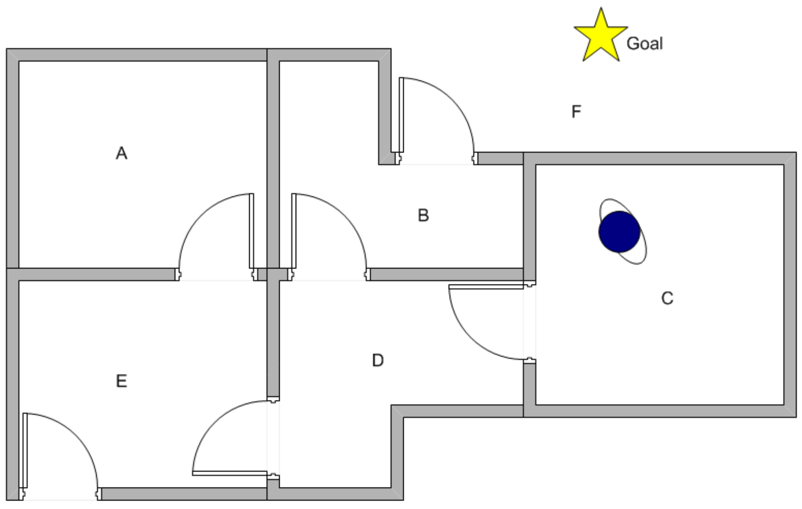
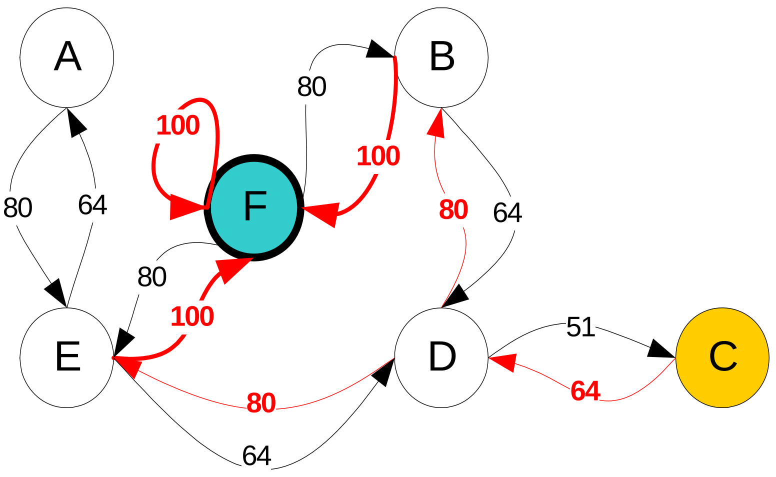
Kardi Teknomo

Q-LEARNING TUTORIAL





Revoledu.com

Q-Learning Tutorial by Kardi Teknomo
Copyright © 2005 - 2012 by Kardi Teknomo
Published by Revoledu.com
Online edition is available at Revoledu.com
Last Update: October 2012

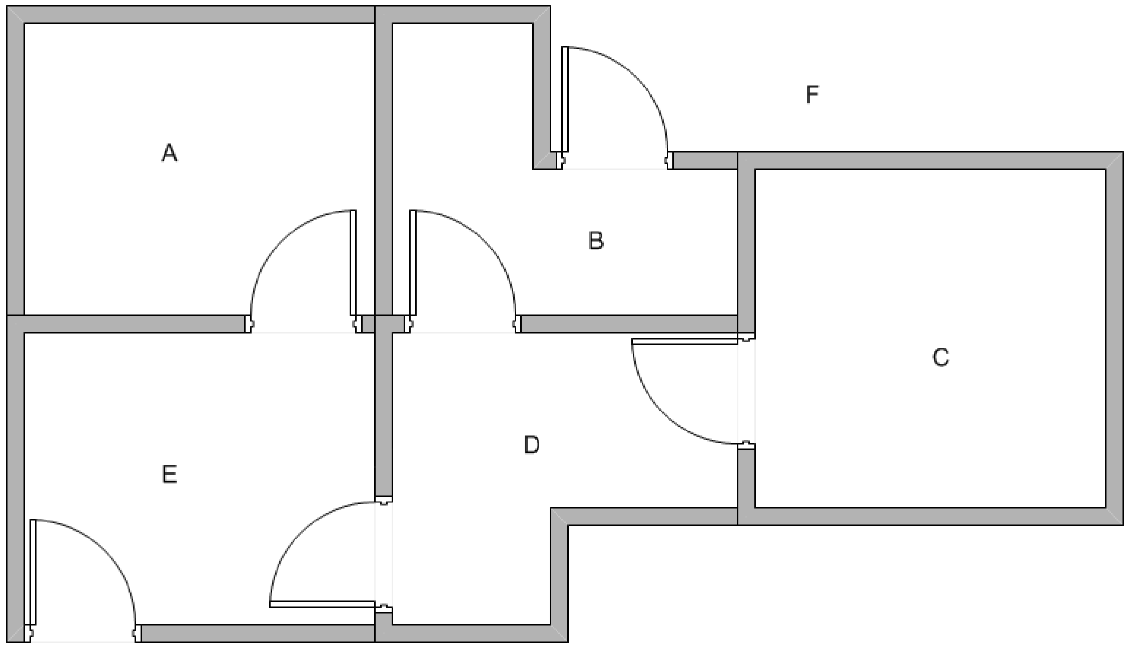
Table of Contents

In this tutorial, you will learn step by step how a single agent learns through training without teacher (unsupervised) in an unknown environment. You will find out part of reinforcement learning algorithm called Q-learning. Reinforcement learning algorithm has been widely used for many applications such as robotics, multi agent system, navigation, motion planning, game, etc.

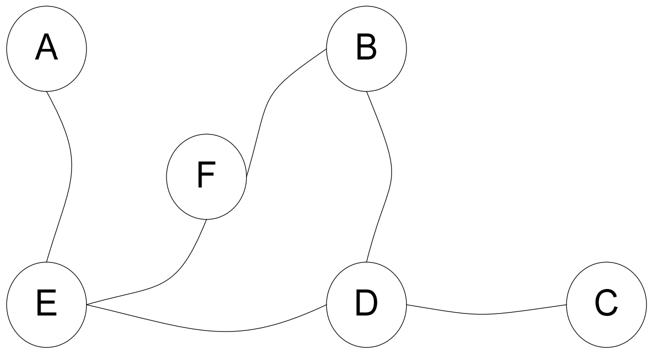
Instead of learning the theory of reinforcement that you can read it from many books and other web sites, this tutorial will introduce the concept through simple but comprehensive numerical examples. If you purchase this tutorial, you will also receive the companion worksheets files and Matlab code.

