for further access for rebuilding, erasing etc. This difference allows the Plot function to be much faster than using the grid methodology directly. The cost to this speed gain is that the objects *must* be available, by name, in the .GlobalEnv.

Details

This class contains two slots, one for the overall arrangement of the plots within the device window, and the second for all the `.spadesGrob` objects within that device window. These `.spadesGrob` objects are the individual "smallest" plot unit.

These `gp*` parameters will specify plot parameters that are available with `gpar()`. `gp` will adjust plot parameters, `gpText` will adjust title and legend text, `gpAxis` will adjust the axes. `size` adjusts point size in a `SpatialPoints` object. These will persist with the original `Plot` call for each individual object. Multiple entries can be used, but they must be named list elements and they must match the `...` items to plot. This is true for a RasterStack also, i.e., the list of named elements must be the same length as the number of layers being plotted. The naming convention used is: `RasterStackName$layerName`, i.e, `landscape$DEM`.

Slots

`arr` An .arrangement object

`spadesGrobList` list. A list of lists of .spadesGrob objects

Author(s)

Eliot McIntire

See Also

`spadesClasses`

.spadesPlotObjects-class

The `.spadesPlotObjects` *class*

Description

This class contains the union of spatialObjects and several other plot-type objects. These are the object classes that the `Plot` function can handle.

Slots

`members` SpatialPoints*, SpatialPolygons*, SpatialLines*, RasterLayer, RasterStack

Author(s)

Eliot McIntire

See Also

`spadesClasses`

.spadesPlottables-class

The `.spadesPlottables` *class*

Description

This class is the union of all .spadesPlotObjects (e.g., RasterLayer*, SpatialPoints*, SpatialPolygons*, ggplot, hist etc.) and `.spadesPlot` class objects. This allows replotting of a `.spadesPlot` object

Slots

`members` `.spadesPlotObjects` and `.spadesPlot`

Author(s)

Eliot McIntire

See Also

spadesClasses

.updateSpadesPlot *Merge two SpaDES Plot objects*

Description

Merges two .spadesPlot objects

Usage

```
.updateSpadesPlot(newSP, curr, ...)

## S4 method for signature '.spadesPlot,list'
.updateSpadesPlot(newSP, curr, ...)

## S4 method for signature '.spadesPlot,missing'
.updateSpadesPlot(newSP, curr, ...)
```

Arguments

newSP	The "new" .spadesPlot object. I.e., the new merges and overwrites into current.
curr	The "current" .spadesPlot object. I.e., the one to be merged into.
...	Additional arguments. Currently none implemented.

Author(s)

Eliot McIntire

addin_newModule *Rstudio addin to create a new module*

Description

Rstudio addin to create a new module

Usage

```
addin_newModule()
```

Author(s)

Alex Chubaty

adj.raw *Fast 'adjacent' function, and Just In Time compiled version*

Description

Faster function for determining the cells of the 4, 8 or bishop neighbours of the `cells`. This is a hybrid function that uses matrix for small numbers of loci (<1e4) and data.table for larger numbers of loci

Usage

```
adj.raw(x = NULL, cells, directions = 8, sort = FALSE, pairs = TRUE,
  include = FALSE, target = NULL, numCol = NULL, numCell = NULL,
  match.adjacent = FALSE, cutoff.for.data.table = 10000, torus = FALSE,
  id = NULL)

adj(x = NULL, cells, directions = 8, sort = FALSE, pairs = TRUE,
  include = FALSE, target = NULL, numCol = NULL, numCell = NULL,
  match.adjacent = FALSE, cutoff.for.data.table = 10000, torus = FALSE,
  id = NULL)
```

Arguments

`x`	Raster* object for which adjacency will be calculated.
`cells`	vector of cell numbers for which adjacent cells should be found. Cell numbers start with 1 in the upper-left corner and increase from left to right and from top to bottom
`directions`	the number of directions in which cells should be connected: 4 (rook's case), 8 (queen's case), or 'bishop' to connect cells with one-cell diagonal moves. Or a neigborhood matrix (see Details)

sort	logical. Whether the outputs should be sorted or not, using Cell IDs of the from cells (and to cells, if match.adjacent is TRUE.
pairs	logical. If TRUE, a matrix of pairs of adjacent cells is returned. If FALSE, a vector of cells adjacent to cells is returned
include	logical. Should the focal cells be included in the result?
target	a vector of cells that can be spread to. This is the inverse of a mask.
numCol	numeric indicating number of columns in the raster. Using this with numCell is a bit faster execution time.
numCell	numeric indicating number of cells in the raster. Using this with numCol is a bit faster execution time.
match.adjacent	logical. Should the returned object be the same as the adjacent function in the raster package.
cutoff.for.data.table	numeric. If the number of cells is above this value, the function uses data.table which is faster with large numbers of cells.
torus	Logical. Should the spread event wrap around to the other side of the raster. Default is FALSE.
id	numeric If not NULL, then function will return "id" column. Default NULL.

Details

Between 4x (large number loci) to 200x (small number loci) speed gains over adjacent in raster package. There is some extra speed gain if NumCol and NumCells are passed rather than a raster. Efficiency gains come from: 1. use data.table internally - no need to remove NAs because wrapped or outside points are just removed directly with data.table - use data.table to sort and fast select (though not fastest possible) 2. don't make intermediate objects; just put calculation into return statement

The steps used in the algorithm are: 1. Calculate indices of neighbouring cells 2. Remove "to" cells that are - <1 or >numCells (i.e., they are above or below raster), using a single modulo calculation - where the modulo of "to" cells is equal to 1 if "from" cells are 0 (wrapped right to left) - or where the modulo of the "to" cells is equal to 0 if "from" cells are 1 (wrapped left to right)

Value

a matrix of one or two columns, from and to.

Author(s)

Eliot McIntire

See Also

adjacent

Examples

```
library(raster)
a <- raster(extent(0, 1000, 0, 1000), res = 1)
sam <- sample(1:length(a), 1e4)
numCol <- ncol(a)
numCell <- ncell(a)
adj.new <- adj(numCol = numCol, numCell = numCell, cells = sam, directions = 8)
adj.new <- adj(numCol = numCol, numCell = numCell, cells = sam, directions = 8,
  include = TRUE)
```

---

agentLocation SELES - *Agent Location at initiation*

Description

Sets the the location of the intiating agents. NOT YET FULLY IMPLEMENTED.

A SELES-like function to maintain conceptual backwards compatability with that simulation tool. This is intended to ease transitions from SELES.

You must know how to use SELES for these to be useful.

Usage

```
agentLocation(map)
```

Arguments

map	A SpatialPoints*, SpatialPolygons*, or Raster* object.

Value

Object of same class as provided as input. If a Raster*, then zeros are converted to NA.

Author(s)

Eliot McIntire

append_attr *Add a module to a* moduleList

Description

Ordinary base lists and vectors do not retain their attributes when subsetted or appended. This function appends items to a list while preserving the attributes of items in the list (but not of the list itself).

Usage

```
append_attr(x, y)

## S4 method for signature 'list,list'
append_attr(x, y)
```

Arguments

x	A list of items with optional attributes.
y	See x.

Details

Similar to updateList but does not require named lists.

Value

An updated list with attributes.

Author(s)

Alex Chubaty and Eliot McIntire

Examples

```
library(igraph) # igraph exports magrittr's pipe operator
tmp1 <- list("apple", "banana") %>% lapply(., `attributes<-`, list(type = "fruit"))
tmp2 <- list("carrot") %>% lapply(., `attributes<-`, list(type = "vegetable"))
append_attr(tmp1, tmp2)
rm(tmp1, tmp2)
```

cache *Cache method for simList class objects*

Description

Because the simList has an environment as one of its slots, the caching mechanism of the archivist package does not work. Here, we make a slight tweak to the cache function. Specifically, we remove all elements that have an environment as part of their attributes. This is generally functions that are loaded from the modules, but also the .envir slot in the simList. Thus, only non-function objects are used as part of the digest call in the digest package (used internally in the cache function).

showCache and clearCache are wrappers around archivist package functions, specific to simList objects. They allow the user a bit of control over what is being cached.

Usage

```
cache(cacheRepo = NULL, FUN, ..., notOlderThan = NULL)

## S4 method for signature 'ANY'
cache(cacheRepo = NULL, FUN, ..., notOlderThan = NULL)

clearCache(sim, afterDate, beforeDate, cacheRepo, ...)

## S4 method for signature 'ANY'
clearCache(sim, afterDate, beforeDate, cacheRepo, ...)

showCache(sim, cacheRepo, ...)

## S4 method for signature 'ANY'
showCache(sim, cacheRepo, ...)
```

Arguments

cacheRepo	A repository used for storing cached objects.
FUN	A function to be called.
...	Other arguments passed to If neither afterDate or beforeDate are provided, then all objects will be removed. If both afterDate and beforeDate are specified, then all objects between afterDate and beforeDate will be deleted.
notOlderThan	load an artifact from the database only if it was created after notOlderThan.
sim	A simList simulation object, generally produced by simInit.
afterDate	Objects cached after this date will be deleted, formatted YYYY-MM-DD.
beforeDate	Objects cached before this date will be deleted, formatted as YYYY-MM-DD.

Details

Normally, a user will access this functionality as an argument in spades.

Value

Identical to cache

Will clear all objects from the cachePath of the sim object

Author(s)

Eliot McIntire

Eliot McIntire

Eliot McIntire

See Also

cache.

cache.

splitTagsLocal.

Examples

```
## Not run:
mySim <- simInit(times=list(start=0.0, end=5.0),
             params=list(.globals=list(stackName="landscape", burnStats = "testStats")),
                 modules=list("randomLandscapes", "fireSpread"),
                 paths=list(modulePath=system.file("sampleModules", package="SpaDES")))
if (require(archivist)) {
  # Call cache function directly
  archivist::createLocalRepo(paths(mySim)$cachePath)
  system.time(outSim <- cache(paths(mySim)$cachePath,
            spades, sim = copy(mySim), .plotInitialTime = NA, notOlderThan = Sys.time()))
  system.time(outSim <- cache(paths(mySim)$cachePath,
              spades, sim = copy(mySim), .plotInitialTime = NA))

  # This functionality can be achieved within a spades call
  # compare caching ... run once to create cache
  system.time(outSim <- spades(copy(mySim), cache = TRUE, notOlderThan = Sys.time(),
                               .plotInitialTime = NA))
  # compare... second time is fast
  system.time(outSimCached <- spades(copy(mySim), cache = TRUE, .plotInitialTime = NA))
  all.equal(outSim, outSimCached)
}

## End(Not run)

## Not run:
clearCache(mySim)

## End(Not run)
```

```
## Not run:
showCache(mySim)

## End(Not run)
```

---

changeObjEnv *Copy or move objects from one environment to another*

Description

This will copy or move (if `rmSrc=TRUE`) objects passed as a character string to a different environment. This is used with a `spades` call to copy or move objects to the `envir` environment object.

Usage

```
changeObjEnv(x, toEnv, fromEnv, rmSrc)

## S4 method for signature 'character,environment,environment,logical'
changeObjEnv(x, toEnv,
  fromEnv, rmSrc)

## S4 method for signature 'character,environment,missing,missing'
changeObjEnv(x, toEnv)

## S4 method for signature 'character,missing,environment,missing'
changeObjEnv(x, fromEnv)

## S4 method for signature 'character,environment,missing,logical'
changeObjEnv(x, toEnv, rmSrc)

## S4 method for signature 'character,missing,environment,logical'
changeObjEnv(x, fromEnv,
  rmSrc)

## S4 method for signature 'character,environment,environment,missing'
changeObjEnv(x, toEnv,
  fromEnv)

## S4 method for signature 'list,ANY,ANY,ANY'
changeObjEnv(x, toEnv, fromEnv, rmSrc)
```

Arguments

`x`	objects passed as character string vector
`toEnv`	environment to copy or move to
`fromEnv`	environment to copy or move from
`rmSrc`	should the source copies of the objects be removed. Default is FALSE.

Author(s)

Eliot Mcintire

Examples

```
e1 <- new.env()
e2 <- new.env()
assign("a1", 1:1e3, envir = e1)
assign("a2", 1:1e3, envir = e1)
objs <- c("a1", "a2")
# move objects between environments

changeObjEnv(objs, fromEnv = e1, toEnv = e2)
```

---

checkModule *Check for the existence of a remote module*

Description

Looks in the remote `repo` for a module named `name`.

Usage

```
checkModule(name, repo)

## S4 method for signature 'character,character'
checkModule(name, repo)

## S4 method for signature 'character,missing'
checkModule(name)
```

Arguments

`name`	Character string giving the module name.
`repo`	GitHub repository name. Default is `"PredictiveEcology/SpaDES-modules"`, which is specified by the global option `spades.moduleRepo`.

Author(s)

Eliot McIntire

checkObject *Check for existence of object(s) referenced by a* objects *slot of a* simList *object*

Description

Check that a named object exists in the provide simList environment slot, and optionally has desired attributes.

Usage

```
checkObject(sim, name, object, layer, ...)

## S4 method for signature 'simList,missing,Raster,character'
checkObject(sim, name, object,
  layer, ...)

## S4 method for signature 'simList,missing,ANY,missing'
checkObject(sim, name, object, layer,
  ...)

## S4 method for signature 'simList,character,missing,missing'
checkObject(sim, name, object,
  layer, ...)

## S4 method for signature 'simList,character,missing,character'
checkObject(sim, name, object,
  layer, ...)

## S4 method for signature 'missing,ANY,missing,ANY'
checkObject(sim, name, object, layer, ...)
```

Arguments

sim	A simList object.
name	A character string specifying the name of an object to be checked.
object	An object. This is mostly used internally, or with layer, because it will fail if the object does not exist.
layer	Character string, specifying a layer name in a Raster, if the name is a Raster* object.
...	Additional arguments. Not implemented.

Value

Invisibly return TRUE indicating object exists; FALSE if not.

Author(s)

Alex Chubaty and Eliot McIntire

See Also

`library`.

`checkParams` *Check use and existence of params passed to simulation.*

Description

Checks that all parameters passed are used in a module, and that all parameters used in a module are passed.

Usage

```
checkParams(sim, coreModules, coreParams, path, ...)

## S4 method for signature 'simList,list,list,character'
checkParams(sim, coreModules,
  coreParams, path, ...)
```

Arguments

`sim`	A simList simulation object.
`coreModules`	List of core modules.
`coreParams`	List of default core parameters.
`path`	The location of the modules' source files.
`...`	Additional arguments. Not implemented.

Value

Invisibly return `TRUE` indicating object exists; `FALSE` if not. Sensible messages are be produced identifying missing parameters.

Author(s)

Alex Chubaty

checkPath *Check filepath.*

Description

Checks the specified filepath for formatting consistencies, such as trailing slashes, etc.

Usage

```
checkPath(path, create)

## S4 method for signature 'character,logical'
checkPath(path, create)

## S4 method for signature 'character,missing'
checkPath(path)

## S4 method for signature '`NULL`,ANY'
checkPath(path)

## S4 method for signature 'missing,ANY'
checkPath()
```

Arguments

path	A character string corresponding to a filepath.
create	A logical indicating whether the path should be created if it doesn't exist. Default is FALSE.

Value

Character string denoting the cleaned up filepath.

See Also

file.exists, dir.create.

checksums *Calculate checksums for a module's data files*

Description

Verify (and optionally write) checksums for data files in a module's data/ subdirectory. The file data/CHECKSUMS.txt contains the expected checksums for each data file. Checksums are computed using SpaDES:::digest, which is simply a wrapper around digest::digest.

Usage

```
checksums(module, path, write)

## S4 method for signature 'character,character,logical'
checksums(module, path, write)

## S4 method for signature 'character,character,missing'
checksums(module, path)
```

Arguments

module	Character string giving the name of the module.
path	Character string giving the path to the module directory.
write	Logical indicating whether to overwrite CHECKSUMS.txt. Default is FALSE, as users should not change this file. Module developers should write this file prior to distributing their module code, and update accordingly when the data change.

Details

Modules may require data that for various reasons cannot be distributed with the module source code. In these cases, the module developer should ensure that the module downloads and extracts the data required. It is useful to not only check that the data files exist locally but that their checksums match those expected. See also downloadData.

Value

A data.frame with columns: result, expectedFile, actualFile, and checksum.

Note

In version 1.2.0 and earlier, two checksums per file were required because of differences in the checksum hash values on Windows and Unix-like platforms. Recent versions use a different (faster) algorithm and only require one checksum value per file. To update your 'CHECKSUMS.txt' files using the new algorithm, see https://github.com/PredictiveEcology/SpaDES/issues/295#issuecomment-246513405.

Author(s)

Alex Chubaty

Examples

```
## Not run:
moduleName <- "my_module"
modulePath <- file.path("path", "to", "modules")

## verify checksums of all data files
checksums(monudleName, modulePath)
```

```
## write new CHECKSUMS.txt file

# 1. verify that all data files are present (and no extra files are present)
list.files(file.path(modulePath, moduleName, "data")

# 2. calculate file checksums and write to file (this will overwrite CHECKSUMS.txt)
checksums(monudleName, modulePath, write = TRUE)

## End(Not run)
```

---

cir *Identify pixels in a circle or ring (donut) around an object.*

Description

Identify the pixels and coordinates that are at a (set of) buffer distance(s) of the objects passed into `coords`. This is similar to `rgeos::gBuffer` but much faster and without the geo referencing information. In other words, it can be used for similar problems, but where speed is important. This code is substantially adapted from `PlotRegionHighlighter::createCircle`.

Usage

```
cir(landscape, coords, loci, maxRadius = ncol(landscape)/4,
  minRadius = maxRadius, allowOverlap = TRUE, allowDuplicates = FALSE,
  includeBehavior = "includePixels", returnDistances = FALSE,
  angles = NA_real_, returnAngles = FALSE, returnIndices = TRUE,
  closest = FALSE, simplify = TRUE)

## S4 method for signature 'RasterLayer,SpatialPoints,missing'
cir(landscape, coords, maxRadius,
  minRadius = maxRadius, allowOverlap, allowDuplicates, includeBehavior,
  returnDistances, angles, returnAngles, returnIndices, closest, simplify)

## S4 method for signature 'RasterLayer,missing,numeric'
cir(landscape, loci, maxRadius,
  minRadius = maxRadius, allowOverlap, allowDuplicates, includeBehavior,
  returnDistances, angles, returnAngles, returnIndices, closest, simplify)

## S4 method for signature 'RasterLayer,missing,missing'
cir(landscape, loci, maxRadius,
  minRadius = maxRadius, allowOverlap, allowDuplicates, includeBehavior,
  returnDistances, angles, returnAngles, returnIndices, closest, simplify)

## S4 method for signature 'RasterLayer,matrix,missing'
cir(landscape, coords, loci,
  maxRadius = ncol(landscape)/4, minRadius = maxRadius,
  allowOverlap = TRUE, allowDuplicates = FALSE,
```

```
includeBehavior = "includePixels", returnDistances = FALSE,
angles = NA_real_, returnAngles = FALSE, returnIndices = TRUE,
closest = FALSE, simplify = TRUE)
```

Arguments

landscape	Raster on which the circles are built.
coords	Either a matrix with 2 (or 3) columns, x and y (and id), representing the coordinates (and an associated id, like cell index), or a SpatialPoints* object around which to make circles. Must be same coordinate system as the landscape argument. Default is missing, meaning it uses the default to loci
loci	Numeric. An alternative to coords. These are the indices on landscape to initiate this function. See coords. Default is one point in centre of landscape..
maxRadius	Numeric vector of length 1 or same length as coords
minRadius	Numeric vector of length 1 or same length as coords. Default is maxRadius, meaning return all cells that are touched by the narrow ring at that exact radius. If smaller than maxRadius, then this will create a buffer or donut or ring.
allowOverlap	Logical. Should duplicates across id be removed or kept. Default TRUE.
allowDuplicates	Logical. Should duplicates within id be removed or kept. Default FALSE. This is useful if the actual x, y coordinates are desired, rather than the cell indices. This will increase the size of the returned object.
includeBehavior	Character string. Currently accepts only "includePixels", the default, and "excludePixels". See details.
returnDistances	Logical. If TRUE, then a column will be added to the returned data.table that reports the distance from coords to every point that was in the circle/donut surrounding coords. Default FALSE, which is faster.
angles	Numeric. Optional vector of angles, in radians, to use. This will create "spokes" outward from coords. Default is NA, meaning, use internally derived angles that will "fill" the circle.
returnAngles	Logical. If TRUE, then a column will be added to the returned data.table that reports the angle from coords to every point that was in the circle/donut surrounding coords. Default FALSE.
returnIndices	Logical. Should the function return a data.table with indices and values of successful spread events, or return a raster with values. See Details.
closest	Logical. When determining non-overlapping circles, should the function give preference to the closest loci or the first one (much faster). Default is FALSE, meaning the faster, though maybe not desired behavior.
simplify	logical. If TRUE, then all duplicate pixels are removed. This means that some x, y combinations will disappear.

Details

This function identifies all the pixels as defined by a donut with inner radius minRadius and outer radius of maxRadius. The includeBehavior defines whether the cells that intersect the radii but whose centres are not inside the donut are included `includePixels` or not `excludePixels` in the returned pixels identified. If this is `excludePixels`, and if a `minRadius` and `maxRadius` are equal, this will return no pixels.

Value

A `matrix` with 4 columns, `id`, `indices`, `x`, `y`. The `x` and `y` indicate the exact coordinates of the `indices` (i.e., cell number) of the `landscape` associated with the ring or circle being identified by this function.

See Also

`rings` which uses `spread` internally. `cir` tends to be faster when there are few starting points, `rings` tends to be faster when there are many starting points. `cir` scales with `maxRadius ^ 2` and `coords`. Another difference between the two functions is that `rings` takes the centre of the pixel as the centre of a circle, whereas `cir` takes the exact coordinates. See example. For the specific case of creating distance surfaces from specific points, see `distanceFromEachPoint`, which is often faster. For the more general GIS buffering, see `rgeos::gBuffer`.

Examples

```
library(raster)
library(data.table)
library(sp)

set.seed(1642)

# circle centred
Ras <- raster(extent(0, 15, 0, 15), res = 1, val = 0)
middleCircle <- cir(Ras)
Ras[middleCircle[, "indices"]] <- 1
circlePoints <- SpatialPoints(middleCircle[, c("x", "y")])
if (interactive()) {
  clearPlot()
  Plot(Ras)
  Plot(circlePoints, addTo = "Ras")
}

# circles non centred
Ras <- randomPolygons(Ras, numTypes = 4)
N <- 2
agent <- SpatialPoints(coords = cbind(x = stats::runif(N, xmin(Ras), xmax(Ras)),
                                      y = stats::runif(N, xmin(Ras), xmax(Ras))))

cirs <- cir(Ras, agent, maxRadius = 15, simplify = TRUE)
cirsSP <- SpatialPoints(coords = cirs[, c("x", "y")])
cirsRas <- raster(Ras)
cirsRas[] <- 0
```

```
cirsRas[cirs[, "indices"]] <- 1

if (interactive()) {
  clearPlot()
  Plot(Ras)
  Plot(cirsRas, addTo = "Ras", cols = c("transparent", "#00000055"))
  Plot(agent, addTo = "Ras")
  Plot(cirsSP, addTo = "Ras")
}

# Example comparing rings and cir
a <- raster(extent(0,30,0,30), res = 1)
hab <- gaussMap(a, speedup = 1) # if raster is large (>1e6 pixels), use speedup>1
radius <- 4
N <- 2
coords <- SpatialPoints(coords = cbind(x = stats::runif(N, xmin(hab), xmax(hab)),
                                       y = stats::runif(N, xmin(hab), xmax(hab))))

# cirs
cirs <- cir(hab, coords, maxRadius = rep(radius, length(coords)), simplify = TRUE)

# rings
loci <- cellFromXY(hab, coordinates(coords))
cirs2 <- rings(hab, loci, maxRadius = radius, minRadius = radius - 1, returnIndices = TRUE)

# Plot both
ras1 <- raster(hab)
ras1[] <- 0
ras1[cirs[, "indices"]] <- cirs[, "id"]

ras2 <- raster(hab)
ras2[] <- 0
ras2[cirs2$indices] <- cirs2$id
if (interactive()) {
  clearPlot()
  Plot(ras1, ras2)
}

a <- raster(extent(0, 100, 0, 100), res = 1)
hab <- gaussMap(a,speedup = 1)
cirs <- cir(hab, coords, maxRadius = 44, minRadius = 0)
ras1 <- raster(hab)
ras1[] <- 0
cirsOverlap <- data.table(cirs)[,list(sumIDs = sum(id)),by = indices]
ras1[cirsOverlap$indices] <- cirsOverlap$sumIDs
if (interactive()) {
  clearPlot()
  Plot(ras1)
}

# Provide a specific set of angles
Ras <- raster(extent(0, 330, 0, 330), res = 1)
Ras[] <- 0
```

```
N <- 2
coords <- cbind(x = stats::runif(N, xmin(Ras), xmax(Ras)),
                y = stats::runif(N, xmin(Ras), xmax(Ras)))
circ <- cir(Ras, coords, angles = seq(0, 2*pi, length.out = 21),
            maxRadius = 200, minRadius = 0, returnIndices = FALSE,
            allowOverlap = TRUE, returnAngles = TRUE)
```

`classFilter` *Filter objects by class*

Description

Based on http://stackoverflow.com/a/5158978/1380598.

Usage

```
classFilter(x, include, exclude, envir)

## S4 method for signature 'character,character,character,environment'
classFilter(x, include,
  exclude, envir)

## S4 method for signature 'character,character,character,missing'
classFilter(x, include,
  exclude)

## S4 method for signature 'character,character,missing,environment'
classFilter(x, include,
  envir)

## S4 method for signature 'character,character,missing,missing'
classFilter(x, include)
```

Arguments

`x`	Character vector of object names to filter, possibly from `ls`.
`include`	Class(es) to include, as a character vector.
`exclude`	Optional class(es) to exclude, as a character vector.
`envir`	The environment ins which to search for objects. Default is the calling environment.

Value

Vector of object names matching the class filter.

Note

inherits is used internally to check the object class, which can, in some cases, return results inconsistent with is. See http://stackoverflow.com/a/27923346/1380598. These (known) cases are checked manually and corrected.

Author(s)

Alex Chubaty

Examples

```
## Not run:
  ## from global environment
  a <- list(1:10)     # class `list`
  b <- letters        # class `character`
  d <- stats::runif(10)      # class `numeric`
  f <- sample(1L:10L) # class `numeric`, `integer`
  g <- lm( jitter(d) ~ d ) # class `lm`
  h <- glm( jitter(d) ~ d ) # class `lm`, `glm`
  classFilter(ls(), include=c("character", "list"))
  classFilter(ls(), include = "numeric")
  classFilter(ls(), include = "numeric", exclude = "integer")
  classFilter(ls(), include = "lm")
  classFilter(ls(), include = "lm", exclude = "glm")
  rm(a, b, d, f, g, h)

## End(Not run)

## from local (e.g., function) environment
local({
  e <- environment()
  a <- list(1:10)     # class `list`
  b <- letters        # class `character`
  d <- stats::runif(10)      # class `numeric`
  f <- sample(1L:10L) # class `numeric`, `integer`
  g <- lm( jitter(d) ~ d ) # class `lm`
  h <- glm( jitter(d) ~ d ) # class `lm`, `glm`
  classFilter(ls(), include=c("character", "list"), envir = e)
  classFilter(ls(), include = "numeric", envir = e)
  classFilter(ls(), include = "numeric", exclude = "integer", envir = e)
  classFilter(ls(), include = "lm", envir = e)
  classFilter(ls(), include = "lm", exclude = "glm", envir = e)
  rm(a, b, d, e, f, g, h)
})

## from another environment
e = new.env(parent = emptyenv())
e$a <- list(1:10)     # class `list`
e$b <- letters        # class `character`
e$d <- stats::runif(10)      # class `numeric`
e$f <- sample(1L:10L) # class `numeric`, `integer`
e$g <- lm( jitter(e$d) ~ e$d ) # class `lm`
```

```
e$h <- glm( jitter(e$d) ~ e$d ) # class `lm`, `glm`
classFilter(ls(e), include=c("character", "list"), envir = e)
classFilter(ls(e), include = "numeric", envir = e)
classFilter(ls(e), include = "numeric", exclude = "integer", envir = e)
classFilter(ls(e), include = "lm", envir = e)
classFilter(ls(e), include = "lm", exclude = "glm", envir = e)
rm(a, b, d, f, g, h, envir = e)
rm(e)
```

`clearPlot` *Clear plotting device*

Description

Under some conditions, a device and its metadata need to be cleared manually. This can be done with either the `new = TRUE` argument within the call to `Plot`. Sometimes, the metadata of a previous plot will prevent correct plotting of a new `Plot` call. Use `clearPlot` to clear the device and all the associated metadata manually.

