qlearningAgents.py (original)

```
# qlearningAgents.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

from game import *
from learningAgents import ReinforcementAgent
from featureExtractors import *

import random,util,math

class QLearningAgent(ReinforcementAgent):
  """
    Q-Learning Agent

    Functions you should fill in:
      - getQValue
      - getAction
      - getValue
      - getPolicy
      - update

    Instance variables you have access to
      - self.epsilon (exploration prob)
      - self.alpha
      - self.gamma

    Functions you
      - self.getLegalActions(state)
        which returns legal actions
        for a state
  """
  def __init__(self, **args):
    "You can initialize Q-values here..."
    ReinforcementAgent.__init__(self, **args)

    "*** YOUR CODE HERE ***"

  def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we never seen
      a state or (state,action) tuple
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()


  def getValue(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

  def getPolicy(self, state):
    """
      Compute the best action to take in a state.  Note that if there
```

Assignment Project Exam Help
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

```
      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):
    """
      Compute the action to take in the current state.  With
      probability self.epsilon, we should take a random action and
      take the best policy action otherwise.  Note that if there are
      no legal actions, which is the case at the terminal state, you
      should choose None as the action.

      HINT: You might want to use util.flipCoin(prob)
      HINT: To pick randomly from a list, use random.choice(list)
    """
    # Pick Action
    legalActions = self.getLegalActions(state)
    action = None
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

    return action

  def update(self, state, action, nextState, reward):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be c
    """
    "*** YOUR CODE
    util.raiseNotD

class PacmanQAgent(QLearningAgent):
  "Exactly the same as QLearningAgent, but with different default parameters"

  def __init__(self, epsilon=0.05,gamma=0.8,alpha=0.2, numTraining=0, **args):
    """
    These default parameters can be changed from the pacman.py command line.
    For example, to change the exploration rate, try:
        python pacman.py -p PacmanQLearningAgent -a epsilon=0.1

    alpha    - learning rate
    epsilon  - exploration rate
    gamma    - discount factor
    numTraining - number of training episodes, i.e. no learning after these many
episodes
    """
    args['epsilon'] = epsilon
    args['gamma'] = gamma
    args['alpha'] = alpha
    args['numTraining'] = numTraining
    self.index = 0  # This is always Pacman
    QLearningAgent.__init__(self, **args)

  def getAction(self, state):
    """
    Simply calls the getAction method of QLearningAgent and then
    informs parent of action for Pacman.  Do not change or remove this
    method.
    """
    action = QLearningAgent.getAction(self,state)
    self.doAction(state,action)
    return action
```

Assignment Project Exam Help
https://eduassistpro.github.io/
Add WeChat edu_assist_pro

```
class ApproximateQAgent(PacmanQAgent):
  """
     ApproximateQLearningAgent

     You should only have to overwrite getQValue
     and update.  All other QLearningAgent functions
     should work as is.
  """
  def __init__(self, extractor='IdentityExtractor', **args):
    self.featExtractor = util.lookup(extractor, globals())()
    PacmanQAgent.__init__(self, **args)

    # You might want to initialize weights here.
    "*** YOUR CODE HERE ***"

  def getQValue(self, state, action):
    """
      Should return Q(state,action) = w * featureVector
      where * is the dotProduct operator
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

  def update(self, state, action, nextState, reward):
    """
       Should update your weights based on transition
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()

  def final(self, state):
    "Called at the
    # call the sup
    PacmanQAgent.f

    # did we finish training?
    if self.episodesSoFar == self.numTrainin
      # you might want to print your weights
      "*** YOUR CODE HERE ***"
      pass
```

Assignment Project Exam Help
https://eduassistpro.github.io/
Add WeChat edu_assist_pro