In [27]:

```
import numpy as np
from frozen_lake import FrozenLakeEnv
import random

env = FrozenLakeEnv()



def epsilon_greedy_action(env,Q,state,epsilon=0.3):
    n = random.uniform(0,1)
    if n<= epsilon:
        return np.random.randint(env.action_space.n)
    else:
        return np.argmax(Q[state])



def Q_Learning(env, episodes=1000, gamma=0.91, alpha=0.1):
    Q = np.zeros([env.nS,env.nA])

    for i in range(episodes):
        finished = False

        env.reset()


        S = env.s

        while not finished:

            A = epsilon_greedy_action(env,Q,S)

            next_S, R, finished, _ = env.step(A)

            #A = epsilon_greedy_action(env,Q,next_S)
```

```
            #Q[S][A] = Q[S][A] + alpha*(R + gamma * Q[next_S][next_A] -
 Q[S][A])
            Q[S][A] = Q[S][A] + alpha * ( R + gamma* np.max(Q[next_S][A
])  - Q[S][A]  )

            S = next_S


    return Q
```

In [28]:
```
  #zaokrąglamy do dwóch miejsc po przecinku
Q = Q_Learning(env,10000)
print(Q)

env.close()
```

```
[[0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.90999769 0.        ]
 [0.         0.         0.99999956 0.        ]
 [0.         0.         0.         0.        ]]
```

In [ ]:
```

```