Assignment 5 by Haobin Tang

1. Write code for the interface for tabular RL algorithms. The core of this interface should be a mapping from a (state, action) pair to a sampling of the (next state, reward) pair. It is important that this interface doesn't present the state-transition probability model or the reward model.

In [1]:
```
import numpy as np
import random as random
```

In [2]:
```
''' Build a simulator exmaple behind the interface. As I discussed with professor Rao, the simulat
or can be both determinisitic
or stochastic, but we should assume we don't know in the RL interface and just send in (state, act
ion) pair and
get (next state, reward)
'''
def simulator_example(state, action):
    next_state=np.random.binomial(20, 0.5, size=None)
    action=state*action
    return next_state, action

def get_mdp_rep_for_rl_tabular(data):
    output=[]
    for i in data:
        next_state, action=simulator_example(i[0],i[1])
        output.append([next_state,action])
    return output
```

In [3]:
```
data=[[0,5],[2,3],[12,2],[6,0],[4,1]]
get_mdp_rep_for_rl_tabular(data)
```

Out[3]:
```
[[12, 0], [15, 6], [7, 24], [10, 0], [10, 4]]
```

2. Implement any tabular Monte-Carlo algorithm for Value Function prediction

In [12]:
```
def MC(size,mc_path, visit="first"):
    V, counts_dict,_=env(size)
    for episode in mc_path:
        _, _, first_visit=env(size)
        for i in episode:
            s=i[0]
            G=i[1]
            if  visit=="first":
                if first_visit[s]==0:
                    counts_dict[s] += 1
                    c = counts_dict[s]
                    V[s] = (V[s] * (c - 1) + G) / c
                    first_visit[s]=1
            if  visit=="every":
                counts_dict[s] += 1
                c = counts_dict[s]
                V[s] = (V[s] * (c - 1) + G) / c
    return V
```

In [7]:
```
def env(size):
    V={}
    counts_dict={}
    first_visit={}
    for i in range(size):
        V[i]=0
        counts_dict[i]=0
        first_visit[i]=0
    return V, counts_dict, first_visit
```

```
In [20]: #mc_path contains the data of episodes. Each episode there are (state, G) pairs.
         mc_path=[
             [[0,5],[1,3],[3,2],[0,2],[5,2]],
             [[0,3],[2,3],[4,3]],
             [[1,6],[2,4],[4,3],[1,2],[5,2]],
             [[1,6],[3,4],[4,3],[3,2],[0,2]],
             [[1,6],[4,4],[4,3],[2,2],[1,2]],
         ]
```

```
In [18]: MC(6, mc_path, "first")

Out[18]: {0: 3.3333333333333335, 1: 5.25, 2: 3.0, 3: 3.0, 4: 3.25, 5: 2.0}
```

```
In [19]: MC(6, mc_path,"every")

Out[19]: {0: 3.0, 1: 4.166666666666667, 2: 3.0, 3: 2.6666666666666665, 4: 3.2, 5: 2.0}
```

3. Implement tabular 1-step TD algorithm for Value Function prediction

```
In [27]: def TD(size,TD_path,gamma,alpha=0):
             V, counts_dict=env(size)

             for episode in TD_path:
                 for j,i in enumerate(episode):
                     s=i[0]
                     r=i[1]
                     if j<len(episode)-1:
                         next_s=episode[j+1][0]
                     else:
                         next_s=i[0]
                     counts_dict[s] += 1
                     if alpha==0:
                         alpha = 1/counts_dict[s]
                     V[s] = V[s] + alpha*(r+gamma*V[next_s]-V[s])
             return V
```

```
In [22]: def env(size):
             V={}
             counts_dict={}
             for i in range(size):
                 V[i]=0
                 counts_dict[i]=0
             return V, counts_dict
```

```
In [29]: #TD_path contains the data of episodes. Each episode there are (state, r) pairs.
         TD_path=[
             [[0,5],[1,3],[3,2],[0,2],[5,2]],
             [[0,3],[2,3],[4,3]],
             [[1,6],[2,4],[4,3],[1,2],[5,2]],
             [[1,6],[3,4],[4,3],[3,2],[0,2]],
             [[1,6],[4,4],[4,3],[2,2],[1,2]],
             [[1,6],[3,4],[4,3],[3,2],[0,2]],
         ]
```

```
In [30]: TD(6,TD_path,1)

Out[30]: {0: 7.0, 1: 11.0, 2: 27.0, 3: 7.0, 4: 17.0, 5: 4.0}
```

```
In [32]: TD(6,TD_path,1,0.8)

Out[32]: {0: 6.08,
          1: 13.432576000000001,
          2: 16.391424,
          3: 7.6874752,
          4: 14.4373248,
          5: 3.2}
```

4. Prove that fixed learning rate (step size alpha) for MC is equivalent to an exponentially decaying average of episode returns

Incremental Monte-Carlo Updates. In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$
\begin{gathered}
V(S_t) = V(S_t) + \alpha(G_t - V(S_t)) \\
V(S_{t0}) = 0 \\
V(S_{t1}) = 0 + \alpha(G_{t1} - 0) = \alpha G_{t1} \\
V(S_{t2}) = \alpha G_{t1} + \alpha(G_{t2} - \alpha G_{t1}) = \alpha * (1 - \alpha) G_{t1} + \alpha G_{t2} \\
\dots \\
V(S_t) = \alpha * (1 - \alpha)^{t-1} G_{t1} + \alpha * (1 - \alpha)^{t-2} G_{t2} + \dots + \alpha G_t
\end{gathered}
$$

Therefore, fixed learning rate (step size alpha) for MC is equivalent to an exponentially decaying average of episode returns.