When no knowledge of the world, what do you do

Model free method:

You venture the world, you live to learn, you crash and burn in order to learn to live.

You experience and for every step you make you improve =

Temporal Difference Learning

Model free approach: Q-learning

- When we address Q-learning we have a model-free method.
- If that is the case we can still train a system, but then we score an action rather than a state.
    - In fact, using the analogy of A* again we can imagine the *h(s)* of $V(s) = g(s) + h(s)$ is incrementally learned rather than "guessed".
    - Q-learning is also a TD method.
- The basic steps of the elementary Q-learning method are:
- An agent is in state s
- For all states s ϵ S:
    - For all actions a_i ϵ A_s:
        - We sample an action a_i
        - We observe the reward and the next state for that action (a_i)
        - We take the action with the highest Q and move to the next state, s'

Q-learning

- learns Q-values – no explicit model for state transitions
- Standard Q-learning algo:

- $Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \Upsilon \max' Q(s',a') - Q(s,a))$

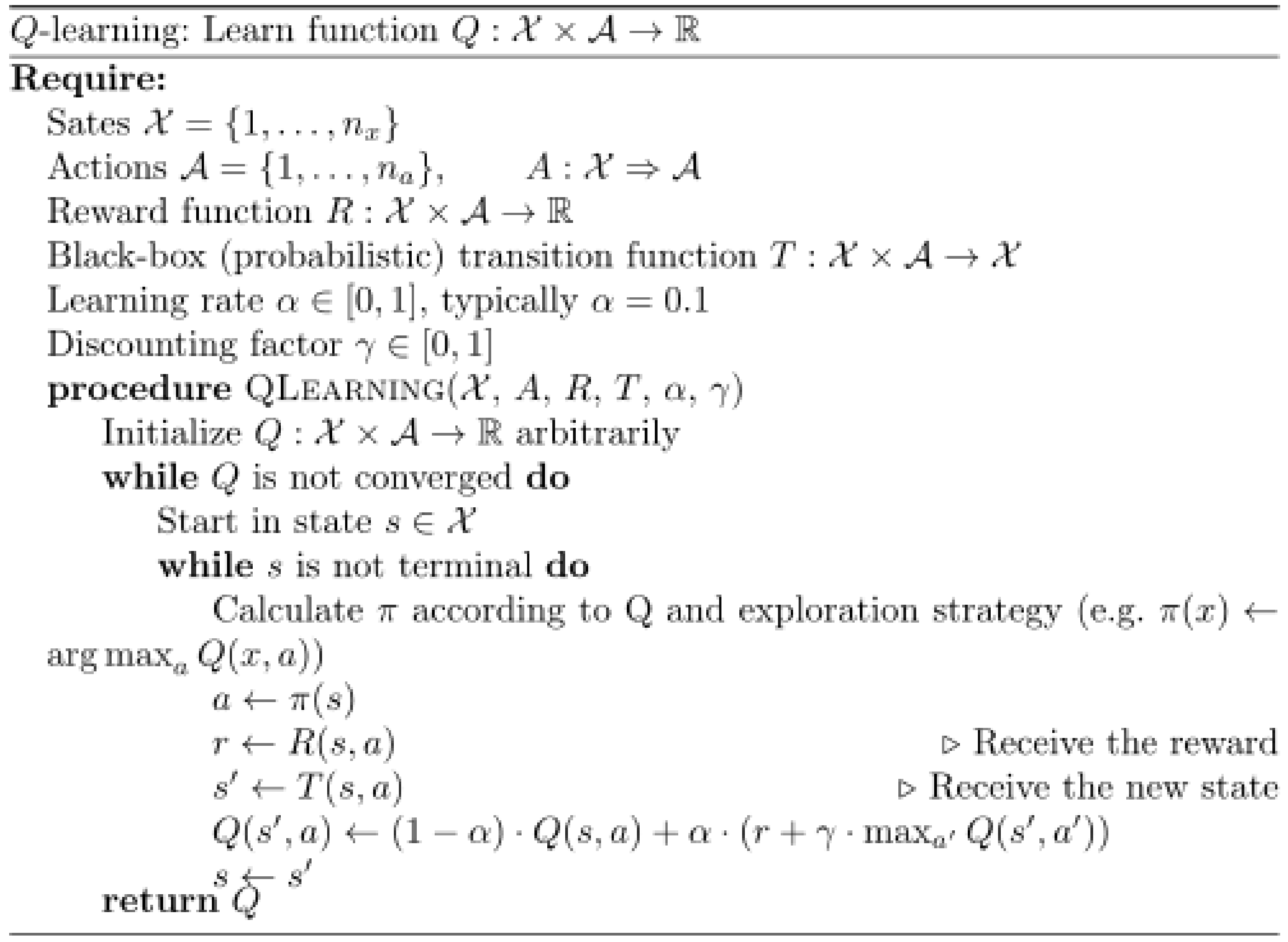
  - This incremental update happens every time an action a is executed in state s leading to the new state s’.