Modeling the environment

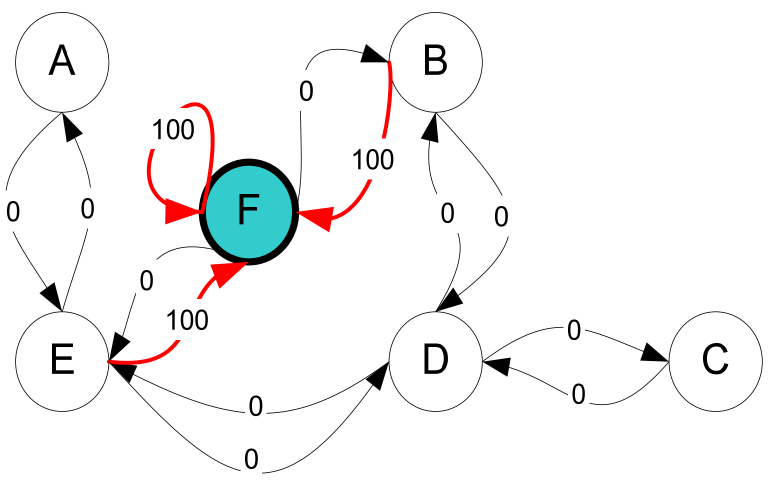
Suppose we have 5 rooms in a building connected by certain doors as shown in the figure below. We give name to each room A to E. We can consider outside of the building as one big room to cover the building, and name it as F. Notice that there are two doors lead to the building from F, that is through room B and room E.



We can simplify the representation of the rooms above by a graph. A graph is a collection of points and simple non-intersecting curves that have certain properties. The length or straightness of the curves (which we called it edge or link) does not matter. Similarly the size and shape of the point (that we called it node or vertex) does not matter. We represent each room as a node and each door as a link. Since we have 6 rooms, our graph below has 6 nodes. Notice that the link between nodes only exists if the actual rooms are connected by a door. The outside of the building is called room F. There are two doors to go out, which is through room B or room E.



We want to set a target room. If we put an agent in any room, we want the agent to go outside the building. In other words, the goal room is the node F. To set this kind of goal, we introduce give a kind of reward value to each door (i.e. edge of the graph). The doors that lead immediately to the goal have instant reward of 100 (see diagram below, they have red arrows). Other doors that do not have direct connection to the target room have zero reward. Because the door is two way (from A can go to E and from E can go back to A), we assign two arrows to each room of the previous graph. Each arrow contains an instant reward value. The graph becomes *state diagram* as shown below



Additional loop with highest reward (100) is given to the goal room (F back to F) so that if the agent arrives at the goal, it will remain there forever. This type of goal is called *absorbing* goal because when the agent reaches the goal state, it will stay in the goal state.

Let us summarized what we have done to model the environment:

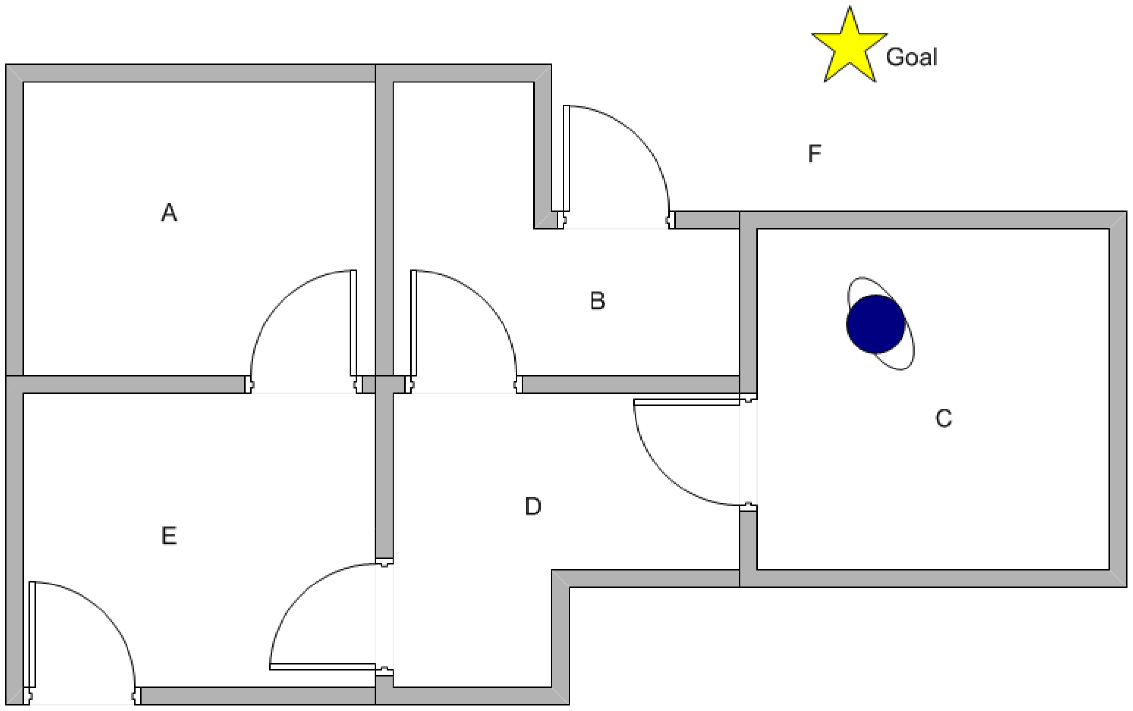
1. Represent the environment into a graph
2. Set the links immediate to the goal node with highest reward value (e.g. 100)
3. Set all other links with zero reward value
4. Add a self loop link to the goal with highest reward value.

Agent, State and Action Introduction

Ladies and gentlemen, now is the time to introduce our superstar agent….

Imagine our agent as a dumb virtual robot that can learn through experience. The agent can pass one room to another but has no knowledge of the environment. It does not know which sequence of doors the agent must pass to go outside the building.

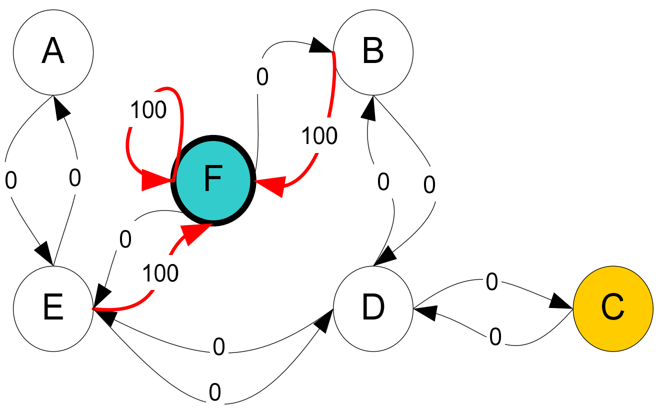
Suppose we want to model some kind of simple evacuation of an agent from any room in the building. Now suppose we have an agent in Room C and we want the agent to learn to reach outside the building (F) (See the diagram below).



