valueIterationAgents.py (original)

```
# valueIterationAgents.py
# -----------------------
# Licensing Information: Please do not distribute or publish solutions to this
# project. You are free to use and extend these projects for educational
# purposes. The Pacman AI projects were developed at UC Berkeley, primarily by
# John DeNero (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# For more info, see http://inst.eecs.berkeley.edu/~cs188/sp09/pacman.html

import mdp, util

from learningAgents import ValueEstimationAgent

class ValueIterationAgent(ValueEstimationAgent):
  """
      * Please read learningAgents.py before reading this.*

      A ValueIterationAgent takes a Markov decision process
      (see mdp.py) on initialization and runs value iteration
      for a given number of iterations using the supplied
      discount factor.
  """
  def __init__(self, mdp, discount = 0.9, iterations = 100):
    """
      Your value iteration agent should take an mdp on
      construction, run the indicated number of iterations
      and then act according to the resulting policy.

      Some useful mdp methods you will use:
          mdp.getStates()
          mdp.getPossibleActions(state)
          mdp.getTransitionStatesAndProbs(state, action)
          mdp.getReward(state, action, nextState)
    """
    self.mdp = mdp
    self.discount = discount
    self.iterations = iterations
    self.values = util.Counter() # A Counter is a dict with default 0

    "*** YOUR CODE HERE ***"

  def getValue(self, state):
    """
      Return the value of the state (computed in __init__).
    """
    return self.values[state]


  def getQValue(self, state, action):
    """
      The q-value of the state action pair
      (after the indicated number of value iteration
      passes).  Note that value iteration does not
      necessarily create this quantity and you may have
      to derive it on the fly.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

  def getPolicy(self, state):
    """
      The policy is the best action in the given state
      according to the values computed by value iteration.
      You may break ties any way you see fit.  Note that if
      there are no legal actions, which is the case at the
```

```
      terminal state, you should return None.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

  def getAction(self, state):
    "Returns the policy at the state (no exploration)."
    return self.getPolicy(state)
```