A not-so formal algo description

The Q-Learning algorithm goes as follows:

- 1. Set the gamma parameter, and environment rewards in matrix R.
- 2. Initialize matrix Q to zero.
- 3. For each episode:
    - Select a random initial state (!!!!!!!!!!)
    - Do While the goal state hasn't been reached.
        - Select one among all possible actions for the current state.
        - Using this possible action, consider going to the next state.
        - Get maximum Q value for this next state based on all possible actions.
        - Compute:
            - $Q(\text{state, action}) = \alpha\ (R(\text{state, action}) + \Upsilon^{*}\ \text{Max}[Q(\text{next state, all actions})) - Q(s.a)]$
        - Set the next state as the current state.
    - End Do
- End For

The formal Q-method



```
$Q$-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Require:
  Sates $\mathcal{X} = \{1, \ldots, n_x\}$
  Actions $\mathcal{A} = \{1, \ldots, n_a\}$,    $A : \mathcal{X} \Rightarrow \mathcal{A}$
  Reward function $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
  Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
  Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
  Discounting factor $\gamma \in [0, 1]$
  procedure QLEARNING($\mathcal{X}$, $A$, $R$, $T$, $\alpha$, $\gamma$)
      Initialize $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
      while $Q$ is not converged do
          Start in state $s \in \mathcal{X}$
          while $s$ is not terminal do
              Calculate $\pi$ according to Q and exploration strategy (e.g. $\pi(x) \leftarrow$
  $\arg\max_a Q(x, a)$)
              $a \leftarrow \pi(s)$
              $r \leftarrow R(s, a)$                                  ▷ Receive the reward
              $s' \leftarrow T(s, a)$                                 ▷ Receive the new state
              $Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
              $s \leftarrow s'$
      return $Q$
```

Q-learning-agent (1)

- Function Q-learning   Return *action a*
    - Input: A *percept* indicating the current state ‘s and *reward signal r’* = R(s)
    - Persistent values:
        - *Q*, a table of action values indexed by state and action, initialized to zero
        - *Ns,*a, a table of frequencies for state-action pairs, initialized to zero (if all moves are equal can be omitted)
        - *s,a,r* the previous state, action and reward, initially null

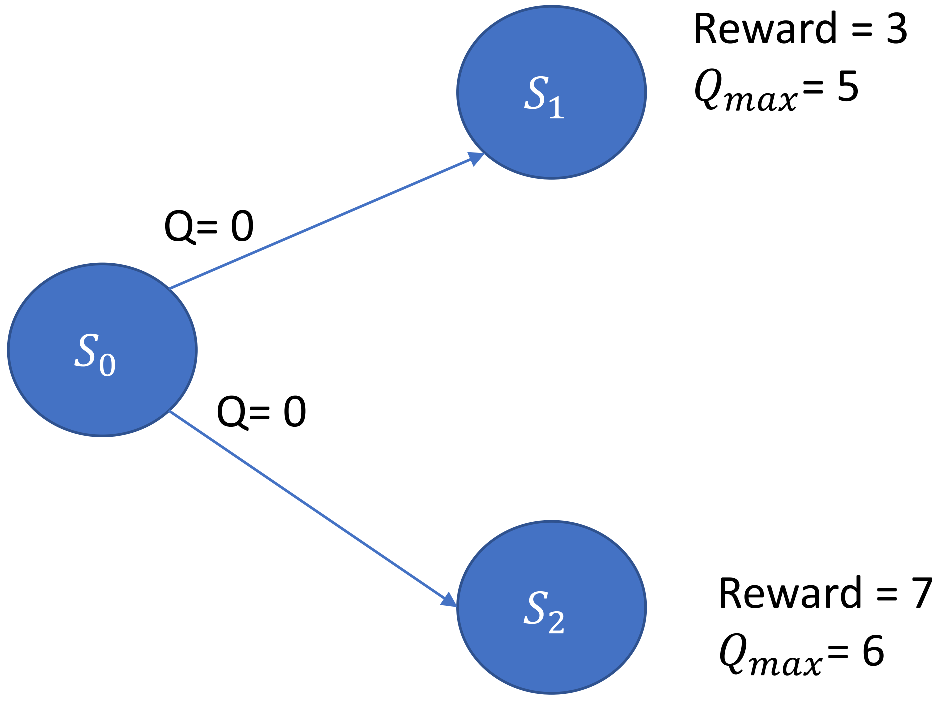
• (continued………………. Next)

