Assignment 3 by Haobin Tang

Explaination: For policy_evaluation, policy_improvement, policy_iteration and value_iteration, the parameters P, nS, nA, gamma are defined as follows:

```
P: nested dictionary
    From gym.core.Environment
    For each pair of states in [1, nS] and actions in [1, nA], P[state][ac
tion] is a
    tuple of the form (probability, nextstate, reward, terminal) where
        - probability: float
            the probability of transitioning from "state" to "nextstate" w
ith "action"
        - nextstate: int
            denotes the state we transition to (in range [0, nS - 1])
        - reward: int
            either 0 or 1, the reward for transitioning from "state" to
            "nextstate" with "action"
        - terminal: bool
          True when "nextstate" is a terminal state, False otherwise
nS: int
    number of states in the environment
nA: int
    number of actions in the environment
gamma: float
    Discount factor. Number in range [0, 1)
```

policy_evaluation

In [2]:

```
import numpy as np
import random
```

In [1]:

```
def policy_evaluation(P, nS, nA, policy, gamma=0.9, tol=1e-3):
    value_function = np.zeros(nS)
    while True:
        prev_value_function = np.zeros(nS)
        for i in range(nS):
            #policy_s=policy[i]
            for prob, next_state, reward, terminal in P[i][policy[i]]:
                prev_value_function[i] += prob * (reward+ gamma * value_function
[next_state])
        var = np.linalg.norm(prev_value_function-value_function, ord=np.inf)
        if var > tol:
            value_function = prev_value_function
        else :
            break
    return value_function
```

policy_improvement

In [4]:

```
def policy_improvement(P, nS, nA, value_from_policy, policy, gamma=0.9):
    new_policy = np.zeros(nS, dtype='int')
    for i in range(nS):
        Qvalue = np.zeros(nA)
        for j in range(nA):
            for prob, next_state, reward, terminal in P[i][j]:
                #if terminal:
                 #   Qvalue[j]+= prob * reward
                #else:
                Qvalue[j]+= prob * (reward + gamma * value_from_policy[next_stat
e])
        new_policy[i]=np.argmax(Qvalue)
    return new_policy
```

policy_iteration

In [5]:

```
def policy_iteration(P, nS, nA, gamma=0.9, tol=10e-3):
    value_function = np.zeros(nS)
    policy = np.zeros(nS, dtype=int)
    while True:
        new_policy=np.zeros(nS, dtype=int)
        value_function=policy_evaluation(P, nS, nA, policy, gamma=0.9, tol=10e-3
)
        new_policy=policy_improvement(P, nS, nA, value_function, policy, gamma=
0.9)

        if (policy==new_policy).all():
           # np.array_equal()
            break
        else:
            policy=new_policy
    return value_function, policy
```

value_iteration

In [6]:

```
def value_iteration(P, nS, nA, gamma=0.9, tol=1e-3):
    value_function = np.zeros(nS)
    policy = np.zeros(nS, dtype=int)
    while True:
        prev_value_function = np.ones(nS)
        for i in range(nS):
            Qvalue = np.zeros(nA)

            for j in range(nA):
                for prob, next_state, reward, terminal in P[i][j]:
                    Qvalue[j] += prob * (reward + gamma * value_function[next_st
ate])

            prev_value_function[i]=np.max(Qvalue)
            policy[i]=np.argmax(Qvalue)

        var=np.linalg.norm(prev_value_function-value_function, ord=np.inf)
        if var>tol:
            value_function=prev_value_function
        else:
            break
    return value_function, policy
```