Define custom environment for deep reinforcement learn

rlR: Define custom environments for Deep Reinforcement learning in R

Environment class

If you want to use this package for your self defined task, you need to implement your own R6 class to represent the environment. Below is a template. Where the listed fields (member variables and methods) must exist in your implemented R6 class. You could define other public and private members as you like.

```
library(rlR)
library(R6)
MyEnv = R6::R6Class("MyEnv",
  public = list(
    state_dim = NULL,  # obligatory field, should be vector. c(28, 28, 3) which is usually the dimensio
    act_cnt = NULL, # obligatory field, should be type integer
    s_r_d_info = NULL,
    step_cnt = NULL,
    initialize = function(...) {
      # put your initialization code here for example
      self$state_dim = 4
      self$act_cnt = 2
      self$step_cnt = 0L
      self$s_r_d_info = vector(mode = "list", length = 4)
      names(self$s_r_d_info) = c("state", "reward", "done", "info")
      self$s_r_d_info[["reward"]] = 1  # design your own reward scheme in step function below
      self$s_r_d_info[["done"]] = FALSE  # whether the episode is finished?
      self$s_r_d_info[["state"]] = array(0, dim = self$state_dim) # state must be array of the same dim
      self$s_r_d_info[["info"]] = list()  # info can be arbitray object
      as.array
    },

    render = function(...) {
      # you could leave this field empty.
    },

    # this function will be called at each step of the learning
    step = function(action) {
      # input your custom code here
      self$step_cnt = self$step_cnt + 1L
      self$s_r_d_info[["reward"]] = rnorm(1L)  # design your own reward scheme in step function below
      if(self$step_cnt > 5L)  self$s_r_d_info[["done"]] = TRUE
      self$s_r_d_info
    },

    # this function will be called at the beginning of the learning and at the end of each episode
    reset = function() {
      # input your custom function FALSE
      self$step_cnt = 0
      self$s_r_d_info[["done"]] = FALSE
```

```
      self$s_r_d_info
    },

    afterAll = function() {
      # what to do after the whole learning is finished?  could be left empty
    }
    ),
  private = list(),
  active = list()
  )
```

Afterwards you could choose one of the available Agents to learn on this newly defined environments.

```
env = MyEnv$new()
listAvailAgent()
## [1] "AgentDQN:deep q learning"
## [2] "AgentFDQN:frozen target deep q learning"
## [3] "AgentDDQN: double deep q learning"
## [4] "AgentPG: policy gradient basic"
## [5] "AgentPGBaseline: policy gradient with baseline"
## [6] "AgentActorCritic: actor critic method"
agent = makeAgent("AgentDQN", env)
## parameters:
## -render: - FALSE-
## -agent.gamma: - 0.99-
## -policy.maxEpsilon: - 1-
## -policy.minEpsilon: - 0.001-
## -policy.decay: - 0.999000499833375-
## -replay.memname: - Uniform-
## -replay.epochs: - 1-
## -interact.maxiter: - 500-
## -console: - FALSE-
## -log: - FALSE-
## -policy.name: - EpsilonGreedy-
## -replay.batchsize: - 64-
## -agent.nn.arch: nhidden- 64-
##  -agent.nn.arch: act1- relu-
##  -agent.nn.arch: act2- linear-
##  -agent.nn.arch: loss- mse-
##  -agent.nn.arch: lr- 0.00025-
##  -agent.nn.arch: kernel_regularizer- regularizer_l2(l=0.0)-
##  -agent.nn.arch: bias_regularizer- regularizer_l2(l=0.0)-
agent$updatePara("console", FALSE)
perf = agent$learn(10)
```