Q-learning-agent (2)

- IF state s is terminal? Then Q [s, None] <-- r’
- IF s is *not null* THEN
  - Increment Ns,a
  - $Q(s,a) \leftarrow Q(s,a) + \alpha(Ns,a[s,a](r + \Upsilon\ maxa'Q[s',a'] - Q[s,a])$
  - $s,a,r \leftarrow s'$, argmaxa’ , f(Q[s’,a’], Ns,a[s,a], r’
- Return a

Basic principles

Q(s,a) ← Q(s,a) + α(R(s) + ϒ max'Q(s',a') – Q(s,a))



A given decision situation at T = t_1

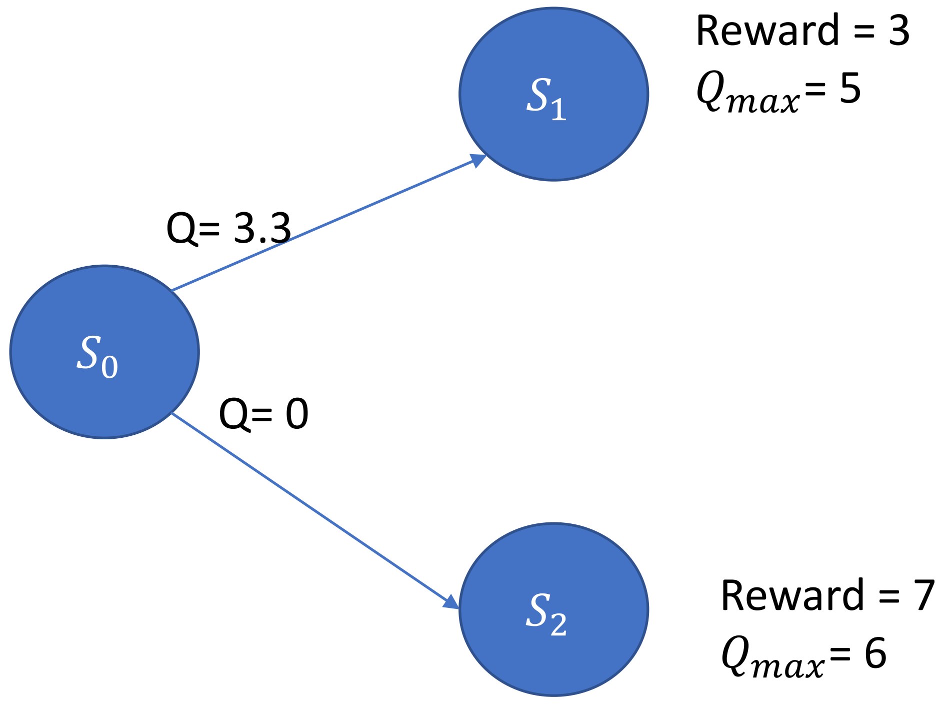
An agent can move to two other states from S_0 . Which one?

The roulette makes its choice according to the probability:

P($S_j, a_{j,i}$) = $\frac{a_{j,i}}{\sum_{i=1}^{A} a_{j,i}}$

Learning rate α = 0,6
Discount factor ϒ = 0,5

Basic principles



An agent can move to two other states from S_0 . Which one?

The roulette makes its choice according to the probability:

P($S_j, a_{j,i}$) = $\frac{a_{j,i}}{\sum_{i=1}^{A} a_{j,i}}$

Learning rate α = 0,6
Discount factor ϒ = 0,5

$Q'(S_0 a_{0,1})$ $= Q(S_0 a_{0,1})$ + α(R(S_1) + ϒ* $Q_{max,S1}$-Q($S_0 a_{0,1}$)
$Q'(S_0 a_{0,1})$ = 0 + 0,6(3+ (0,5*5)-0 = 3.3

Q-value propagation

Recall:

