note set Q

Conceptual View of Q-Learning

- Q-Learning is a form of Reinforcement Learning.

- It uses a random exploration paired with delayed reward to generate values for a "Q table".

- For some examples - the Q table is used to select the best action to move to a goal state given the agent is in a particular state

$$S(i) + a(j) \longrightarrow S(k)$$

$$\text{state}(i) + \text{action}(j) \xrightarrow[\text{results in}]{} \text{State}(k)$$

state (i): usualy- in Q learning there are a finite number of states an agent can be in

action (j): there are some # of actions available to the agent. Basic systems would have all actions for each state

State(k): new state as a result of action (j)

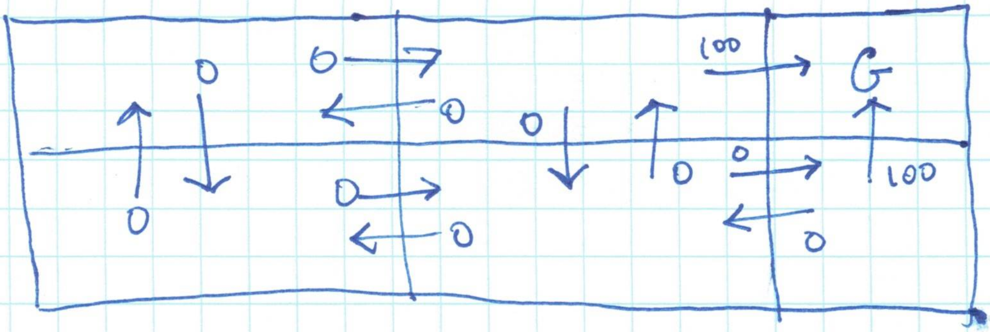


table shows immediate reward

- Only moving to the goal state provides a reward - that is an imediate reward.
- Every where else no immediate reward is given.

> A key part of Q learning is exploration of the state space.

- Agent is placed at a random state, and a random action is taken

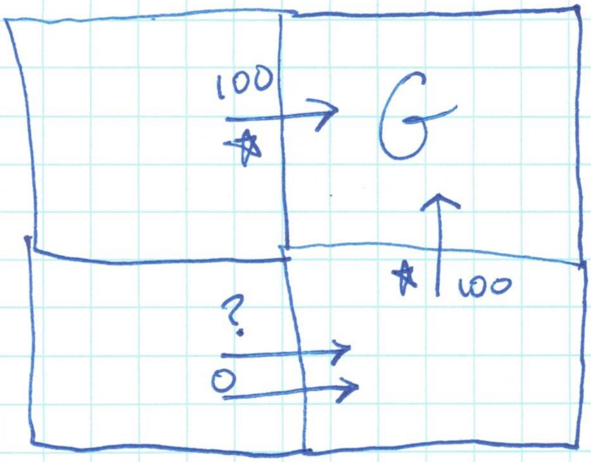
> it is the exploration that fills out the state table.

filling out the table relies on the concept of delayed reward

delay factor

delayed reward = imediate reward at state + η * reward of immediate successor state

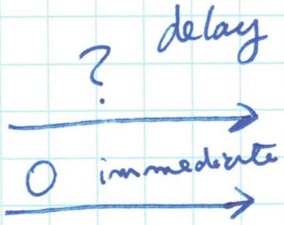
current



G - goal state

☆ - immediate reward for moving to the goal state

? - here we need to assign reward based on delay



let $\eta = .9$

delayed reward = immediate reward + η * reward of immediate successor state

$$90 = 0 + .9 * 100$$

most reward available from this state (you have to pick the correct action to get it)

③

Basic Q Algorithm

```
for (j=0 ; j< nEpochs; j++) {
    myState = select Random State

    while (myState != goal State) {
        myAction = select Random Action

        next State =
            qMatrix ~~adj~~ [myState][myAction]
        immediate reward = ~~State~~ immediate [nextState].immediate_reward

        delayed reward = immediate reward +
                $\gamma$ * max (all rewards from successor states)

qTable →
        stateTable[myState][myAction]
            ~~delayedReward~~ = delayedReward

        myState = nexState
    }
}
```

(4)

3	4	5
0	1	2

qImmediate[nStates] =

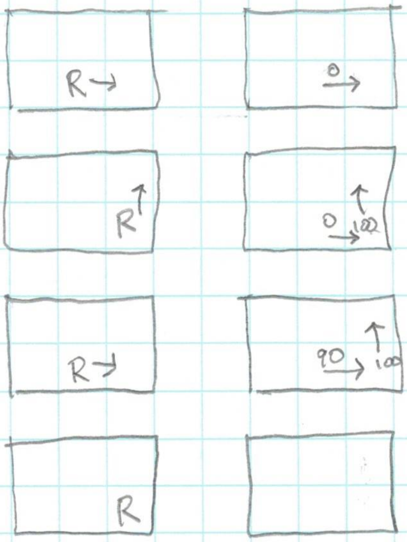
[0, 0, 0, 0, 0, 0]

```
                     left, right, up, down
        qAdj = [ [-1, 1, 3, -1],              qReward = [ [-1, 0, 0, -1],
                 [0, 2, 4, -1],                             [0, 0, 0, -1],
 qAdj[nStates]                                qReward
     [nActions]  [1, -1, 5, -1],              [nStates]
                                              [nActions]    [0, -1, 0, -1],
                 [-1, 4, -1, 0],                            [-1, 0, -1, 0],
 nStates = 6     [3, 5, -1, 1]                              [0, 0, -1, 0]
 nActions = 4    [4, -1, -1, 2]]                            [0, -1, -1, 0]]

 goal = 5                         lastState = nStates - 1     epochs = 0
 qImmediate[goal] = 100           lastAction = nActions - 1
 exploring = true                 η = .9
 while(exploring) {
     myState = random(0, lastState)
     while (myState != goal) {
        /* Get a valid random action. */
        myAction = random(0, lastAction)
        while (qAdj[myState][myAction] != -1) {
              myAction = random(0, lastAction)
        }
        nextState = qAdj[myState][myAction]
        immediadeReward = qImmediate[nextState]
        delayedReward = max(qReward[nextState]) * γ
        totalReward = immediateReward + delayedReward
        qReword[myState][myAction] = totalReward
        myState = nextState
    }

    epochs ++

    if (epochs == maxEpochs) {
        exploring = false
    }
}
```



Basic Q Algorithm

qImmediate is array that holds immeddiate reward given when agent moves to that state from an adjacent state

qMatrix is an array that holds the adjacent states given an action

qTable holds the cummulitive rewards of recieved when moving from one state to another

```
for (j=0; j< nEpochs; j++) {
        myState = selectRandomState();

        While (myState != goalState) {
              myAction = selectRandomAction();
              nextState = qMatrix [myState] [myAction]
              immediateReward = qImmediate [next
                                            state]

              delayedReward = max (qTable
                                   [nextState]
                              * decay
        totalReward = immediateReward +
                      delayedReward

        qTable[myState][myAction] =
                           totalReward

        myState = next state
    } /* end while */

} /* end for */
```