Usage

```
clearPlot(dev = dev.cur(), removeData = TRUE, force = FALSE)

## S4 method for signature 'numeric,logical'
clearPlot(dev = dev.cur(), removeData = TRUE,
  force = FALSE)

## S4 method for signature 'numeric,missing'
clearPlot(dev, force)

## S4 method for signature 'missing,logical'
clearPlot(removeData, force)

## S4 method for signature 'missing,missing'
clearPlot(dev = dev.cur(), removeData = TRUE,
  force = FALSE)
```

Arguments

`dev`	Numeric. Device number to clear.
`removeData`	Logical indicating whether any data that was stored in the `.spadesEnv` should also be removed; i.e., not just the plot window wiped.
`force`	Logical or "all". Sometimes the graphics state cannot be fixed by a simple clearPlot(). If TRUE, this will close the device and reopen the same device number. If "all", then all spades related data from all devices will be cleared, in addition to device closing and reopening.

Author(s)

Eliot McIntire

`clearStubArtifacts` *Clear erroneous archivist artifacts*

Description

When an archive object is being saved, if this is occurring at the same time as another process doing the same thing, a stub of a artifact occurs. This function will clear those stubs.

Usage

```
clearStubArtifacts(repoDir = NULL)

## S4 method for signature 'ANY'
clearStubArtifacts(repoDir = NULL)
```

Arguments

`repoDir`	A character denoting an existing directory of the Repository for which metadata will be returned. If it is set to NULL (by default), it will use the repoDir specified in `archivist::setLocalRepo`.

Value

Done for its side effect on the repoDir

Author(s)

Eliot McIntire

`clickValues` *Mouse interactions with Plots*

Description

These functions use `grid.locator`. The primary two user-level functions are `clickValues` and `clickExtent`. These functions automatically select the correct viewport (i.e., map) where the mouse clicks occured so the user does not have to manually specify which map is being clicked on. This works for `Raster*`, `SpatialPoints*`, and `SpatialPolygons*` objects.

Usage

```
clickValues(n = 1)

clickExtent(devNum = NULL, plot.it = TRUE)

clickCoordinates(n = 1)

.clickCoord(X, n = 1, gl = NULL)
```

Arguments

n	The number of mouse clicks to do.
devNum	The device number for the new plot to be plotted on.
plot.it	Logical. If TRUE a new plotting window is made for the new extent. Default TRUE.
X	The raster object whose values will be returned where mouse clicks occur.
gl	An object created by a call to grid.locator.

Details

clickValues is equivalent to running X[SpatialPoints(locator(n))], where X is the raster being clicked on, in base graphics. This function determines which place in the grid.layout was clicked and makes all appropriate calculations to determine the value on the raster(s) at that or those location(s). It should be noted that when zooming in to rasters, plotting of rasters will only allow for complete pixels to be plotted, even if the extent is not perfectly in line with pixel edges. As a result, when values returned by this function may be slightly off (<0.5 pixel width).

clickExtent is for drawing an extent with two mouse clicks on a given Plotted map.

clickCoordinates is the workhorse function that determines which plot has been clicked on and passes this plot name and the clicked coordinates to .clickCoord.

.clickCoord is intended for internal use and is called by other functions here.

Value

clickValues returns the layer names and values at the clicked points. clickExtent invisibly returns the extent object, and optionally plots it in a new device window. clickCoordinates returns the xy coordinates in the units of the plot clicked on.

Author(s)

Eliot McIntire

Eliot McIntire

Eliot McIntire

Eliot McIntire

copy *Copy a simList object*

Description

Because a simList works with an environment to hold all objects, all objects within that slot are pass-by-reference. That means it is not possible to simply copy an object with an assignment operator: the two objects will share the same objects. As one simList object changes so will the other. when this is not the desired behaviour, use this function.

Usage

```
copy(sim, objects = TRUE)

## S4 method for signature 'simList,logical'
copy(sim, objects = TRUE)

## S4 method for signature 'simList,missing'
copy(sim)
```

Arguments

sim	A simList object.
objects	Whether the objects contained within the simList environment should be copied. Default = TRUE, which may be slow.

createsOutput *Define an output object of a module*

Description

Used to specify an output object's name, class, description and other specifications.

Usage

```
createsOutput(objectName, objectClass, desc, ...)

## S4 method for signature 'ANY,ANY,ANY'
createsOutput(objectName, objectClass, desc, ...)

## S4 method for signature 'character,character,character'
createsOutput(objectName, objectClass,
  desc, ...)
```

Arguments

objectName	Character string to define the output object's name.
objectClass	Character string to specify the output object's class.
desc	Text string providing a brief description of the output object.
...	Other specifications of the output object.

Value

A data.frame suitable to be passed to outputObjects in a module's metadata.

Author(s)

Yong Luo

Examples

```
outputObjects <- dplyr::bind_rows(
  createsOutput(objectName = "outputObject1", objectClass = "character",
                desc = "this is for example"),
  createsOutput(objectName = "outputObject2", objectClass = "numeric",
                desc = "this is for example",
                otherInformation = "I am the second output object")
)
```

---

defineModule *Define a new module.*

Description

Specify a new module's metadata as well as object and package dependecies. Packages are loaded during this call.

Usage

```
defineModule(sim, x)

## S4 method for signature '.simList,list'
defineModule(sim, x)
```

Arguments

sim	A simList object.
x	A named list containing the parameters used to construct a new .moduleDeps object.

Value

Updated `simList` object.

Required metadata elements

`name`	Module name. Must match the filename (without the `.R` extension). This is currently not parsed by SpaDES
`description`	Brief description of the module. This is currently not parsed by SpaDES; it is for human readers only.
`keywords`	Author-supplied keywords. This is currently not parsed by SpaDES; it is for human readers only.
`childModules`	If this contains any character vector, then it will be treated as a parent module. If this is a parent module, th
`authors`	Module author information (as a vector of `person` objects. This is currently not parsed by SpaDES; it is for
`version`	Module version number (will be coerced to `numeric_version` if a character or numeric are supplied). The
`spatialExtent`	The spatial extent of the module supplied via `raster::extent`. This is currently unimplemented. Once im
`timeframe`	Vector (length 2) of POSIXt dates specifying the temporal extent of the module. Currently unimplemented.
`timeunit`	Time scale of the module (e.g., "day", "year"). This MUST be specified. It indicates what '1' unit of time n
`citation`	List of character strings specifying module citation information. Alternatively, a list of filenames of `.bib` o
`documentation`	List of filenames refering to module documentation sources. This is currently not parsed by SpaDES; it is f
`reqdPkgs`	List of R package names required by the module. These packages will be loaded when `simInit` is called.
`parameters`	A data.frame specifying the parameters used in the module. Usually produced by `rbind`-ing the outputs of
`inputObjects`	A `data.frame` specifying the data objects expected as inputs to the module, with columns `objectName` (cla
`outputObjects`	A `data.frame` specifying the data objects output by the module, with columns identical to those in `inputO`

Author(s)

Alex Chubaty

Examples

```
## Not run:
  # a default version of the defineModule is created with a call to newModule

  newModule("test", path = tempdir())
  # file.edit(file.path(tempdir(), "test", "test.R"))

 # The default defineModule created by newModule is currently (SpaDES version 1.2.0.9010):
  defineModule(sim, list(
    name = "test",
    description = "insert module description here",
    keywords = c("insert key words here"),
    authors = c(person(c("First", "Middle"), "Last",
                    email="email@example.com", role=c("aut", "cre"))),
    childModules = character(0),
    version = numeric_version("1.2.0.9010"),spatialExtent =
                                raster::extent(rep(NA_real_, 4)),
    timeframe = as.POSIXlt(c(NA, NA)),
  timeunit = NA_character_, # e.g., "year",
```

```
  citation = list("citation.bib"),
documentation = list("README.txt", "test.Rmd"),
  reqdPkgs = list(),
  parameters = rbind(
    #defineParameter("paramName", "paramClass", value, min, max,
    "parameter description")),
    defineParameter(".plotInitialTime", "numeric", NA, NA, NA,
    "This describes the simulation time at which the first plot event should occur"),
    defineParameter(".plotInterval", "numeric", NA, NA, NA,
    "This describes the simulation time at which the first plot event should occur"),
    defineParameter(".saveInitialTime", "numeric", NA, NA, NA,
    "This describes the simulation time at which the first save event should occur"),
    defineParameter(".saveInterval", "numeric", NA, NA, NA,
    "This describes the simulation time at which the first save event should occur")
  ),
  inputObjects = data.frame(
    objectName = NA_character_,
    objectClass = NA_character_,
    sourceURL = "",
    other = NA_character_,
    stringsAsFactors = FALSE
  ),
  outputObjects = data.frame(
    objectName = NA_character_,
    objectClass = NA_character_,
    other = NA_character_,
    stringsAsFactors = FALSE
  )
))


## End(Not run)
```

`defineParameter` *Define a parameter used in a module*

Description

Used to specify a parameter's name, value, and set a default.

Usage

```
defineParameter(name, class, default, min, max, desc)

## S4 method for signature 'character,character,ANY,ANY,ANY,character'
defineParameter(name,
  class, default, min, max, desc)
```

```
  ## S4 method for signature 'character,character,ANY,missing,missing,character'
defineParameter(name,
  class, default, desc)

## S4 method for signature 'missing,missing,missing,missing,missing,missing'
defineParameter()
```

Arguments

name	Character string giving the parameter name.
class	Character string giving the parameter class.
default	The default value to use when none is specified by the user. Non-standard evaluation is used for the expression.
min	With max, used to define a suitable range of values. Non-standard evaluation is used for the expression.
max	With min, used to define a suitable range of values. Non-standard evaluation is used for the expression.
desc	Text string providing a brief description of the parameter.

Value

data.frame

Author(s)

Alex Chubaty

Examples

```
parameters = rbind(
  defineParameter("lambda", "numeric", 1.23, desc = "intrinsic rate of increase"),
  defineParameter("P", "numeric", 0.2, 0, 1, "probability of attack")
)
```

---

depsEdgeList *Build edge list for module dependency graph*

Description

Build edge list for module dependency graph

Usage

```
depsEdgeList(sim, plot)

## S4 method for signature 'simList,logical'
depsEdgeList(sim, plot)

## S4 method for signature 'simList,missing'
depsEdgeList(sim, plot)
```

Arguments

sim	A simList object.
plot	Logical indicating whether the edgelist (and subsequent graph) will be used for plotting. If TRUE, duplicated rows (i.e., multiple object dependencies between modules) are removed so that only a single arrow is drawn connecting the modules. Default is FALSE.

Value

A data.table whose first two columns give a list of edges and remaining columns the attributes of the dependency objects (object name, class, etc.).

Author(s)

Alex Chubaty

---

depsGraph *Build a module dependency graph*

Description

Build a module dependency graph

Usage

```
depsGraph(sim, plot)

## S4 method for signature 'simList,logical'
depsGraph(sim, plot)

## S4 method for signature 'simList,missing'
depsGraph(sim)
```

Arguments

sim	A simList object.
plot	Logical indicating whether the edgelist (and subsequent graph) will be used for plotting. If TRUE, duplicated rows (i.e., multiple object dependencies between modules) are removed so that only a single arrow is drawn connecting the modules. Default is FALSE.

Value

An igraph object.

Author(s)

Alex Chubaty

---

dev *Specify where to plot*

Description

Switch to an existing plot device, or if not already open, launch a new graphics device based on operating system used. On Windows and Mac, if no x is provided, then this will open or switch to the first non R Studio device, which is much faster than the png-based R Studio Plot device. Currently, this will not open anything new

Usage

```
dev(x, ...)
```

Arguments

x	The number of a plot device. If missing, will open a new non-RStudio plotting device
...	Additional arguments passed to newPlot.

Details

For example, dev(6) switches the active plot device to device #6. If it doesn't exist, it opens it. NOTE: if devices 1-5 don't exist they will be opened too.

Value

Opens a new plot device on the screen. Invisibly returns the device number selected.

Author(s)

Eliot McIntire and Alex Chubaty

directionFromEachPoint *Calculate distances and directions between many points and many grid cells*

Description

This is a modification of distanceFromPoints for the case of many points. This version can often be faster for a single point because it does not return a RasterLayer. This is different than distanceFromPoints because it does not take the minimum distance from the set of points to all cells. Rather this returns the every pair-wise point distance. As a result, this can be used for doing inverse distance weightings, seed rain, cumulative effects of distance-based processes etc. If memory limitation is an issue, maxDistance will keep memory use down, but with the consequences that there will be a maximum distance returned. This function has the potential to use a lot of memory if there are a lot of from and to points.

This is meant to be used internally.

Usage

```
directionFromEachPoint(from, to = NULL, landscape)

.pointDirection(from, to)
```

Arguments

from	matrix with 2 or 3 columns, x and y, representing x and y coordinates of "from" cell, and optional "id" which will be returned, and if "id" column is in to, it will be matched with that.
to	matrix with 2 or 3 columns (or optionally more, all of which will be returned), x and y, representing x and y coordinates of "to" cells, and optional "id" which will be matched with "id" from from. It makes no sense to have "id" column here with no "id" column in from
landscape	RasterLayer. optional. This is only used if to is NULL, in which case all cells are considered to

Details

directionFromEachPoint calls .pointDirection, which is not intended to be called directly by the user.

If knowing the which from cell matches with which to cell is important, put a column "id" (e.g., starting cell) in the from matrix.

Value

A sorted matrix on `id` with same number of rows as `to`, but with one extra column, `angles` indicating the angle in radians between from and to. For speed, this angle will be between -pi/2 and 3*pi/2. If the user wants this between say, 0 and 2*pi, then `angles %% (2*pi)` will do the trick. See example.

See Also

`distanceFromEachPoint`, which will also return directions if `angles` is TRUE.

Examples

```
library(raster)
N <- 2
dirRas <- raster(extent(0,40,0,40), res = 1)
coords <- cbind(x = round(runif(N, xmin(dirRas), xmax(dirRas)))+0.5,
                y = round(runif(N, xmin(dirRas), xmax(dirRas)))+0.5,
                id = 1:N)

dirs1 <- directionFromEachPoint(from = coords, landscape = dirRas)
require(CircStats)
dirs1[, "angles"] <- deg(dirs1[,"angles"] %% (2*pi))
indices <- cellFromXY(dirRas,dirs1[, c("x", "y")])
minDir <- tapply(dirs1[,"angles"], indices, function(x) min(x)) # minimum angle
dirRas[] <- as.vector(minDir)
if (interactive()) {
  clearPlot()
  Plot(dirRas)
  library(sp)
  start <- SpatialPoints(coords[, c("x", "y"), drop = FALSE])
  Plot(start, addTo = "dirRas")
}
```

---

distanceFromEachPoint *Calculate distances and directions between many points and many grid cells*

Description

This is a modification of `distanceFromPoints` for the case of many points. This version can often be faster for a single point because it does not return a RasterLayer. This is different than `distanceFromPoints` because it does not take the minimum distance from the set of points to all cells. Rather this returns the every pair-wise point distance. As a result, this can be used for doing inverse distance weightings, seed rain, cumulative effects of distance-based processes etc. If memory limitation is an issue, maxDistance will keep memory use down, but with the consequences that there will be a maximum distance returned. This function has the potential to use a lot of memory if there are a lot of `from` and `to` points.

Usage

```
distanceFromEachPoint(from, to = NULL, landscape, angles = NA_real_,
  maxDistance = NA_real_, cumulativeFn = NULL, distFn = function(dist)
  1/(1 + dist), cl, ...)

.pointDistance(from, to, angles = NA, maxDistance = NA_real_)
```

Arguments

from	matrix with 2 or 3 columns, x and y, representing x and y coordinates of "from" cell, and optional "id" which will be matched with "id" from to
to	matrix with 2 or 3 columns (or optionally more, all of which will be returned), x and y, representing x and y coordinates of "to" cells, and optional "id" which will be matched with "id" from from. Default is all cells.
landscape	RasterLayer. optional. This is only used if to is NULL, in which case all cells are considered to
angles	Logical. If TRUE, then the function will return angles in radians, as well as distances.
maxDistance	Numeric in units of number of cells. The algorithm will build the whole surface (from from to to), but will remove all distances that are above this distance. Using this will keep memory use down.
cumulativeFn	A function that can be used to incrementally accumulate values in each to location, as the function iterates through each from. See Details.
distFn	A function. This can be a function of landscape, fromCells, toCells, and dist. If cumulativeFn is supplied, then this will be used to convert the distances to some other set of units that will be accumulated by the cumulativeFn. See Details and examples.
cl	A cluster object. Optional. This would generally be created using parallel::makeCluster or equivalent. This is an alternative way, instead of beginCluster(), to use parallelism for this function, allowing for more control over cluster use.
...	Any additional objects needed for distFn.

Details

This function is cluster aware. If there is a cluster running, it will use it. To start a cluster use beginCluster, with N being the number of cores to use. See examples in experiment.

If the user requires an id (indicating the from cell for each to cell) to be returned with the fuction, the user must add an identifier to the from matrix, such as "id". Otherwise, the function will only return the coordinates and distances.

distanceFromEachPoint calls .pointDistance, which is not intended to be called directly by the user.

This function has the potential to return a very large object, as it is doing pairwise distances (and optionally directions) between from and to. If there are memory limitations because there are many from and many to points, then cumulativeFn and distFn can be used. These two functions together will be used iteratively through the from points. The distFn should be a transformation of

distances to be used by the cumulativeFn function. For example, if distFn is 1/(1+x), the default, and cumulativeFn is `+`, then it will do a sum of inverse distance weights. See examples.

Value

A sorted matrix on id with same number of rows as to, but with one extra column, "dists" indicating the distance between from and to.

See Also

rings, cir, distanceFromPoints, which can all be made to do the same thing, under specific combinations of arguments. But each has different primary use cases. Each is also faster under different conditions. For instance, if maxDistance is relatively small compared to the number of cells in the landscape, then cir will likely be faster. If a minimum distance from all cells in the landscape to any cell in from, then distanceFromPoints will be fastest. This function scales best when there are many to points or all cells are used to = NULL (which is default).

Examples

```
library(raster)
N <- 2
distRas <- raster(extent(0,40,0,40), res = 1)
coords <- cbind(x = round(runif(N, xmin(distRas), xmax(distRas)))+0.5,
                y = round(runif(N, xmin(distRas), xmax(distRas)))+0.5)

# inverse distance weights
dists1 <- distanceFromEachPoint(coords, landscape = distRas)
indices <- cellFromXY(distRas,dists1[,c("x","y")])
invDist <- tapply(dists1[,"dists"], indices, function(x) sum(1/(1+x))) # idw function
distRas[] <- as.vector(invDist)
if (interactive()) {
  clearPlot()
  Plot(distRas)
}

# With iterative summing via cumulativeFn to keep memory use low, with same result
dists1 <- distanceFromEachPoint(coords[, c("x", "y"), drop = FALSE],
                                landscape = distRas, cumulativeFn = `+`)
idwRaster <- raster(distRas)
idwRaster[] <- dists1[,"val"]
if (interactive()) Plot(idwRaster)

all(idwRaster[] == distRas[]) # TRUE

# A more complex example of cumulative inverse distance sums, weighted by the value
#  of the origin cell
ras <- raster(extent(0,34, 0,34), res = 1, val = 0)
rp <- randomPolygons(ras, numTypes = 10) ^ 2
N <- 15
cells <- sample(ncell(ras), N)
coords <- xyFromCell(ras, cells)
distFn <- function(landscape, fromCell, dist) landscape[fromCell] / (1 + dist)
```

```
# beginCluster(3) # can do parallel
dists1 <- distanceFromEachPoint(coords[, c("x", "y"), drop = FALSE],
               landscape = rp, distFn = distFn, cumulativeFn = `+`)

# endCluster() # if beginCluster was run
idwRaster <- raster(ras)
idwRaster[] <- dists1[,"val"]
if (interactive()) {
  clearPlot()
  Plot(rp, idwRaster)
  sp1 <- SpatialPoints(coords)
  Plot(sp1, addTo="rp")
  Plot(sp1, addTo="idwRaster")
}
```

`divergentColors` *Divergent colour palette*

Description

Creates a palette for the current session for a divergent-color graphic with a non-symmetric range. Based on ideas from Maureen Kennedy, Nick Povak, and Alina Cansler.

Usage

```
divergentColors(start.color, end.color, min.value, max.value, mid.value = 0,
  mid.color = "white")

## S4 method for signature 'character,character,numeric,numeric'
divergentColors(start.color,
  end.color, min.value, max.value, mid.value = 0, mid.color = "white")
```

Arguments

`start.color`	Start colour to be passed to `colorRampPalette`.
`end.color`	End colour to be passed to `colorRampPalette`.
`min.value`	Numeric minimum value corresponding to `start.colour`. If attempting to change the color of a Raster layer, this can be set to minValue(RasterObject)
`max.value`	Numeric maximum value corresponding to `end.colour`. If attempting to change the color of a Raster layer, this can be set to maxValue(RasterObject)
`mid.value`	Numeric middle value corresponding to `mid.colour`. Default is `0`.
`mid.color`	Middle colour to be passed to `colorRampPalette`. Defaults to `"white"`.

Value

A diverging colour palette.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

colorRampPalette

Examples

```
divergentColors("darkred", "darkblue", -10, 10, 0, "white")
```

---

doEvent.checkpoint *Simulation checkpoints.*

Description

Save and reload the current state of the simulation, including the state of the random number generator, by scheduling checkpoint events.

Usage

```
doEvent.checkpoint(sim, eventTime, eventType, debug = FALSE)

checkpointLoad(file)

.checkpointSave(sim, file)

checkpointFile(object)

## S4 method for signature '.simList'
checkpointFile(object)

checkpointFile(object) <- value

## S4 replacement method for signature '.simList'
checkpointFile(object) <- value

checkpointInterval(object)

## S4 method for signature '.simList'
checkpointInterval(object)

checkpointInterval(object) <- value

## S4 replacement method for signature '.simList'
checkpointInterval(object) <- value
```

Arguments

sim	A simList simulation object.
eventTime	A numeric specifying the time of the next event.
eventType	A character string specifying the type of event: one of either "init", "load", or "save".
debug	Optional logical flag determines whether sim debug info will be printed (default debug = FALSE).
file	The checkpoint file.
object	A simList simulation object.
value	The object to be stored at the slot.

Details

checkpointLoad and .checkpointSave code based on: https://raw.githubusercontent.com/achubaty/r-tools/master/checkpoint.R

RNG save code adapted from: http://www.cookbook-r.com/Numbers/Saving_the_state_of_the_random_number_generator/ and https://stackoverflow.com/questions/13997444/

Value

Returns the modified simList object.

Author(s)

Alex Chubaty

See Also

.Random.seed.

Other functions to access elements of a simList object: .addDepends, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

downloadData *Download module data*

Description

Download external data for a module if not already present in the module directory or if there is a checksum mismatch indicating that the file is not the correct one.

Usage

```
downloadData(module, path, quiet)

## S4 method for signature 'character,character,logical'
downloadData(module, path, quiet)

## S4 method for signature 'character,missing,missing'
downloadData(module)

## S4 method for signature 'character,missing,logical'
downloadData(module, quiet)

## S4 method for signature 'character,character,missing'
downloadData(module, path)
```

Arguments

module	Character string giving the name of the module.
path	Character string giving the path to the module directory.
quiet	Logical. This is passed to download.file. Default is FALSE.

Value

Invisibly, a list of downloaded files.

Author(s)

Alex Chubaty

---

downloadModule *Download a module from a SpaDES module GitHub repository*

Description

Download a .zip file of the module and extract (unzip) it to a user-specified location.

Usage

```
downloadModule(name, path, version, repo, data, quiet)


  ## S4 method for signature
## 'character,character,character,character,logical,logical'
downloadModule(name,
  path, version, repo, data, quiet)
```

```
  ## S4 method for signature 'character,missing,missing,missing,missing,missing'
downloadModule(name)

## S4 method for signature 'character,ANY,ANY,ANY,ANY,ANY'
downloadModule(name, path, version,
  repo, data, quiet)
```

Arguments

name	Character string giving the module name.
path	Character string giving the location in which to save the downloaded module.
version	The module version to download. (If not specified, or NA, the most recent version will be retrieved.)
repo	GitHub repository name. Default is "PredictiveEcology/SpaDES-modules", which is specified by the global option spades.moduleRepo.
data	Logical. If TRUE, then the data that is identified in the module metadata will be downloaded, if possible. Default if FALSE.
quiet	Logical. This is passed to download.file. Default is FALSE.

Details

Currently only works with a public GitHub repository, where modules are in a modules directory in the root tree on the master branch.

NOTE: the default is to overwrite any existing files in the case of a conflict.

Value

A list of length 2. The first elemet is a character vector containing a character vector of extracted files for the module. The second element is a tbl with details about the data that is relevant for the function, including whether it was downloaded or not, whether it was renamed (because there was a local copy that had the wrong file name).

Author(s)

Alex Chubaty

---

dwrpnorm2 *Vectorized wrapped normal density function*

Description

This is a modified version of dwrpnorm found in CircStats to allow for multiple angles at once (i.e., vectorized).

Usage

```
dwrpnorm2(theta, mu, rho, sd = 1, acc = 1e-05, tol = acc)
```

Arguments

theta	value at which to evaluate the density function, measured in radians.
mu	mean direction of distribution, measured in radians.
rho	mean resultant length of distribution.
sd	different way of select rho, see details below.
acc	parameter defining the accuracy of the estimation of the density. Terms are added to the infinite summation that defines the density function until successive estimates are within acc of each other.
tol	the same as acc.

Author(s)

Eliot McIntire

Examples

```
# Values for which to evaluate density
theta <- c(1:500)*2*pi/500
# Compute wrapped normal density function
density <- c(1:500)
for(i in 1:500) density[i] <- dwrpnorm2(theta[i], pi, .75)
if (interactive()) plot(theta, density)
# Approximate area under density curve
sum(density*2*pi/500)
```

---

dyears *SpaDES time units*

Description

SpaDES modules commonly use approximate durations that divide with no remainder among themselves. For example, models that simulate based on a "week" timestep, will likely want to fall in lock step with a second module that is a "year" timestep. Since, weeks, months, years don't really have this behaviour because of: leap years, leap seconds, not quite 52 weeks in a year, months that are of different duration, etc. We have generated a set of units that work well together that are based on the astronomical or "Julian" year. In an astronomical year, leap years are added within each year with an extra 1/4 day, (i.e., 1 year == 365.25 days); months are defined as year/12, and weeks as year/52.

Usage

```
dyears(x)

## S4 method for signature 'numeric'
dyears(x)

dmonths(x)

## S4 method for signature 'numeric'
dmonths(x)

dweeks(x)

## S4 method for signature 'numeric'
dweeks(x)

dweek(x)

dmonth(x)

dyear(x)

dsecond(x)

dday(x)

dhour(x)

dNA(x)

## S4 method for signature 'ANY'
dNA(x)
```

Arguments

x	numeric. Number of the desired units

Details

When these units are not correct, a module developer can create their own time unit using, and create a function to calculate the number of seconds in that unit using the "d" prefix (for duration), following the `lubridate` package standard: `dfortNight <- function(x) lubridate::duration(dday(14))`. Then the module developer can use "fortNight" as the module's time unit.

Value

Number of seconds within each unit

Author(s)

Eliot McIntire

`envir` *Simulation environment*

Description

Accessor functions for the `.envir` slot in a `simList` object. These are included for advanced users.

Usage

```
envir(object)

## S4 method for signature 'simList'
envir(object)

envir(object) <- value

## S4 replacement method for signature 'simList'
envir(object) <- value
```

Arguments

`object`	A `simList` simulation object.
`value`	The object to be stored at the slot.

Details

Currently, only get and set methods are defined. Subset methods are not.

Value

Returns or sets the value of the slot from the `simList` object.

Author(s)

Alex Chubaty

See Also

SpaDES, specifically the section 1.2.8 on simList environment.

Other functions to access elements of a `simList` object: `.addDepends`, `doEvent.checkpoint`, `events`, `globals`, `inputs`, `ls.simList`, `ls.str.simList`, `modules`, `objs`, `packages`, `params`, `paths`, `progressInterval`, `times`

equalExtent *Assess whether a list of extents are all equal*

Description

Assess whether a list of extents are all equal

Usage

```
equalExtent(extents)

## S4 method for signature 'list'
equalExtent(extents)
```

Arguments

extents	list of extents objects

Author(s)

Eliot McIntire

eventDiagram *Simulation event diagram*

Description

Create a Gantt Chart representing the events in a completed simulation. This event diagram is constructed using the completed event list To change the number of events shown, provide an n argument.