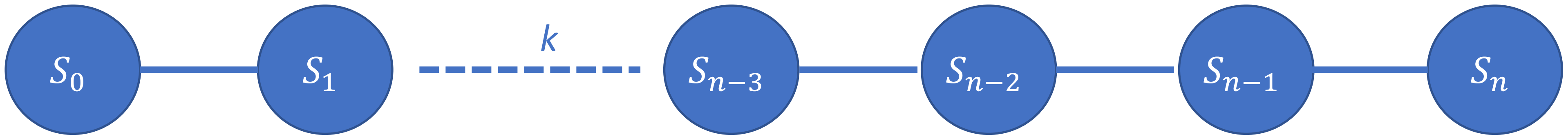
$$\boldsymbol{G_t} = \boldsymbol{R_{k+1}} + \boldsymbol{\gamma R_{k+2}} + \boldsymbol{\gamma^2 R_{k+3}} + \cdots = \sum_{k=0} \Upsilon^k \boldsymbol{R_{k+1}} \qquad \Upsilon \ \boldsymbol{\epsilon} \ [\boldsymbol{0}, \boldsymbol{1}]$$

Works for Q-learning too:

$Q'(S_{n-3}a_{n-3,i}) = 5{,}4 * 0{,}5^k * 0{,}6$

$Q'(S_{n-2}a_{n-2,i}) = 60 * 0{,}5 * 0{,}6 = 18$

Reward = 100



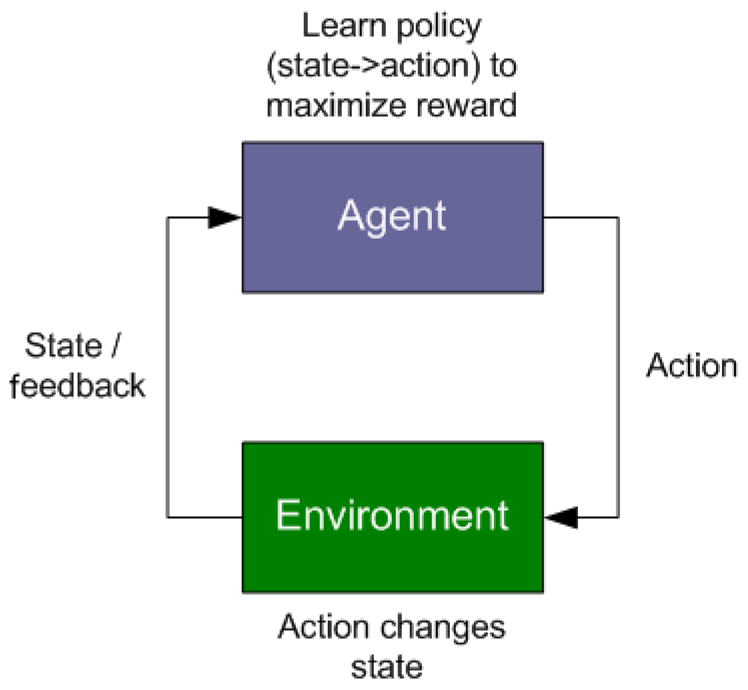
$Q'(S_{n-3}a_{n-3,i}) = 5{,}4 * 0{,}5^{k+1} * 0{,}6$

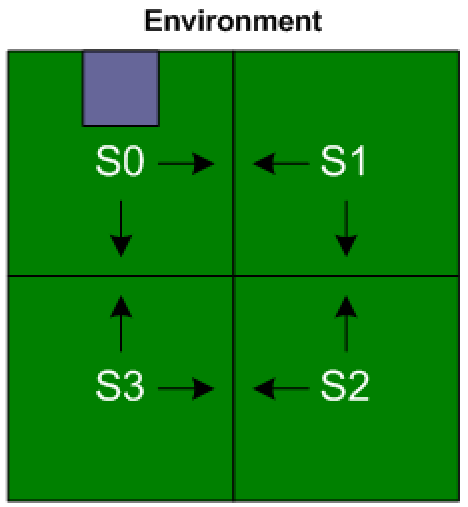
$Q'(S_{n-3}a_{n-3,i}) = 18 * 0{,}5 * 0{,}6 = 5{,}4$

