Package 'reinforcelearn'

April 9, 2019

Type Package

Title Reinforcement Learning

Version 0.2.1

Description Implements reinforcement learning environments and algorithms as described in Sutton & Barto (1998, ISBN:0262193981).
The Q-Learning algorithm can be used with function approximation,
eligibility traces (Singh & Sutton (1996) <doi:10.1007/BF00114726>)
and experience replay (Mnih et al. (2013) <arXiv:1312.5602>).

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.0.0)

RoxygenNote 6.1.1

BugReports https://github.com/markusdumke/reinforcelearn/issues

URL http://markusdumke.github.io/reinforcelearn

SystemRequirements (Python and gym only required if gym environments are used)

Imports checkmate (>= 1.8.4), R6 (>= 2.2.2), nnet (>= 7.3-12), purrr (>= 0.2.4)

Suggests reticulate, keras, knitr, rmarkdown, testthat, covr, lintr

VignetteBuilder knitr

NeedsCompilation no

Author Markus Dumke [aut, cre]

Maintainer Markus Dumke <markusdumke@gmail.com>

Repository CRAN

Date/Publication 2019-04-09 11:50:08 UTC

R topics documented:

`CliffWalking` *Cliff Walking*

Description

Gridworld environment for reinforcement learning from Sutton & Barto (2017). Grid of shape 4x12 with a goal state in the bottom right of the grid. Episodes start in the lower left state. Possible actions include going left, right, up and down. Some states in the lower part of the grid are a cliff, so taking a step into this cliff will yield a high negative reward of - 100 and move the agent back to the starting state. Elsewise rewards are - 1, for the goal state 0.

Arguments

...	[any] Arguments passed on to makeEnvironment.

Details

This is the gridworld (goal state denoted G, cliff states denoted C, start state denoted S):

```
. . . . . . . . . . . .
. . . . . . . . . . . .
. . . . . . . . . . . .
S C C C C C C C C C C G
```

Usage

```
makeEnvironment("cliff.walking", ...)
```

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the `done` flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

References

Sutton and Barto (Book draft 2017): Reinforcement Learning: An Introduction Example 6.6

Examples

```
env = makeEnvironment("cliff.walking")
```

`Eligibility` *Eligibility traces*

Description

Eligibility traces.

Arguments

`lambda`	[`numeric(1)` in (0, 1)] Trace decay parameter.
`traces`	[`character(1)`] Type of eligibility trace update. One of `c("replace", "accumulate")`.

Details

Algorithms supporting eligibility traces:

- QLearning

Examples

```
alg = makeAlgorithm("qlearning", lambda = 0.8, traces = "accumulate")
```

`Environment` *Custom Reinforcement Learning Environment*

Description

Custom Reinforcement Learning Environment

Arguments

`step`	[`function(self, action)`] Custom step function.
`reset`	[`function(self)`] Custom reset function.
`visualize`	[`function(self)`] Optional custom visualization function.
`discount`	[`numeric(1)` in (0, 1)] Discount factor.
`action.names`	[`named integer`] Optional action names for a discrete action space.

Usage

```
makeEnvironment("custom", step, reset, visualize = NULL, discount = 1, action.names = NULL)
```

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the `done` flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

Examples

```
step = function(self, action) {
  state = list(mean = action + rnorm(1), sd = runif(1))
  reward = rnorm(1, state[[1]], state[[2]])
  done = FALSE
  list(state, reward, done)
}

reset = function(self) {
  state = list(mean = 0, sd = 1)
  state
}

env = makeEnvironment(step = step, reset = reset)
env$reset()
env$step(100)
```

EpsilonGreedyPolicy *Epsilon Greedy Policy*

Description

Epsilon Greedy Policy

Arguments

epsilon	[numeric(1) in [0, 1]] Ratio of random exploration in epsilon-greedy action selection.

Usage

```
makePolicy("epsilon.greedy", epsilon = 0.1)
makePolicy("greedy")
```

Examples

```
policy = makePolicy("epsilon.greedy", epsilon = 0.1)
```

getEligibilityTraces *Get eligibility traces*

Description

Returns the eligibility traces of the agent.