How to make our agent to learn from experience?

Before we discuss about how the agent will learn (using Q learning) in the next section, let us discuss about some terminologies of *state* and *action*.

We call each room (including outside the building) as a *state*. Agent's movement from one room to another room is called *action*. Let us draw back our state diagram. State is depicted using node in the state diagram, while action is represented by the arrow. In short, we just rename each node as a state and each link as an action.



Suppose now the agent is in state C. From state C, the agent can go to state D because the state C is connected to D. From state C, however, the agent cannot directly go to state B because there is no direct door connecting room B and C (thus, no arrow). From state D, the agent can go either to state B or state E or back to state C (look at the arrow out of state D). If the agent is in state E, then three possible actions are to go to state A, or state F or state D. If agent is state B, it can go either to state F or state D. From state A, it can only go back to state E.

We can put the state diagram above and the instant reward values into the following reward table, or matrix **R**.

	Action to go to state					
Agent now in state	A	B	C	D	E	F
A	-	-	-	-	0	-
B	-	-	-	0	-	100
C	-	-	-	0	-	-
D	-	0	0	-	0	-
E	0	-	-	0	-	100
F	-	0	-	-	0	100

The minus sign in the table says that the row state has no action to go to column state. For example, State A cannot go to state B (because no door connecting room A and B, remember?)

In summary, a reward matrix **R** is simply a weighted adjacency matrix with weight zero if there is a connection between states and weight 100 for the links that immediately go to the goal state (including the self-loop).

Q Learning

In the last section of this tutorial, we have modeled the environment and the reward system for our agent. This section will describe learning algorithm called *Q learning* (which is a simplification of reinforcement learning).

At this point, we have already a model the environment in term of reward system represented as a matrix R.

$$\mathbf{R} = \begin{array}{cc} state \backslash action & \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} & \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{array}$$

Now we need to put similar matrix name **Q** in the brain of our agent that will represent the memory of what the agent have learned through many experiences. The row of matrix Q represents current state of the agent, the column of matrix Q pointing to the action to go to the next state.

In the beginning, we say that the agent know nothing, thus we put **Q** as zero matrix. In this example, for the simplicity of explanation, we assume the number of state is known (to be six). In more general case, you can start with zero matrix of single cell. It is a simple task to add more column and rows in **Q** matrix if a new state is found.

The transition rule of this Q learning is a very simple formula

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

The formula above have meaning that the entry value in matrix **Q** (that is row represent state and column represent action) is equal to corresponding entry of matrix **R** added by a multiplication of a learning parameter γ and maximum value of **Q** for all action in the next state.

Q Learning Algorithm

Our virtual agent will learn through experience without teacher (this is called unsupervised learning). The agent will explore state to state until it reaches the goal. We call each exploration as an *episode*. In one episode the agent will move from initial state until the goal state. Once the agent arrives at the goal state, program goes to the next episode. The algorithm below has been proved to be convergence (See Sutton and Barto for the proof).

Q Learning
Given: State diagram with a goal state (represented by matrix R)
Find: Minimum path from any initial state to the goal state (represented by matrix Q)

Q Learning Algorithm goes as follow

```
1. Set parameter $\gamma$, and environment reward matrix R
2. Initialize matrix Q as zero matrix
3. For each episode:
   A. Select random initial state
   B. Do while not reach goal state
         a. Select one among all possible actions for the current state
         b. Using this possible action, consider to go to the next state
         c. Get maximum Q value of this next state based on all possible actions
         d. Compute
            $\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max[\mathbf{Q}(next\ state, all\ actions)]$
         e. Set the next state as the current state
      End Do
   End For
```

The above algorithm is used by the agent to learn from experience or training. Each episode is equivalent to one training session. In each training session, the agent explores the environment (represented by Matrix **R**), get the reward (or none) until it reach the goal state. The purpose of the training is to enhance the 'brain' of our agent that represented by **Q** matrix. More training will give better **Q** matrix that can be used by the agent to move in *optimal* way. In this case, if the **Q** matrix has been enhanced, instead of exploring around and go back and forth to the same room, the agent will find the fastest route to the goal state.

Parameter γ has range value of 0 to 1($0 \le \gamma < 1$). If γ is closer to zero, the agent will tend to consider only immediate reward. If γ is closer to one, the agent will consider future reward with greater weight, willing to delay the reward.

To use the **Q** matrix, the agent traces the sequence of states, from the initial state until goal state. The algorithm is as simple as finding action that makes maximum Q for current state:

```
Algorithm to utilize the Q matrix
    Input: Q matrix, initial state
        1. Set current state = initial state
        2. From current state, find action that produce maximum Q
           value
        3. Set current state = next state
        4. Go to 2 until current state = goal state
```

The algorithm above will return sequence of current state from initial state until goal state.

Q Learning Numerical Example

To understand how the **Q** learning algorithm works, we will go through several steps of numerical examples. The rest of the steps can be can be confirm using the program (please check either MS Excel file or Matlab code companion of this tutorial)

Let us set the value of parameter $\gamma = 0.8$ and initial state as room B.

First we set matrix **Q** as a zero matrix.

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

I put again the instant reward matrix **R** that represents the environment in here for your convenient.

$$\mathbf{R} = \begin{array}{c} state \backslash action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{array}$$

Look at the second row (state B) of matrix **R.** There are two possible actions for the current state B, that is to go to state D, or go to state F. By random selection, we select to go to F as our action.

Now we consider that suppose we are in state F. Look at the sixth row of reward matrix **R** (i.e. state F). It has 3 possible actions to go to state B, E or F.

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(B,F) = \mathbf{R}(B,F) + 0.8 \cdot Max\left\{\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)\right\} = 100 + 0.8 \cdot 0 = 100$$

Since matrix **Q** that is still zero, $\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)$ are all zero. The result of computation $\mathbf{Q}(B,F)$ is also 100 because of the instant reward.

The next state is F, now become the current state. Because F is the goal state, we finish one episode. Our agent's brain now contain updated matrix **Q** as

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

For the next episode, we start with initial random state. This time for instance we have state D as our initial state.

Look at the fourth row of matrix **R;** it has 3 possible actions, that is to go to state B, C and E. By random selection, we select to go to state B as our action.

