# **REINFORCEMENT LEARNING**

**Flowchart**

**Pseudocode**

For each s, a, initialize table entry Q(s,a) <- 0

Observe current state s

Do forever:

Select an action a and execute it

Receive immediate reward r

Observe the new state s'

Update the table entry for Q(s, a) a

This procedural approach can be translated into steps as follows:

1. Initialize the Q-values table, **Q(s, a)**.
2. Observe the current state, **s**.
3. Choose an action, **a**, for that state
4. Take the action, and observe the reward, **r**, as well as the new state, **s'**.
5. Update the Q-value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

**ALGORITHM AND IMPLEMENTATION**

**Initialize the Q-values table, Q(s, a).**

self.Q = [ [ initQ for a in range( noActions[s] ) ] for s in range( noStates ) ]

**Choose an action, a, for that state**

def explore( self ):

self.getAction = lambda st : self.exp.getAction( st, self.Q[st], self.maxQ[st], self.maxA[st], self.maxN[st] )

def exploit( self ):

self.getAction = lambda st : choice( self.maxA[st], self.maxN[st] )

**Update the Q-value for the state using the observed reward and the maximum reward**

if update:

V = self.maxQ[st\_]

self.Q[st][at] += alpha\*( rt + gamma\*V - self.Q[st][at] )

if updatenext == "always":

self.maxQ[st] = max( self.Q[st] )

self.V[st] = self.maxQ[st]

self.maxA[st] = [ a for a in range( self.A[st] ) if self.Q[st][a] == self.maxQ[st] ]

self.maxN[st] = len( self.maxA[st] )

if updatenext == "now":

for s in range( self.S ):

self.maxQ[s] = max( self.Q[s] )

self.V[s] = self.maxQ[s]

self.maxA[s] = [ a for a in range( self.A[s] ) if self.Q[s][a] == self.maxQ[s] ]

self.maxN[s] = len( self.maxA[s] )