Usage

```
eventDiagram(sim, n, startDate, ...)

## S4 method for signature 'simList,numeric,character'
eventDiagram(sim, n, startDate, ...)

## S4 method for signature 'simList,missing,character'
eventDiagram(sim, n, startDate, ...)

## S4 method for signature 'simList,missing,missing'
eventDiagram(sim, n, startDate, ...)
```

Arguments

sim	A simList object (typically corresponding to a completed simulation).
n	The number of most recently completed events to plot.
startDate	A character representation of date in YYYY-MM-DD format.
...	Additional arguments passed to mermaid. Useful for specifying height and width.

Details

Simulation time is presented on the x-axis, starting at date 'startDate'. Each module appears in a color-coded row, within which each event for that module is displayed corresponding to the sequence of events for that module. Note that only the start time of the event is meaningful is these figures: the width of the bar associated with a particular module's event DOES NOT correspond to an event's "duration".

Based on this StackOverflow answer: http://stackoverflow.com/a/29999300/1380598.

Value

Plots an event diagram as Gantt Chart, invisibly returning a mermaid object.

Note

A red vertical line corresponding to the current date may appear on the figure. This is useful for Gantt Charts generally but can be considered a 'bug' here.

Author(s)

Alex Chubaty

See Also

mermaid.

---

events *Simulation event lists*

Description

Accessor functions for the events and completed slots of a simList object. By default, the event lists are shown when the simList object is printed, thus most users will not require direct use of these methods.

events	Scheduled simulation events (the event queue).
completed	Completed simulation events.

Usage

```
events(object, unit)

## S4 method for signature '.simList,character'
events(object, unit)

## S4 method for signature '.simList,missing'
events(object, unit)

events(object) <- value

## S4 replacement method for signature '.simList'
events(object) <- value

current(object, unit)

## S4 method for signature '.simList,character'
current(object, unit)

## S4 method for signature '.simList,missing'
current(object, unit)

current(object) <- value

## S4 replacement method for signature '.simList'
current(object) <- value

completed(object, unit)

## S4 method for signature '.simList,character'
completed(object, unit)

## S4 method for signature '.simList,missing'
completed(object, unit)

completed(object) <- value

## S4 replacement method for signature '.simList'
completed(object) <- value
```

Arguments

object	A simList simulation object.
unit	Character. One of the time units used in SpaDES.
value	The object to be stored at the slot.

Details

Currently, only get and set methods are defined. Subset methods are not.

Value

Returns or sets the value of the slot from the `simList` object.

Note

Each event is represented by a `data.table` row consisting of:

`eventTime`	The time the event is to occur.
`moduleName`	The module from which the event is taken.
`eventType`	A character string for the programmer-defined event type.

See Also

`SpaDES`, specifically the section 1.2.6 on Simulation event queues.

Other functions to access elements of a `simList` object: `.addDepends`, `doEvent.checkpoint`, `envir`, `globals`, `inputs`, `ls.simList`, `ls.str.simList`, `modules`, `objs`, `packages`, `params`, `paths`, `progressInterval`, `times`

`expectsInput` *Define an input object that the module expects.*

Description

Used to specify an input object's name, class, description, source url and other specifications.

Usage

```
expectsInput(objectName, objectClass, desc, sourceURL, ...)

## S4 method for signature 'ANY,ANY,ANY,ANY'
expectsInput(objectName, objectClass, desc,
  sourceURL, ...)

## S4 method for signature 'character,character,character,character'
expectsInput(objectName,
  objectClass, desc, sourceURL, ...)

## S4 method for signature 'character,character,character,missing'
expectsInput(objectName,
  objectClass, desc, sourceURL, ...)
```

Arguments

objectName	Character string to define the input object's name.
objectClass	Character string to specify the input object's class.
desc	Text string providing a brief description of the input object.
sourceURL	Character string to specify an URL to reach the input object, default is NA.
...	Other specifications of the input object.

Value

A data.frame suitable to be passed to inputObjects in a module's metadata.

Author(s)

Yong Luo

Examples

```
inputObjects <- dplyr::bind_rows(
  expectsInput(objectName = "inputObject1", objectClass = "character",
               desc = "this is for example", sourceURL = "not available"),
  expectsInput(objectName = "inputObject2", objectClass = "numeric",
               desc = "this is for example", sourceURL = "not available",
               otherInformation = "I am the second input object")
)
```

---

experiment *Run an experiment using* spades

Description

This is essentially a wrapper around the spades call that allows for multiple calls to spades. This function will use a single processor, or multiple processors if beginCluster has been run first or a cluster objects is passed in the cl argument (gives more control to user).

Usage

```
experiment(sim, replicates = 1, params, modules, objects = list(), inputs,
  dirPrefix = "simNum", substrLength = 3, saveExperiment = TRUE,
  experimentFile = "experiment.RData", clearSimEnv = FALSE, notOlderThan,
  cl, ...)

## S4 method for signature 'simList'
experiment(sim, replicates = 1, params, modules,
  objects = list(), inputs, dirPrefix = "simNum", substrLength = 3,
  saveExperiment = TRUE, experimentFile = "experiment.RData",
  clearSimEnv = FALSE, notOlderThan, cl, ...)
```

Arguments

sim	A simList simulation object, generally produced by simInit.
replicates	The number of replicates to run of the same simList. See details and examples.
params	Like for simInit, but for each parameter, provide a list of alternative values. See details and examples.
modules	Like for simInit, but a list of module names (as strings). See details and examples.
objects	Like for simInit, but a list of named lists of named objects. See details and examples.
inputs	Like for simInit, but a list of inputs data.frames. See details and examples.
dirPrefix	String vector. This will be concatenated as a prefix on the directory names. See details and examples.
substrLength	Numeric. While making outputPath for each spades call, this is the number of characters kept from each factor level. See details and examples.
saveExperiment	Logical. Should params, modules, inputs, sim, and resulting experimental design be saved to a file. If TRUE are saved to a single list called experiment. Default TRUE.
experimentFile	String. Filename if saveExperiment is TRUE; saved to outputPath(sim) in .RData format. See Details.
clearSimEnv	Logical. If TRUE, then the envir(sim) of each simList in the return list is emptied. This is to reduce RAM load of large return object. Default FALSE.
notOlderThan	Date. Passed to SpaDES::cache to update the cache. Default is NULL, meaning don't update the cache. If Sys.time() is provided then, it will force a recache. Ignored if cache is FALSE.
cl	A cluster object. Optional. This would generally be created using parallel::makeCluster or equivalent. This is an alternative way, instead of beginCluster(), to use parallelism for this function, allowing for more control over cluster use.
...	Passed to spades. Specifically, debug, .plotInitialTime, .saveInitialTime, cache and/or notOlderThan. Caching is still experimental. It is tested to work under some conditions, but not all. See details.

Details

Generally, there are 2 reasons to do this: replication and varying simulation inputs to accomplish some sort of simulation experiment. This function deals with both of these cases. In the case of varying inputs, this function will attempt to create a fully factorial experiment among all levels of the variables passed into the function. If all combinations do not make sense, e.g., if parameters and modules are varied, and some of the parameters don't exist in all combinations of modules, then the function will do an "all meaningful combinations" factorial experiment. Likewise, fully factorial combinations of parameters and inputs may not be the desired behaviour. The function requires a simList object, acting as the basis for the experiment, plus optional inputs and/or objects and/or params and/or modules and/or replications.

This function requires a complete simList: this simList will form the basis of the modifications as passed by params, modules, inputs, and objects. All params, modules, inputs or objects passed into

this function will override the corresponding params, modules, inputs, or identically named objects that are in the sim argument.

This function is parallel aware, using the same mechanism as used in the raster package. Specifically, if you start a cluster using beginCluster, then this experiment function will automatically use that cluster. It is always a good idea to stop the cluster when finished, using endCluster.

Here are generic examples of how params, modules, objects, and inputs should be structured.

```
params = list(moduleName = list(paramName = list(val1, val2))).

modules = list(c("module1","module2"), c("module1","module3"))

objects = list(objName = list(object1=object1, object2=object2))

inputs = list(       data.frame(file = pathToFile1, loadTime = 0, objectName = "landscape",
)
```

Output directories are changed using this function: this is one of the dominant side effects of this function. If there are only replications, then a set of subdirectories will be created, one for each replicate. If there are varying parameters and or modules, outputPath is updated to include a subdirectory for each level of the experiment. These are not nested, i.e., even if there are nested factors, all subdirectories due to the experimental setup will be at the same level. Replicates will be one level below this. The subdirectory names will include the module(s), parameter names, the parameter values, and input index number (i.e., which row of the inputs data.frame). The default rule for naming is a concatenation of:

1. The experiment level (arbitrarily starting at 1). This is padded with zeros if there are many experiment levels.

2. The module, parameter name and parameter experiment level (not the parameter value, as values could be complex), for each parameter that is varying.

3. The module set.

4. The input index number

5. Individual identifiers are separated by a dash.

6. Module - Parameter - Parameter index triplets are separated by underscore.

e.g., a folder called: 01-fir_spr_1-car_N_1-inp_1 would be the first experiment level (01), the first parameter value for the spr* parameter of the fir* module, the first parameter value of the N parameter of the car* module, and the first input dataset provided.

This subdirectory name could be long if there are many dimensions to the experiment. The parameter substrLength determines the level of truncation of the parameter, module and input names for these subdirectories. For example, the resulting directory name for changes to the spreadprob parameter in the fireSpread module and the N parameter in the caribouMovement module would be: 1_fir_spr_1-car_N_1 if substrLength is 3, the default.

Replication is treated slightly differently. outputPath is always 1 level below the experiment level for a replicate. If the call to experiment is not a factorial experiment (i.e., it is just replication), then the default is to put the replicate subdirectories at the top level of outputPath. To force this one level down, dirPrefix can be used or a manual change to outputPath before the call to experiment.

dirPrefix can be used to give custom names to directories for outputs. There is a special value, "simNum", that is used as default, which is an arbitrary number associated with the experiment. This

corresponds to the row number in the `attr(sims, "experiment")`. This `"simNum"` can be used with other strings, such as `dirPrefix = c("expt", "simNum")`.

The experiment structure is kept in two places: the return object has an attribute, and a file named `experiment.RData` (see argument `experimentFile`) located in `outputPath(sim)`.

`substrLength`, if `0`, will eliminate the subfolder naming convention and use only `dirPrefix`.

If `cache = TRUE` is passed, then this will pass this to `spades`, with the additional argument `replicate = x`, where x is the replicate number. That means that if a user runs `experiment` with `replicate = 4` and `cache = TRUE`, then SpaDES will run 4 replicates, caching the results, including replicate = 1, replicate = 2, replicate = 3, and replicate = 4. Thus, if a second call to experiment with the exact same simList is passed, and `replicates = 6`, the first 4 will be taken from the cached copies, and replicate 5 and 6 will be run (and cached) as normal. If `notOlderThan` used with a time that is more recent than the cached copy, then a new spades will be done, and the cached copy will be deleted from the cache repository, so there will only ever be one copy of a particular replicate for a particular simList. NOTE: caching may not work as desired on a Windows machine because the sqlite database can only be written to one at a time, so there may be collisions.

Value

Invisibly returns a list of the resulting `simList` objects from the fully factorial experiment. This list has an attribute, which a list with 2 elements: the experimental design provided in a wide data.frame and the experiment values in a long data.frame. There is also a file saved with these two data.frames. It is named whatever is passed into `experimentFile`. Since returned list of `simList` objects may be large, the user is not obliged to return this object (as it is returned invisibly). Clearly, there may be objects saved during simulations. This would be determined as per a normal `spades` call, using `outputs` like, say, `outputs(sims[[1]])`.

Author(s)

Eliot McIntire

See Also

`simInit`, `SpaDES`

Examples

```
if (interactive()) {
  library(igraph) # use %>% in a few examples
  library(raster)

  tmpdir <- file.path(tempdir(), "examples")

  # Create a default simList object for use through these examples
  mySim <- simInit(
    times = list(start = 0.0, end = 2.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
      # Turn off interactive plotting
      fireSpread = list(.plotInitialTime = NA),
      caribouMovement = list(.plotInitialTime = NA),
```

```
    randomLandscapes = list(.plotInitialTime = NA)
  ),
  modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES"),
               outputPath = tmpdir),
  # Save final state of landscape and caribou
 outputs = data.frame(objectName = c("landscape", "caribou"), stringsAsFactors = FALSE)
)

# Example 1 - test alternative parameter values
# Create an experiment - here, 2 x 2 x 2 (2 levels of 2 params in fireSpread,
#    and 2 levels of 1 param in caribouMovement)

# Here is a list of alternative values for each parameter. They are length one
#   numerics here -- e.g., list(0.2, 0.23) for spreadprob in fireSpread module,
#   but they can be anything, as long as it is a list.
experimentParams <- list(fireSpread = list(spreadprob = list(0.2, 0.23),
                                           nFires = list(20, 10)),
                         caribouMovement = list(N = list(100, 1000)))

sims <- experiment(mySim, params = experimentParams)

# see experiment:
attr(sims, "experiment")

# Read in outputs from sims object
FireMaps <- do.call(stack, lapply(1:NROW(attr(sims, "experiment")$expDesign),
                                  function(x) sims[[x]]$landscape$Fires))
if (interactive()) Plot(FireMaps, new = TRUE)

# Or reload objects from files, useful if sim objects too large to store in RAM
caribouMaps <- lapply(sims, function(sim) {
  caribou <- readRDS(outputs(sim)$file[outputs(sim)$objectName == "caribou"])
})
names(caribouMaps) <- paste0("caribou", 1:8)
# Plot whole named list
if (interactive()) Plot(caribouMaps, size = 0.1)

# Example 2 - test alternative modules
# Example of changing modules, i.e., caribou with and without fires
# Create an experiment - here, 2 x 2 x 2 (2 levels of 2 params in fireSpread,
#    and 2 levels of 1 param in caribouMovement)
experimentModules <- list(
  c("randomLandscapes", "fireSpread", "caribouMovement"),
  c("randomLandscapes", "caribouMovement"))
sims <- experiment(mySim, modules = experimentModules)
attr(sims, "experiment")$expVals # shows 2 alternative experiment levels

# Example 3 - test alternative parameter values and modules
# Note, this isn't fully factorial because all parameters are not
#   defined inside smaller module list
sims <- experiment(mySim, modules = experimentModules, params = experimentParams)
attr(sims, "experiment")$expVals # shows 10 alternative experiment levels
```

```
# Example 4 - manipulate manipulate directory names -
#  "simNum" is special value for dirPrefix, it is converted to 1, 2, ...
sims <- experiment(mySim, params = experimentParams, dirPrefix = c("expt", "simNum"))
attr(sims, "experiment")$expVals # shows 8 alternative experiment levels, 24 unique
                                 #   parameter values

# Example 5 - doing replicate runs -
sims <- experiment(mySim, replicates = 2)
attr(sims, "experiment")$expDesign # shows 2 replicates of same experiment

# Example 6 - doing replicate runs, but within a sub-directory
sims <- experiment(mySim, replicates = 2, dirPrefix = c("expt"))
lapply(sims, outputPath) # shows 2 replicates of same experiment, within a sub directory

# Example 7 - doing replicate runs, of a complex, non factorial experiment.
# Here we do replication, parameter variation, and module variation all together.
# This creates 20 combinations.
# The experiment function tries to make fully factorial, but won't
# if all the levels don't make sense. Here, changing parameter values
# in the fireSpread module won't affect the simulation when the fireSpread
# module is not loaded:
# library(raster)
# beginCluster(20) # if you have multiple clusters available, use them here to save time
sims <- experiment(mySim, replicates = 2, params = experimentParams,
                   modules = experimentModules,
                   dirPrefix = c("expt", "simNum"))
# endCluster() # end the clusters
attr(sims, "experiment")

# Example 8 - Use replication to build a probability map.
# For this to be meaningful, we need to provide a fixed input landscape,
#   not a randomLandscape for each experiment level. So requires 2 steps.
# Step 1 - run randomLandscapes module twice to get 2 randomly
#  generated landscape maps. We will use 1 right away, and we will
#  use the two further below
mySimRL <- simInit(
  times = list(start = 0.0, end = 0.1, timeunit = "year"),
  params = list(
    .globals = list(stackName = "landscape"),
    # Turn off interactive plotting
    randomLandscapes = list(.plotInitialTime = NA)
  ),
  modules = list("randomLandscapes"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES"),
               outputPath = file.path(tmpdir, "landscapeMaps1")),
 outputs = data.frame(objectName = "landscape", saveTime = 0, stringsAsFactors = FALSE)
)
# Run it twice to get two copies of the randomly generated landscape
mySimRLOut <- experiment(mySimRL, replicate = 2)

#extract one of the random landscapes, which will be passed into next as an object
landscape <- mySimRLOut[[1]]$landscape
```

```
# here we don't run the randomLandscapes module; instead we pass in a landscape
#  as an object, i.e., a fixed input
mySimNoRL <- simInit(
  times = list(start = 0.0, end = 1, timeunit = "year"), # only 1 year to save time
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
    # Turn off interactive plotting
    fireSpread = list(.plotInitialTime = NA),
    caribouMovement = list(.plotInitialTime = NA)
  ),
  modules = list("fireSpread", "caribouMovement"), # No randomLandscapes modules
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES"),
               outputPath = tmpdir),
  objects = c("landscape"), # Pass in the object here
 # Save final state (the default if saveTime is not specified) of landscape and caribou
 outputs = data.frame(objectName = c("landscape", "caribou"), stringsAsFactors = FALSE)
)

# Put outputs into a specific folder to keep them easy to find
outputPath(mySimNoRL) <- file.path(tmpdir, "example8")
sims <- experiment(mySimNoRL, replicates = 8) # Run experiment
attr(sims, "experiment") # shows the experiment, which in this case is just replicates

# list all files that were saved called 'landscape'
landscapeFiles <- dir(outputPath(mySimNoRL), recursive = TRUE, pattern = "landscape",
                      full.names = TRUE)

# Can read in Fires layers from disk since they were saved, or from the sims
#  object
# Fires <- lapply(sims, function(x) x$landscape$Fires) %>% stack
Fires <- lapply(landscapeFiles, function(x) readRDS(x)$Fires) %>% stack()
Fires[Fires > 0] <- 1 # convert to 1s and 0s
fireProb <- sum(Fires)/nlayers(Fires) # sum them and convert to probability
if (interactive()) Plot(fireProb, new = TRUE)

# Example 9 - Pass in inputs, i.e., input data objects taken from disk
#  Here, we, again, don't provide randomLandscapes module, so we need to
#  provide an input stack called lanscape. We point to the 2 that we have
#  saved to disk in Example 8
mySimInputs <- simInit(
  times = list(start = 0.0, end = 2.0, timeunit = "year"),
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
    # Turn off interactive plotting
    fireSpread = list(.plotInitialTime = NA),
    caribouMovement = list(.plotInitialTime = NA)
  ),
  modules = list("fireSpread", "caribouMovement"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES"),
               outputPath = tmpdir),
  # Save final state of landscape and caribou
 outputs = data.frame(objectName = c("landscape", "caribou"), stringsAsFactors = FALSE)
```

```
  )
  landscapeFiles <- dir(tmpdir, pattern = "landscape_year0", recursive = TRUE, full.names = TRUE)

  # Varying inputs files - This could be combined with params, modules, replicates also
  outputPath(mySimInputs) <- file.path(tmpdir, "example9")
  sims <- experiment(mySimInputs,
                     inputs = lapply(landscapeFiles,function(filenames) {
                       data.frame(file = filenames, loadTime = 0,
                                  objectName = "landscape",
                                  stringsAsFactors = FALSE) })
   )

  # load in experimental design object
  experiment <- load(file = file.path(tmpdir, "example9", "experiment.RData")) %>% get()
  print(experiment) # shows input files and details

  # Example 10 - Use a very simple output dir name using substrLength = 0,
  #   i.e., just the simNum is used for outputPath of each spades call
  outputPath(mySim) <- file.path(tmpdir, "example10")
  sims <- experiment(mySim, modules = experimentModules, replicates = 2,
                     substrLength = 0)
  lapply(sims, outputPath) # shows that the path is just the simNum
  experiment <- load(file = file.path(tmpdir, "example10", "experiment.RData")) %>% get()
  print(experiment) # shows input files and details

  # Example 11 - use clearSimEnv = TRUE to remove objects from simList
  # This will shrink size of return object, which may be useful because the
  #  return from experiment function may be a large object (it is a list of
  # simLists). To see size of a simList, you have to look at the objects
  #  contained in the  envir(simList).  These can be obtained via objs(sim)
  sapply(sims, function(x) object.size(objs(x))) %>% sum + object.size(sims)
  # around 3 MB
  # rerun with clearSimEnv = TRUE
  sims <- experiment(mySim, modules = experimentModules, replicates = 2,
                     substrLength = 0, clearSimEnv = TRUE)
  sapply(sims, function(x) object.size(objs(x))) %>% sum + object.size(sims)
  # around 250 kB, i.e., all the simList contents except the objects.

  # Example 12 - pass in objects
  experimentObj <- list(landscape = lapply(landscapeFiles, readRDS) %>%
                                    setNames(paste0("landscape",1:2)))
  # Pass in this list of landscape objects
  sims <- experiment(mySimNoRL, objects = experimentObj)

  # Remove all temp files
  unlink(tmpdir, recursive = TRUE)
}
```

`fileExt` *Extract the file extension of a file*

Description

Extract the file extension of a file

Usage

```
fileExt(x)
```

Arguments

x	List or charcter vector of file names.

Value

A character vector of file extensions.

Author(s)

Eliot McIntire and Alex Chubaty

`fileName` *Extract filename (without extension) of a file*

Description

Extract filename (without extension) of a file

Usage

```
fileName(x)
```

Arguments

x	List or charcter vector

Value

A character vector.

Author(s)

Eliot McIntire

findObjects *Find objects if passed as character strings*

Description

Objects are passed into simList via `simInit` call or `objects(simList)` assignment. This function is an internal helper to find those objects from their environments by searching the call stack.

Usage

```
.findObjects(objects, functionCall = "simInit")
```

Arguments

`objects`	A character vector of object names
`functionCall`	A character string identifying the function name to be searched in the call stack. Default is "simInit"

Author(s)

Eliot McIntire

gaussMap *Produce a* `raster` *of a random Gaussian process.*

Description

This is a wrapper for the `RFsimulate` function in the RandomFields package. The main addition is the `speedup` argument which allows for faster map generation. A `speedup` of 1 is normal and will get progressively faster as the number increases, at the expense of coarser pixel resolution of the pattern generated

Usage

```
gaussMap(x, scale = 10, var = 1, speedup = 10, inMemory = FALSE, ...)
```

Arguments

`x`	A spatial object (e.g., a `RasterLayer`).
`scale`	The spatial scale in map units of the Gaussian pattern.
`var`	Spatial variance.
`speedup`	An index of how much faster than normal to generate maps.
`inMemory`	Should the RasterLayer be forced to be in memory? Default `FALSE`.
`...`	Additional arguments to `raster`.

Value

A raster map of extent `ext` with a Gaussian random pattern.

See Also

`RFsimulate` and `extent`

Examples

```
## Not run:
library(RandomFields)
library(raster)
nx <- ny <- 100L
r <- raster(nrows = ny, ncols = nx, xmn = -nx/2, xmx = nx/2, ymn = -ny/2, ymx = ny/2)
speedup <- max(1, nx/5e2)
map1 <- gaussMap(r, scale = 300, var = 0.03, speedup = speedup, inMemory = TRUE)
Plot(map1)

## End(Not run)
```

---

getColors *Get colours for plotting Raster* objects.*

Description

Get colours for plotting Raster* objects.

Usage

```
getColors(object)

## S4 method for signature 'Raster'
getColors(object)

## S4 method for signature 'ANY'
getColors(object)

## S4 method for signature 'SpatialPoints'
getColors(object)
```

Arguments

`object`	A `Raster*` object.

Value

Returns a named list of colors.

Author(s)

Alex Chubaty

#@importClassesFrom NetLogoRClasses agentMatrix

See Also

setColors<-, brewer.pal

---

getFileName *Get the name of a* source*-ed file*

Description

Use getFileName in a file that is source-ed. Based on http://stackoverflow.com/a/1816487/1380598.

Usage

```
getFileName(fullname)

## S4 method for signature 'logical'
getFileName(fullname)
```

Arguments

fullname	Logical (default FALSE) indicating whether the full path should be returned.

Value

Character string representing the filename.

Author(s)

Alex Chubaty

getModuleVersion *Find the latest module version from a SpaDES module repository*

Description

Modified from http://stackoverflow.com/a/25485782/1380598.

Usage

```
getModuleVersion(name, repo)

## S4 method for signature 'character,character'
getModuleVersion(name, repo)

## S4 method for signature 'character,missing'
getModuleVersion(name)
```

Arguments

name	Character string giving the module name.
repo	GitHub repository name. Default is "PredictiveEcology/SpaDES-modules", which is specified by the global option spades.moduleRepo.

Author(s)

Alex Chubaty

globals *Get and set simulation globals.*

Description

globals accesses or sets the "globals" slot in the simList.

Usage

```
globals(object)

## S4 method for signature '.simList'
globals(object)

globals(object) <- value

## S4 replacement method for signature '.simList'
globals(object) <- value
```

Arguments

object	A simList simulation object.
value	The object to be stored at the slot.

See Also

SpaDES, specifically the section 1.2.1 on Simulation Parameters.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

gpar *Importing some grid functions*

Description

Currently only the gpar function is imported. This is a convenience so that users can change Plot arguments without having to load the entire grid package.

Usage

```
gpar(...)

## S4 method for signature 'ANY'
gpar(...)
```

Arguments

...	Any number of named arguments.

See Also

gpar

heading *Heading between spatial points.*

Description

Determines the heading between spatial points.

Usage

```
heading(from, to)

## S4 method for signature 'SpatialPoints,SpatialPoints'
heading(from, to)

## S4 method for signature 'matrix,matrix'
heading(from, to)

## S4 method for signature 'matrix,SpatialPoints'
heading(from, to)

## S4 method for signature 'SpatialPoints,matrix'
heading(from, to)
```

Arguments

from	The starting position; an object of class SpatialPoints.
to	The ending position; an object of class SpatialPoints.

Value

The heading between the points, in degrees.

Author(s)

Eliot McIntire

Examples

```
library(sp)
N <- 10L                # number of agents
x1 <- stats::runif(N, -50, 50) # previous X location
y1 <- stats::runif(N, -50, 50) # previous Y location
x0 <- stats::rnorm(N, x1, 5)   # current X location
y0 <- stats::rnorm(N, y1, 5)   # current Y location

# using SpatialPoints
prev <- SpatialPoints(cbind(x = x1, y = y1))
curr <- SpatialPoints(cbind(x = x0, y = y0))
```

```
heading(prev, curr)

# using matrix
prev <- matrix(c(x1, y1), ncol = 2, dimnames = list(NULL, c("x","y")))
curr <- matrix(c(x0, y0), ncol = 2, dimnames = list(NULL, c("x","y")))
heading(prev, curr)

#using both
prev <- SpatialPoints(cbind(x = x1, y = y1))
curr <- matrix(c(x0, y0), ncol = 2, dimnames = list(NULL, c("x","y")))
heading(prev, curr)

prev <- matrix(c(x1, y1), ncol = 2, dimnames = list(NULL, c("x","y")))
curr <- SpatialPoints(cbind(x = x0, y = y0))
heading(prev, curr)
```

`initialize,simList-method` *Generate a* `simList` *object*

Description

Given the name or the definition of a class, plus optionally data to be included in the object, `new` returns an object from that class.

Usage

```
## S4 method for signature 'simList'
initialize(.Object)
```

Arguments

`.Object`	A `simList` object.

`initiateAgents` `SELES` - *Initiate agents*

Description

Sets the the number of agents to initiate. THIS IS NOT FULLY IMPLEMENTED.

A `SELES`-like function to maintain conceptual backwards compatability with that simulation tool. This is intended to ease transitions from SELES.

You must know how to use SELES for these to be useful.