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

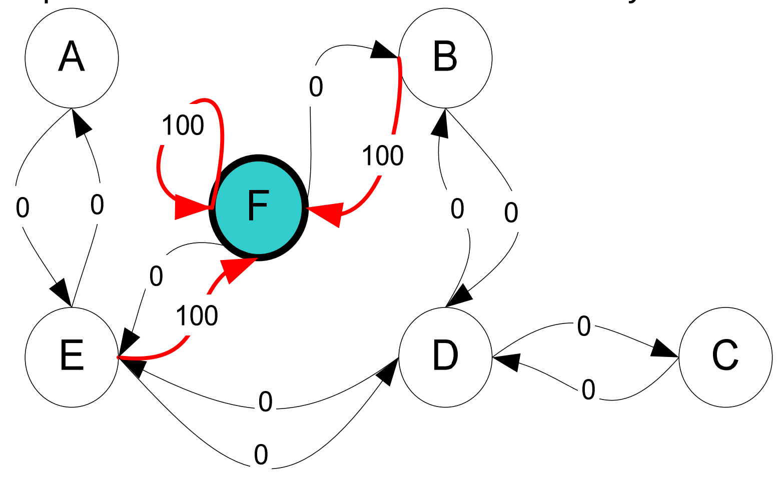
$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$
$$\mathbf{Q}(D,B) = \mathbf{R}(D,B) + 0.8 \cdot Max\left\{\mathbf{Q}(B,D), \mathbf{Q}(B,F)\right\} = 0 + 0.8 \cdot Max\left\{0,100\right\} = 80$$

We use the updated matrix **Q** from the last episode. $\mathbf{Q}(B,D) = 0$ and $\mathbf{Q}(B,F) = 100$. The result of computation $\mathbf{Q}(D,B) = 80$ because of the reward is zero. The **Q** matrix becomes

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The next state is B, now become the current state. We repeat the inner loop in Q learning algorithm because state B is not the goal state.

For the new loop, the current state is state B. I copy again the state diagram that represent instant reward matrix **R** for your convenient.



There are two possible actions from the current state B, that is to go to state D, or go to state F. By lucky draw, our action selected is state F.

Now we think of state F that has 3 possible actions to go to state B, E or F. We compute the Q value using the maximum value of these possible actions.

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(B,F) = \mathbf{R}(B,F) + 0.8 \cdot Max\left\{\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)\right\}$$

$$= 100 + 0.8 \cdot Max\left\{0,0,0\right\} = 100$$

The entries of updated Q matrix contain $\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)$ are all zero. The result of computation $\mathbf{Q}(B,F)$ is also 100 because of the instant reward. This result does not change the Q matrix.

Because F is the goal state, we finish this episode. Our agent's brain now contain updated matrix **Q** as

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

If our agent learns more and more experience through many episodes, it will finally reach convergence values of Q matrix as

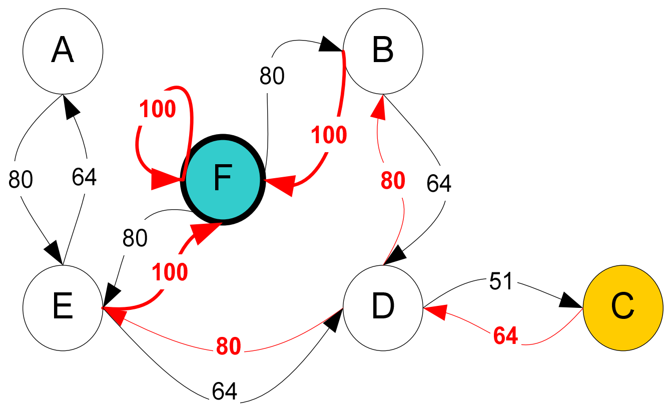
$$\mathbf{Q} = \begin{array}{c} state \backslash action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 400 & - \\ - & - & - & 320 & - & 500 \\ - & - & - & 320 & - & - \\ - & 400 & 256 & - & 400 & - \\ 320 & - & - & 320 & - & 500 \\ - & 400 & - & - & 400 & 500 \end{bmatrix} \end{array}$$

This **Q** matrix, then can be normalized into a percentage by dividing all valid entries with the highest number (divided by 500 in this case) becomes

$$\widehat{\mathbf{Q}} = \begin{array}{c} state \backslash action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 80 & - \\ - & - & - & 64 & - & 100 \\ - & - & - & 64 & - & - \\ - & 80 & 51 & - & 80 & - \\ 64 & - & - & 64 & - & 100 \\ - & 80 & - & - & 80 & 100 \end{bmatrix} \end{array}$$

Once the Q matrix reaches almost the convergence value, our agent can reach the goal in an optimum way. To trace the sequence of states, it can easily compute by finding action that makes maximum Q for this state

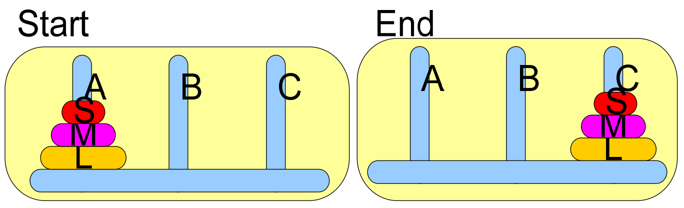
For example from initial State C, it can use the Q matrix as follow:
From State C the maximum Q produces action to go to state D
From State D the maximum Q has two alternatives to go to state B or E.
Suppose we choose arbitrary to go to B
From State B the maximum value produces action to go to state F
Thus the sequence is C – D – B – F



(Check the MS Excel worksheet companion of this tutorial. It is done without any programming).

Q Learning Example: Tower of Hanoi

To wet your appetite more about this Q learning, I make another simple example on solitary game of Tower of Hanoi.



We have three towers name A, B and C and 3 disks of different size name S (small), M (medium), and L (large). In the beginning, all the disks are in tower A and the goal is to move the disks, one at a time, so that in the final state all the three disks are in tower C. The constraint is the size of the disk: a smaller disk cannot be put below a larger disk. Thus, you cannot put disk S below M or L, and you cannot put disk M below L. The game is very simple and the goal is to get the minimum number of disk movement.
Can you think of a solution?

Before going to the solution, let us consider the states of the game. Our simple three disk Tower of Hanoi game may have many states. Can you think of how many states does Tower of Hanoi have?