Usage

```
getEligibilityTraces(agent)
```

Arguments

agent	[Agent] An agent created by makeAgent.

Value

A matrix with the eligibility traces.

getReplayMemory *Get replay memory.*

Description

Returns the replay memory of the agent.

Usage

```
getReplayMemory(agent)
```

Arguments

agent	[Agent] An agent created by makeAgent.

Value

A list containing the experienced observations, actions and rewards.

getStateValues *Get state values.*

Description

Get state value function from action value function.

Usage

```
getStateValues(action.vals)
```

Arguments

action.vals	[matrix] Action value matrix.

getValueFunction *Get weights of value function.*

Description

Returns the weights of the value function representation of the agent.

Usage

```
getValueFunction(agent)
```

Arguments

agent	[Agent] An agent created by makeAgent.

Value

For a value function table this will return a matrix, for a neural network a list with the weights of the layers.

Gridworld *Gridworld*

Description

Creates gridworld environments.

Arguments

shape	[integer(2)] Shape of the gridworld (number of rows x number of columns).
goal.states	[integer] Goal states in the gridworld.
cliff.states	[integer] Cliff states in the gridworld.
reward.step	[integer(1)] Reward for taking a step.
cliff.transition.states	[integer] States to which the environment transitions if stepping into the cliff. If it is a vector, all states will have equal probability. Only used when `cliff.transition.done == FALSE`, else specify the `initial.state` argument.
reward.cliff	[integer(1)] Reward for taking a step in the cliff state.
diagonal.moves	[logical(1)] Should diagonal moves be allowed?
wind	[integer] Strength of the upward wind in each cell.
cliff.transition.done	[logical(1)] Should the episode end after stepping into the cliff?
stochasticity	[numeric(1)] Probability of random transition to any of the neighboring states when taking any action.
...	[any] Arguments passed on to makeEnvironment.

Details

A gridworld is an episodic navigation task, the goal is to get from start state to goal state.

Possible actions include going left, right, up or down. If `diagonal.moves = TRUE` diagonal moves are also possible, leftup, leftdown, rightup and rightdown.

When stepping into a cliff state you get a reward of `reward.cliff`, usually a high negative reward and transition to a state specified by `cliff.transition.states`.

In each column a deterministic wind specified via `wind` pushes you up a specific number of grid cells (for the next action).

A stochastic gridworld is a gridworld where with probability `stochasticity` the next state is chosen at random from all neighbor states independent of the actual action.

If an action would take you off the grid, the new state is the nearest cell inside the grid. For each step you get a reward of `reward.step`, until you reach a goal state, then the episode is done.

States are enumerated row-wise and numeration starts with 0. Here is an example 4x4 grid:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

So a board position could look like this (G: goal state, x: current state, C: cliff state):

G	o	o	o
o	o	o	o
o	x	o	o
o	o	o	C

Usage

```
makeEnvironment("gridworld", shape = NULL, goal.states = NULL, cliff.states = NULL, reward.step = -1,
```

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the done flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

Examples

```
# Gridworld Environment (Sutton & Barto Example 4.1)
env1 = makeEnvironment("gridworld", shape = c(4L, 4L), goal.states = 0L,
  initial.state = 15L)
env1$reset()
env1$visualize()
env1$step(0L)
env1$visualize()

# Windy Gridworld (Sutton & Barto Example 6.5)
env2 = makeEnvironment("gridworld", shape = c(7, 10), goal.states = 37L,
```

```
  reward.step = -1, wind = c(0, 0, 0, 1, 1, 1, 2, 2, 1, 0),
  initial.state = 30L)

# Cliff Walking (Sutton & Barto Example 6.6)
env3 = makeEnvironment("gridworld", shape = c(4, 12), goal.states = 47L,
  cliff.states = 37:46, reward.step = -1, reward.cliff = -100,
  cliff.transition.states = 36L, initial.state = 36L)
```

`GymEnvironment` *Gym Environment*

Description

Reinforcement learning environment from OpenAI Gym.

Arguments

`gym.name`	[`character(1)`] Name of gym environment, e.g. `"CartPole-v0"`.
`...`	[any] Arguments passed on to makeEnvironment.