$Q'(S_{n-1}a_{n-1,i}) = 100 * 0{,}6 = 60$

Learning rate α = 0,6
Discount factor ϒ = 0,5

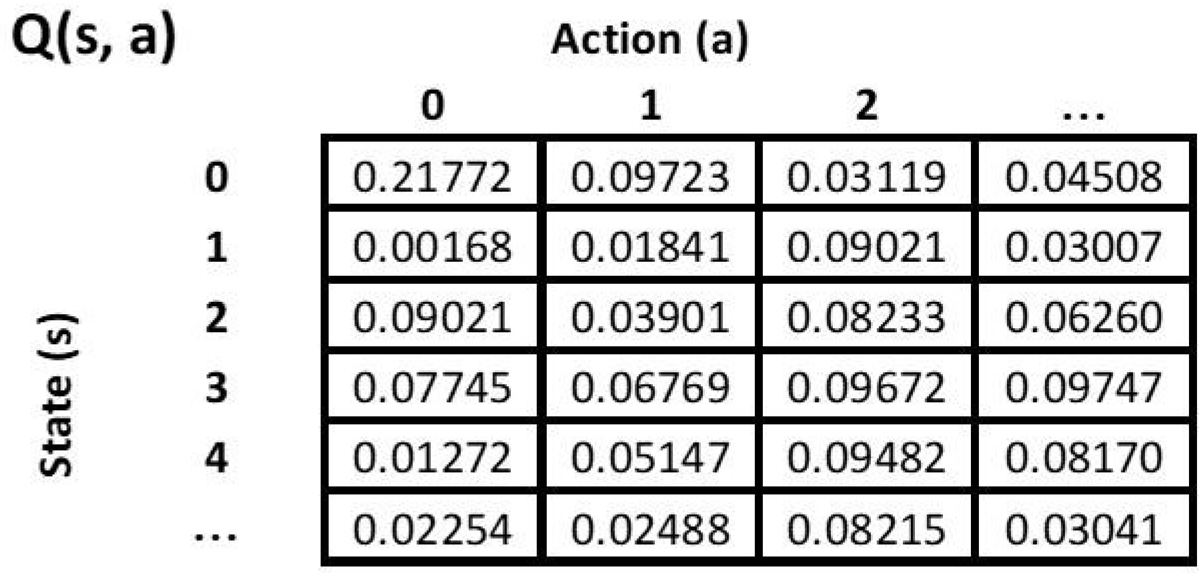
A tabular approach





State	Action	Q-Value
S_0	East	Q_{S1}
S_0	South	Q_{S3}
S_1	West	Q_{S0}
S_1	South	Q_{S2}
S_2	West	Q_{S3}
S_2	North	Q_{S1}
S_3	East	Q_{S2}
S_3	North	Q_{S0}

Basic Q-learning requires a table to maintain Q-values (and Rewards)



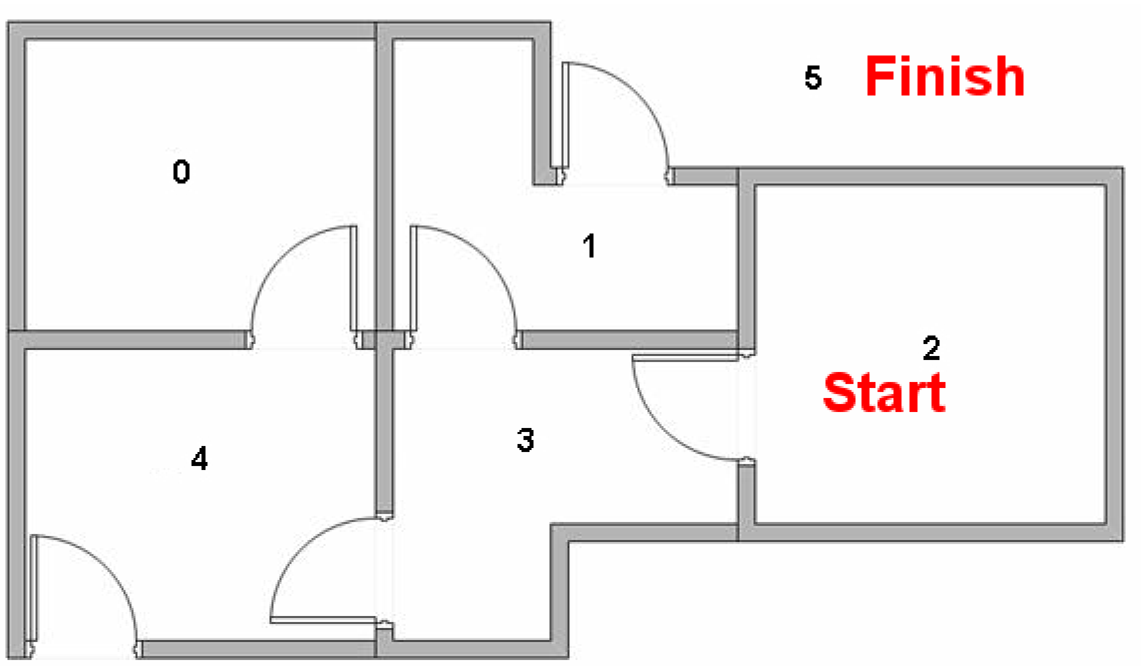
Q(s, a)		Action (a)			
		0	1	2	...
State (s)	0	0.21772	0.09723	0.03119	0.04508
	1	0.00168	0.01841	0.09021	0.03007
	2	0.09021	0.03901	0.08233	0.06260
	3	0.07745	0.06769	0.09672	0.09747
	4	0.01272	0.05147	0.09482	0.08170
	...	0.02254	0.02488	0.08215	0.03041