Usage

```
initiateAgents(map, numAgents, probInit, asSpatialPoints = TRUE, indices)

## S4 method for signature 'Raster,missing,missing,ANY,missing'
initiateAgents(map, numAgents,
  probInit, asSpatialPoints)

## S4 method for signature 'Raster,missing,Raster,ANY,missing'
initiateAgents(map, probInit,
  asSpatialPoints)

## S4 method for signature 'Raster,numeric,missing,ANY,missing'
initiateAgents(map, numAgents,
  probInit, asSpatialPoints = TRUE, indices)

## S4 method for signature 'Raster,numeric,Raster,ANY,missing'
initiateAgents(map, numAgents,
  probInit, asSpatialPoints)

## S4 method for signature 'Raster,missing,missing,ANY,numeric'
initiateAgents(map, numAgents,
  probInit, asSpatialPoints = TRUE, indices)
```

Arguments

map	RasterLayer with extent and resolution of desired return object
numAgents	numeric resulting from a call to numAgents
probInit	a Raster resulting from a probInit call
asSpatialPoints	logical. Should returned object be RasterLayer or SpatialPointsDataFrame (default)
indices	numeric. Indices of where agents should start

Value

A SpatialPointsDataFrame, with each row representing an individual agent

Author(s)

Eliot McIntire

Examples

```
library(dplyr)
library(raster)
map <- raster(xmn = 0, xmx = 10, ymn = 0, ymx = 10, val = 0, res = 1)
map <- gaussMap(map, scale = 1, var = 4, speedup = 1)
pr <- probInit(map, p = (map/maxValue(map))^2)
```

```
agents <- initiateAgents(map, 100, pr)
if (interactive()) {
  clearPlot()
  Plot(map)
  Plot(agents, addTo = "map")
}

# Note, can also produce a Raster representing agents,
# then the number of points produced can't be more than
# the number of pixels:
agentsRas <- initiateAgents(map, 30, pr, asSpatialPoints = FALSE)
if (interactive()) Plot(agentsRas)

# Check that the agents are more often at the higher probability areas based on pr
out <- data.frame(stats::na.omit(crosstab(agentsRas, map)), table(round(map[]))) %>%
   dplyr::mutate(selectionRatio = Freq/Freq.1) %>%
   dplyr::select(-Var1, -Var1.1) %>%
   dplyr::rename(Present = Freq, Avail = Freq.1, Type = Var2)
out
```

`inputs` *Inputs and outputs*

Description

These functions are one of two mechanisms to add the information about which input files to load in a `spades` call and the information about which output files to save. The other way is to pass them as arguments to a `simInit` call.

`inputArgs` and `outputArgs` are ways to specify any arguments that are needed for file loading and file saving. This is still somewhat experimental.

Usage

```
inputs(object)

## S4 method for signature '.simList'
inputs(object)

inputs(object) <- value

## S4 replacement method for signature '.simList'
inputs(object) <- value

outputs(object)

## S4 method for signature '.simList'
outputs(object)
```

```
outputs(object) <- value

## S4 replacement method for signature '.simList'
outputs(object) <- value

inputArgs(object)

## S4 method for signature '.simList'
inputArgs(object)

inputArgs(object) <- value

## S4 replacement method for signature '.simList'
inputArgs(object) <- value

outputArgs(object)

## S4 method for signature '.simList'
outputArgs(object)

outputArgs(object) <- value

## S4 replacement method for signature '.simList'
outputArgs(object) <- value
```

Arguments

object	A simList simulation object.
value	The object to be stored at the slot. See Details.

Details

Accessor functions for the inputs and outputs slots in a simList object.

Value

Returns or sets the value(s) of the input or output slots in the simList object.

inputs

inputs accepts a data.frame, with up to 7 columns. Columns are:

file	required, a character string indicating the file path. There is no default.
objectName	optional, character string indicating the name of the object that the loaded file will be assigned to in the simLis
fun	optional, a character string indicating the function to use to load that file. Defaults to the known extentions in S
package	optional character string indicating the package in which to find the fun);
loadTime	optional numeric, indicating when in simulation time the file should be loaded. The default is the highest priori
interval	optional numeric, indicating at what interval should this same exact file be reloaded from disk, e.g,. 10 would r

arguments	is a list of lists of named arguments, one list for each fun. For example, if fun="raster", arguments = list

Currently, only file is required. All others will be filled with defaults if not specified.

See the modules vignette for more details (browseVignettes("SpaDES")).

outputs

outputs accepts a data.frame similar to the inputs data.frame, but with up to 6 columns.

objectName	required, character string indicating the name of the object in the simList that will be saved to disk (without th
file	optional, a character string indicating the file path to save to. The default is to concatenate objectName with the
fun	optional, a character string indicating the function to use to save that file. The default is saveRDS
package	optional character string indicating the package in which to find the fun);
saveTime	optional numeric, indicating when in simulation time the file should be saved. The default is the lowest priority
arguments	is a list of lists of named arguments, one list for each fun. For example, if fun="write.csv", arguments = li

See the modules vignette for more details (browseVignettes("SpaDES")).

Note

The automatic file type handling only adds the correct extension from a given fun and package. It does not do the inverse, from a given extension find the correct fun and package.

See Also

SpaDES, specifically the section 1.2.2 on loading and saving.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

Examples

```
#######################
# inputs
#######################

# Start with a basic empty simList
sim <- simInit()

test <- 1:10
library(igraph) # for %>%
tmpdir <- file.path(tempdir(), "inputs") %>% checkPath(create = TRUE)
tmpFile <- file.path(tmpdir, "test.rds")
saveRDS(test, file = tmpFile)
inputs(sim) <- data.frame(file = tmpFile) # using only required column, "file"
inputs(sim) # see that it is not yet loaded, but when it is scheduled to be loaded
simOut <- spades(sim)
```

```
inputs(simOut) # confirm it was loaded
simOut$test

# can put data.frame for inputs directly inside simInit call
allTifs <- dir(system.file("maps", package = "SpaDES"),
               full.names = TRUE, pattern = "tif")

# next: objectNames are taken from the filenames (without the extension)
# This will load all 5 tifs in the SpaDES sample directory, using
#   the raster fuction in the raster package, all at time = 0
if (require("rgdal", quietly = TRUE)) {
  sim <- simInit(
    inputs = data.frame(
      files = allTifs,
      functions = "raster",
      package = "raster",
      loadTime = 0,
      stringsAsFactors = FALSE)
    )

  ##############################
  #A fully described inputs object, including arguments:
  files <- dir(system.file("maps", package = "SpaDES"),
               full.names = TRUE, pattern = "tif")
  # arguments must be a list of lists. This may require I() to keep it as a list
  #   once it gets coerced into the data.frame.
  arguments = I(rep(list(native = TRUE), length(files)))
  filelist = data.frame(
     objectName = paste0("Maps", 1:5),
     files = files,
     functions = "raster::raster",
     arguments = arguments,
     loadTime = 0,
     intervals = c(rep(NA, length(files) - 1), 10)
  )
  inputs(sim) <- filelist
  spades(sim)
}


# Clean up after
unlink(tmpdir, recursive = TRUE)

#######################
# outputs
#######################

library(igraph) # for %>%
tmpdir <- file.path(tempdir(), "outputs") %>% checkPath(create = TRUE)
tmpFile <- file.path(tmpdir, "temp.rds")
tempObj <- 1:10

# Can add data.frame of outputs directly into simInit call
```

```
sim <- simInit(objects=c("tempObj"),
  outputs=data.frame(objectName="tempObj"),
  paths=list(outputPath=tmpdir))
outputs(sim) # To see what will be saved, when, what filename
sim <- spades(sim)
outputs(sim) # To see that it was saved, when, what filename

# Also can add using assignment after a simList object has been made
sim <- simInit(objects=c("tempObj"),
  paths=list(outputPath=tmpdir))
outputs(sim) <- data.frame(objectName = "tempObj", saveTime=1:10)
sim <- spades(sim)
outputs(sim) # To see that it was saved, when, what filename.

# can do highly variable saving
tempObj2 <- paste("val",1:10)
df1 <- data.frame(col1 = tempObj, col2 = tempObj2)
sim <- simInit(objects=c("tempObj", "tempObj2", "df1"),
  paths=list(outputPath=tmpdir))
outputs(sim) = data.frame(
     objectName = c(rep("tempObj",2), rep("tempObj2", 3), "df1"),
     saveTime = c(c(1,4), c(2,6,7), end(sim)),
     fun = c(rep("saveRDS", 5), "write.csv"),
     package = c(rep("base", 5), "utils"),
     stringsAsFactors = FALSE)
# since write.csv has a default of adding a column, x, with rownames, must add additional
#   argument for 6th row in data.frame (corresponding to the write.csv function)
outputArgs(sim)[[6]] <- list(row.names=FALSE)
sim <- spades(sim)
outputs(sim)

# read one back in just to test it all worked as planned
newObj <- read.csv(dir(tmpdir, pattern="second10.csv", full.name=TRUE))
newObj

# using saving with SpaDES-aware methods
# To see current ones SpaDES can do
.saveFileExtensions()

library(raster)
if (require(rgdal)) {
  ras <- raster(ncol=4, nrow=5)
  ras[] <- 1:20

  sim <- simInit(objects=c("ras"),
    paths=list(outputPath=tmpdir))
  outputs(sim) = data.frame(
       file="test",
       fun = "writeRaster",
       package = "raster",
       objectName = "ras",
       stringsAsFactors = FALSE)
```

```
  outputArgs(sim)[[1]] <- list(format="GTiff") # see ?raster::writeFormats
  simOut <- spades(sim)
  outputs(simOut)
  newRas <- raster(dir(tmpdir, full.name=TRUE, pattern=".tif"))
  all.equal(newRas, ras) # Should be TRUE
}
# Clean up after
unlink(tmpdir, recursive = TRUE)
```

---

inRange *Test whether a number lies within range* [a,b]

Description

Default values of a=0; b=1 allow for quick test if x is a probability.

Usage

```
inRange(x, a = 0, b = 1)
```

Arguments

x	values to be tested
a	lower bound (default 0)
b	upper bound (default 1)

Author(s)

Alex Chubaty

Examples

```
set.seed(100)
x <- stats::rnorm(4) # -0.50219235  0.13153117 -0.07891709  0.88678481
inRange(x, 0, 1)
```

inSeconds *Convert time units*

Description

In addition to using the lubridate package, some additional functions to work with times are provided.

This function takes a numeric with a "unit" attribute and converts it to another numeric with a different time attribute. If the units passed to argument units are the same as attr(time, "unit"), then it simply returns input time.

Usage

```
inSeconds(unit, envir)

## S4 method for signature 'character,environment'
inSeconds(unit, envir)

## S4 method for signature '`NULL`,missing'
inSeconds(unit)

## S4 method for signature 'character,missing'
inSeconds(unit)

convertTimeunit(time, unit, envir)

## S4 method for signature 'numeric,character,environment'
convertTimeunit(time, unit, envir)

## S4 method for signature 'numeric,missing,missing'
convertTimeunit(time)

## S4 method for signature 'numeric,character,missing'
convertTimeunit(time, unit)

.spadesTimes

spadesTimes()

checkTimeunit(unit, envir)

## S4 method for signature 'character,missing'
checkTimeunit(unit, envir)

## S4 method for signature 'character,environment'
checkTimeunit(unit, envir)
```

Arguments

unit	Character. One of the time units used in SpaDES or user defined time unit, given as the unit name only. See details.
envir	An environment. This is where to look up the function definition for the time unit. See details.
time	Numeric. With a unit attribute, indicating the time unit of the input numeric. See Details.

Format

An object of class character of length 6.

Details

Current pre-defined units are found within the spadesTimes() function. The user can define a new unit. The unit name can be anything, but the function definition must be of the form, "dunitName", e.g., dyear or dfortNight. The unit name is the part without the d and the function name definition includes the "d". This new function, e.g., #' `dfortNight <- function(x) lubridate::duration(dday(14))` can be placed anywhere in the search path or in a module.

Because of R scoping, if envir is a simList environment, then this function will search there first, then up the current search() path. Thus, it will find a user defined or module defined unit before a SpaDEs unit. This means that a user can override the dyear given in SpaDES, for example, which is 365.25 days, with `dyear <- function(x) lubridate::duration(dday(365))`

If time has no units attribute, then it is assumed to be seconds.

Value

A numeric vector of length 1, with unit attribute set to "seconds".

Author(s)

Alex Chubaty & Eliot McIntire

Eliot McIntire

---

layerNames *Extract the layer names of Spatial Objects*

Description

There are already methods for Raster* objects. This adds methods for SpatialPoints*, SpatialLines*, and SpatialPolygons*, returning an empty character vector of length 1. This function was created to give consistent, meaningful results for all classes of objects plotted by Plot.

Usage

```
layerNames(object)

## S4 method for signature 'list'
layerNames(object)

## S4 method for signature 'ANY'
layerNames(object)

## S4 method for signature 'Raster'
layerNames(object)

## S4 method for signature '.spadesPlot'
layerNames(object)

## S4 method for signature 'igraph'
layerNames(object)
```

Arguments

object	A Raster*, SpatialPoints*, SpatialLines*, or SpatialPolygons* object; or list of these.

Author(s)

Eliot McIntire

---

loadPackages *Load packages.*

Description

Load and optionally install additional packages.

Usage

```
loadPackages(packageList, install = FALSE, quiet = TRUE)

## S4 method for signature 'character'
loadPackages(packageList, install = FALSE,
  quiet = TRUE)

## S4 method for signature 'list'
loadPackages(packageList, install = FALSE, quiet = TRUE)

## S4 method for signature '`NULL`'
loadPackages(packageList, install = FALSE, quiet = TRUE)
```

Arguments

packageList	A list of character strings specifying the names of packages to be loaded.
install	Logical flag. If required packages are not already installed, should they be installed?
quiet	Logical flag. Should the final "packages loaded" message be suppressed?

Value

Specified packages are loaded and attached using `require()`, invisibly returning a logical vector of successes.

Author(s)

Alex Chubaty

See Also

`require`.

Examples

```
## Not run:
  pkgs <- list("ggplot2", "lme4")
  loadPackages(pkgs) # loads packages if installed
  loadPackages(pkgs, install = TRUE) # loads packages after installation (if needed)

## End(Not run)
```

---

ls.simList *List simulation objects*

Description

Return a vector of character strings giving the names of the objects in the specified simulation environment. Can be used with a `simList` object, because the method for this class is simply a wrapper for calling `ls` on the simulation environment stored in the `simList` object.

Usage

```
ls.simList(name)

## S4 method for signature 'simList'
ls(name)

objects.simList(name)

## S4 method for signature 'simList'
objects(name)
```

Arguments

name	A simList object.

See Also

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.str.simList, modules, objs, packages, params, paths, progressInterval, times

---

ls.str.simList *List simulation objects and their structure*

Description

A variation of applying str to each matched name. Can be used with a simList object, because the method for this class is simply a wrapper for calling ls on the simulation environment stored in the simList object.

export

Usage

```
ls.str.simList(name)

## S4 method for signature 'missing,simList'
ls.str(name)

## S4 method for signature 'simList,missing'
ls.str(pos)
```

Arguments

name	A simList object.
pos	A simList object, used only if name not provided.

See Also

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, modules, objs, packages, params, paths, progressInterval, times

makeLines *Make* SpatialLines *object from two* SpatialPoints *objects*

Description

The primary conceived usage of this is to draw arrows following the trajectories of agents.

Usage

```
makeLines(from, to)

## S4 method for signature 'SpatialPoints,SpatialPoints'
makeLines(from, to)
```

Arguments

from	Starting spatial coordinates (SpatialPointsDataFrame).
to	Ending spatial coordinates (SpatialPointsDataFrame).

Value

A SpatialLines object. When this object is used within a Plot call and the length argument is specified, then arrow heads will be drawn. See examples.

Author(s)

Eliot McIntire

Examples

```
library(sp)
# Make 2 objects
caribou1 <- SpatialPoints(cbind(x = stats::runif(10, -50, 50), y = stats::runif(10, -50, 50)))
caribou2 <- SpatialPoints(cbind(x = stats::runif(10, -50, 50), y = stats::runif(10, -50, 50)))

caribouTraj <- makeLines(caribou1, caribou2)
  if (interactive()) {
    clearPlot()
    Plot(caribouTraj, length = 0.1)
  }

# or  to a previous Plot
## Not run:
filelist <- data.frame(files =
     dir(file.path(find.package("SpaDES", quiet = FALSE), "maps"),
         full.names = TRUE, pattern = "tif"),
     functions = "rasterToMemory",
     packages = "SpaDES")
```

```
# Load files to memory (using rasterToMemory)
sim1 <- loadFiles(filelist = filelist)
caribouTraj <- makeLines(caribou1, caribou2)

  if (interactive()) {
    clearPlot()
    Plot(sim1$DEM)
    Plot(caribouTraj, addTo = "sim1$DEM", length = 0.1)
  }

## End(Not run)
```

`maxTimeunit` *Determine the largest timestep unit in a simulation*

Description

Determine the largest timestep unit in a simulation

Usage

```
maxTimeunit(sim)

## S4 method for signature 'simList'
maxTimeunit(sim)
```

Arguments

`sim`	A `simList` simulation object.

Value

The timeunit as a character string. This defaults to `NA` if none of the modules has explicit units.

Author(s)

Eliot McIntire and Alex Chubaty

mergeRaster *Merge split raster tiles into a single raster layer.*

Description

Recombine the split tiles from splitRaster into a single RasterLayer.

Usage

```
mergeRaster(x)

## S4 method for signature 'list'
mergeRaster(x)
```

Arguments

x	A list of split raster tiles (i.e., from splitRaster).

Details

mergeRaster differs from merge in how overlapping tile regions are handled: merge retains the values of the first raster in the list. This has the consequence of retaining the values from the buffered region in the first tile in place of the values from the neighbouring tile. On the otherhand, mergeRaster retains the values of the tile region, over the values in any buffered regions. This is useful for reducing edge effects when performing raster operations involving contagious processes. To use the average of cell values, or do another computation, use mosaic.

Value

A RasterLayer object.

Author(s)

Yong Luo and Alex Chubaty

See Also

merge, mosaic

Examples

```
require(raster)

# an example with dimensions:
# nrow = 77
# ncol = 101
# nlayers = 3
b <- brick(system.file("external/rlogo.grd", package = "raster"))
r <- b[[1]] # use first layer only
```

```
nx <- 3
ny <- 4

tmpdir <- file.path(tempdir(), "splitRaster-example")

y0 <- splitRaster(r, nx, ny, path = file.path(tmpdir, "y0")) # no buffer

# buffer: 10 pixels along both axes
y1 <- splitRaster(r, nx, ny, c(10, 10), path = file.path(tmpdir, "y1"))

# buffer: half the width and length of each tile
y2 <- splitRaster(r, nx, ny, c(0.5, 0.5), path = file.path(tmpdir, "y2"))

# parallel cropping
if (interactive()) {
  n <- pmin(parallel::detectCores(), 4) # use up to 4 cores
  beginCluster(n)
  y3 <- splitRaster(r, nx, ny, c(0.7, 0.7), path = file.path(tmpdir, "y3"))
  endCluster()
}

# the original raster:
if (interactive()) plot(r) # may require a call to `dev()` if using RStudio

# the split raster:
layout(mat = matrix(seq_len(nx*ny), ncol = nx, nrow = ny))
plotOrder <- c(4, 8, 12, 3, 7, 11, 2, 6, 10, 1, 5, 9)
if (interactive()) invisible(lapply(y0[plotOrder], plot))

# can be recombined using `raster::merge`
m0 <- do.call(merge, y0)
all.equal(m0, r) ## TRUE

m1 <- do.call(merge, y1)
all.equal(m1, r) ## TRUE

m2 <- do.call(merge, y2)
all.equal(m2, r) ## TRUE

# or recombine using SpaDES::mergeRaster
n0 <- mergeRaster(y0)
all.equal(n0, r) ## TRUE

n1 <- mergeRaster(y1)
all.equal(n1, r) ## TRUE

n2 <- mergeRaster(y2)
all.equal(n2, r) ## TRUE

unlink(tmpdir, recursive = TRUE)
```

minTimeunit *Determine the smallest timeunit in a simulation*

Description

When modules have different timeunit, SpaDES automatically takes the smallest (e.g., "second") as the unit for a simulation.

Usage

```
minTimeunit(sim)

## S4 method for signature 'simList'
minTimeunit(sim)

## S4 method for signature 'list'
minTimeunit(sim)
```

Arguments

sim	A simList simulation object.

Value

The timeunit as a character string. This defaults to "second" if none of the modules has explicit units.

Author(s)

Eliot McIntire

moduleCoverage *Calculate module coverage of unit tests*

Description

Calculate the test coverage by unit tests for the module and its functions.

Usage

```
moduleCoverage(name, path)

## S4 method for signature 'character,character'
moduleCoverage(name, path)

## S4 method for signature 'character,missing'
moduleCoverage(name)
```

Arguments

name	Character string. The module's name.
path	Character string. The path to the module directory (default is the current working directory).

Value

Return a list of two coverage objects and two data.table objects. The two coverage objects are named 'moduleCoverage' and 'functionCoverage'. The 'moduleCoverage' object contains the percent value of unit test coverage for the module. The 'functionCoverage' object contains percentage values for unit test coverage for each function defined in the module. Please use shine to view the coverage information. Two data.tables give the information of all the tested and untested functions in the module.

Note

When running this function, the test files must be strictly placed in the 'tests/testthat/' directory under module path. To automatically generate this folder, please set unitTests = TRUE when creating a new module using newModule. To accurately test your module, the test filename must follw the format test-functionName.R.

Author(s)

Yong Luo

See Also

newModule.

Examples

```
## Not run:
 library(igraph) # for %>%
 library(SpaDES)
 tmpdir <- file.path(tempdir(), "coverage")
 modulePath <- file.path(tmpdir, "Modules") %>% checkPath(create = TRUE)
 moduleName <- "forestAge" # sample module to test
 downloadModule(name = moduleName, path = modulePath) # download sample module
 testResults <- moduleCoverage(name = moduleName, path = modulePath)
 shine(testResults$moduleCoverage)
 shine(testResults$functionCoverage)
 unlink(tmpdir, recursive = TRUE)

## End(Not run)
```

moduleDiagram *Simulation module dependency diagram*

Description

Create a network diagram illustrating the simplified module dependencies of a simulation. Offers a less detailed view of specific objects than does plotting the depsEdgeList directly with objectDiagram.

Usage

```
moduleDiagram(sim, type, showParents, ...)

## S4 method for signature 'simList,character,logical'
moduleDiagram(sim, type, showParents, ...)

## S4 method for signature 'simList,missing,ANY'
moduleDiagram(sim, type, showParents, ...)
```

Arguments

sim	A simList object (typically corresponding to a completed simulation).
type	Character string, either "rgl" for igraph::rglplot or "tk" for igraph::tkplot. Default missing, which uses regular plot.
showParents	Logical. If TRUE, then any children that are grouped into parent modules will be grouped together by colored blobs. Internally, this is calling moduleGraph. Default FALSE.
...	Additional arguments passed to plotting function specfied by type.

Value

Plots module dependency diagram.

Author(s)

Alex Chubaty

See Also

igraph, moduleGraph for a version that accounts for parent and children module structure.

moduleGraph *Build a module dependency graph*

Description

This is still experimental, but this will show the hierarchical structure of parent and children modules and return a list with an igraph object and an igraph communities object, showing the groups. Currently only tested with relatively simple structures.

Usage

```
moduleGraph(sim, plot, ...)

## S4 method for signature 'simList,logical'
moduleGraph(sim, plot, ...)

## S4 method for signature 'simList,missing'
moduleGraph(sim, plot, ...)
```

Arguments

sim	A simList object.
plot	Logical indicating whether the edgelist (and subsequent graph) will be used for plotting. If TRUE, duplicated rows (i.e., multiple object dependencies between modules) are removed so that only a single arrow is drawn connecting the modules. Default is FALSE.
...	Arguments passed to Plot

Value

A list with 2 elements, an igraph object and an igraph communities object.

@importFrom igraph graph_from_data_frame cluster_optimal edges # already with import igraph

Author(s)

Eliot McIntire

See Also

moduleDiagram

`moduleMetadata` *Parse and extract module metadata*

Description

Parse and extract module metadata

Usage

```
moduleMetadata(module, path, sim)

## S4 method for signature 'character,character,ANY'
moduleMetadata(module, path)

## S4 method for signature 'character,missing,missing'
moduleMetadata(module)

## S4 method for signature 'ANY,missing,simList'
moduleMetadata(module, sim)
```

Arguments

`module`	Character string. Your module's name.
`path`	Character string specifying the file path to modules directory. Default is to use the `spades.modulePath` option.
`sim`	A `simList` simulation object, generally produced by `simInit`.

Value

A list of module metadata, matching the structure in `defineModule`.

Author(s)

Alex Chubaty

See Also

`defineModule`

Examples

```
path <- system.file(package = "SpaDES", "sampleModules")
sampleModules <- dir(path)
x <- moduleMetadata(sampleModules[3], path)

# using simList
mySim <- simInit(
   times = list(start = 2000.0, end = 2002.0, timeunit = "year"),
```

```
  params = list(
    .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
  ),
  modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
  paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
)
moduleMetadata(sim = mySim)
```

`modules` *Simulation modules and dependencies*

Description

Accessor functions for the `depends` and `modules` slots in a `simList` object. These are included for advanced users.

`depends`	List of simulation module dependencies. (advanced)
`modules`	List of simulation modules to be loaded. (advanced)
`inputs`	List of loaded objects used in simulation. (advanced)

Usage

```
modules(object, hidden = FALSE)

## S4 method for signature '.simList'
modules(object, hidden = FALSE)

modules(object) <- value

## S4 replacement method for signature '.simList'
modules(object) <- value

depends(object)

## S4 method for signature '.simList'
depends(object)

depends(object) <- value

## S4 replacement method for signature '.simList'
depends(object) <- value
```

Arguments

`object`	A `simList` simulation object.

hidden	Logical. If TRUE, show the default core modules.
value	The object to be stored at the slot.

Details

Currently, only get and set methods are defined. Subset methods are not.

Value

Returns or sets the value of the slot from the simList object.

Author(s)

Alex Chubaty

See Also

SpaDES, specifically the section 1.2.7 on Modules and dependencies.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, objs, packages, params, paths, progressInterval, times

---

move *Move*

Description

Wrapper for selecting different animal movement methods.

This version uses just turn angles and step lengths to define the correlated random walk.

Usage

```
move(hypothesis = "crw", ...)

crw(agent, extent, stepLength, stddev, lonlat, torus = FALSE)

## S4 method for signature 'SpatialPointsDataFrame'
crw(agent, extent, stepLength, stddev,
  lonlat, torus = FALSE)

## S4 method for signature 'SpatialPoints'
crw(agent, extent, stepLength, stddev, lonlat,
  torus = FALSE)
```

Arguments

hypothesis	Character vector, length one, indicating which movement hypothesis/method to test/use. Currently defaults to 'crw' (correlated random walk) using crw.
...	arguments passed to the function in hypothesis
agent	A SpatialPoints* object. If a SpatialPointsDataFrame, 2 of the columns must be x1 and y1, indicating the previous location. If a SpatialPoints object, then x1 and y1 will be assigned randomly.
extent	An optional Extent object that will be used for torus.
stepLength	Numeric vector of length 1 or number of agents describing step length.
stddev	Numeric vector of length 1 or number of agents describing standard deviation of wrapped normal turn angles.
lonlat	Logical. If TRUE, coordinates should be in degrees. If FALSE coordinates represent planar ('Euclidean') space (e.g. units of meters)
torus	Logical. Should the crw movement be wrapped to the opposite side of the map, as determined by the extent argument. Default FALSE.

Details

This simple version of a correlated random walk is largely the version that was presented in Turchin 1998, but it was also used with bias modifications in McIntire, Schultz, Crone 2007.

Value

A SpatialPointsDataFrame object with updated spatial position defined by a single occurence of step length(s) and turn angle(s).

Author(s)

Eliot McIntire

Eliot McIntire

References

Turchin, P. 1998. Quantitative analysis of movement: measuring and modeling population redistribution in animals and plants. Sinauer Associates, Sunderland, MA.

McIntire, E. J. B., C. B. Schultz, and E. E. Crone. 2007. Designing a network for butterfly habitat restoration: where individuals, populations and landscapes interact. Journal of Applied Ecology 44:725-736.

See Also

pointDistance

newModule *Create new module from template.*

Description

Autogenerate a skeleton for a new SpaDES module, a template for a documentation file, a citation file, a license file, a README.txt file, and a folder that contains unit tests information. The newModuleDocumentation will not generate the module file, but will create the other files.

Usage

```
newModule(name, path, ...)

## S4 method for signature 'character,character'
newModule(name, path, ...)

## S4 method for signature 'character,missing'
newModule(name, path, ...)
```

Arguments

name	Character string specfying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
...	Additonal arguments. Currently, only the following are supported:
	open. Logical. Should the new module file be opened after creation? Default TRUE.
	unitTests. Logical. Should the new module include unit test files? Default TRUE. Unit testing relies on the testthat package.
	type. Character string specifying one of "child" (default), or "parent".
	children. Required when type = "parent". A character vector specifying the names of child modules.

Details

All files will be created within a subfolder named name within the path.

Value

Nothing is returned. The new module file is created at 'path/name.R', as well as ancillary files for documentation, citation, license, readme, and unit tests folder.