Since the game is rather simple, I found out that the state combinations of tower of Hanoi with 3 disks are 27 states. Here are the 27 states of the tower of Hanoi:

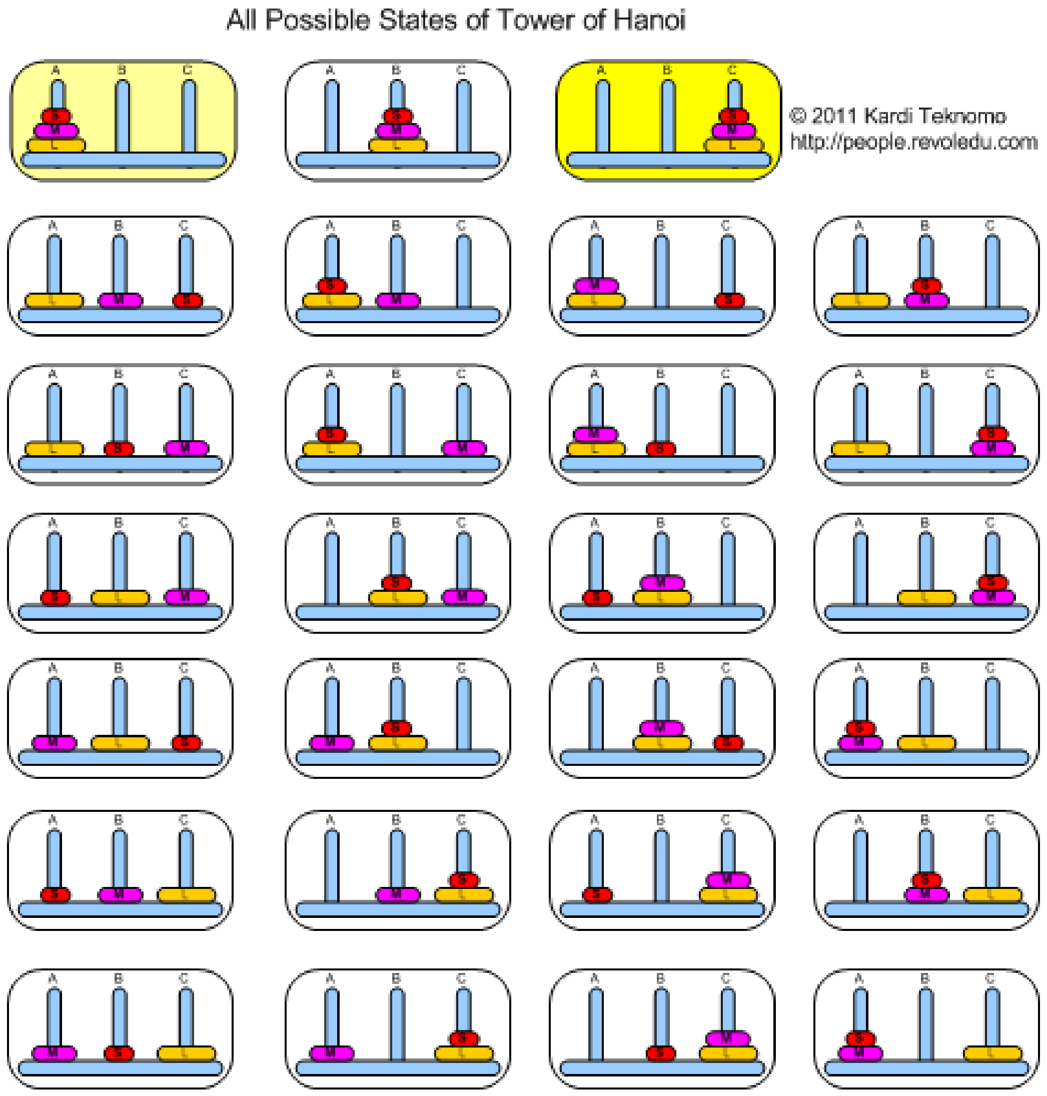
3 disks in a tower	**1 disk in a tower**	**(SL) disks in a tower**	**(ML) disks in a tower**	**(SM) disks in a tower**
(SML)**	LMS	(SL)M*	(ML)S*	(SM)L*
(SML)	LSM	(SL)*M	(ML)*S	(SM)*L
**(SML)	MLS	M(SL)*	S(ML)*	L(SM)*
	SLM	*(SL)M	*(ML)S	*(SM)L
	MSL	M*(SL)	S*(ML)	L*(SM)
	SML	*M(SL)	*S(ML)	*L(SM)

State notation SML represents disk S is in tower A, disk M is in tower B and disk L is in tower C.

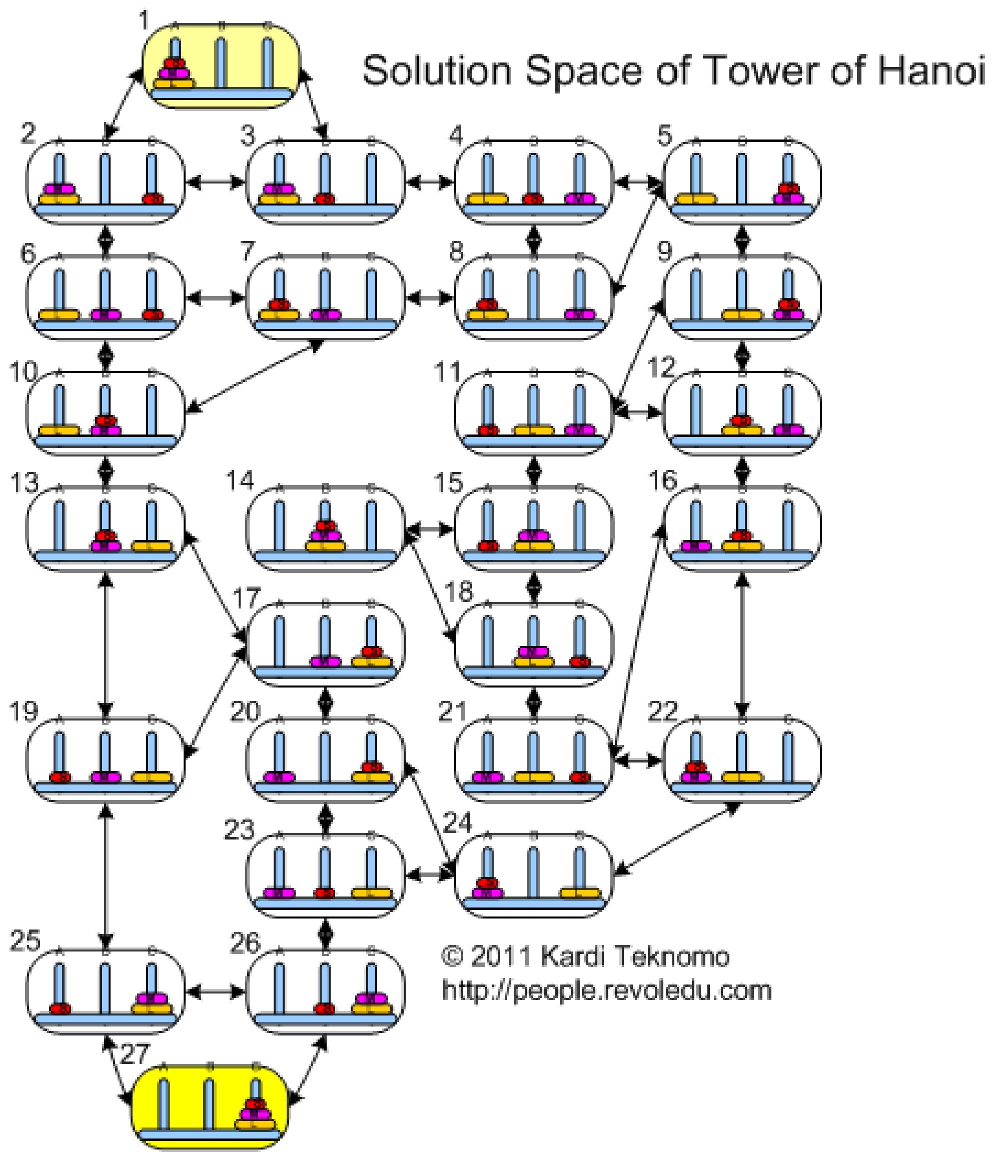
Symbol (SML) represents disk S is on top of disk M and both of them are on top of disk L and the parentheses symbol is to indicate that the three disks are located in one tower. The * is to indicate that there is no disk in the tower. Thus, state *(SML)* represents empty tower A and C while tower B contains the three disks.