Works only with discrete states
Requires computational power and memory for larger tasks

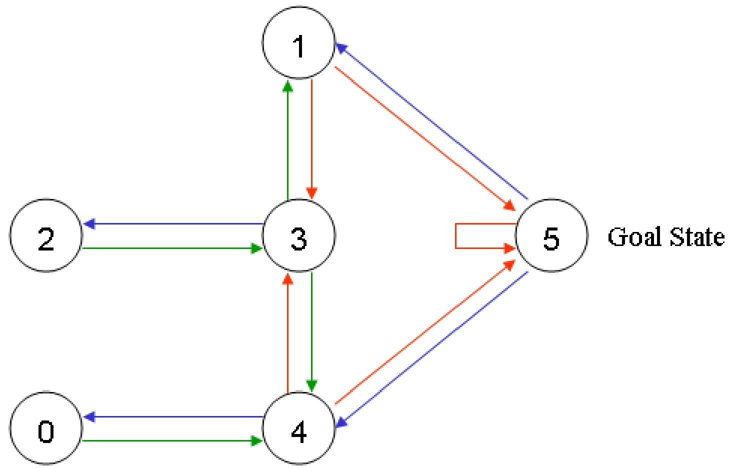
Example

Suppose we have 5 rooms in a building connected by doors as shown in the figure next. We'll number each room 0 through 4. The outside of the building can be thought of as one big room (5). Notice that doors 1 and 4 lead into the building from room 5 (outside).

(sourced from http://mnemstudio.org/)

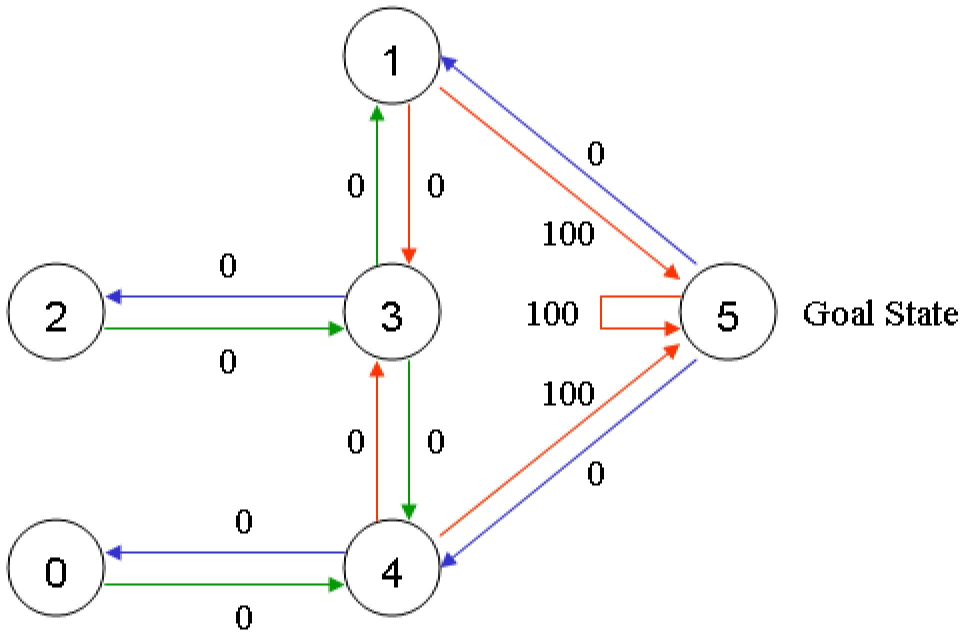

5
Finish
0
1
2
Start
3
4

This can be modelled in the usual way – as a graph



For this example, we'd like to put an agent in any room, and from that room, go outside the building (this will be our target room). In other words, the goal room is number 5. To set this room as a goal, we'll associate a reward value to each door (i.e. link between nodes). The doors that lead immediately to the goal have an instant reward of 100. Other doors not directly connected to the target room have zero reward. Because doors are two-way (0 leads to 4, and 4 leads back to 0), two arrows are assigned to each room. Each arrow contains an instant reward value, as shown below:

Reward assigned to goal state



Room 5 loops back to itself with a reward of 100, and all other direct connections to the goal room carry a reward of 100.

