In [2]:

```
# Q-Table Learning
import gym
import numpy as np
```

In [3]:

```
# Load the environment
env = gym.make('FrozenLake-v0')
```

```
[2017-08-03 10:10:54,396] Making new env: FrozenLake-v0
```

In [4]:

```
# Implement Q- Learning Algorithm

#Initialize table with all zeros
Q = np.zeros([env.observation_space.n,env.action_space.n])
# Set learning parameters
lr = .8
y = .95
num_episodes = 2000
#create lists to contain total rewards and steps per episode
#jList = []
rList = []
for i in range(num_episodes):
    #Reset environment and get first new observation
    s = env.reset()
    rAll = 0
    d = False
    j = 0
    #The Q-Table learning algorithm , heere we are taking till 99 just to avoi
d exploytation
    while j < 99:
        j+=1
        #Choose an action by greedily (with noise) picking from Q table
        a = np.argmax(Q[s,:] + np.random.randn(1,env.action_space.n)*
(1./(i+1)))
        # The above function is taking maximum value of a state + randome numb
er between 1, nof of possible action * 1/(i+1)
        # 1/(i+1) more and more episodes so the randome noise should reduce as
 the number of apisodes will increase for model stability

    #Get new state and reward from environment
        s1,r,d,_ = env.step(a)
        #Update Q-Table with new knowledge, this is using temporal differencin
g factor
        # Q value of current state = current state value + learning rate* (rew
ard + discounting factor
        # * (MAX of Qvalue of next state with all possible actions-current sta
te))
        Q[s,a] = Q[s,a] + lr*(r + y*np.max(Q[s1,:]) - Q[s,a])
        rAll += r
        s = s1
        if d == True:
            break
    #jList.append(j)
    rList.append(rAll)
```

```
In [9]: print ("Score over time: " +  str(sum(rList)/num_episodes))
```

```
Score over time: 0.532
```

```
In [8]: print ("Final Q-Table Values")
        print (Q)
```

```
Final Q-Table Values
[[  6.96832246e-02   6.94033722e-03   5.69810322e-03   6.85127481e-03]
 [  5.70115299e-05   3.70157845e-03   1.29387070e-03   1.25213888e-01]
 [  1.56434348e-03   1.01918817e-03   2.96259116e-03   2.08913881e-01]
 [  3.18405722e-04   9.08422144e-05   3.05669493e-04   5.28372703e-02]
 [  1.24557569e-01   4.69284591e-03   4.86855708e-03   3.11977451e-03]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  6.19331694e-02   1.73674708e-05   4.61703563e-06   7.22170571e-04]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  9.61372060e-04   1.02513116e-04   1.53308435e-05   2.97164711e-01]
 [  0.00000000e+00   6.46474954e-01   3.09616849e-03   0.00000000e+00]
 [  7.84311051e-01   0.00000000e+00   4.40992847e-04   6.23307087e-04]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   5.13606974e-01   6.25589169e-03]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   9.69087318e-01]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]]
```