Similarly, symbol (SM) represents disk S is on top of disk M and the parentheses is to indicate the two disks are located together in one tower. Thus state (SM)L* indicates disk S and M are in tower A, disk L is in tower B and empty tower C.

Diagram below shows all 27 possible state of the Tower of Hanoi.

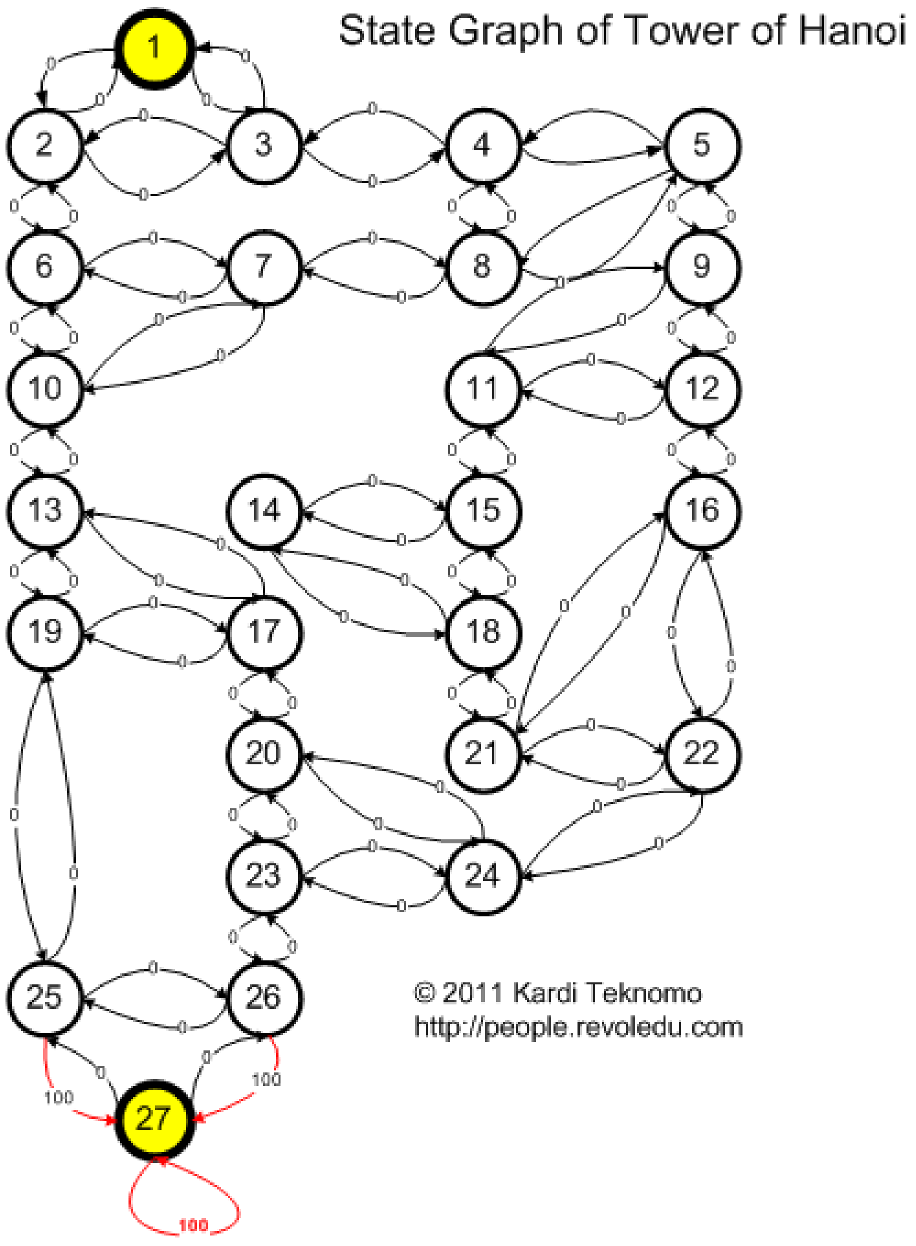

All Possible States of Tower of Hanoi
© 2011 Kardi Teknomo
http://people.revoledu.com
A
B
C
S
M
L

From all possible states, we can create all possible transitions between states. That is our next move. Graph below is my solution. I trace all possible combination of states and all possible combination of transition between states from the starting state until the final state. Let us call the diagram below as the solution space of the tower of Hanoi.