Note

On Windows there is currently a bug in RStudio that prevents the editor from opening when `file.edit` is called. Similarly, in RStudio on OS X / macOS, there is an issue opening files where they are opened in an overlayed window rather than a new tab. `file.edit` does work if the user types it at the command prompt. A message with the correct lines to copy and paste is provided.

Author(s)

Alex Chubaty and Eliot McIntire

See Also

Other module creation helpers: `newModuleCode`, `newModuleDocumentation`, `newModuleTests`

Examples

```
## Not run:
  ## create a "myModule" module in the "modules" subdirectory.
  newModule("myModule", "modules")

  ## create a new parent module in the "modules" subdirectory.
  newModule("myParentModule", "modules", type = "parent", children = c("child1", "child2"))

## End(Not run)
```

---

newModuleCode *Create new module code file*

Description

Create new module code file

Usage

```
newModuleCode(name, path, open, type, children)

## S4 method for signature 'character,character,logical,character,character'
newModuleCode(name,
  path, open, type, children)
```

Arguments

name	Character string specifying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
open	Logical. Should the new module file be opened after creation? Default TRUE.
type	Character string specifying one of "child" (default), or "parent".
children	Required when type = "parent". A character vector specifying the names of child modules.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Other module creation helpers: newModuleDocumentation, newModuleTests, newModule

---

newModuleDocumentation

Create new module documentation

Description

Create new module documentation

Usage

```
newModuleDocumentation(name, path, open, type, children)


  ## S4 method for signature 'character,character,logical,character,character'
newModuleDocumentation(name,
  path, open, type, children)

## S4 method for signature 'character,missing,logical,ANY,ANY'
newModuleDocumentation(name, open)

## S4 method for signature 'character,character,missing,ANY,ANY'
newModuleDocumentation(name,
  path)

## S4 method for signature 'character,missing,missing,ANY,ANY'
newModuleDocumentation(name)
```

Arguments

name	Character string specifying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
open	Logical. Should the new module file be opened after creation? Default TRUE.
type	Character string specifying one of "child" (default), or "parent".
children	Required when type = "parent". A character vector specifying the names of child modules.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Other module creation helpers: newModuleCode, newModuleTests, newModule

---

newModuleTests *Create template testing structures for new modules*

Description

Create template testing structures for new modules

Usage

```
newModuleTests(name, path, open)

## S4 method for signature 'character,character,logical'
newModuleTests(name, path, open)
```

Arguments

name	Character string specifying the name of the new module.
path	Character string. Subdirectory in which to place the new module code file. The default is the current working directory.
open	Logical. Should the new module file be opened after creation? Default TRUE.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Other module creation helpers: newModuleCode, newModuleDocumentation, newModule

newPlot *Open a new plotting window*

Description

Open a new plotting window

Usage

```
newPlot(noRStudioGD = TRUE, ...)
```

Arguments

noRStudioGD	Logical Passed to dev.new. Default is TRUE to avoid using RStudio graphics device, which is slow.
...	Additional arguments.

Note

dev.new is supposed to be the correct way to open a new window in a platform-generic way; however, doesn't work in RStudio (#116).

Author(s)

Eliot McIntire and Alex Chubaty

See Also

dev.

newProgressBar *Progress bar*

Description

Shows a progress bar that is scaled to simulation end time.

Usage

```
newProgressBar(sim)
```

Arguments

sim	A simList simulation object.

Details

The progress bar object is stored in a separate environment, `.spadesEnv`.

Author(s)

Alex Chubaty

Eliot McIntire

`normPath` *Normalize filepath*

Description

Checks the specified filepath for formatting consistencies: 1) use slash instead of backslash; 2) do tilde etc. expansion; 3) remove trailing slash.

Usage

```
normPath(path)

## S4 method for signature 'character'
normPath(path)

## S4 method for signature 'list'
normPath(path)

## S4 method for signature '`NULL`'
normPath(path)

## S4 method for signature 'missing'
normPath()
```

Arguments

`path`	A character vector of filepaths.

Value

Character vector of cleaned up filepaths.

numAgents *SELES - Number of Agents to initiate*

Description

Sets the the number of agents to initiate. THIS IS NOT YET FULLY IMPLEMENTED.

A SELES-like function to maintain conceptual backwards compatability with that simulation tool. This is intended to ease transitions from SELES.

You must know how to use SELES for these to be useful.

Usage

```
numAgents(N, probInit)
```

Arguments

N	Number of agents to intitate (integer scalar).
probInit	Probability of initalizing an agent at the location.

Value

A numeric, indicating number of agents to start

Author(s)

Eliot McIntire

numLayers *Find the number of layers in a Spatial Object*

Description

There are already methods for Raster* in the raster package. Adding methods for list, SpatialPolygons, SpatialLines, and SpatialPoints, gg, histogram, igraph. These latter classes return 1.

Usage

```
numLayers(x)

## S4 method for signature 'list'
numLayers(x)

## S4 method for signature '.spadesPlot'
numLayers(x)
```

```
## S4 method for signature 'Raster'
numLayers(x)

## S4 method for signature 'Spatial'
numLayers(x)

## S4 method for signature 'ANY'
numLayers(x)
```

Arguments

x	A .spadesPlotObjects object or list of these.

Value

The number of layers in the object.

Author(s)

Eliot McIntire

---

objectDiagram *Simulation object dependency diagram*

Description

Create a sequence diagram illustrating the data object dependencies of a simulation. Offers a more detailed view of specific objects than does plotting the depsEdgeList directly with moduleDiagram.

Usage

```
objectDiagram(sim, ...)

## S4 method for signature 'simList'
objectDiagram(sim, ...)
```

Arguments

sim	A simList object (typically corresponding to a completed simulation).
...	Additional arguments passed to mermaid. Useful for specifying height and width.

Value

Plots a sequence diagram, invisibly returning a mermaid object.

Author(s)

Alex Chubaty

See Also

mermaid.

---

objs *Extract or replace an object from the simulation environment*

Description

The [[and $ operators provide "shortcuts" for accessing objects in the simulation environment. I.e., instead of using envir(sim)$object or envir(sim)[["object"]], one can simply use sim$object or sim[["object"]].

Usage

```
objs(x, ...)

## S4 method for signature 'simList'
objs(x, ...)

objs(x) <- value

## S4 replacement method for signature 'simList'
objs(x) <- value

## S4 method for signature 'simList,ANY,ANY'
x[[i, j, ..., drop]]

## S4 replacement method for signature 'simList,ANY,ANY,ANY'
x[[i]] <- value

## S4 method for signature 'simList'
x$name

## S4 replacement method for signature 'simList'
x$name <- value
```

Arguments

x	A simList object from which to extract element(s) or in which to replace element(s).
...	see i.
value	Any R object.

i	Indices specifying elements to extract or replace.
j	see i.
drop	not implemented.
name	A literal character string or a name.

Details

objs can take ... arguments passed to ls, allowing, e.g. all.names=TRUE objs<- requires takes a named list of values to be assigned in the simulation envirment.

Value

Returns or sets a list of objects in the simList environment.

See Also

SpaDES, specifically the section 1.2.1 on Simulation Parameters.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, packages, params, paths, progressInterval, times

---

openModules *Open all modules nested within a base directory*

Description

This is just a convenience wrapper for opening several modules at once, recursively. A module is defined as any file that ends in .R or .r and has a directory name identical to its filename. Thus, this must be case sensitive.

Usage

```
openModules(name, path)

## S4 method for signature 'character,character'
openModules(name, path)

## S4 method for signature 'missing,missing'
openModules()

## S4 method for signature 'missing,character'
openModules(path)

## S4 method for signature 'character,missing'
openModules(name)

## S4 method for signature 'simList,missing'
openModules(name)
```

Arguments

name	Character vector with names of modules to open. If missing, then all modules will be opened within the basedir.
path	Character string of length 1. The base directory within which there are only module subdirectories.

Value

Nothing is returned. All file are open via `file.edit`.

Note

On Windows there is currently a bug in RStudio that prevents the editor from opening when `file.edit` is called. `file.edit` does work if the user types it at the command prompt. A message with the correct lines to copy and paste is provided.

Author(s)

Eliot McIntire

Examples

```
## Not run: openModules("~\SpaDESModules")
```

---

packages *Get simulation package dependencies*

Description

Get simulation package dependencies

Usage

```
packages(sim)

## S4 method for signature '.simList'
packages(sim)
```

Arguments

sim	A `simList` object.

Value

A sorted character vector of package names.

Author(s)

Alex Chubaty

See Also

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, params, paths, progressInterval, times

---

paddedFloatToChar *Convert numeric to character with padding*

Description

Convert numeric to character with padding

Usage

```
paddedFloatToChar(x, padL = ceiling(log10(x + 1)), padR = 3, pad = "0")
```

Arguments

x	numeric. Number to be converted to character with padding
padL	numeric. Desired number of digits on left side of decimal. If not enough, pad will be used to pad.
padR	numeric. Desired number of digits on right side of decimal. If not enough, pad will be used to pad.
pad	character to use as padding (nchar(pad)==1 must be TRUE). Passed to str_pad

Value

Character string representing the filename.

Author(s)

Eliot McIntire and Alex Chubaty

Examples

```
paddedFloatToChar(1.25)
paddedFloatToChar(1.25, padL = 3, padR = 5)
```

params *Get and set simulation parameters.*

Description

params and P access the parameter slot in the simList. params has a replace method, so can be used to update a parameter value.

P is a concise way to access parameters within a module. It works more like a namespaced function in the sense that the module from which it is called is the default place it will look for the parameter. To access a parameter from within a module, you can use P(sim)$paramName instead of params(sim)$moduleName$paramName

Usage

```
params(object)

## S4 method for signature '.simList'
params(object)

params(object) <- value

## S4 replacement method for signature '.simList'
params(object) <- value

P(object, module = NULL, param = NULL)

## S4 method for signature '.simList'
P(object, module = NULL, param = NULL)

parameters(object, asDF = FALSE)

## S4 method for signature '.simList'
parameters(object, asDF = FALSE)

p(object, module = NULL, param = NULL)
```

Arguments

object	A simList simulation object.
value	The object to be stored at the slot.
module	Optional character string indicating which module params should come from.
param	Optional character string indicating which parameter to choose.
asDF	Logical. For parameters, if TRUE, this will produce a single data.frame of all model parameters. If FALSE, then it will return a data.frame with 1 row for each parameter within nested lists, with the same structure as params.

Value

Returns or sets the value of the slot from the `simList` object.

Note

The differences between P, params and being explicit with passing arguments are mostly a question of speed and code compactness. The computationally fastest way to get a parameter is to specify moduleName and parameter name, as in: `P(sim, "moduleName", "paramName")` (replacing moduleName and paramName with your specific module and parameter names), but it is more verbose than P(sim)$paramName. Note: the important part for speed (e.g., 2-4x faster) is specifying the moduleName. Specifying the parameter name is <5

`P` is a function in `shiny` and html packages. This can produce namespace clashes that are difficult to detect, as the errors are not meaningful. The name of this function may be changed in the future to remove this potential conflict. In the mean time, use SpaDES::P(sim) if you are concerned with the potential conflicts with a shiny app.

See Also

SpaDES, specifically the section 1.2.1 on Simulation parameters.

Other functions to access elements of a `simList` object: `.addDepends`, `doEvent.checkpoint`, `envir`, `events`, `globals`, `inputs`, `ls.simList`, `ls.str.simList`, `modules`, `objs`, `packages`, `paths`, `progressInterval`, `times`

Examples

```
modules = list("randomLandscapes")
paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
mySim <- simInit(modules = modules, paths = paths,
                 params = list(.globals = list(stackName = "landscape")))
parameters(mySim)
```

---

patchSize *Patch size*

Description

Patch size

Usage

```
patchSize(patches)
```

Arguments

`patches`	Number of patches.

paths *Specify paths for modules, inputs, and outputs*

Description

Accessor functions for the paths slot in a simList object.

Usage

```
paths(object)

## S4 method for signature '.simList'
paths(object)

paths(object) <- value

## S4 replacement method for signature '.simList'
paths(object) <- value

cachePath(object)

## S4 method for signature '.simList'
cachePath(object)

cachePath(object) <- value

## S4 replacement method for signature '.simList'
cachePath(object) <- value

inputPath(object)

## S4 method for signature '.simList'
inputPath(object)

inputPath(object) <- value

## S4 replacement method for signature '.simList'
inputPath(object) <- value

outputPath(object)

## S4 method for signature '.simList'
outputPath(object)

outputPath(object) <- value

## S4 replacement method for signature '.simList'
```

```
outputPath(object) <- value

modulePath(object)

## S4 method for signature '.simList'
modulePath(object)

modulePath(object) <- value

## S4 replacement method for signature '.simList'
modulePath(object) <- value
```

Arguments

object	A simList simulation object.
value	The object to be stored at the slot.

Details

These are ways to add or access the file paths used by spades. There are four file paths: cachePath, modulePath, inputPath, and outputPath. Each has a function to get or set the value in a simList object. If no paths are specified, the defaults are as follows:

- cachePath: getOption("spades.cachePath");
- inputPath: getOption("spades.modulePath");
- modulePath: getOption("spades.inputPath");
- inputPath: getOption("spades.outputPath").

Value

Returns or sets the value of the slot from the simList object.

See Also

SpaDES, specifically the section 1.2.4 on Simulation Paths.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, progressInterval, times

Plot *Plot: Fast, optimally arranged, multipanel plotting function with SpaDES*

Description

The main plotting function accompanying SpaDES. This can take objects of type Raster*, SpatialPoints*, SpatialPolygons*, and any combination of those. These can be provided as individual objects, or a named list. If a named list, the names either represent a different original object in the calling environment and that will be used, or if the names don't exist in the calling environment, then they will be copied to the .spadesEnv for reuse later. It can also handle ggplot2 objects or base::histogram objects created via call to exHist <- hist(1:10, plot = FALSE). It can also take arguments as if it were a call to plot. In this latter case, the user should be explicit about naming the plot area using addTo. Customization of the ggplot2 elements can be done as a normal ggplot2 plot, then added with Plot(ggplotObject).

Re-plot to a specific device

Usage

```
Plot(..., new = FALSE, addTo = NULL, gp = gpar(), gpText = gpar(),
  gpAxis = gpar(), axes = FALSE, speedup = 1, size = 5, cols = NULL,
  col = NULL, zoomExtent = NULL, visualSqueeze = NULL, legend = TRUE,
  legendRange = NULL, legendText = NULL, pch = 19, title = TRUE,
  na.color = "#FFFFFF00", zero.color = NULL, length = NULL, arr = NULL,
  plotFn = "plot")

## S4 method for signature 'ANY'
Plot(..., new = FALSE, addTo = NULL, gp = gpar(),
  gpText = gpar(), gpAxis = gpar(), axes = FALSE, speedup = 1,
  size = 5, cols = NULL, col = NULL, zoomExtent = NULL,
  visualSqueeze = NULL, legend = TRUE, legendRange = NULL,
  legendText = NULL, pch = 19, title = TRUE, na.color = "#FFFFFF00",
  zero.color = NULL, length = NULL, arr = NULL, plotFn = "plot")

## S4 method for signature 'simList'
Plot(..., new = FALSE, addTo = NULL, gp = gpar(),
  gpText = gpar(), gpAxis = gpar(), axes = FALSE, speedup = 1,
  size = 5, cols = NULL, col = NULL, zoomExtent = NULL,
  visualSqueeze = NULL, legend = TRUE, legendRange = NULL,
  legendText = NULL, pch = 19, title = TRUE, na.color = "#FFFFFF00",
  zero.color = NULL, length = NULL, arr = NULL, plotFn = "plot")

rePlot(toDev = dev.cur(), fromDev = dev.cur(), ...)
```

Arguments

...	A combination of spatialObjects or some non-spatial objects. See details.

new	Logical. If TRUE, then the previous named plot area is wiped and a new one made; if FALSE, then the ... plots will be added to the current device, adding or rearranging the plot layout as necessary. Default is FALSE. This currently works best if there is only one object being plotted in a given Plot call. However, it is possible to pass a list of logicals to this, matching the length of the ... objects. Use clearPlot to clear the whole plotting device.
addTo	Character vector, with same length as .... This is for overplotting, when the overplot is not to occur on the plot with the same name, such as plotting a SpatialPoints* object on a RasterLayer.
gp	A gpar object, created by gpar function, to change plotting parameters (see grid package).
gpText	A gpar object for the title text. Default gpar(col = "black").
gpAxis	A gpar object for the axes. Default gpar(col = "black").
axes	Logical or "L", representing the left and bottom axes, over all plots.
speedup	Numeric. The factor by which the number of pixels is divided by to plot rasters. See Details.
size	Numeric. The size, in points, for SpatialPoints symbols, if using a scalable symbol.
cols	(also col) Character vector or list of character vectors of colours. See details.
col	(also cols) Alternative to cols to be consistent with plot. cols takes precedence, if both are provided.
zoomExtent	An Extent object. Supplying a single extent that is smaller than the rasters will call a crop statement before plotting. Defaults to NULL. This occurs after any downsampling of rasters, so it may produce very pixelated maps.
visualSqueeze	Numeric. The proportion of the white space to be used for plots. Default is 0.75.
legend	Logical idicating whether a legend should be drawn. Default is TRUE.
legendRange	Numeric vector giving values that, representing the lower and upper bounds of a legend (i.e., 1:10 or c(1,10) will give same result) that will override the data bounds contained within the grobToPlot.
legendText	Character vector of legend value labels. Defaults to NULL, which results in a pretty numeric representation. If Raster* has a Raster Attribute Table (rat; see raster package), this will be used by default. Currently, only a single vector is accepted. The length of this must match the length of the legend, so this is mosty useful for discrete-valued rasters.
pch	see ?par.
title	Logical or character string. If logical, it indicates whether to print the object name as the title above the plot. If a character string, it will print this above the plot. NOTE: the object name is used with addTo, not the title.
na.color	Character string indicating the color for NA values. Default transparent.
zero.color	Character string indicating the color for zero values, when zero is the minimum value, otherwise, zero is treated as any other color. Default transparent.
length	Numeric. Optional length, in inches, of the arrow head.

arr	A vector of length 2 indicating a desired arrangement of plot areas indicating number of rows, number of columns. Default NULL, meaning let Plot function do it automatically.
plotFn	An optional function name to do the plotting internally, e.g., "barplot" to get a barplot() call. Default "plot".
toDev	Numeric. Which device should the new rePlot be plotted to. Default is current device.
fromDev	Numeric. Which device should the replot information be taken from. Default is current device

Details

NOTE: Plot uses the `grid` package; therefore, it is NOT compatible with base R graphics. Also, because it does not by default wipe the plotting device before plotting, a call to `clearPlot` is helpful to resolve many errors. Careful use of the other device tools, such as `dev.off()` and `dev.list()` might also clear problems that may arise.

If `new = TRUE`, a new plot will be generated, but only in the figure region that has the same name as the object being plotted. This is different than calling `clearPlot(); Plot(Object)`, i.e,. directly before creating a new Plot. `clearPlot()` will clear the entire plotting device. When `new = FALSE`, any plot that already exists will be overplotted, while plots that have not already been plotted will be added. This function rearranges the plotting device to maximize the size of all the plots, minimizing white space. If using the RStudio IDE, it is recommended to make and use a new device with `dev()`, because the built in device is not made for rapid redrawing. The function is based on the grid package.

Each panel in the multipanel plot must have a name. This name is used to overplot, rearrange the plots, or overlay using `addTo` when necessary. If the `...` are named spatialObjects, then `Plot` will use these names. However, this name will not persist when there is a future call to `Plot` that forces a rearrangement of the plots. A more stable way is to use the object names directly, and any layer names (in the case of `RasterLayer` or `RasterStack` objects). If plotting a RasterLayer and the layer name is "layer" or the same as the object name, then, for simplicity, only the object name will be used. In other words, only enough information is used to uniquely identify the plot.

Because of modularity, Plot must have access to the original objects that were plotted. These objects will be used if a subsequent Plot event forces a rearrangement of the Plot device. Rather than saving all the plot information (including the data) at each Plot call (this is generally too much data to constantly make copies), the function saves a pointer to the original R object. If the plot needs to be rearranged because of a future addition, then Plot will search for that original object that created the first plots, and replot them. This has several consequences. First, that object must still exist and in the same environment. Second, if that object has changed between the first time it is plot and any subsequent time it is replotted (via a forced rearrangement), then it will take the object *as it exists*, not as it existed. Third, if passing a named list of objects, Plot will either create a link to objects with those names in the calling environment (e.g., .GlobalEnv) or, if they do not exist, then Plot will make a copy in the hidden .spadesEnv for later reuse.

`cols` is a vector of colours that can be understood directly, or by `colorRampePalette`, such as `c("orange", "blue")`, will give a colour range from orange to blue, interploated. If a list, it will be used, in order, for each item to be plotted. It will be recycled if it is shorter than the objects to be plotted. Note that when this approach to setting colours is used, any overplotting will revert to the `colortable` slot of the object, or the default for rasters, which is `terrain.color()`

cols can also accept RColorBrewer colors by keyword if it is character vector of length 1. i.e., this cannot be used to set many objects by keyword in the same Plot call. Default terrain.color(). See Details.

Some coloring will be automatic. If the object being plotted is a Raster, then this will take the colorTable slot (can be changed via setColors() or other ways). If this is a SpatialPointsDataFrame, this function will use a column called colors and apply these to the symbols.

Silently, one hidden object is made, .spadesPlot in the .spadesEnv environment, which is used for arranging plots in the device window, and identifying the objects to be replotted if rearranging is required, subsequent to a new = FALSE additional plot.

This function is optimized to allow modular Plotting. This means that several behaviours will appear unusual. For instance, if a first call to Plot is made, the legend will reflect the current color scheme. If a second or subsequent call to Plot is made with the same object but with different colours (e.g., with cols), the legend will not update. This behaviour is made with the decision that the original layer takes precedence and all subsequent plots to that same frame are overplots only.

speedup is not a precise number because it is faster to plot an un-resampled raster if the new resampling is close to the original number of pixels. At the moment, for rasters, this is set to 1/3 of the original pixels. In other words, speedup will not do anything if the factor for speeding up is not high enough (i.e., >3). If no sub-sampling is desired, use a speedup value less than 0.1.

These gp* parameters will specify plot parameters that are available with gpar(). gp will adjust plot parameters, gpText will adjust title and legend text, gpAxis will adjust the axes. size adjusts point size in a SpatialPoints object. These will persist with the original Plot call for each individual object. Multiple entries can be used, but they must be named list elements and they must match the ... items to plot. This is true for a RasterStack also, i.e., the list of named elements must be the same length as the number of layers being plotted. The naming convention used is: RasterStackName$layerName, i.e, landscape$DEM.

Value

Invisibly returns the .spadesPlot class object. If this is assigned to an object, say obj, then this can be plotted again with Plot(obj). This object is also stored in the locked .spadesEnv, so can simply be replotted with rePlot() or on a new device with rePlot(n), where n is the new device number.

Author(s)

Eliot McIntire

See Also

clearPlot, gpar, raster, par, SpatialPolygons, grid.polyline, ggplot, dev

Examples

```
## Not run:
library(sp)
library(raster)
library(rgdal)
library(igraph)
```

```
library(RColorBrewer)
# Make list of maps from package database to load, and what functions to use to load them
filelist <-
   data.frame(files =
     dir(file.path(
       find.package("SpaDES", quiet = FALSE), "maps"),
       full.names = TRUE, pattern= "tif"),
     functions = "rasterToMemory",
     packages = "SpaDES",
     stringsAsFactors = FALSE)

# Load files to memory (using rasterToMemory)
mySim <- loadFiles(filelist = filelist)

# put layers into a single stack for convenience
landscape <- stack(mySim$DEM, mySim$forestCover, mySim$forestAge,
   mySim$habitatQuality, mySim$percentPine)

# can change color palette
setColors(landscape, n = 50) <- list(DEM=topo.colors(50),
                           forestCover = brewer.pal(9, "Set1"),
                           forestAge = brewer.pal("Blues", n=8),
                           habitatQuality = brewer.pal(9, "Spectral"),
                           percentPine = brewer.pal("GnBu", n=8))

# Make a new raster derived from a previous one; must give it a unique name
habitatQuality2 <- landscape$habitatQuality ^ 0.3
names(habitatQuality2) <- "habitatQuality2"

# make a SpatialPoints object
caribou <- cbind(x = stats::runif (1e2, -50, 50), y = stats::runif (1e2, -50, 50)) %>%
  SpatialPoints(coords = .)

# use factor raster to give legends as character strings
ras <- raster(extent(0,3,0,4), vals = sample(1:4, size=12, replace=TRUE), res = 1)
# needs to have a data.frame with ID as first column - see ?raster::ratify
levels(ras) <- data.frame(ID=1:4, Name=paste0("Level",1:4))
if (interactive()) Plot(ras, new=TRUE)

# Arbitrary values for factors, including zero and not all levels represented in raster
levs <- c(0:5,7:12)
ras <- raster(extent(0,3,0,2), vals = c(1,1,3,5,8,9), res = 1)
levels(ras) <- data.frame(ID=levs, Name=LETTERS[c(1:3,8:16)])
if (interactive()) Plot(ras, new=TRUE)

# Arbitrary values for factors, including zero and not all levels represented in raster
levs <- c(0:5,7:23)
ras <- raster(extent(0,3,0,2), vals = c(1,1,3,5,8,9), res = 1)
levels(ras) <- data.frame(ID=levs, Name=LETTERS[1:23])
if (interactive()) Plot(ras, new=TRUE)

# SpatialPolygons
Sr1 = Polygon(cbind(c(2, 4, 4, 1, 2), c(2, 3, 5, 4, 2))*20-50)
```

```
Sr2 = Polygon(cbind(c(5, 4, 2, 5), c(2, 3, 2, 2))*20 - 50)
Srs1 = Polygons(list(Sr1), "s1")
Srs2 = Polygons(list(Sr2), "s2")
SpP = SpatialPolygons(list(Srs1, Srs2), 1:2)

if (interactive()) {
  clearPlot()
  Plot(ras)

  # dev(2)

  clearPlot()
  Plot(landscape)

  # Can overplot, using addTo
  Plot(caribou, addTo = "landscape$forestAge", size = 4, axes = FALSE)

  # can add a plot to the plotting window
  Plot(caribou, new = FALSE)

  # Can add two maps with same name, if one is in a stack; they are given
  #  unique names based on object name
  Plot(landscape, caribou, mySim$DEM)

  # can mix stacks, rasters, SpatialPoint*
  Plot(landscape, habitatQuality2, caribou)

  # can mix stacks, rasters, SpatialPoint*, and SpatialPolygons*
  Plot(landscape, caribou)
  Plot(habitatQuality2, new = FALSE)
  Plot(SpP)
  Plot(SpP, addTo = "landscape$forestCover", gp = gpar(lwd = 2))

  # provide arrangement, NumRow, NumCol
  Plot(SpP, arr = c(1,4), new=TRUE)

  # example base plot
  clearPlot()
  Plot(1:10, 1:10, addTo = "test", new=TRUE) # if there is no "test" then it will make it
  Plot(4,5, pch=22, col = "blue", addTo = "test") # if there is no "test" then it will make it
  obj1 <- rnorm(1e2)
  Plot(obj1, axes = "L")

  # Can plot named lists of objects (but not base objects yet)
  ras1 <- ras2 <- ras
  a <- list();for(i in 1:2) a[[paste0("ras",i)]] <- get(paste0("ras",i))
  a$SpP <- SpP
  clearPlot()
  Plot(a)


}
```

```
## End(Not run)
```

POM *Use Pattern Oriented Modeling to fit unknown parameters*

Description

This is very much in alpha condition. It has been tested on simple problems, as shown in the examples, with up to 2 parameters. It appears that DEoptim is the superior package for the stochastic problems. This should be used with caution as with all optimization routines. This function can nevertheless take optim or genoud as optimizers, using `stats::optim` or `rgenoud::genoud`, respectively. However, these latter approaches do not seem appropriate for stochastic problems, and have not been widely tested and are not supported within POM.