Details

For available gym environments take a look at https://gym.openai.com/envs.

Usage

```
makeEnvironment("gym", gym.name, ...)
```

Installation

For installation of the python package `gym` see https://github.com/openai/gym#installation. Then install the R package `reticulate`.

Methods

- `$close()` Close visualization window.
- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the done flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

Examples

```
## Not run:
# Create an OpenAI Gym environment.
# Make sure you have Python, gym and reticulate installed.
env = makeEnvironment("gym", gym.name = "MountainCar-v0")
env$reset()
env$close()

## End(Not run)
```

`interact` *Interaction between agent and environment.*

Description

Run interaction between agent and environment for specified number of steps or episodes.

Usage

```
interact(env, agent, n.steps = Inf, n.episodes = Inf,
  max.steps.per.episode = Inf, learn = TRUE, visualize = FALSE)
```

Arguments

`env`	[Environment] Reinforcement learning environment created by makeEnvironment.
`agent`	[Agent] Agent created by makeAgent.
`n.steps`	[integer(1)] Number of steps to run.
`n.episodes`	[integer(1)] Number of episodes to run.
`max.steps.per.episode`	[integer(1)] Maximal number of steps allowed per episode.
`learn`	[logical(1)] Should the agent learn?
`visualize`	[logical(1)] Visualize the interaction between agent and environment?

Value

`list` Return and number of steps per episode.

Examples

```
env = makeEnvironment("windy.gridworld")
agent = makeAgent("softmax", "table", "qlearning")
interact(env, agent, n.episodes = 10L)
```

`makeAgent` *Create Agent.*

Description

An agent consists of a policy and (optional) a value function representation and (optional) a learning algorithm.

Usage

```
makeAgent(policy, val.fun = NULL, algorithm = NULL,
  preprocess = identity, replay.memory = NULL, policy.args = list(),
  val.fun.args = list(), algorithm.args = list())
```

Arguments

`policy`	[`character(1)` | Policy] A policy. If you pass a string the policy will be created via makePolicy.
`val.fun`	[`character(1)` | ValueFunction] A value function representation. If you pass a string the value function will be created via makeValueFunction.
`algorithm`	[`character(1)` | Algorithm] An algorithm. If you pass a string the algorithm will be created via makeAlgorithm.
`preprocess`	[`function`] A function which preprocesses the state so that the agent can learn on this.
`replay.memory`	[`ReplayMemory`] Replay memory for experience replay created by makeReplayMemory.
`policy.args`	[`list`] Arguments passed on to `args` in makePolicy.
`val.fun.args`	[`list`] Arguments passed on to `args` in makeValueFunction.
`algorithm.args`	[`list`] Arguments passed on to `args` in makeAlgorithm.

Examples

```
agent = makeAgent("softmax", "table", "qlearning")
```

makeAlgorithm *Make reinforcement learning algorithm.*

Description

Make reinforcement learning algorithm.

Usage

```
makeAlgorithm(class, args = list(), ...)
```

Arguments

class	[character(1)] Algorithm. One of c("qlearning").
args	[list] Optional list of named arguments passed on to the subclass. The arguments in ... take precedence over values in this list. We strongly encourage you to use one or the other to pass arguments to the function but not both.
...	[any] Optional named arguments passed on to the subclass. Alternatively these can be given using the args argument.

Representations

- QLearning

Examples

```
alg = makeAlgorithm("qlearning")
```

makeEnvironment *Create reinforcement learning environment.*

Description

This function creates an environment for reinforcement learning.

Usage

```
makeEnvironment(class = "custom", discount = 1, ...)
```

Arguments

`class`	[`character(1)`] Class of environment. One of `c("custom", "mdp", "gym", "gridworld")`.
`discount`	[`numeric(1)` in (0, 1)] Discount factor.
`...`	[`any`] Arguments passed on to the specific environment.

Details

Use the `step` method to interact with the environment.

Note that all states and actions are numerated starting with 0!

For a detailed explanation and more examples have a look at the vignette "How to create an environment?".

Value

R6 class of class Environment.

