Lab 3: Frozen Lake Value Iteration

In [1]:
```
import gym
env=gym.make("FrozenLake-v1",render_mode="human")
```

In [2]:
```
print(env.observation_space)
```

```
Discrete(16)
```

In [3]:
```
print(env.action_space)
```

```
Discrete(4)
```

In [4]:
```
env.P[0][2]
```

Out[4]:
```
[(0.3333333333333333, 4, 0.0, False),
 (0.3333333333333333, 1, 0.0, False),
 (0.3333333333333333, 0, 0.0, False)]
```

In [ ]:
```
import time
state=env.reset()
print("Time stamp-0")
t=0
env.render()
n=10
for i in range(n):
    action=env.action_space.sample()
    newstate,reward,done,info,x=env.step(action)
    print("Time stamp {}".format(t+1))
    t=t+1
    env.render()
    time.sleep(2)
    if done:
        break
```

In [1]:
```
import gym
import time
import numpy as np
env=gym.make("FrozenLake-v1",render_mode="human",is_slippery=False)
#env.reset()
#env.render()
```

In [2]:
```
def value_iteration(env):
    n=100
    threshold=1e-100
    gamma=0.9
    value_table=np.zeros(env.observation_space.n)
    for i in range(n):
        updated=np.copy(value_table)
        for j in range(env.observation_space.n):
            q_values=[sum([prob*(r+gamma*updated[s]) for prob,s,r,x in env.P[j][a]])
            value_table[j]=max(q_values)

        if (np.sum(np.fabs(updated-value_table)) <= threshold):
            break
    return value_table
```

In [3]:
```
def extract_policy(value_table):
    gamma=0.9
    policy=np.zeros(env.observation_space.n)
    for j in range(env.observation_space.n):
        q_values=[sum([prob*(r+gamma*value_table[s]) for prob,s,r,x in env.P[j][a]])
        policy[j]=np.argmax(np.array(q_values))
    return policy
```

In [4]:
```
optimal_function=value_iteration(env)
optimal_policy=extract_policy(optimal_function)
print(optimal_policy)
```

```
[1. 2. 1. 0. 1. 0. 1. 0. 2. 1. 1. 0. 0. 2. 2. 0.]
```

In [5]:
```
print(type(optimal_policy))
```

```
<class 'numpy.ndarray'>
```

In [6]:
```
print(type(optimal_policy[0]))
```

```
<class 'numpy.float64'>
```

In [7]:
```
print(optimal_function)
```

```
[0.59049 0.6561  0.729   0.6561  0.6561  0.      0.81    0.      0.729
 0.81    0.9     0.      0.      0.9     1.      0.      ]
```

In [8]:
```
print(optimal_function)
```

```
[0.59049 0.6561  0.729   0.6561  0.6561  0.      0.81    0.      0.729
 0.81    0.9     0.      0.      0.9     1.      0.      ]
```

In [9]:
```
policy=optimal_policy.reshape(4,4)
```

In [10]:
```
def show():
    state=env.reset()[0]
    while(True):
        y=(state+1)%4-1
        x=(state)//4
        print(x,y)
        action=int(policy[x][y])
        state,reward,done,info,x=env.step(action)
        #print("Time stamp {}".format(i))
        print(state)
        env.render()
        time.sleep(0.5)
        if done:
            env.reset()
            show()
```

In [ ]:
```
show()
```

```
0 0
4
1 0
8
2 0
9
2 1
13
3 1
14
3 2
15
0 0
4
1 0
8
2 0
9
2 1
```

In [3]:
```
env.P
```

Out[3]:
```
{0: {0: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False)],
  1: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 1, 0.0, False)],
  2: [(0.3333333333333333, 4, 0.0, False),
   (0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False)],
  3: [(0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 0, 0.0, False)]},
 1: {0: [(0.3333333333333333, 1, 0.0, False),
   (0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 5, 0.0, True)],
  1: [(0.3333333333333333, 0, 0.0, False),
   (0.3333333333333333, 5, 0.0, True),
   (0.3333333333333333, 2, 0.0, False)],
  2: [(0.3333333333333333, 5, 0.0, True),
```