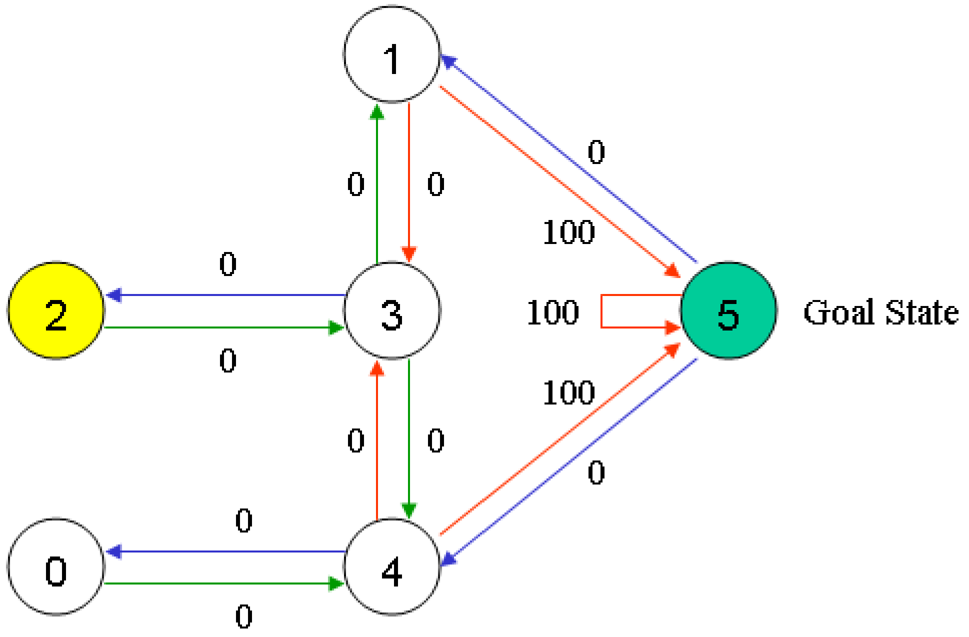
Pursue the highest reward

In Q-learning, the goal is to reach the state with the highest reward, so that if the agent arrives at the goal, it will remain there forever. This type of goal is called an "absorbing goal".

Imagine our agent as a dumb virtual robot that can learn through experience. The agent can pass from one room to another but has no knowledge of the environment, and doesn't know which sequence of doors lead to the outside.

Suppose we want to model some kind of simple evacuation of an agent from any room in the building. Now suppose we have an agent in Room 2 and we want the agent to learn to reach outside the house (5).

Room 2 is start state
Room 5 (outdoor) is goal state



Suppose the agent is in state 2. From state 2, it can go to state 3 because state 2 is connected to 3. From state 2, however, the agent cannot directly go to state 1 because there is no direct door connecting room 1 and 2 (thus, no arrows). From state 3, it can go either to state 1 or 4 or back to 2 (look at all the arrows about state 3). If the agent is in state 4, then the three possible actions are to go to state 0, 5 or 3. If the agent is in state 1, it can go either to state 5 or 3. From state 0, it can only go back to state 4.

The Q update

The agent starts out knowing nothing, the matrix Q is initialized to zero.

In this example, for the simplicity of explanation, we assume the number of states is known (to be six).

If we didn't know how many states were involved, the matrix Q could start out with only one element. It is a simple task to add more columns and rows in matrix Q if a new state is found.

Reward matrix R

$$\begin{array}{cc} & \text{Action} \\ \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \end{array}$$

$$R = \begin{array}{c|cccccc|} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Hitting the wall is penalized
So is cycling (beyond the goal state)

The Q update

The agent starts out knowing nothing, the matrix Q is initialized to zero.

In this example, for the simplicity of explanation, we assume the number of states is known (to be six).

If we didn't know how many states were involved, the matrix Q could start out with only one element. It is a simple task to add more columns and rows in matrix Q if a new state is found.

The Q matrix is updated incrementally

- Now we'll add a similar matrix, "Q", to the brain of our agent, representing the memory of what the agent has learned through experience.

- The rows of matrix Q represent the current state of the agent, and the columns represent the possible actions leading to the next state (the links between the nodes)

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The transition rule

The transition rule of Q learning is a very simple formula:
Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

According to this formula, a value assigned to a specific element of matrix Q, is equal to the sum of the corresponding value in matrix R and the learning parameter Gamma, multiplied by the maximum value of Q for all possible actions in the next state. (assuming the learning coefficient $\alpha = 1$)

Updating the Q matrix

Initially

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$R = \begin{array}{c|cccccc} & & & \text{Action} & & & \\ \text{State} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Assume agent is in state 5. Look at the sixth row of the reward matrix R (i.e. state 5). It has 3 possible actions: go to state 1, 4 or 5.