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the `done` flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

Environments

- Environment
- GymEnvironment
- MdpEnvironment
- Gridworld
- MountainCar

Examples

```
step = function(self, action) {
  state = list(mean = action + rnorm(1), sd = runif(1))
  reward = rnorm(1, state[[1]], state[[2]])
  done = FALSE
  list(state, reward, done)
}

reset = function(self) {
```

```
  state = list(mean = 0, sd = 1)
  state
}

env = makeEnvironment(step = step, reset = reset, discount = 0.9)
env$reset()
env$step(100)

# Create a Markov Decision Process.
P = array(0, c(2, 2, 2))
P[, , 1] = matrix(c(0.5, 0.5, 0, 1), 2, 2, byrow = TRUE)
P[, , 2] = matrix(c(0, 1, 0, 1), 2, 2, byrow = TRUE)
R = matrix(c(5, 10, -1, 2), 2, 2, byrow = TRUE)
env = makeEnvironment("mdp", transitions = P, rewards = R)

env$reset()
env$step(1L)

# Create a Gridworld.
grid = makeEnvironment("gridworld", shape = c(4, 4),
  goal.states = 15, initial.state = 0)
grid$visualize()

## Not run:
# Create an OpenAI Gym environment.
# Make sure you have Python, gym and reticulate installed.
env = makeEnvironment("gym", gym.name = "MountainCar-v0")

# Take random actions for 200 steps.
env$reset()
for (i in 1:200) {
  action = sample(env$actions, 1)
  env$step(action)
  env$visualize()
}
env$close()

## End(Not run)
```

`makePolicy` *Create policy.*

Description

Reinforcement learning policies.

Usage

```
makePolicy(class = "random", args = list(), ...)
```

Arguments

`class`	[`character(1)`] Class of policy. One of `c("random", "epsilon.greedy", "greedy", "softmax")`.
`args`	[`list`] Optional list of named arguments passed on to the subclass. The arguments in ... take precedence over values in this list. We strongly encourage you to use one or the other to pass arguments to the function but not both.
`...`	[`any`] Optional named arguments passed on to the subclass. Alternatively these can be given using the `args` argument.

Value

`list(name, args)` List with the name and optional args. This list can then be passed onto makeAgent, which will construct the policy accordingly.

Policies

- RandomPolicy
- GreedyPolicy
- EpsilonGreedyPolicy
- SoftmaxPolicy

Examples

```
policy = makePolicy("random")
policy = makePolicy("epsilon.greedy", epsilon = 0.1)
```

`makeReplayMemory` *Experience Replay*

Description

Create replay memory for experience replay.

Usage

```
makeReplayMemory(size = 100L, batch.size = 16L)
```

Arguments

`size`	[`integer(1)`] Size of replay memory.
`batch.size`	[`integer(1)`] Batch size.

Details

Sampling from replay memory will be uniform.

Value

`list(size, batch.size)` This list can then be passed onto makeAgent, which will construct the replay memory accordingly.

Examples

```
memory = makeReplayMemory(size = 100L, batch.size = 16L)
```

---

makeValueFunction *Value Function Representation*

Description

A representation of the value function.

Usage

```
makeValueFunction(class, args = list(), ...)
```

Arguments

`class`	[`character(1)`] Class of value function approximation. One of `c("table", "neural.network")`.
`args`	[`list`] Optional list of named arguments passed on to the subclass. The arguments in ... take precedence over values in this list. We strongly encourage you to use one or the other to pass arguments to the function but not both.
`...`	[`any`] Optional named arguments passed on to the subclass. Alternatively these can be given using the `args` argument.

Value

`list(name, args)` List with the name and optional args. This list can then be passed onto makeAgent, which will construct the value function accordingly.

Representations

- ValueTable
- ValueNetwork

Examples

```
val = makeValueFunction("table", n.states = 16L, n.actions = 4L)
# If the number of states and actions is not supplied, the agent will try
# to figure these out from the environment object during interaction.
val = makeValueFunction("table")
```

`MdpEnvironment` *MDP Environment*

Description

Markov Decision Process environment.