Usage

```
POM(sim, params, objects, objFn, cl, optimizer = "DEoptim", sterr = FALSE,
  ..., objFnCompare = "MAD", optimControl = NULL, NaNRetries = NA,
  logObjFnVals = FALSE, weights)

## S4 method for signature 'simList,character'
POM(sim, params, objects, objFn, cl,
  optimizer = "DEoptim", sterr = FALSE, ..., objFnCompare = "MAD",
  optimControl = NULL, NaNRetries = NA, logObjFnVals = FALSE, weights)
```

Arguments

`sim`	A `simList` simulation object, generally produced by `simInit`.
`params`	Character vector of parameter names that can be changed by the optimizer. These must be accessible with `params(sim)` internally.
`objects`	A optional named list (must be specified if objFn is not). The names of each list element must correspond to an object in the `.GlobalEnv` and the list elements must be objects or functions of objects that can be accessed in the ls(sim) internally. These will be used to create the objective function passed to the optimizer. See details and examples.
`objFn`	An optional objective function to be passed into `optimizer`. If missing, then `POM` will use `objFnCompare` and `objects` instead. If using `POM` with a SpaDES simulation, this objFn must contain a spades call internally, followed by a derivation of a value that can be minimized but the `optimizer`. It must have, as first argument, the values for the parameters. See example.
`cl`	A cluster object. Optional. This would generally be created using parallel::makeCluster or equivalent. This is an alternative way, instead of `beginCluster()`, to use parallelism for this function, allowing for more control over cluster use.

optimizer	The function to use to optimize. Default is "DEoptim". Currently it can also be "optim" or "rgenoud", which use stats::optim or rgenoud::genoud, respectively. The latter two do not seem optimal for stochastic problems and have not been widely tested.
sterr	Logical. If using optimizer = "optim", the hessian can be calculated. If this is TRUE, then the standard errors can be estimated using that hessian, assuming normality.
...	All objects needed in objFn
objFnCompare	Character string. Either, "MAD" or "RMSE" indicating that inside the objective function, data and prediction will be compared by Mean Absolute Deviation or Root Mean Squared Error. Default is "MAD".
optimControl	List of control arguments passed into the control of each optimization routine. Currently, only passed to DEoptim.control when optimizer is "DEoptim"
NaNRetries	Numeric. If greater than 1, then the function will retry the objective function for a total of that number of times if it results in an NaN. In general this should not be used as the objective function should be made so that it doesn't produce NaN. But, sometimes it is difficult to diagnose stochastic results.
logObjFnVals	Logical or Character string indicating a filename. Ignored if objFn is supplied. If TRUE (and there is no objFn supplied), then the value of the individual patterns will be output the console if being run interactively or to a tab delimited text file named ObjectiveFnValues.txt (or that passed by the user here) at each evaluation of the POM created objective function. See details.
weights	Numeric. If provided, this vector will be multiplied by the standardized mean absolute deviations as describe in objects. This has the effect of weighing each pattern to a user specified amount in the objective function.

Details

There are two ways to use this function, via 1) objFn or 2) objects.

1. The user can pass the entire objective function to the objFn argument that will be passed directly to the optimizer. For this, the user will likely need to pass named objects as part of the ....
2. The slightly simpler approach is to pass a list of 'actual data–simulated data' pairs as a named list in objects and specify how these objects should be compared via objFnCompare (whose default is Mean Absolute Deviation or "MAD").

Option 1 offers more control to the user, but may require more knowledge. Option 1 should likely contain a call to simInit(copy(simList)) and spades internally. See examples that show simple examples of each type, option 1 and option 2. In both cases, params is required to indicate which parameters can be varied in order to achieve the fit.

Currently, option 1 only exists when optimizer is "DEoptim", the default.

The upper and lower limits for parameter values are taken from the metadata in the module. Thus, if the module metadata does not define the upper and lower limits, or these are very wide, then the optimization may have troubles. Currently, there is no way to override these upper and lower

limits; the module metadata should be changed if there needs to be different parameter limits for optimization.

objects is a named list of data–pattern pairs. Each of these pairs will be assessed against one another using the objFnCompare, after standardizing each independently. The standardization, which only occurs if the abs(data value < 1), is: mean(abs(derived value - data value))/mean(data value). If the data value is between -1 and 1, then there is no standardization. If there is more than one data–pattern pair, then they will simply be added together in the objective function. This gives equal weight to each pair. If the user wishes to put different weight on each pattern, a weights vector can be provided. This will be used to multiply the standardized values described above. Alternatively, the user may wish to weight them differently, in which case, their relative scales can be adjusted.

There are many options that can be passed to DEoptim, (the details of which are in the help), using optimControl. The defaults sent from POM to DEoptim are: steptol = 3 (meaning it will start assessing convergence after 3 iterations (WHICH MAY NOT BE SUFFICIENT FOR YOUR PROBLEM), NP = 10 * length(params) (meaning the population size is 10 x the number of parameters) and itermax = 200 (meaning it won't go past 200 iterations). These and others may need to be adjusted to obtain good values. NOTE: DEoptim does not provide a direct estimate of confidence intervals. Also, convergence may be unreliable, and may occur because itermax is reached. Even when convergence is indicated, the estimates are not guaranteed to be global optima. This is different than other optimizers that will normally indicate if convergence was not achieved at termination of the optimization.

Using this function with a parallel cluster currently requires that you pass optimControl = list(parallelType = 1), and possibly package and variable names (and does not yet accept the cl argument). See examples. This setting will use all available threads on your computer. Future versions of this will allow passing of a custom cluster object via cl argument. POM will automatically determine packages to load in the spawned cluster (via SpaDES::packages) and it will load all objects in the cluster that are necessary, by sending names(objects) to parVar in DEoptim.control.

Setting logObjFnVals to TRUE may help diagnosing some problems. Using the POM derived objective function, essentially all patterns are treated equally. This may not give the correct behavior for the objective function. Because the POM weighs the patterns equally, it may be useful to set all patterns so that their expected values are each close to the same value, e.g., 1. If they are all expected to be near that value, then their sum will give relatively equal weights to each of the patterns. This can, of course, be manipulated to give higher weight to some patterns over others, by setting their expected value to be scaled to 1, then multiplied by some weight. This would have to be done in the preparation of the patterns with the objects argument.

Value

A list with at least 2 elements. The first (or first several) will be the returned object from the optimizer. The second (or last if there are more than 2), named args is the set of arguments that were passed into the control of the optimizer.

Author(s)

Eliot McIntire

See Also

spades, makeCluster, simInit

Examples

```
if (interactive()) {
  set.seed(89462)
  library(parallel)
  mySim <- simInit(
    times = list(start = 0.0, end = 2.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
      fireSpread = list(nFires = 5),
      randomLandscapes = list(nx = 300, ny = 300)
    ),
    modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
    paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
  )

  # Since this is a made up example, we don't have real data
  #  to run POM against. Instead, we will run the model once,
  #  take the values at the end of the simulation as if they
  #  are real data, then rerun the POM function next,
  #  comparing these "data" with the simulated values
  #  using Mean Absolute Deviation
  outData <- spades(copy(mySim), .plotInitialTime = NA)

  # Extract the "true" data, in this case, the "proportion of cells burned"
  # Function defined that will use landscape$Fires map from simList,
  #  i.e., sim$landscape$Fires
  #  the return value being compared via MAD with propCellBurnedData
  propCellBurnedFn <- function(landscape) {
    sum(getValues(landscape$Fires)>0) / ncell(landscape$Fires)
  }
  # visualize the burned maps of true "data"
  propCellBurnedData <- propCellBurnedFn(outData$landscape)
  clearPlot()
  if(interactive()) {
    Fires <- outData$landscape$Fires # Plot doesn't do well with many nested layers
    Plot(Fires)
  }
  # Example 1 - 1 parameter
  # In words, this says, "find the best value of spreadprob such that
  #  the proportion of the area burned in the simulation
  #  is as close as possible to the proportion area burned in
  #  the "data", using \code{DEoptim()}.

  # Can use cluster if computer is multi-threaded (but not yet via cl arg, which is not
  #                                                 implemented yet in DEoptim)
  # This example uses parallelType = 1 in DEoptim. For this, you must manually
  #  pass all packages and variables as character strings.
  # cl <- makeCluster(detectCores() - 1) # not implemented yet in DEoptim
  out1 <- POM(mySim, "spreadprob",
              list(propCellBurnedData = propCellBurnedFn), # data = pattern pair
              #optimControl = list(parallelType = 1),
              logObjFnVals = TRUE)
```

```
## Once cl arg is available from DEoptim, this will work:
# out1 <- POM(mySim, "spreadprob", cl = cl,
#            list(propCellBurnedData = propCellBurnedFn)) # data = pattern pair

# Example 2 - 2 parameters
# Function defined that will use caribou from sim$caribou, with
#  the return value being compared via MAD with NPattern
#  module, parameter N, is from 10 to 1000)
caribouFn <- function(caribou) length(caribou)

# Extract "data" from simList object (normally, this would be actual data)
NPattern <- caribouFn(outData$caribou)

aTime <- Sys.time()
parsToVary <- c("spreadprob", "N")
out2 <- POM(mySim, parsToVary,
            list(propCellBurnedData = propCellBurnedFn,
                 NPattern = caribouFn), logObjFnVals = TRUE)
                 #optimControl = list(parallelType = 1))
                 #cl = cl) # not yet implemented, waiting for DEoptim
bTime <- Sys.time()
# check that population overlaps known values (0.225 and 100)
apply(out2$member$pop, 2, quantile, c(0.025, 0.975))
hists <- apply(out2$member$pop, 2, hist, plot = FALSE)
clearPlot()
for (i in seq_along(hists)) Plot(hists[[i]], addTo = parsToVary[i],
                                 title = parsToVary[i], axes = TRUE)

print(paste("DEoptim", format(bTime - aTime)))
#stopCluster(cl) # not yet implemented, waiting for DEoptim

# Example 3 - using objFn instead of objects

# list all the parameters in the simList, from these, we select to vary
params(mySim)

# Objective Function Example:
#   objective function must have several elements
#   - first argument must be parameter vector, passed to and used by DEoptim
#   - likely needs to take sim object, likely needs a copy
#      because of pass-by-reference semantics of sim objects
#   - pass data that will be used internally for objective function
objFnEx <- function(pars, # param values
                    sim, # simList object
                   NPattern, propCellBurnedData, caribouFn, propCellBurnedFn) { # data

  # make a copy of simList because it will possibly be altered by spades call
  sim1 <- SpaDES::copy(sim)

  # take the parameters and assign them to simList
  params(sim1)$fireSpread$spreadprob <- pars[1]
  params(sim1)$caribouMovement$N <- pars[2]
```

```
  # run spades, without plotting
  out <- spades(sim1, .plotInitialTime = NA)

  # calculate outputs
  propCellBurnedOut <- propCellBurnedFn(out$landscape)
  NPattern_Out <- caribouFn(out$caribou)

  minimizeFn <- abs(NPattern_Out - NPattern) +
                abs(propCellBurnedOut - propCellBurnedData)

  # have more info reported to console, if desired
  # cat(minimizeFn)
  # cat(" ")
  # cat(pars)
  # cat("\n")

  return(minimizeFn)
}

# Run DEoptim with custom objFn, identifying 2 parameters to allow
#  to vary, and pass all necessary objects required for the
#  objFn

# choose 2 of them to vary. Need to identify them in params & inside objFn
# Change optimization parameters to alter how convergence is achieved
out5 <- POM(mySim, params = c("spreadprob", "N"),
            objFn = objFnEx,
            NPattern = NPattern,
            propCellBurnedData = propCellBurnedData,
            caribouFn = caribouFn,
            propCellBurnedFn = propCellBurnedFn,
           #cl = cl, # uncomment for cluster # not yet implemented, waiting for DEoptim
            # see ?DEoptim.control for explanation of these options
            optimControl = list(
              NP = 100, # run 100 populations, allowing quantiles to be calculated
           initialpop = matrix(c(runif(100, 0.2, 0.24), runif(100, 80, 120)), ncol = 2),
              parallelType = 1
            )
          )

# Can also use an optimizer directly -- miss automatic parameter bounds,
#  and automatic objective function using option 2
library(DEoptim)
out7 <- DEoptim(fn = objFnEx,
                sim = mySim,
                NPattern = NPattern,
                propCellBurnedData = propCellBurnedData,
                caribouFn = caribouFn,
                propCellBurnedFn = propCellBurnedFn,
                # cl = cl, # uncomment for cluster
                # see ?DEoptim.control for explanation of these options
                control = DEoptim.control(steptol = 3, # parallelType = 3,
```

```
                                        parallelType = 1,
                                    packages = list("raster", "SpaDES", "RColorBrewer"),
                                        parVar = list("objFnEx")),
                  initialpop = matrix(c(runif(40, 0.2, 0.24), runif(40, 80, 120)), ncol = 2),
                    lower = c(0.2, 80), upper = c(0.24, 120))
}
```

---

probInit SELES - *Probability of Initiation*

Description

Describes the probability of initiation of agents or events. THIS IS NOT FULLY IMPLEMENTED.

A SELES-like function to maintain conceptual backwards compatability with that simulation tool. This is intended to ease transitions from SELES.

You must know how to use SELES for these to be useful.

Usage

```
probInit(map, p = NULL, absolute = NULL)
```

Arguments

map	A spatialObjects object. Currently, only provides CRS and, if p is not a raster, then all the raster dimensions.
p	probability, provided as a numeric or raster
absolute	logical. Is p absolute probabilities or relative?

Value

An RasterLayer with probabilities of initialization. There are several combinations of inputs possible and they each result in different behaviors.

If p is numeric or Raster and between 0 and 1, it is treated as an absolute probability, and a map will be produced with the p value(s) everywhere.

If p is numeric or Raster and not between 0 and 1, it is treated as a relative probability, and a map will be produced with p/max(p) value(s) everywhere

If absolute is provided, it will override the previous statements, unless absolute is TRUE and p is not between 0 and 1 (i.e., is not a probability)

Author(s)

Eliot McIntire

progressInterval *Get and set simulation progress bar details*

Description

The progress bar can be set in two ways in SpaDES. First, by setting values in the .progress list element in the params list element passed to `simInit`. Second, at the `spades` call itself, which can be simpler. See examples.

Usage

```
progressInterval(object)

## S4 method for signature '.simList'
progressInterval(object)

progressInterval(object) <- value

## S4 replacement method for signature '.simList'
progressInterval(object) <- value

progressType(object)

## S4 method for signature '.simList'
progressType(object)

progressType(object) <- value

## S4 replacement method for signature '.simList'
progressType(object) <- value
```

Arguments

`object`	A `simList` simulation object.
`value`	The object to be stored at the slot.

Details

Progress Bar: Progress type can be one of `"text"`, `"graphical"`, or `"shiny"`. Progress interval can be a numeric. These both can get set by passing a `.progress=list(type="graphical", interval=1)` into the `simInit` call. See examples.

See Also

Other functions to access elements of a `simList` object: `.addDepends`, `doEvent.checkpoint`, `envir`, `events`, `globals`, `inputs`, `ls.simList`, `ls.str.simList`, `modules`, `objs`, `packages`, `params`, `paths`, `times`

Examples

```
## Not run:
mySim <- simInit(times=list(start=0.0, end=100.0),
                 params=list(.globals=list(stackName="landscape"),
                             .progress=list(type="text", interval=10),
                             .checkpoint = list(interval = 10, file = "chkpnt.RData")),
                 modules=list("randomLandscapes"),
                 paths=list(modulePath=system.file("sampleModules", package="SpaDES")))

# progress bar
progressType(mySim) # "text"
progressInterval(mySim) # 10

# parameters
params(mySim) # returns all parameters in all modules
              # including .global, .progress, .checkpoint
globals(mySim) # returns only global parameters

# checkpoint
checkpointFile(mySim) # returns the name of the checkpoint file
                      # In this example, "chkpnt.RData"
checkpointInterval(mySim) # 10

## End(Not run)
```

`randomPolygons` *randomPolygons*

Description

Produces a raster of random polygons. These are built with the `spread` function internally.

Usage

```
randomPolygons(ras = raster(extent(0, 15, 0, 15), res = 1, vals = 0),
  numTypes = 2, ...)
```

Arguments

`ras`	A raster that whose extent will be used for the randomPolygons
`numTypes`	Numeric value. The number of unique polygon types to use. This will be overridden by `p`, `A` or `minpatch`, if any of these are vectors.
`...`	Other arguments passed to spread. No known uses currently.

Value

A map of extent `ext` with random polygons.

References

Saura, S. and Martinez-Millan, J. (2000) Landscape patterns simulation with a modified random clusters method. Landscape Ecology, 15, 661–678.

See Also

spread and raster

Examples

```
set.seed(1234)
Ras <- randomPolygons(numTypes = 5)
if (interactive()) {
  clearPlot()
  Plot(Ras, cols = c("yellow", "dark green", "blue", "dark red"))
}

library(raster)
# more complex patterning, with a range of patch sizes
a <- randomPolygons(numTypes = 400, raster(extent(0, 50, 0, 50), res = 1, vals = 0))
a[a<320] <- 0
a[a>=320] <- 1
suppressWarnings(clumped <- clump(a)) # (warning sometimes occurs, but not important)
aHist <- hist(table(getValues(clumped)), plot = FALSE)
if (interactive()) {
  clearPlot()
  Plot(a)
  Plot(aHist)
}
```

---

rasterizeReduced *Convert reduced representation to full raster*

Description

Convert reduced representation to full raster

Usage

```
rasterizeReduced(reduced, fullRaster, plotCol, mapcode = names(fullRaster),
  ...)
```

Arguments

reduced	data.frame or data.table that has at least one column of codes that are represented in the fullRaster.
fullRaster	RasterLayer of codes used in reduced that represents a spatial representation of the data

plotCol	a character, length 1, with the name of the column in reduced that whose value will be plotted
mapcode	a character, length 1, with the name of the column in reduced that is represented in fullRaster
...	Other arguments. None used yet.

Value

A RasterLayer of with same dimensions as fullRaster representing plotCol spatially, according to the join between the mapcodeAll contained within reduced and fullRaster

Author(s)

Eliot McIntire

See Also

raster

Examples

```
library(data.table)
library(raster)
Ras <- raster(extent(0,15,0,15), res=1)
fullRas <- randomPolygons(Ras, numTypes=2)
names(fullRas) <- "mapcodeAll"
uniqueComms <- unique(fullRas)
reducedDT <- data.table(mapcodeAll=uniqueComms,
   communities=sample(1:1000,length(uniqueComms)),
   biomass=rnbinom(length(uniqueComms),mu=4000,0.4))
biomass <- rasterizeReduced(reducedDT, fullRas, "biomass")

# The default key is the layer name of the fullRas, so even
# if the reducedDT is miskeyed
setkey(reducedDT, biomass)

communities <- rasterizeReduced(reducedDT, fullRas, "communities")
setColors(communities) <- c("blue", "orange", "red")
if (interactive()) {
  clearPlot()
  Plot(biomass, communities, fullRas)
}
```

rasterToMemory *Read raster to memory*

Description

Wrapper to the raster function, that creates the raster object in memory, even if it was read in from file.

Usage

```
rasterToMemory(x, ...)

## S4 method for signature 'ANY'
rasterToMemory(x, ...)
```

Arguments

x	An object passed directly to the function raster (e.g., character string of a filename).
...	Additional arguments to raster.

Value

A raster object whose values are stored in memory.

Author(s)

Eliot McIntire and Alex Chubaty

See Also

raster.

rings *Identifies all cells within a ring around the focal cells*

Description

Identifies the cell numbers of all cells within a ring defined by minimum and maximum distances from focal cells. Uses spread under the hood, with specific values set. Under many situations, this will be faster than using rgeos::gBuffer twice (once for smaller ring and once for larger ring, then removing the smaller ring cells).

Usage

```
rings(landscape, loci = NA_real_, id = FALSE, minRadius = 2,
  maxRadius = 5, allowOverlap = FALSE, returnIndices = FALSE,
  returnDistances = TRUE, ...)

## S4 method for signature 'RasterLayer'
rings(landscape, loci = NA_real_, id = FALSE,
  minRadius = 2, maxRadius = 5, allowOverlap = FALSE,
  returnIndices = FALSE, returnDistances = TRUE, ...)
```

Arguments

landscape	A RasterLayer object. This defines the possible locations for spreading events to start and spread into. This can also be used as part of stopRule. Require input.
loci	A vector of locations in landscape. These should be cell indexes. If user has x and y coordinates, these can be converted with cellFromXY.
id	Logical. If TRUE, returns a raster of events ids. If FALSE, returns a raster of iteration numbers, i.e., the spread history of one or more events. NOTE: this is overridden if returnIndices is TRUE.
minRadius	Numeric. Minimum radius to be included in the ring. Note: this is inclusive, i.e., >=
maxRadius	Numeric. Maximum radius to be included in the ring. Note: this is inclusive, i.e., <=
allowOverlap	Logical. If TRUE, then individual events can overlap with one another, i.e., they do not interact. Currently, this is slower than if allowOverlap is FALSE. Default is FALSE.
returnIndices	Logical. Should the function return a data.table with indices and values of successful spread events, or return a raster with values. See Details.
returnDistances	Logical. Should the function inclue a column with the individual cell distances from the locus where that event started. Default is FALSE. See Details.
...	Any other argument passed to spread

Value

This will return a data.table with columns as described in spread when returnIndices = TRUE.

Author(s)

Eliot McIntire

See Also

cir which uses a different algorithm. cir tends to be faster when there are few starting points, rings tends to be faster when there are many starting points. Another difference between the two functions is that rings takes the centre of the pixel as the centre of a circle, whereas cir takes the exact coordinates. See example. rgeos::gBuffer

Examples

```
library(raster)

# Make random forest cover map
emptyRas <- raster(extent(0,1e2,0,1e2), res = 1)

# start from two cells near middle
loci <- (ncell(emptyRas)/2 - ncol(emptyRas))/2 + c(-3, 3)

# Allow overlap
emptyRas[] <- 0
Rings <- rings(emptyRas, loci = loci, allowOverlap = TRUE, returnIndices = TRUE)
# Make a raster that adds together all id in a cell
wOverlap <- Rings[,list(sumEventID=sum(id)),by="indices"]
emptyRas[wOverlap$indices] <- wOverlap$sumEventID
if (interactive()) {
  clearPlot()
  Plot(emptyRas)
}

# No overlap is default, occurs randomly
emptyRas[] <- 0
Rings <- rings(emptyRas, loci = loci, minRadius = 7, maxRadius = 9, returnIndices = TRUE)
emptyRas[Rings$indices] <- Rings$id
if (interactive()) {
  clearPlot()
  Plot(emptyRas)
}

# Variable ring widths, including centre cell for smaller one
emptyRas[] <- 0
Rings <- rings(emptyRas, loci = loci, minRadius = c(0,7), maxRadius = c(8, 18),
               returnIndices = TRUE)
emptyRas[Rings$indices] <- Rings$id
if (interactive()) {
  clearPlot()
  Plot(emptyRas)
}
```

---

rndstr *Generate random strings*

Description

Generate a vector of random alphanumeric strings each of an arbitrary length.

Usage

```
rndstr(n, len, characterFirst)
```

```
## S4 method for signature 'numeric,numeric,logical'
rndstr(n, len, characterFirst)

## S4 method for signature 'numeric,numeric,missing'
rndstr(n, len)

## S4 method for signature 'numeric,missing,logical'
rndstr(n, characterFirst)

## S4 method for signature 'missing,numeric,logical'
rndstr(len, characterFirst)

## S4 method for signature 'numeric,missing,missing'
rndstr(n)

## S4 method for signature 'missing,numeric,missing'
rndstr(len)

## S4 method for signature 'missing,missing,logical'
rndstr(characterFirst)

## S4 method for signature 'missing,missing,missing'
rndstr(n, len, characterFirst)
```

Arguments

n	Number of strings to generate (default 1). Will attempt to coerce to integer value.
len	Length of strings to generate (default 8). Will attempt to coerce to integer value.
characterFirst	Logical, if TRUE, then a letter will be the first character of the string (useful if being used for object names).

Value

Character vector of random strings.

Author(s)

Alex Chubaty and Eliot McIntire

Examples

```
set.seed(11)
rndstr()
rndstr(len = 10)
rndstr(characterFirst = FALSE)
rndstr(n = 5, len = 10)
rndstr(n = 5)
```

```
rndstr(n = 5, characterFirst = TRUE)
rndstr(len = 10, characterFirst = TRUE)
rndstr(n = 5, len = 10, characterFirst = TRUE)
```

---

saveFiles *Save objects using* .saveObjects *in* params *slot of* simInit

---

Description

In the simInit call, a parameter called .saveObjects can be provided in each module. This must be a character string vector of all object names to save. These objects will then be saved whenever a call to saveFiles is made.

Usage

```
saveFiles(sim)
```

Arguments

sim	A simList simulation object.

Details

The file names will be equal to the object name plus time(sim) is appended at the end. The files are saved as .rds files, meaning, only one object gets saved per file. For objects saved using this function, the module developer must create save events that schedule a call to saveFiles.

There are 3 ways to save objects using SpaDES.

1. Model-level saving

Using the outputs slot in the simInit call. See second example in simInit. This can be convenient because it gives overall control of many modules at a time, and there is an implicit scheduling that gets created during the simInit call.

2. Module-level saving

Using the saveFiles function inside a module. This must be accompanied by a .saveObjects list element in the params slot in the simInit call. Usually a module developer will create this method for future users of their module.

3. User saving

A user can save any object at any time inside their module. This is the least modular approach.

Note

It is not possible to schedule separate saving events for each object that is listed in the .saveObjects.

Author(s)

Eliot McIntire

Alex Chubaty

Examples

```
## Not run:

# This will save the "caribou" object at the save interval of 1 unit of time
#  in the outputPath location
outputPath <- file.path(tempdir(), "test_save")
times <- list(start = 0, end = 6, "month")
parameters <- list(
  .globals = list(stackName = "landscape"),
  caribouMovement = list(
    .saveObjects = "caribou",
    .saveInitialTime = 1, .saveInterval = 1
  ),
  randomLandscapes = list(.plotInitialTime = NA, nx = 20, ny = 20))

modules <- list("randomLandscapes", "caribouMovement")
paths <- list(
  modulePath = system.file("sampleModules", package = "SpaDES"),
  outputPath = savePath
)
mySim <- simInit(times = times, params = parameters, modules = modules,
                 paths = paths)

# The caribou module has a saveFiles(sim) call, so it will save caribou
spades(mySim)
dir(outputPath)

# remove the files
file.remove(dir(savePath, full.names = T))


## End(Not run)
```

---

scheduleEvent *Schedule a simulation event*

Description

Adds a new event to the simulation's event queue, updating the simulation object.

Usage

```
scheduleEvent(sim, eventTime, moduleName, eventType, eventPriority)
```

```
## S4 method for signature 'simList,numeric,character,character,numeric'
scheduleEvent(sim,
  eventTime, moduleName, eventType, eventPriority)

## S4 method for signature 'simList,`NULL`,character,character,numeric'
scheduleEvent(sim,
  eventTime, moduleName, eventType, eventPriority)

## S4 method for signature 'simList,numeric,character,character,missing'
scheduleEvent(sim,
  eventTime, moduleName, eventType, eventPriority)
```

Arguments

sim	A simList simulation object.
eventTime	A numeric specifying the time of the next event.
moduleName	A character string specifying the module from which to call the event.
eventType	A character string specifying the type of event from within the module.
eventPriority	A numeric specifying the priority of the event. Lower number means higher priority.

Details

Based on code from chapter 7.8.3 of Matloff (2011): "Discrete event simulation". Here, we implement a simulation in a more modular fashion so it's easier to add submodules to the simulation. We use S4 classes and methods, and use 'data.table' instead of 'data.frame' to implement the event queue (because it is much faster).

Value

Returns the modified simList object.

Author(s)

Alex Chubaty

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

Examples

```
## Not run:
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn") # default priority
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .normal()) # default priority

 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .normal()-1) # higher priority
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .normal()+1) # lower priority
```

```
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .highest()) # highest priority
 scheduleEvent(x, time(sim) + 1.0, "firemodule", "burn", .lowest()) # lowest priority

## End(Not run)
```

---

setColors<- *Set colours for plotting Raster* objects.*

Description

setColors works as a replacement method or a normal function call.

Usage

```
setColors(object, ..., n) <- value

## S4 replacement method for signature 'RasterLayer,numeric,character'
setColors(object, ..., n) <- value

## S4 replacement method for signature 'RasterLayer,missing,character'
setColors(object, ..., n) <- value

## S4 replacement method for signature 'RasterStack,numeric,list'
setColors(object, ..., n) <- value

## S4 replacement method for signature 'Raster,missing,list'
setColors(object, ..., n) <- value

setColors(object, value, n)

## S4 method for signature 'RasterLayer,character,numeric'
setColors(object, value, n)

## S4 method for signature 'RasterLayer,character,missing'
setColors(object, value)
```

Arguments

object	A Raster* object.
...	Additional arguments to colorRampPalette.
n	An optional vector of values specifiying the number of levels from which to interpolate the color palette.
value	Named list of hex color codes (e.g., from RColorBrewer::brewer.pal), corresponding to the names of RasterLayers in x.

Value

Returns a Raster with the `colortable` slot set to `values`.

Author(s)

Alex Chubaty

See Also

`brewer.pal`, `colorRampPalette`.