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

$$Q(1, 5) = R(1, 5) + 0.8 * \text{Max}[\underbrace{Q(5, 1), Q(5, 4), Q(5, 5)]}_{\text{The road onwards is analyzed, values gathered from the present Q matrix}} = 100 + 0.8 * 0 = 100$$

The memory of the reward is injected into the brain of the agent = The updated Q-matrix

New value= 100 inserted here

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

For the next episode, we start with a randomly chosen initial state. This time, we have state 3 as our initial state.

Look at the fourth row of matrix R; it has 3 possible actions: go to state 1, 2 or 4. By random selection, we select to go to state 1 as our action.

Now we imagine that we are in state 1. Look at the second row of reward matrix R (i.e. state 1). It has 2 possible actions: go to state 3 or state 5. Then, we compute the Q value:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
Q(3, 1) = R(3, 1) + 0.8 * Max[Q(1, 2), Q(1, 5)] = 0 + 0.8 * Max(0, 100) = 80

We use the updated matrix Q from the last episode. Q(1, 3) = 0 and Q(1, 5) = 100. The result of the computation is Q(3, 1) = 80 because the reward is zero. The matrix Q becomes:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

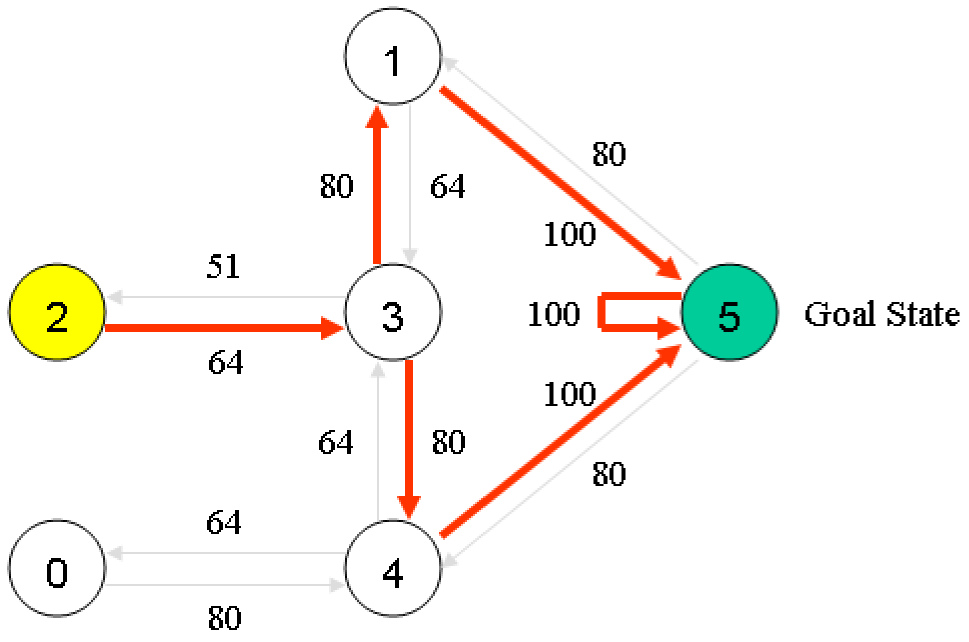
After several episodes

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{array}\right] \end{array}$$

After normalization

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{array}\right] \end{array}$$

After several episodes when the system converges and exploitation rather than exploration dominates (steady state)



For example, from initial State 2, the agent can use the matrix Q as a guide:
From State 2 the maximum Q values suggests the action to go to state 3.
From State 3 the maximum Q values suggest two alternatives: go to state 1 or 4. Suppose we arbitrarily choose to go to 1.
From State 1 the maximum Q values suggests the action to go to state 5.
Thus the sequence is 2 - 3 - 1 - 5.

Q-learning basics

- $Q_{k+1}(s,a) = E[R_{t+1} + \gamma max_{a'} Q_k(S_{t+1}, a') | S_t = s, A_t = a]$
  - This suggests that the we build state values incrementally.
  - For the action *a* in state *s* the value assigned at iteration *k+1* is the expected

    value of R_{t+1} at the state where a takes the agent plus the discounted value of the best action, *a'* afterwards. Q-learning propagates rewards rom states that lies ahead backwards along the string of selected actions.