Arguments

`transitions`	[`array (n.states x n.states x n.actions)`] State transition array.
`rewards`	[`matrix (n.states x n.actions)`] Reward array.
`initial.state`	[`integer`] Optional starting state. If a vector is given a starting state will be randomly sampled from this vector whenever `reset` is called. Note that states are numerated starting with 0. If `initial.state = NULL` all non-terminal states are possible starting states.
`...`	[`any`] Arguments passed on to makeEnvironment.

Usage

```
makeEnvironment("MDP", transitions, rewards, initial.state, ...)
```

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the `done` flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

Examples

```
# Create a Markov Decision Process.
P = array(0, c(2, 2, 2))
P[, , 1] = matrix(c(0.5, 0.5, 0, 1), 2, 2, byrow = TRUE)
P[, , 2] = matrix(c(0, 1, 0, 1), 2, 2, byrow = TRUE)
R = matrix(c(5, 10, -1, 2), 2, 2, byrow = TRUE)
env = makeEnvironment("mdp", transitions = P, rewards = R)
env$reset()
env$step(1L)
```

MountainCar *Mountain Car*

Description

The classical mountain car problem for reinforcement learning.

Arguments

`...`	[any] Arguments passed on to makeEnvironment.

Format

An object of class `R6ClassGenerator` of length 24.

Details

The classical Mountain Car task the action is one of 0, 1, 2, in the continuous version the action is in [-1, 1].

Usage

```
makeEnvironment("MountainCar", ...)
makeEnvironment("MountainCarContinuous", ...)
```

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the `done` flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

Examples

```
env = makeEnvironment("mountain.car")
env$reset()
env$step(1L)

env = makeEnvironment("mountain.car.continuous")
env$reset()
env$step(0.62)
```

---

nHot *Make n hot vector.*

Description

Make n hot vector.

Usage

```
nHot(x, len, out = "matrix")
```

Arguments

x	[integer] Which features are active?
len	[integer(1)] Length of the feature vector.
out	[character(1)] Format of the output. Can be a vector or a matrix.

Value

`matrix(1, len)` A one-row matrix with `len` columns with every entry 0 except the columns specified by `x` which are 1.

Examples

```
nHot(c(1, 3), 5)
nHot(c(1, 3), 5, out = "vector")
```

QLearning *Q-Learning*

Description

Q-Learning algorithm.

Arguments

lambda	[numeric(1) in (0, 1)] Trace decay parameter.
traces	[character(1)] Type of eligibility trace update. One of c("replace", "accumulate").

Details

To use eligibility traces specify lambda and traces.

Usage

```
makeAlgorithm("qlearning", lambda, traces)
```

See Also

Eligibility

Examples

```
alg = makeAlgorithm("qlearning", lambda = 0.8, traces = "accumulate")
```

RandomPolicy *Random Policy*

Description

Random Policy

Usage

```
makePolicy("random")
```

Examples

```
pol = makePolicy("random")
```

`reinforcelearn` *Reinforcement Learning.*

Description

Implementations of reinforcement learning algorithms and environments.

Environments

- makeEnvironment
- Environment
- GymEnvironment
- MdpEnvironment
- Gridworld
- WindyGridworld
- CliffWalking
- MountainCar
- MountainCarContinuous

Policies

- makePolicy
- EpsilonGreedyPolicy
- GreedyPolicy
- SoftmaxPolicy
- RandomPolicy

Value Function Representations

- makeValueFunction
- ValueTable
- ValueNetwork

Algorithms

- makeAlgorithm
- QLearning

Extensions

- makeReplayMemory
- Eligibility

Agent

- makeAgent
- getValueFunction
- getReplayMemory
- getEligibilityTraces

Interaction

- interact

`SoftmaxPolicy` *Softmax Policy*

Description

Softmax Policy

Usage

```
makePolicy("softmax")
```

Examples

```
pol = makePolicy("softmax")
```

`tiles` *Tile Coding*

Description

Implementation of Sutton's tile coding software version 3.

