```python
# valueIterationAgents.py
# -----------------------
# Licensing Information:  You are free to use or extend these projects for
# educational purposes provided that (1) you do not distribute or publish
# solutions, (2) you retain this notice, and (3) you provide clear
# attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
#
# Attribution Information: The Pacman AI projects were developed at UC Berkeley.
# The core projects and autograders were primarily created by John DeNero
# (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# Student side autograding was added by Brad Miller, Nick Hay, and
# Pieter Abbeel (pabbeel@cs.berkeley.edu).


# valueIterationAgents.py
# -----------------------
# Licensing Information:  You are free to use or extend these projects for
# educational purposes provided that (1) you do not distribute or publish
# solutions, (2) you retain this notice, and (3) you provide clear
# attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
#
# Attribution Information: The Pacman AI projects were developed at UC Berkeley.
# The core projects and autograders were primarily created by John DeNero
# (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# Student side autograding was added by Brad Miller, Nick Hay, and
# Pieter Abbeel (pabbeel@cs.berkeley.edu).


import mdp, util

from learningAgents import ValueEstimationAgent
import collections

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: mdp.MarkovDecisionProcess, 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)
              mdp.isTerminal(state)
        """
        self.mdp = mdp
        self.discount = discount
        self.iterations = iterations
        self.values = util.Counter() # A Counter is a dict with default 0
        self.runValueIteration()

    def runValueIteration(self):
        """
          Run the value iteration algorithm. Note that in standard
          value iteration, V_k+1(...) depends on V_k(...)'s.
        """
        for i in range(self.iterations):
            new_values = {}
            for state in self.mdp.getStates():
                if self.mdp.isTerminal(state):
                    new_values[state] = 0
                else:
                    max_Q = float('-inf')
                    for action in self.mdp.getPossibleActions(state):
                        Q_value = self.computeQValueFromValues(state, action)
                        if Q_value > max_Q:
                            max_Q = Q_value
                    new_values[state] = max_Q
            self.values = new_values

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

```python
def computeQValueFromValues(self, state, action):
    """
      Compute the Q-value of action in state from the
      value function stored in self.values.
    """
    Q = 0
    for next_state, P in self.mdp.getTransitionStatesAndProbs(state, action):
        V = self.values[next_state]
        R = self.mdp.getReward(state, action, next_state)
        Q += P * (R + self.discount * V)
    return Q
    #util.raiseNotDefined()

def computeActionFromValues(self, state):
    """
      The policy is the best action in the given state
      according to the values currently stored in self.values.

      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.
    """
    if self.mdp.isTerminal(state):
        return None
    best_action = None
    max_Q = float('-inf')
    for action in self.mdp.getPossibleActions(state):
        Q_value = self.computeQValueFromValues(state, action)
        # in case of a tie we keep the first action
        if Q_value > max_Q:
            max_Q = Q_value
            best_action = action
    return best_action
    #util.raiseNotDefined()

def getPolicy(self, state):
    return self.computeActionFromValues(state)

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

def getQValue(self, state, action):
    return self.computeQValueFromValues(state, action)
```