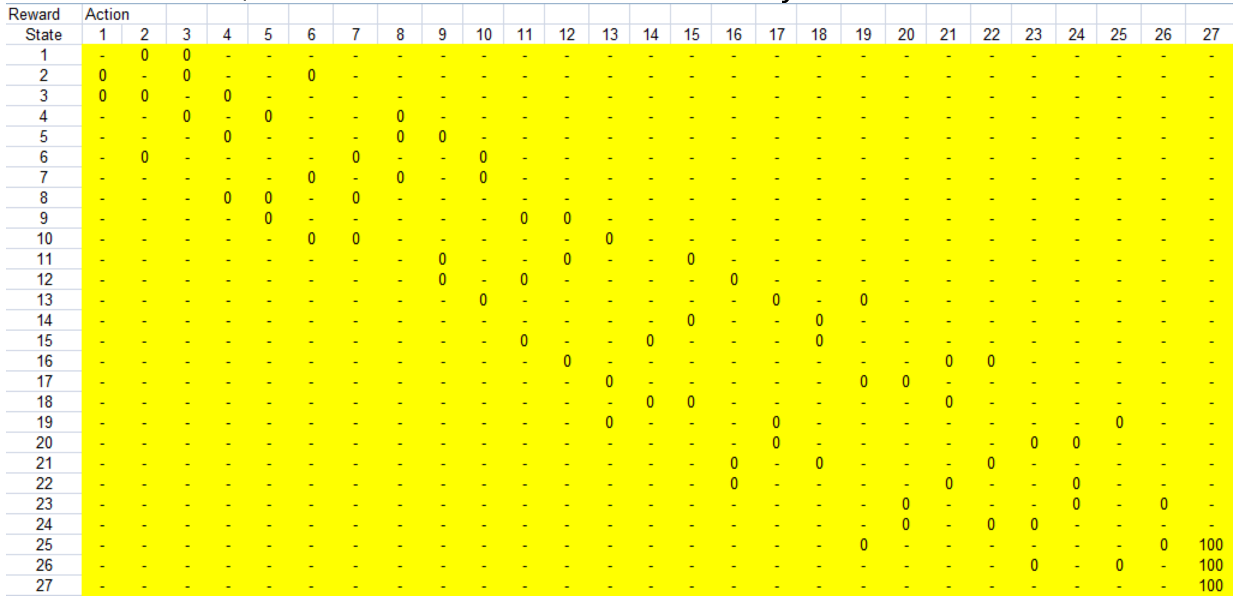
In the next section, you will learn the solution of the Tower of Hanoi using Q-Learning.

To use Q learning to solve this problem, I transform the solution space graph above into state-diagram with reward value of zero to all links except the direct link heading to the goal and the loop in the goal. See the state diagram below.



To solve this problem, we transform the state diagram above into **R** matrix as shown in the picture below (check the MS Excel worksheet companion of this tutorial. It is done without any programming).

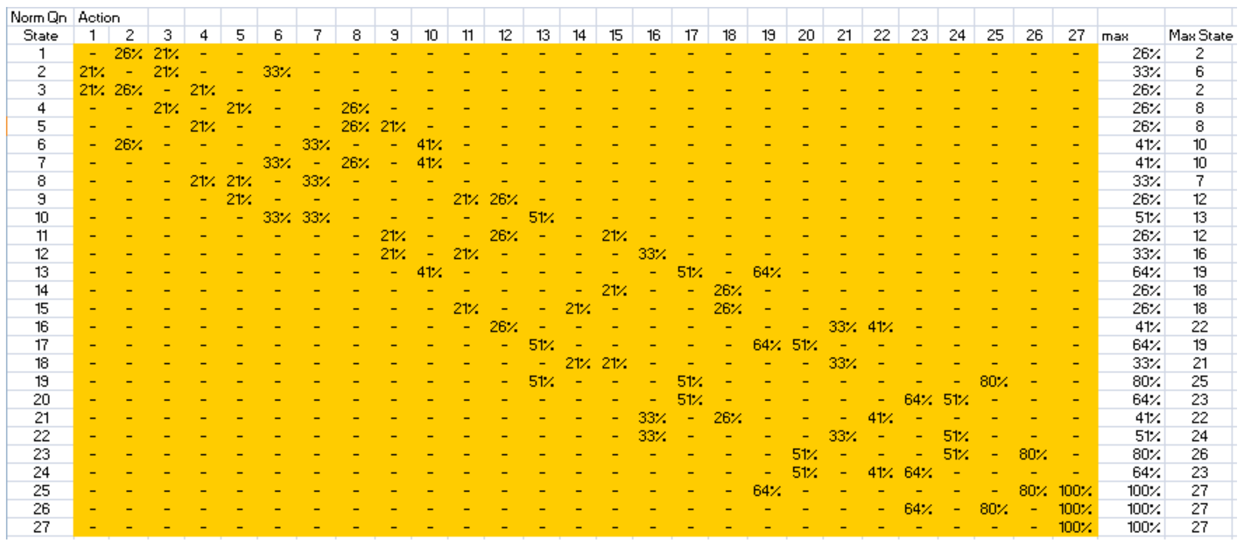
Remember the R matrix is simply an adjacency matrix of the state graph above, with special attention to the links toward the goal. All links toward the goal (including an additional self loop) must have high value (100) while all other links have zero values. Non-connected nodes are not considered, and therefore the value is infinity.



Reward	Action																										
State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	0	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	0	0	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	0	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	0	-	-	-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	-	0	-	-	-	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	-	-	-	-	-	0	-	0	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	-	-	-	0	0	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	0	-	-	-	-	-	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	0	0	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-	0	-	-	0	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	0	-	0	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	0	-	0	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	0	-	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-	-	-	0	-	-	0	-	-	-	0	-	-	-	-	-	-	-	-	-
16	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	0	0	-	-	-	-	-
17	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	0	0	-	-	-	-	-	-	-
18	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	-	-	-	-	-	0	-	-	-	-	-	-
19	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	0	-	-	-	-	-	-	-	0	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	0	0	-	-	-
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	0	-	-	-	0	-	-	-	-	-
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	0	-	-	0	-	-	-
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	0	-	0	-
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	0	0	-	-	-	-
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	0	100
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	0	-	100
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100