Examples

```
library(raster)
library(igraph) # need pipe for one example below

ras <- raster(matrix(c(0,0,1,2), ncol=2, nrow=2))

# Use replacement method
setColors(ras, n=3) <- c("red", "blue", "green")
if (interactive()) {
  clearPlot()
  Plot(ras)
}

# Use function method
ras <- setColors(ras, n=3, c("red", "blue", "yellow"))
if (interactive()) {
  clearPlot()
  Plot(ras)
}

# Using the wrong number of colors, e.g., here 2 provided,
# for a raster with 3 values... causes interpolation, which may be surprising
ras <- setColors(ras, c("red", "blue"))
if (interactive()) {
  clearPlot()
  Plot(ras)
}

# Real number rasters - interpolation is used
ras <- raster(matrix(runif(9), ncol=3, nrow=3)) %>%
  setColors(c("red", "yellow")) # interpolates when real numbers, gives warning
if (interactive()) {
  clearPlot()
  Plot(ras)
}

# Factor rasters, can be contiguous (numerically) or not, in this case not:
ras <- raster(matrix(sample(c(1,3,6), size=9, replace=TRUE), ncol=3, nrow=3))
levels(ras) <- data.frame(ID=c(1,3,6), Names=c("red", "purple", "yellow"))
ras <- setColors(ras, n=3, c("red", "purple", "yellow"))
```

```
if (interactive()) {
  clearPlot()
  Plot(ras)
}

# if a factor rastere, and not enough labels are provided, then a warning
#   will be given, and colors will be interpolated
#   The level called purple is not purple, but interpolated betwen red and yellow
ras <- setColors(ras, c("red", "yellow"))
if (interactive()) {
  clearPlot()
  Plot(ras)
}
```

---

shine *Display a simple, interactive shiny app of the simList*

Description

Currently, this is quite simple. It creates a side bar with the simulation times, plus a set of tabs, one for each module, with numeric sliders. Currently, this does not treat NAs correctly. Also, it is slow (shiny is not built to be fast out of the box). There are two buttons, one to run the entire spades call, the other to do just one time step at a time. It can be repeatedly pressed.

Usage

```
shine(sim, title = "SpaDES App", debug = FALSE, filesOnly = FALSE, ...)

## S4 method for signature 'simList'
shine(sim, title = "SpaDES App", debug = FALSE,
  filesOnly = FALSE, ...)
```

Arguments

sim	a simList object
title	character string. The title of the shiny page.
debug	Logical. If TRUE, then will show spades event debugger in console.
filesOnly	Logical. If TRUE, then the server.R, ui.R files will be written to a temp location, with a message indicating where they are. Publishing this to shinyapps.io is currently very buggy, and will likely not work as desired.
...	additional arguments. Currently not used

Note

Many module parameters are only accessed by modules at the start of a model run. So, even if the user changes them mid run, there won't be an effect on the model runs until `Reset` is pressed, and one of the Run buttons is pressed.

`.plotInterval` changes will only affect plots that are the base layer of a given plot image. If there are layers on top of a base layer (e.g., an agent on top of a raster layer), the .plotInterval of the overlayed layers is ignored.

Examples

```
## Not run:
 mySim <- simInit(
   times <- list(start = 0.0, end = 20.0),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
 )

shine(mySim)

# To publish to shinyapps.io, need files. THis is not reliable yet.
shine(mySim, filesOnly = TRUE)

# if the user wants to see the events go by, which can help with debugging:
shine(mySim, debug = TRUE)

## End(Not run)
```

`show,simList-method` *Show an Object*

Description

Show an Object

Usage

```
## S4 method for signature 'simList'
show(object)
```

Arguments

`object`	`simList`

simInit *Initialize a new simulation*

Description

Create a new simulation object, the "sim" object. This object is implemented using an `environment` where all objects and functions are placed. Since environments in `R` are pass by reference, "putting" objects in the sim object does no actual copy. This is also the location of all parameters, and other important simulation information, such as times, paths, modules, and module load order. See more details below.

Usage

```
simInit(times, params, modules, objects, paths, inputs, outputs, loadOrder)


  ## S4 method for signature
## 'list,list,list,list,list,data.frame,data.frame,character'
simInit(times,
  params, modules, objects, paths, inputs, outputs, loadOrder)

## S4 method for signature 'ANY,ANY,ANY,character,ANY,ANY,ANY,ANY'
simInit(times, params,
  modules, objects, paths, inputs, outputs, loadOrder)

## S4 method for signature 'ANY,ANY,character,ANY,ANY,ANY,ANY,ANY'
simInit(times, params,
  modules, objects, paths, inputs, outputs, loadOrder)

## S4 method for signature 'ANY,ANY,ANY,ANY,ANY,ANY,ANY,ANY'
simInit(times, params, modules,
  objects, paths, inputs, outputs, loadOrder)
```

Arguments

`times`	A named list of numeric simulation start and end times (e.g., `times = list(start = 0.0, end = 10.0)`
`params`	A list of lists of the form list(moduleName=list(param1=value, param2=value)). See details.
`modules`	A named list of character strings specfying the names of modules to be loaded for the simulation. Note: the module name should correspond to the R source file from which the module is loaded. Example: a module named "caribou" will be sourced form the file 'caribou.R', located at the specified `modulePath(simList)` (see below).
`objects`	(optional) A vector of object names (naming objects that are in the calling environment of the `simInit`, which is often the `.GlobalEnv` unless used programmatically – NOTE: this mechanism will fail if object name is in a package dependency), or a named list of data objects to be passed into the simList (more

reliable). These objects will be accessible from the simList as a normal list, e.g,. `mySim$obj`.

`paths` An optional named list with up to 4 named elements, `modulePath`, `inputPath`, `outputPath`, and `cachePath`. See details.

`inputs` A `data.frame`. Can specify from 1 to 6 columns with following column names: `objectName` (character, required), `file` (character), `fun` (character), `package` (character), `interval` (numeric), `loadTime` (numeric). See `inputs` and vignette("ii-modules") section about inputs.

`outputs` A `data.frame`. Can specify from 1 to 5 columns with following column names: `objectName` (character, required), `file` (character), `fun` (character), `package` (character), `saveTime` (numeric). See `outputs` and `vignette("ii-modules")` section about outputs.

`loadOrder` An optional list of module names specfiying the order in which to load the modules. If not specified, the module load order will be determined automatically.

Details

Calling this simInit function does several things including the following: - sources all module files, placing all function definitions in the sim object - optionally copies objects from the global environment to the sim object - optionally loads objects from disk - schedules all "init" events from all modules - assesses module dependencies via the inputs and outputs identified in their metadata - determines time units of modules and how they fit together

`params` can only contain updates to any parameters that are defined in the metadata of modules. Take the example of a module named, `Fire`, which has a parameter named `.plotInitialTime`. In the metadata of that moduel, it says TRUE. Here we can override that default with: `list(Fire=list(.plotInitialTime=NA` effectively turning off plotting. Since this is a list of lists, one can override the module defaults for multiple parameters from multiple modules all at once, with say: `list(Fire = list(.plotInitialTime = NA, .plotInte`

We implement a discrete event simulation in a more modular fashion so it is easier to add modules to the simulation. We use S4 classes and methods, and use `data.table` instead of `data.frame` to implement the event queue (because it is much faster).

`paths` specifies the location of the module source files, the data input files, and the saving output files. If no paths are specified the defaults are as follows:

- `cachePath`: `getOption("spades.cachePath")`;
- `inputPath`: `getOption("spades.modulePath")`;
- `modulePath`: `getOption("spades.inputPath")`;
- `inputPath`: `getOption("spades.outputPath")`.

Value

A `simList` simulation object, pre-initialized from values specified in the arguments supplied.

Note

The user can opt to run a simpler simInit call without inputs, outputs, and times. These can be added later with the accessor methods (See example). These are not required for initializing the simulation via simInit. `modules`, `paths`, `params`, and `objects` are all needed for initialization.

Author(s)

Alex Chubaty and Eliot McIntire

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

See Also

spades, times, params, objs, paths, modules, inputs, outputs

Examples

```
## Not run:
 mySim <- simInit(
   times = list(start = 0.0, end = 2.0, timeunit = "year"),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
 )
 spades(mySim, .plotInitialTime = NA)

 # Change more parameters, removing plotting
 wantPlotting <- FALSE
 mySim <- simInit(
   times = list(start = 0.0, end = 2.0, timeunit = "year"),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned"),
     fireSpread = list(.plotInitialTime=wantPlotting),
     #caribouMovement = list(.plotInitialTime=wantPlotting),
     #randomLandscapes = list(.plotInitialTime=wantPlotting)
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
 )
 outSim <- spades(mySim)

# A little more complicated with inputs and outputs
if (require(rgdal)) {
   mapPath <- system.file("maps", package = "SpaDES")
   mySim <- simInit(
     times = list(start = 0.0, end = 2.0, timeunit = "year"),
     params = list(
       .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
     ),
     modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
     paths = list(modulePath = system.file("sampleModules", package = "SpaDES"),
                  outputPath = tempdir()),
     inputs = data.frame(
```

```
    files = dir(file.path(mapPath), full.names = TRUE, pattern = "tif")[1:2],
    functions = "raster",
    package = "raster",
    loadTime = 0,
    stringsAsFactors = FALSE),
  outputs = data.frame(
    expand.grid(objectName = c("caribou","landscape"),
    saveTime = 1:2,
    stringsAsFactors = FALSE))
)

# Use accessors for inputs, outputs, times
mySim2 <- simInit(modules = list("randomLandscapes", "fireSpread",
                                 "caribouMovement"),
                  params = list(.globals = list(stackName = "landscape",
                                                burnStats = "nPixelsBurned")),
                  paths = list(modulePath = system.file("sampleModules",
                                                        package = "SpaDES"),
                               outputPath = tempdir()))
# add by accessor: note need current in times() accessor
times(mySim2) <- list(current=0, start = 0.0, end = 2.0, timeunit = "year")
inputs(mySim2) <- data.frame(
    files = dir(file.path(mapPath), full.names = TRUE, pattern = "tif")[1:2],
    functions = "raster",
    package = "raster",
    loadTime = 3,
    stringsAsFactors = FALSE)
outputs(mySim2) <- data.frame(
    expand.grid(objectName = c("caribou","landscape"),
    saveTime = 1:2,
    stringsAsFactors = FALSE))
all.equal(mySim, mySim2) # TRUE
}

## End(Not run)
```

---

spades *Run a spatial discrete event simulation*

Description

Based on code from chapter 7.8.3 of Matloff (2011): "Discrete event simulation". Here, we implement a simulation in a more modular fashion so it's easier to add submodules to the simulation. We use S4 classes and methods, and use 'data.table' instead of 'data.frame' to implement the event queue (because it is much faster).

Usage

```
spades(sim, debug = FALSE, progress = NA, cache, .plotInitialTime = NULL,
```

```
  .saveInitialTime = NULL, notOlderThan, ...)

## S4 method for signature 'simList,ANY,ANY,missing'
spades(sim, debug = FALSE,
  progress = NA, cache, .plotInitialTime = NULL, .saveInitialTime = NULL,
  notOlderThan, ...)

## S4 method for signature 'ANY,ANY,ANY,logical'
spades(sim, debug = FALSE, progress = NA,
  cache, .plotInitialTime = NULL, .saveInitialTime = NULL, notOlderThan,
  ...)
```

Arguments

sim	A simList simulation object, generally produced by simInit.
debug	Optional logical flag or character vector indicating what to print to console at each event. See details. Default is debug=FALSE.
progress	Logical (TRUE or FALSE show a graphical progress bar), character ("graphical", "text") or numeric indicating the number of update intervals to show in a graphical progress bar.
cache	Logical. If TRUE, then the spades call will be cached. This means that if the call is made again with the same simList, then spades will return the return value from the previous run of that exact same simList. Default FALSE. See Details.
.plotInitialTime	Numeric. Temporarily override the .plotInitialTime parameter for all modules. See Details.
.saveInitialTime	Numeric. Temporarily override the .plotInitialTime parameter for all modules. See Details.
notOlderThan	Date. Passed to SpaDES::cache to update the cache. Default is NULL, meaning don't update the cache. If Sys.time() is provided then, it will force a recache. Ignored if cache is FALSE.
...	Any. Can be used to make a unique cache identity, such as "replicate = 1". This will be included in the SpaDES::cache call, so will be unique and thus spades will not use a cached copy as long as anything passed in ... is unique, i.e., not cached previously.

Details

The is the workhorse function in the SpaDES package. It runs simulations by implementing the rules outlined in the simList.

This function gives simple access to two sets of module parameters: .plotInitialTime and with .plotInitialTime. The primary use of these arguments is to temporarily turn off plotting and saving. "Temporary" means that the simList is not changed, so it can be used again with the simList values reinstated. To turn off plotting and saving, use .plotInitialTime = NA or .saveInitialTime = NA. NOTE: if a module did not use .plotInitialTime or .saveInitialTime, then these arguments will not do anything.

If cache is TRUE, this allows for a seamless way to "save" results of a simulation. The user does not have to intentionally do any saving manually. Instead, upon a call to spades in which the simList is identical, the function will simply return the result that would have come if it had been rerun. Use this with caution, as it will return exactly the result from a previous run, even if there is stochasticity internally. Caching is only based on the input simList. See also experiment for the same mechanism, but it can be used with replication.

If debug is specified, it can be Logical or character vector. In all cases, something will be printed to the console immediately before each event is being executed. If TRUE, then the event immediately following will be printed as it runs (equivalent to current(sim)). If a character string, then it can be one of the many simList accessors, such as events, params, or "simList" (print the entire simList), or any R expression. If an R expression if will be evaluated with access to the sim object. If this is more than one character string, then all will be printed to the screen in their sequence.

Value

Invisibly returns the modified simList object.

Note

The debug option is primarily intended to facilitate building simulation models by the user. Will print additional outputs informing the user of updates to the values of various simList slot components.

Author(s)

Alex Chubaty

References

Matloff, N. (2011). The Art of R Programming (ch. 7.8.3). San Fransisco, CA: No Starch Press, Inc.. Retrieved from https://www.nostarch.com/artofr.htm

See Also

simInit, SpaDES

experiment for using replication with spades.

Examples

```
## Not run:
 mySim <- simInit(
   times = list(start = 0.0, end = 2.0, timeunit = "year"),
   params = list(
     .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
   ),
   modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
   paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
 )
 spades(mySim)
```

```
# Different debug options
spades(mySim, debug = TRUE) # Fastest
spades(mySim, debug = "simList")
spades(mySim, debug = "print(table(sim$landscape$Fires[]))")

# Can turn off plotting, and inspect the output simList instead
out <- spades(mySim, .plotInitialTime = NA) # much faster
completed(out) # shows completed events

# use cache -- simInit should generally be rerun each time a spades call is made
#   to guarantee that it is identical. Here, run spades call twice, first
#   time to establish cache, second time to return cached result
for(i in 1:2) {
  mySim <- simInit(
    times = list(start = 0.0, end = 2.0, timeunit = "year"),
    params = list(
      .globals = list(stackName = "landscape", burnStats = "nPixelsBurned")
    ),
    modules = list("randomLandscapes", "fireSpread", "caribouMovement"),
    paths = list(modulePath = system.file("sampleModules", package = "SpaDES"))
  )
  print(system.time(out <- spades(mySim, cache = TRUE)))
}

## End(Not run)
```

---

spadesClasses *Classes defined in SpaDES*

Description

These S4 classes are defined within SpaDES. "dot" classes are not exported and are therefore intended for internal use only.

Simulation classes

simList	The 'simList' class
.moduleDeps	Descriptor object for specifying SpaDES module dependecies
.simDeps	Defines all simulation dependencies for all modules within a SpaDES simulation

Plotting classes - used within Plot

New classes

.spadesPlot	Main class for Plot - contains .spadesGrob and .arrangement objects
.spadesGrob	GRaphical OBject used by SpaDES - smallest unit
.arrangement	The layout or "arrangement" of plot objects

Unions of existing classes

.spadesPlottables	The union of all object classes Plot can accept
.spadesPlotObjects	The union of spatialObjects and several others
spatialObjects	The union of several spatial classes

Author(s)

Eliot McIntire and Alex Chubaty

See Also

Plot, simInit

---

spadesMaps *Dummy maps included with* SpaDES

Description

All maps included with SpaDES are randomly generated maps created by gaussMap(). These are located within the maps folder of the package, and are used in the vignettes. Use system.file(package="SpaDES", "maps") to locate the maps directory on your system.

Format

raster

Details

DEM.tif: converted to a a small number of discrete levels (in 100m hypothetical units)

habitatQuality.tif: made to look like a continuous habitat surface, rescaled to 0 to 1

forestAge.tif: rescaled to possible forest ages in a boreal forest setting

forestCover.tif: rescaled to possible forest cover in a boreal forest setting

percentPine.tif: rescaled to percentages.

spatialObjects-class *The* spatialObjects *class*

Description

This class is the union of several spatial objects from raster and sp packages. Notably missing is RasterBrick, for now.

Slots

members SpatialPoints*, SpatialPolygons*, SpatialLines*, RasterLayer, RasterStack

Author(s)

Eliot McIntire

See Also

spadesClasses

specificNumPerPatch *Initiate a specific number of agents in a map of patches*

Description

Instantiate a specific number of agents per patch. The user can either supply a table of how many to initiate in each patch, linked by a column in that table called pops.

Usage

```
specificNumPerPatch(patches, numPerPatchTable = NULL, numPerPatchMap = NULL)
```

Arguments

patches	RasterLayer of patches, with some sort of a patch id.
numPerPatchTable	A data.frame or data.table with a column named pops that matches the patches patch ids, and a second column num.in.pop with population size in each patch.
numPerPatchMap	A RasterLayer exactly the same as patches but with agent numbers rather than ids as the cell values per patch.

Value

A raster with 0s and 1s, where the 1s indicate starting locations of agents following the numbers above.

Examples

```
library(raster)
library(data.table)
set.seed(1234)
Ntypes <- 4
ras <- randomPolygons(numTypes = Ntypes)
if(interactive()) {
  clearPlot()
  Plot(ras)
}

# Use numPerPatchTable
patchDT <- data.table(pops = 1:Ntypes, num.in.pop = c(1, 3, 5, 7))
rasAgents <- specificNumPerPatch(ras, patchDT)
rasAgents[is.na(rasAgents)] <- 0
#Plot(rasAgents)

library(testthat)
expect_true(all(unname(table(ras[rasAgents])) == patchDT$num.in.pop))

# Use numPerPatchMap
rasPatches <- ras
for (i in 1:Ntypes) {
  rasPatches[rasPatches==i] <- patchDT$num.in.pop[i]
}
if (interactive()) {
  clearPlot()
  Plot(ras, rasPatches)
}
rasAgents <- specificNumPerPatch(ras, numPerPatchMap = rasPatches)
rasAgents[is.na(rasAgents)] <- 0
if(interactive()) {
  clearPlot()
  Plot(rasAgents)
}
```

`splitRaster` *Split a RasterLayer into multiple tiles*

Description

Divides up a raster into an arbitrary number of pieces (tiles). Split rasters can be recombined using `do.call(merge, y)` or `mergeRaster(y)`, where `y <- splitRaster(x)`.

Usage

```
splitRaster(r, nx, ny, buffer, path, cl)

## S4 method for signature 'RasterLayer,integer,integer,numeric,character'
```

```
splitRaster(r, nx, ny,
  buffer, path, cl)

## S4 method for signature 'RasterLayer,numeric,numeric,integer,character'
splitRaster(r, nx, ny,
  buffer, path)

## S4 method for signature 'RasterLayer,numeric,numeric,integer,missing'
splitRaster(r, nx, ny,
  buffer)

## S4 method for signature 'RasterLayer,numeric,numeric,numeric,character'
splitRaster(r, nx, ny,
  buffer, path)

## S4 method for signature 'RasterLayer,numeric,numeric,numeric,missing'
splitRaster(r, nx, ny,
  buffer)

## S4 method for signature 'RasterLayer,numeric,numeric,missing,character'
splitRaster(r, nx, ny,
  path)

## S4 method for signature 'RasterLayer,numeric,numeric,missing,missing'
splitRaster(r, nx, ny)

## S4 method for signature 'RasterLayer,integer,integer,missing,character'
splitRaster(r, nx, ny,
  path)

## S4 method for signature 'RasterLayer,integer,integer,missing,missing'
splitRaster(r, nx, ny)
```

Arguments

r	The raster to be split.
nx	The number of tiles to make along the x-axis.
ny	The number of tiles to make along the y-axis.
buffer	Numeric vector of length 2 giving the size of the buffer along the x and y axes. If these values less than or equal to 1 are used, this is interpreted as the number of pixels (cells) to use as a buffer. Values between 0 and 1 are interpreted as proportions of the number of pixels in each tile (rounded up to an integer value). Default is c(0, 0), which means no buffer.
path	Character specifying the directory to which the split tiles will be saved. If missing, the function creates a subfolder in the current working directory based on the raster's name (i.e., using names(x)).
cl	A cluster object. Optional. This would generally be created using parallel::makeCluster

or equivalent. This is an alternative way, instead of `beginCluster()`, to use parallelism for this function, allowing for more control over cluster use.

Details

This function is parallel-aware, using the same mechanism as used in the `raster` package. Specifically, if you start a cluster using `beginCluster`, then this function will automatically use that cluster. It is always a good idea to stop the cluster when finished, using `endCluster`.

Value

A list (length `nx*ny`) of cropped raster tiles.

Author(s)

Alex Chubaty and Yong Luo

See Also

`do.call`, `merge`, `mergeRaster`.

Examples

```
require(raster)

# an example with dimensions:
# nrow = 77
# ncol = 101
# nlayers = 3
b <- brick(system.file("external/rlogo.grd", package = "raster"))
r <- b[[1]] # use first layer only
nx <- 3
ny <- 4

tmpdir <- file.path(tempdir(), "splitRaster-example")

y0 <- splitRaster(r, nx, ny, path = file.path(tmpdir, "y0")) # no buffer

# buffer: 10 pixels along both axes
y1 <- splitRaster(r, nx, ny, c(10, 10), path = file.path(tmpdir, "y1"))

# buffer: half the width and length of each tile
y2 <- splitRaster(r, nx, ny, c(0.5, 0.5), path = file.path(tmpdir, "y2"))

# parallel cropping
if (interactive()) {
  n <- pmin(parallel::detectCores(), 4) # use up to 4 cores
  beginCluster(n)
  y3 <- splitRaster(r, nx, ny, c(0.7, 0.7), path = file.path(tmpdir, "y3"))
  endCluster()
}
```

```
# the original raster:
if (interactive()) plot(r) # may require a call to `dev()` if using RStudio

# the split raster:
layout(mat = matrix(seq_len(nx*ny), ncol = nx, nrow = ny))
plotOrder <- c(4, 8, 12, 3, 7, 11, 2, 6, 10, 1, 5, 9)
if (interactive()) invisible(lapply(y0[plotOrder], plot))

# can be recombined using `raster::merge`
m0 <- do.call(merge, y0)
all.equal(m0, r) ## TRUE

m1 <- do.call(merge, y1)
all.equal(m1, r) ## TRUE

m2 <- do.call(merge, y2)
all.equal(m2, r) ## TRUE

# or recombine using SpaDES::mergeRaster
n0 <- mergeRaster(y0)
all.equal(n0, r) ## TRUE

n1 <- mergeRaster(y1)
all.equal(n1, r) ## TRUE

n2 <- mergeRaster(y2)
all.equal(n2, r) ## TRUE

unlink(tmpdir, recursive = TRUE)
```

---

spokes *Identify outward radiating spokes from initial points*

Description

This is a generalized version of a notion of a viewshed. The main difference is that there can be many "viewpoints".

Usage

```
spokes(landscape, coords, loci, maxRadius = ncol(landscape)/4,
  minRadius = maxRadius, allowOverlap = TRUE, stopRule = NULL,
  includeBehavior = "includePixels", returnDistances = FALSE,
  angles = NA_real_, nAngles = NA_real_, returnAngles = FALSE,
  returnIndices = TRUE, ...)

## S4 method for signature 'RasterLayer,SpatialPoints,missing'
spokes(landscape, coords, loci,
  maxRadius = ncol(landscape)/4, minRadius = maxRadius,
```

```
allowOverlap = TRUE, stopRule = NULL, includeBehavior = "includePixels",
returnDistances = FALSE, angles = NA_real_, nAngles = NA_real_,
returnAngles = FALSE, returnIndices = TRUE, ...)
```

Arguments

landscape	Raster on which the circles are built.
coords	Either a matrix with 2 (or 3) columns, x and y (and id), representing the coordinates (and an associated id, like cell index), or a SpatialPoints* object around which to make circles. Must be same coordinate system as the landscape argument. Default is missing, meaning it uses the default to loci
loci	Numeric. An alternative to coords. These are the indices on landscape to initiate this function. See coords. Default is one point in centre of landscape..
maxRadius	Numeric vector of length 1 or same length as coords
minRadius	Numeric vector of length 1 or same length as coords. Default is maxRadius, meaning return all cells that are touched by the narrow ring at that exact radius. If smaller than maxRadius, then this will create a buffer or donut or ring.
allowOverlap	Logical. Should duplicates across id be removed or kept. Default TRUE.
stopRule	A function. If the spokes are to stop. This can be a function of landscape, fromCell, toCell, x (distance from coords cell), or any other named argument passed into the ... of this function. See examples.
includeBehavior	Character string. Currently accepts only "includePixels", the default, and "excludePixels". See details.
returnDistances	Logical. If TRUE, then a column will be added to the returned data.table that reports the distance from coords to every point that was in the circle/donut surrounding coords. Default FALSE, which is faster.
angles	Numeric. Optional vector of angles, in radians, to use. This will create "spokes" outward from coords. Default is NA, meaning, use internally derived angles that will "fill" the circle.
nAngles	Numeric, length one. Alternative to angles. If provided, the function will create a sequence of angles from 0 to 2*pi, with a length nAngles, and not including 2*pi. Will not be used if angles is provided, and will show warning of both are given.
returnAngles	Logical. If TRUE, then a column will be added to the returned data.table that reports the angle from coords to every point that was in the circle/donut surrounding coords. Default FALSE.
returnIndices	Logical. Should the function return a data.table with indices and values of successful spread events, or return a raster with values. See Details.
...	Objects to be used by stopRule function. See examples.

Value

A matrix containing columns id (representing the row numbers of `coords`), angles (from `coords` to each point along the spokes), x and y coordinates of each point along the spokes, the corresponding indices on the `landscape` Raster, dists (the distances between each `coords` and each point along the spokes), and stop, indicating if it was a point that caused a spoke to stop going outwards due to `stopRule`.

Author(s)

Eliot McIntire

Examples

```
library(raster)
library(sp)

set.seed(1234)

Ras <- raster(extent(0, 10, 0, 10), res = 1, val = 0)
rp <- randomPolygons(Ras, numTypes = 10)
if (interactive()) {
  clearPlot()
  Plot(rp)
}
angles <- seq(0,pi*2,length.out = 17)
angles <- angles[-length(angles)]
N <- 2
loci <- sample(ncell(rp), N)
coords <- SpatialPoints(xyFromCell(rp, loci))
stopRule <- function(landscape) landscape < 3
d2 <- spokes(rp, coords = coords, stopRule = stopRule,
             minRadius = 0, maxRadius = 50,
             returnAngles = TRUE, returnDistances = TRUE,
             allowOverlap = TRUE, angles = angles, returnIndices = TRUE)

# Assign values to the "patches" that were in the viewshed of a ray
rasB <- raster(Ras)
rasB[] <- 0
rasB[d2[d2[, "stop"] == 1,"indices"]] <- 1
if (interactive()) {
  Plot(rasB, addTo = "rp", zero.color = "transparent", cols = "red")
  # can plot it as raster or spatial points
  # Plot(rasB, addTo = "rp", zero.color = "transparent", cols = "black")
  if (NROW(d2) > 0) {
    sp1 <- SpatialPoints(d2[,c("x",'y')])
    Plot(sp1, addTo = "rp", pch = 19, size = 5, speedup = 0.1)
  }
  Plot(coords, addTo = "rp", pch = 19, size = 6, cols = "blue", speedup = 0.1)
}
```

spread *Simulate a spread process on a landscape.*

Description

This can be used to simulate fires, seed dispersal, calculation of iterative, concentric landscape values (symmetric or asymmetric) and many other things. Essentially, it starts from a collection of cells (`loci`) and spreads to neighbours, according to the `directions` and `spreadProbLater` arguments. This can become quite general, if `spreadProbLater` is 1 as it will expand from every loci until all cells in the landscape have been covered. With `id` set to `TRUE`, the resulting map will be classified by the index of the cell where that event propagated from. This can be used to examine things like fire size distributions.