Usage

```
tiles(iht, n.tilings, state, action = integer(0))

iht(max.size)
```

Arguments

`iht`	[IHT] A hash table created with `iht`.
`n.tilings`	[integer(1)] Number of tilings.
`state`	[vector(2)] A two-dimensional state observation. Make sure to scale the observation to unit variance before.
`action`	[integer(1)] Optional: If supplied the action space will also be tiled. All distinct actions will result in different tile numbers.
`max.size`	[integer(1)] Maximal size of hash table.

Details

Tile coding is a way of representing the values of a vector of continuous variables as a large binary vector with few 1s and many 0s. The binary vector is not represented explicitly, but as a list of the components that are 1s. The main step is to partition, or tile, the continuous space multiple times and select one tile from each tiling, that corresponding the the vector's value. Each tile is converted to an element in the big binary vector, and the list of the tile (element) numbers is returned as the representation of the vector's value. Tile coding is recommended as a way of applying online learning methods to domains with continuous state or action variables. [copied from manual]

See detailed manual on the web. In comparison to the Python implementation indices start with 1 instead of 0. The hash table is implemented as an environment, which is an attribute of an R6 class.

Make sure that the size of the hash table is large enough, else an error will be triggered, when trying to assign a value to a full hash table.

Value

`iht` creates a hash table, which can then be passed on to `tiles`. `tiles` returns an integer vector of size `n.tilings` with the active tile numbers.

References

Sutton and Barto (Book draft 2017): Reinforcement Learning: An Introduction

Examples

```
# Create hash table
hash = iht(1024)

# Partition state space using 8 tilings
tiles(hash, n.tilings = 8, state = c(3.6, 7.21))
tiles(hash, n.tilings = 8, state = c(3.7, 7.21))
tiles(hash, n.tilings = 8, state = c(4, 7))
tiles(hash, n.tilings = 8, state = c(- 37.2, 7))
```

ValueNetwork *Value Network*

Description

Neural network representing the action value function Q.

Arguments

model	[keras model] A keras model. Make sure that the model has been compiled.

Usage

```
makeValueFunction("neural.network", model)
```

Examples

```
## Not run:
library(keras)
model = keras_model_sequential()
model %>% layer_dense(20, input_shape = 10, activation = "relu")
model %>% layer_dense(4, activation = "softmax")
keras::compile(model, loss = "mae", optimizer = keras::optimizer_sgd(lr = 0.4))

val = makeValueFunction("neural.network", model = model)

## End(Not run)
```

ValueTable *Value Table*

Description

Table representing the action value function Q.

Arguments

n.states	[integer(1)] Number of states (rows in the value function).
n.actions	[integer(1)] Number of actions (columns in the value function).
step.size	[numeric(1)] Step size (learning rate) for gradient descent update.

Details

You can specify the shape of the value table. If omitted the agent will try to configure these automatically from the environment during interaction (therefore the environment needs to have a `n.states` and `n.actions` attribute).

Usage

```
makeValueFunction("table", n.states = NULL, n.actions = 1L, step.size = 0.1, initial.value = NULL)
```

Examples

```
val = makeValueFunction("table", n.states = 20L, n.actions = 4L)
```

WindyGridworld *Windy Gridworld*

Description

Windy Gridworld problem for reinforcement learning. Actions include going left, right, up and down. In each column the wind pushes you up a specific number of steps (for the next action). If an action would take you off the grid, you remain in the previous state. For each step you get a reward of -1, until you reach into a terminal state.

Arguments

`...`	[any] Arguments passed on to makeEnvironment.

Details

This is the gridworld (goal state denoted G, start state denoted S). The last row specifies the upward wind in each column.

```
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
S . . . . . . G . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
0 0 0 1 1 1 2 2 1 0
```

Usage

```
makeEnvironment("windy.gridworld", ...)
```

Methods

- `$step(action)`
  Take action in environment. Returns a list with `state`, `reward`, `done`.
- `$reset()`
  Resets the `done` flag of the environment and returns an initial state. Useful when starting a new episode.
- `$visualize()`
  Visualizes the environment (if there is a visualization function).

References

Sutton and Barto (Book draft 2017): Reinforcement Learning: An Introduction Example 6.5

Examples

```
env = makeEnvironment("windy.gridworld")
```

Index