Usage

```
spread(landscape, loci = NA_real_, spreadProb = 0.23, persistence = 0,
  mask = NA, maxSize = 100000000L, directions = 8L,
  iterations = 1000000L, lowMemory = getOption("spades.lowMemory"),
  returnIndices = FALSE, returnDistances = FALSE, mapID = NULL,
  id = FALSE, plot.it = FALSE, spreadProbLater = NA_real_,
  spreadState = NA, circle = FALSE, circleMaxRadius = NA_real_,
  stopRule = NA, stopRuleBehavior = "includeRing", allowOverlap = FALSE,
  asymmetry = NA_real_, asymmetryAngle = NA_real_, ...)

## S4 method for signature 'RasterLayer'
spread(landscape, loci = NA_real_,
  spreadProb = 0.23, persistence = 0, mask = NA, maxSize = 100000000L,
  directions = 8L, iterations = 1000000L,
  lowMemory = getOption("spades.lowMemory"), returnIndices = FALSE,
  returnDistances = FALSE, mapID = NULL, id = FALSE, plot.it = FALSE,
  spreadProbLater = NA_real_, spreadState = NA, circle = FALSE,
  circleMaxRadius = NA_real_, stopRule = NA,
  stopRuleBehavior = "includeRing", allowOverlap = FALSE,
  asymmetry = NA_real_, asymmetryAngle = NA_real_, ...)
```

Arguments

`landscape`	A `RasterLayer` object. This defines the possible locations for spreading events to start and spread into. This can also be used as part of `stopRule`. Require input.
`loci`	A vector of locations in `landscape`. These should be cell indexes. If user has x and y coordinates, these can be converted with `cellFromXY`.
`spreadProb`	Numeric or rasterLayer. The overall probability of spreading, or probability raster driven. Default is `0.23`. If a `spreadProbLater` is provided, then this is only used for the first iteration. Also called Escape probability. See section on "Breaking out of spread events".

persistence A length 1 probability that an active cell will continue to burn, per time step.

mask non-NULL, a RasterLayer object congruent with landscape whose elements are 0,1, where 1 indicates "cannot spread to". Currently not implemented, but identical behavior can be achieved if spreadProb has zeros in all unspreadable locations.

maxSize Numeric. Maximum number of cells for a single or all events to be spread. Recycled to match loci length, if it is not as long as loci. See section on Breaking out of spread events.

directions The number adjacent cells in which to look; default is 8 (Queen case). Can only be 4 or 8.

iterations Number of iterations to spread. Leaving this NULL allows the spread to continue until stops spreading itself (i.e., exhausts itself).

lowMemory Logical. If true, then function uses package ff internally. This is slower, but much lower memory footprint.

returnIndices Logical. Should the function return a data.table with indices and values of successful spread events, or return a raster with values. See Details.

returnDistances
Logical. Should the function inclue a column with the individual cell distances from the locus where that event started. Default is FALSE. See Details.

mapID Deprecated use id

id Logical. If TRUE, returns a raster of events ids. If FALSE, returns a raster of iteration numbers, i.e., the spread history of one or more events. NOTE: this is overridden if returnIndices is TRUE.

plot.it If TRUE, then plot the raster at every iteraction, so one can watch the spread event grow.

spreadProbLater
Numeric or rasterLayer. If provided, then this will become the spreadProb after the first iteration. See details.

spreadState Data.table. This should be the output of a previous call to spread, where returnIndices was TRUE. Default NA, meaning the spread is starting from loci. See Details.

circle Logical. If TRUE, then outward spread will be by equidistant rings, rather than solely by adjacent cells (via directions arg.). Default is FALSE. Using circle = TRUE can be dramatically slower for large problems. Note, this should usually be used with spreadProb = 1.

circleMaxRadius
Numeric. A further way to stop the outward spread of events. If circle is TRUE, then it will grow to this maximum radius. See section on Breaking out of spread events. Default to NA.

stopRule A function which will be used to assess whether each individual cluster should stop growing. This function can be an argument of "landscape", "id", "cells", and any other named vectors, a named list of named vectors, or a named data.frame of with column names passed to spread in the ... . Default NA meaning that spreading will not stop as a function of the landscape. See section on Breaking out of spread events and examples.

stopRuleBehavior
: Character. Can be one of "includePixel", "excludePixel", "includeRing", "excludeRing". If stopRule contains a function, this argument is used determine what to do with the cell(s) that caused the rule to be TRUE. See details. Default is "includeRing" which means to accept the entire ring of cells that caused the rule to be TRUE.

allowOverlap
: Logical. If TRUE, then individual events can overlap with one another, i.e., they do not interact. Currently, this is slower than if allowOverlap is FALSE. Default is FALSE.

asymmetry
: A numeric indicating the ratio of the asymmetry to be used. Default is NA, indicating no asymmetry. See details. This is still experimental. Use with caution.

asymmetryAngle
: A numeric indicating the angle in degrees (0 is "up", as in North on a map), that describes which way the asymmetry is.

...
: Additional named vectors or named list of named vectors required for stopRule. These vectors should be as long as required e.g., length loci if there is one value per event.

Details

For large rasters, a combination of lowMemory = TRUE and returnIndices = TRUE will use the least amount of memory.

This function can be interrupted before all active cells are exhausted if the iterations value is reached before there are no more active cells to spread into. If this is desired, returnIndices should be TRUE and the output of this call can be passed subsequently as an input to this same function. This is intended to be used for situations where external events happen during a spread event, or where one or more arguments to the spread function change before a spread event is completed. For example, if it is desired that the spreadProb change before a spread event is completed because, for example, a fire is spreading, and a new set of conditions arise due to a change in weather.

asymmetry is currently used to modify the spreadProb in the following way. First for each active cell, spreadProb is converted into a length 2 numeric of Low and High spread probabilities for that cell: spreadProbsLH <- (spreadProb*2) // (asymmetry+1)*c(1,asymmetry), whose ratio is equal to asymmetry. Then, using asymmetryAngle, the angle between the initial starting point of the event and all potential cells is found. These are converted into a proportion of the angle from -asymmetryAngle to asymmetryAngle using: angleQuality <- (cos(angles - rad(asymmetryAngle))+1)/2

These are then converted to multiple spreadProbs by spreadProbs <- lowSpreadProb+(angleQuality * diff(spreadPro
To maintain an expected spreadProb that is the same as the asymmetric spreadProbs, these are then rescaled so that the mean of the asymmetric spreadProbs is always equal to spreadProb at every iteration: spreadProbs <- spreadProbs - diff(c(spreadProb,mean(spreadProbs)))

Value

Either a RasterLayer indicating the spread of the process in the landscape or a data.table if returnIndices is TRUE. If a RasterLayer, then it represents every cell in which a successful spread event occurred. For the case of, say, a fire this would represent every cell that burned. If allowOverlap is TRUE, This Raster layer will represent the sum of the individual event ids (which are numerics seq_along(loci). This will generally be of minimal use because it won't be possible to distinguish if event 2 overlapped with event 5 or if it was just event 7.

If `returnIndices` is `TRUE`, then this function returns a `data.table` with columns:

`id`	an arbitrary ID `1:length(loci)` identifying unique clusters of spread events, i.e., all cells that have been sp
`initialLocus`	the initial cell number of that particular spread event.
`indices`	The cell indices of cells that have been touched by the spread algorithm.
`active`	a logical indicating whether the cell is active (i.e., could still be a source for spreading) or not (no spreading

This will generally be more useful when `allowOverlap` is `TRUE`.

Breaking out of spread events

There are 4 ways for the spread to "stop" spreading. Here, each "event" is defined as all cells that are spawned from a single starting loci. So, one spread call can have multiple spreading "events". The ways outlines below are all acting at all times, i.e., they are not mutually exclusive. Therefore, it is the user's responsibility to make sure the different rules are interacting with each other correctly. Using `spreadProb` or `maxSize` are computationally fastest, sometimes dramatically so.

`spreadProb`	Probabilistically, if spreadProb is low enough, active spreading events will stop. In practice, active sprea
`maxSize`	This is the number of cells that are "successfully" turned on during a spreading event. This can be vector
`circleMaxRadius`	If `circle` is TRUE, then this will be the maximum radius reached, and then the event will stop. This is v
`stopRule`	This is a func tion that can use "landscape", "id", "cells", or any named vector passed into `spread` in the

The spread function does not return the result of this stopRule. If, say, an event has both `circleMaxRadius` and `stopRule`, and it is the `circleMaxRadius` that caused the event spreading to stop, there will be no indicator returned from this function that indicates which rule caused the stop.

`stopRule` has many use cases. One common use case is evaluating a neighbourhood around a focal set of points. This provides, therefore, an alternative to the `buffer` function or `focal` function. In both of those cases, the window/buffer size must be an input to the function. Here, the resulting size can be emergent based on the incremental growing and calculating of the `landscape` values underlying the spreading event.

`stopRuleBehavior`

This determines how the `stopRule` should be implemented. Because spreading occurs outwards in concentric circles or shapes, one cell width at a time, there are 4 possible ways to interpret the logical inequality defined in `stopRule`. In order of number of cells included in resulting events, from most cells to fewest cells:

`"includeRing"`	Will include the entire ring of cells that, as a group, caused `stopRule` to be `TRUE`.
`"includePixel"`	Working backwards from the entire ring that caused the `stopRule` to be `TRUE`, this will iteratively random
`"excludePixel"`	Like `"includePixel"`, but it will not add back the cell that causes `stopRule` to be `TRUE`
`"excludeRing"`	Analogous to `"excludePixel"`, but for the entire final ring of cells added. This will exclude the entire rin

Author(s)

Eliot McIntire

Steve Cumming

See Also

rings which uses spread but with specific argument values selected for a specific purpose. distanceFromPoints

Examples

```
library(raster)
library(RColorBrewer)

# Make random forest cover map
emptyRas <- raster(extent(0,1e2,0,1e2), res = 1)
hab <- randomPolygons(emptyRas, numTypes = 40)
names(hab) = "hab"
mask <- raster(emptyRas)
mask <- setValues(mask, 0)
mask[1:5000] <- 1
numCol <- ncol(emptyRas)
numCell <- ncell(emptyRas)
directions <- 8

# Can use transparent as a color
setColors(hab) <- paste(c("transparent", brewer.pal(8, "Greys")))

if (interactive()) {
  clearPlot()
  Plot(hab, speedup = 3) # note speedup is equivalent to making pyramids,
                                    # so, some details are lost
}

# initiate 10 fires
startCells <- as.integer(sample(1:ncell(emptyRas),10))
fires <- spread(hab, loci = startCells,
                0.235, 0, NULL, 1e8, 8, 1e6, id = TRUE)
#set colors of raster, including a transparent layer for zeros
setColors(fires, 10) <- c("transparent", brewer.pal(8,"Reds")[5:8])
if (interactive()) {
  Plot(fires)
  Plot(fires,addTo = "hab")

  #alternatively, set colors using cols= in the Plot function
  clearPlot()
  Plot(hab)
  Plot(fires) # default color range makes zero transparent.
  # Instead, to give a color to the zero values, use \code{zero.color=}
  Plot(fires, addTo = "hab",
       cols = colorRampPalette(c("orange","darkred"))(10))
  hab2 <- hab
  Plot(hab2)
```

```
  Plot(fires, addTo = "hab2", zero.color = "white",
     cols = colorRampPalette(c("orange","darkred"))(10))
  # or overplot the original (NOTE: legend stays at original values)
  Plot(fires, cols = topo.colors(10))
}

####################
## Continue event by passing interrupted object into spreadState
####################

## Interrupt a spread event using iterations - need returnIndices = TRUE to use outputs
##   as new inputs in next iteration
fires <- spread(hab, loci = as.integer(sample(1:ncell(hab), 10)), returnIndices = TRUE,
                0.235, 0, NULL, 1e8, 8, iterations = 3, id = TRUE)
fires[,list(size=length(initialLocus)), by=id]  # See sizes of fires

fires2 <- spread(hab, loci=NA_real_, returnIndices = TRUE, 0.235,
                 0, NULL, 1e8, 8, iterations = 2, id = TRUE,
                 spreadState=fires)
# NOTE events are assigned arbitrary IDs, starting at 1

## Add new fires to the already burning fires
fires3 <- spread(hab, loci = as.integer(sample(1:ncell(hab), 10)), returnIndices = TRUE,
                     0.235, 0, NULL, 1e8, 8, iterations = 1, id = TRUE,
                                          spreadState=fires)
fires3[,list(size=length(initialLocus)), by=id]  # See sizes of fires
# NOTE old ids are maintained, new events get ids begining above previous
# maximum (e.g., new fires 11 to 20 here)

## Use data.table and loci...
fires <- spread(hab, loci = as.integer(sample(1:ncell(hab), 10)), returnIndices = TRUE,
                0.235, 0, NULL, 1e8, 8, iterations = 2, id = TRUE)
fullRas <- raster(hab)
fullRas[] <- 1:ncell(hab)
burned <- fires[active == FALSE]
burnedMap <- rasterizeReduced(burned, fullRas, "id", "indices")
if (interactive()) {
  clearPlot()
  Plot(burnedMap)
}

####################
## stopRule examples
####################

# examples with stopRule, which means that the eventual size is driven by the values on the raster
#  passed in to the landscape argument
set.seed(1234)
stopRule1 <- function(landscape) sum(landscape)>50
stopRuleA <- spread(hab, loci = as.integer(sample(1:ncell(hab), 10)), 1, 0,
              NULL, maxSize = 1e6, 8, 1e6, id = TRUE, circle = TRUE, stopRule = stopRule1)

set.seed(1234)
```

```
stopRule2 <- function(landscape) sum(landscape)>100
# using stopRuleBehavior = "excludePixel"
stopRuleB <- spread(hab, loci = as.integer(sample(1:ncell(hab), 10)), 1, 0,
              NULL, maxSize = 1e6, 8, 1e6, id = TRUE, circle = TRUE, stopRule = stopRule2,
                stopRuleBehavior = "excludePixel")

# using stopRuleBehavior = "includeRing", means that end result is slightly larger patches, as a
#  complete "iteration" of the spread algorithm is used.
set.seed(1234)
stopRuleB_NotExact <- spread(hab, loci = as.integer(sample(1:ncell(hab), 10)), 1, 0,
              NULL, maxSize = 1e6, 8, 1e6, id = TRUE, circle = TRUE, stopRule = stopRule2)
if (interactive()) {
  clearPlot()
  Plot(stopRuleA, stopRuleB, stopRuleB_NotExact)
}

# Test that the stopRules work
# stopRuleA was not exact, so each value will "overshoot" the stopRule, here it was hab>50
foo <- cbind(vals=hab[stopRuleA], id = stopRuleA[stopRuleA>0]);
tapply(foo[,"vals"], foo[,"id"], sum) # Correct ... all are above 50

# stopRuleB was exact, so each value will be as close as possible while rule still is TRUE
#  Because we have discrete cells, these numbers will always slightly under the rule
foo <- cbind(vals=hab[stopRuleB], id = stopRuleB[stopRuleB>0]);
tapply(foo[,"vals"], foo[,"id"], sum) # Correct ... all are above 50

# stopRuleB_notExact will overshoot
foo <- cbind(vals=hab[stopRuleB_NotExact], id = stopRuleB_NotExact[stopRuleB_NotExact > 0]);
tapply(foo[,"vals"], foo[,"id"], sum) # Correct ... all are above 50


# Cellular automata shapes
# Diamonds - can make them with: a boolean raster, directions = 4,
#    stopRule in place, spreadProb = 1
diamonds <- spread(hab>0, spreadProb = 1, directions = 4,
   id = TRUE, stopRule = stopRule2)
if (interactive()) {
  clearPlot()
  Plot(diamonds)
}

# Squares - can make them with: a boolean raster, directions = 8,
#    stopRule in place, spreadProb = 1
squares <- spread(hab>0, spreadProb = 1, directions = 8,
   id = TRUE, stopRule = stopRule2)
if (interactive()) Plot(squares)

# Interference shapes - can make them with: a boolean raster, directions = 8,
#    stopRule in place, spreadProb = 1
stopRule2 <- function(landscape) sum(landscape)>200
squashedDiamonds <- spread(hab>0, spreadProb = 1, loci = (ncell(hab)-ncol(hab))/2 + c(4, -4),
   directions = 4, id = TRUE, stopRule = stopRule2)
if (interactive()) {
```

```
  clearPlot()
  Plot(squashedDiamonds)
}

# Circles with spreadProb < 1 will give "more" circular shapes, but definitely not circles
stopRule2 <- function(landscape) sum(landscape)>200
seed <- sample(1e4,1)
set.seed(seed)
circlish <- spread(hab>0, spreadProb = 0.23, loci = (ncell(hab)-ncol(hab))/2 + c(4, -4),
   directions = 8, id = TRUE, circle = TRUE)#, stopRule = stopRule2)
set.seed(seed)
regularCA <- spread(hab>0, spreadProb = 0.23, loci = (ncell(hab)-ncol(hab))/2 + c(4, -4),
   directions = 8, id = TRUE)#, stopRule = stopRule2)
if (interactive()) {
  clearPlot()
  Plot(circlish, regularCA)
}


####################
# complex stopRule
####################

initialLoci <- sample(seq_len(ncell(hab)), 2)#(ncell(hab)-ncol(hab))/2 + c(4, -4)
endSizes <- seq_along(initialLoci)*200

# Can be a function of landscape, id, and/or any other named
#   variable passed into spread

stopRule3 <- function(landscape, id, endSizes) sum(landscape)>endSizes[id]

TwoCirclesDiffSize <- spread(hab, spreadProb = 1, loci = initialLoci, circle = TRUE,
   directions = 8, id = TRUE, stopRule = stopRule3, endSizes = endSizes,
   stopRuleBehavior = "excludePixel")
# or using named list of named elements:
#TwoCirclesDiffSize <- spread(hab, spreadProb = 1, loci = initialLoci, circle = TRUE,
#    directions = 8, id = TRUE, stopRule = stopRule3,
#    vars = list(endSizes = endSizes), stopRuleBehavior = "excludePixel")

if (interactive()) {
  clearPlot()
  Plot(TwoCirclesDiffSize)
}
cirs <- getValues(TwoCirclesDiffSize)
vals <- tapply(hab[TwoCirclesDiffSize], cirs[cirs>0], sum)

# Stop if sum of landscape is big or mean of quality is too small
quality <- raster(hab)
quality[] <- runif(ncell(quality), 0, 1)
stopRule4 <- function(landscape, quality, cells) (sum(landscape)>20) | (mean(quality[cells])<0.3)

TwoCirclesDiffSize <- spread(hab, spreadProb = 1, loci = initialLoci, circle = TRUE,
   directions = 8, id = TRUE, stopRule = stopRule4, quality = quality,
```

```
   stopRuleBehavior = "excludePixel")

##############
# allowOverlap
##############
 set.seed(3113)
 initialLoci <- as.integer(sample(1:ncell(hab), 10))
 # using "landscape", "id", and a variable passed in
 maxVal <- rep(500,length(initialLoci))
 # define stopRule
 stopRule2 <- function(landscape,id,maxVal) sum(landscape)>maxVal[id]
 circs <- spread(hab, spreadProb = 1, circle = TRUE, loci = initialLoci, stopRule = stopRule2,
                  id = TRUE, allowOverlap = TRUE, stopRuleBehavior = "includeRing",
                                      maxVal = maxVal, returnIndices = TRUE)
 (vals <- tapply(hab[circs$indices], circs$id, sum))
 vals<=maxVal # All TRUE
 overlapEvents <- raster(hab)
 overlapEvents[] <- 0
 toMap <- circs[,sum(id),by=indices]
 overlapEvents[toMap$indices] <- toMap$V1
if (interactive()) {
  clearPlot()
  Plot(overlapEvents)
}
```

`times` *Time usage in* SpaDES

Description

Functions for the `simtimes` slot of a `simList` object and its elements. To maintain modularity, the behavior of these functions depends on where they are used. In other words, different modules can have their own timeunit. SpaDES converts these to seconds when running a simulation, but shows the user time in the units of the model as shown with `timeunit(sim)`

Usage

```
times(x, ...)

## S4 method for signature '.simList'
times(x)

times(x) <- value

## S4 replacement method for signature '.simList'
times(x) <- value
```

```
time(x, unit, ...)

## S4 method for signature '.simList,missing'
time(x)

## S4 method for signature '.simList,character'
time(x, unit)

time(x) <- value

## S4 replacement method for signature '.simList'
time(x) <- value

end(x, unit, ...)

## S4 method for signature '.simList,missing'
end(x)

## S4 method for signature '.simList,character'
end(x, unit)

end(x) <- value

## S4 replacement method for signature '.simList'
end(x) <- value

start(x, unit, ...)

## S4 method for signature '.simList,missing'
start(x)

## S4 method for signature '.simList,character'
start(x, unit)

start(x) <- value

## S4 replacement method for signature '.simList'
start(x) <- value

timeunit(x)

## S4 method for signature '.simList'
timeunit(x)

timeunit(x) <- value

## S4 replacement method for signature '.simList'
timeunit(x) <- value
```

```
timeunits(x)

## S4 method for signature '.simList'
timeunits(x)
```

Arguments

x	A simList simulation object.
...	Additional parameters.
value	A time, given as a numeric, optionally with a unit attribute, but this will be deduced from the model time units or module time units (if used within a module)
unit	Character. One of the time units used in SpaDES.

Details

NOTE: These have default behavior that is based on the calling frame timeunit. When used inside a module, then the time is in the units of the module. If used in an interactive mode, then the time will be in the units of the spades simulation.

Additonal methods are provided to access the current, start, and end times of the simulation:

time	Current simulation time.
start	Simulation start time.
end	Simulation end time.
timeunit	Simulation timeunit.
timeunits	Module timeunits.
times	List of all simulation times (current, start, end, timeunit).

timeunit will extract the current units of the time used in a simulation (i.e., within a spades call). If it is set within a simInit, e.g., times=list(start=0, end=52, timeunit = "week"), it will set the units for that simulation. By default, a simInit call will use the smallest unit contained within the metadata for the modules being used. If there are parent modules, then the parent module timeunit will be used even if one of its children is a smaller timeunit. If all modules, including parents, are set to NA, timeunit defaults to seconds. If parents are set to NA, then the set of modules defined by that parent module will be given the smallest units of the children.

Currently, available units are "second", "hours", day", "week", "month", and "year" can be used in the metadata of a module.

The user can also define a new unit. The unit name can be anything, but the function definition must be of the form, dunitName, e.g., dyear or dfortnight. The unit name is the part without the 'd' and the function name definition includes the 'd'. This new function, e.g., dfortNight <- function(x) lubridate::duration(dd can be placed anywhere in the search path or in a module.

timeunits will extract the current units of the time of all modules used in a simulation. This is different from timeunit because it is not necessarily associated with a spades call.

In many cases, the "simpler" use of each of these functions may be slower computationally. For instance, it is much faster to use time(sim, "year") than time(sim). So as a module developer, it is advantageous to write out the longer one, minimizing the looking up that R must do.

Value

Returns or sets the value of the slot from the simList object.

Author(s)

Alex Chubaty and Eliot McIntire

See Also

SpaDES, specifically the section 1.2.5 on Simulation times.

Other functions to access elements of a simList object: .addDepends, doEvent.checkpoint, envir, events, globals, inputs, ls.simList, ls.str.simList, modules, objs, packages, params, paths, progressInterval

---

transitions SELES - *Transitioning to next time step*

Description

Describes the probability of an agent successfully persisting until next time step. THIS IS NOT YET FULLY IMPLEMENTED.

A SELES-like function to maintain conceptual backwards compatability with that simulation tool. This is intended to ease transitions from SELES.

You must know how to use SELES for these to be useful.

Usage

```
transitions(p, agent)
```

Arguments

p	realized probability of persisting (i.e., either 0 or 1).
agent	SpatialPoints* object.

Value

Returns new SpatialPoints* object with potentially fewer agents.

Author(s)

Eliot McIntire

updateList *Update elements of a named list with elements of a second named list*

Description

Merge two named list based on their named entries. Where any element matches in both lists, the value from the second list is used in the updated list. Subelements are not examined and are simply replaced. If one list is empty, then it returns the other one, unchanged.

Usage

```
updateList(x, y)

## S4 method for signature 'list,list'
updateList(x, y)

## S4 method for signature '`NULL`,list'
updateList(x, y)

## S4 method for signature 'list,`NULL`'
updateList(x, y)

## S4 method for signature '`NULL`,`NULL`'
updateList(x, y)
```

Arguments

x	a named list
y	a named list

Value

A named list, with elements sorted by name. The values of matching elements in list y replace the values in list x.

Author(s)

Alex Chubaty

Examples

```
L1 <- list(a = "hst", b = NA_character_, c = 43)
L2 <- list(a = "gst", c = 42, d = list(letters))
updateList(L1, L2)

updateList(L1, NULL)
updateList(NULL, L2)
updateList(NULL, NULL) # should return empty list
```

`versionWarning` *Compare module version against SpaDES package version and warn if incompatible*

Description

Performs a basic check to ensure the module version is compatible with the SpaDES package version. Compatibility is best assured when both versions are equal. If module version < spades version, there is likely no problem, as SpaDES should be backwards compatible. However, if module version > spades version, the user needs to update their version of SpaDES because module compatibility cannot be assured.

Usage

```
versionWarning(moduleName, moduleVersion)

## S4 method for signature 'character'
versionWarning(moduleName, moduleVersion)
```

Arguments

`moduleName`	Character string providing the module name.
`moduleVersion`	The module version as either a character, numeric, or numeric version (e.g., extracted from module metadata). Is coerced to `numeric_version`.

Value

Logical (invisibly) indicating whether the module is compatible with the version of the SpaDES package. Will also produce a warning if not compatible.

Author(s)

Alex Chubaty

`wrap` *Wrap coordinates or pixels in a torus-like fashion*

Description

Generally for model development purposes.

Usage

```
wrap(X, bounds, withHeading)

## S4 method for signature 'matrix,Extent,missing'
wrap(X, bounds)

## S4 method for signature 'SpatialPoints,ANY,missing'
wrap(X, bounds)

## S4 method for signature 'matrix,Raster,missing'
wrap(X, bounds)

## S4 method for signature 'matrix,Raster,missing'
wrap(X, bounds)

## S4 method for signature 'matrix,matrix,missing'
wrap(X, bounds)

## S4 method for signature 'SpatialPointsDataFrame,Extent,logical'
wrap(X, bounds, withHeading)

## S4 method for signature 'SpatialPointsDataFrame,Raster,logical'
wrap(X, bounds, withHeading)

## S4 method for signature 'SpatialPointsDataFrame,matrix,logical'
wrap(X, bounds, withHeading)
```

Arguments

X	A SpatialPoints* object, or matrix of coordinates
bounds	Either a Raster*, Extent, or bbox object defining bounds to wrap around
withHeading	logical. If TRUE, then the previous points must be wrapped also so that the subsequent heading calculation will work. Default FALSE. See details.

Details

If withHeading used, then X must be a SpatialPointsDataFrame that contains two columns, x1 and y1, with the immediately previous agent locations.

Value

Same class as X, but with coordinates updated to reflect the wrapping

Author(s)

Eliot McIntire

Examples

```
library(raster)
xrange <- yrange <- c(-50, 50)
hab <- raster(extent(c(xrange, yrange)))
hab[] <- 0

# initialize agents
N <- 10

# previous points
x1 <- rep(0, N)
y1 <- rep(0, N)
# initial points
starts <- cbind(x = stats::runif(N, xrange[1], xrange[2]),
                y = stats::runif(N, yrange[1], yrange[2]))

# create the agent object
agent <- SpatialPointsDataFrame(coords = starts, data = data.frame(x1, y1))


ln <- rlnorm(N, 1, 0.02) # log normal step length
sd <- 30 # could be specified globally in params

if (interactive()) {
  clearPlot()
  Plot(hab, zero.color = "white", axes = "L")
}
for(i in 1:10) {
  agent <- SpaDES::crw(agent = agent,
                       extent = extent(hab), stepLength = ln,
                       stddev = sd, lonlat = FALSE, torus = TRUE)
  if (interactive()) Plot(agent, addTo = "hab", axes = TRUE)
}
```

---

zipModule *Create a zip archive of a module subdirectory*

Description

The most common use of this would be from a "modules" directory, rather than inside a given module.

Usage

```
zipModule(name, path, version, ...)

## S4 method for signature 'character,character,character'
zipModule(name, path, version, ...)
```

```
## S4 method for signature 'character,missing,character'
zipModule(name, path, version, ...)

## S4 method for signature 'character,missing,missing'
zipModule(name, path, version, ...)

## S4 method for signature 'character,character,missing'
zipModule(name, path, version, ...)
```

Arguments

name	Character string giving the module name.
path	A file path to a directory containing the module subdirectory.
version	The module version.
...	Additional arguments to zip: e.g., add "-q" using flags="-q -r9X" (the default flags are "-r9X").

Author(s)

Eliot McIntire and Alex Chubaty

Index