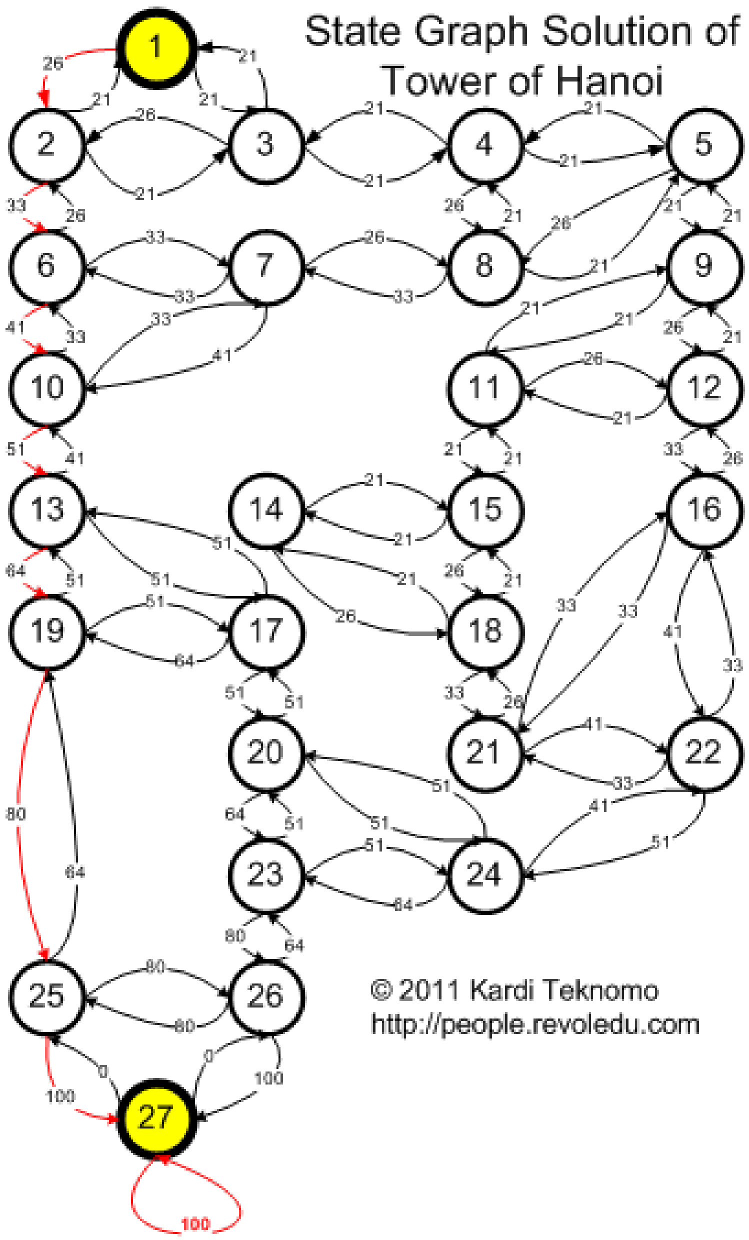
Then transform the **R** matrix into **Q** matrix and normalized it. Do you still remember, how to transform R matrix into Q matrix?

The result of normalized **Q** matrix is given below. These values can then put back into the state graph to get the solution.



Norm Qn	Action																												
State	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	max	Max State
1	-	26%	21%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	2
2	21%	-	21%	-	-	33%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33%	6
3	21%	26%	-	21%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	2
4	-	-	21%	-	21%	-	-	26%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	8
5	-	-	-	21%	-	-	-	26%	21%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	8
6	-	26%	-	-	-	-	33%	-	-	41%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	41%	10
7	-	-	-	-	-	33%	-	26%	-	41%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	41%	10
8	-	-	-	21%	21%	-	33%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33%	7
9	-	-	-	-	21%	-	-	-	-	-	21%	26%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	26%	12
10	-	-	-	-	-	33%	33%	-	-	-	-	-	51%	-	-	-	-	-	-	-	-	-	-	-	-	-	-	51%	13
11	-	-	-	-	-	-	-	-	21%	-	-	26%	-	-	21%	-	-	-	-	-	-	-	-	-	-	-	-	26%	12
12	-	-	-	-	-	-	-	-	21%	-	21%	-	-	-	-	33%	-	-	-	-	-	-	-	-	-	-	-	33%	16
13	-	-	-	-	-	-	-	-	-	41%	-	-	-	-	-	-	51%	-	64%	-	-	-	-	-	-	-	-	64%	19
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	21%	-	-	26%	-	-	-	-	-	-	-	-	-	26%	18
15	-	-	-	-	-	-	-	-	-	-	21%	-	-	21%	-	-	-	26%	-	-	-	-	-	-	-	-	-	26%	18
16	-	-	-	-	-	-	-	-	-	-	-	26%	-	-	-	-	-	-	-	-	33%	41%	-	-	-	-	-	41%	22
17	-	-	-	-	-	-	-	-	-	-	-	-	51%	-	-	-	-	-	64%	51%	-	-	-	-	-	-	-	64%	19
18	-	-	-	-	-	-	-	-	-	-	-	-	-	21%	21%	-	-	-	-	-	33%	-	-	-	-	-	-	33%	21
19	-	-	-	-	-	-	-	-	-	-	-	-	51%	-	-	-	51%	-	-	-	-	-	-	-	80%	-	-	80%	25
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	51%	-	-	-	-	-	64%	51%	-	-	-	64%	23
21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33%	-	26%	-	-	-	41%	-	-	-	-	-	41%	22
22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	33%	-	-	-	-	33%	-	-	51%	-	-	-	51%	24
23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	51%	-	-	-	51%	-	80%	-	80%	26
24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	51%	-	41%	64%	-	-	-	-	64%	23
25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	64%	-	-	-	-	-	-	80%	100%	100%	27
26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	64%	-	80%	-	100%	100%	27
27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100%	100%	27

You can also see the solution in the state diagram below. The red color of arrow is the optimum path (minimum path) from initial state to the goal state.

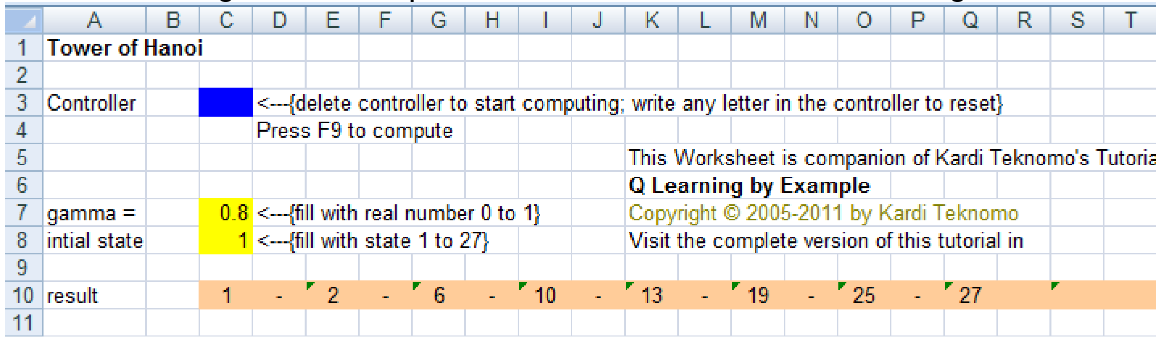

State Graph Solution of Tower of Hanoi
© 2011 Kardi Teknomo
http://people.revoledu.com

Actually, the Q values in this diagram will lead to any initial state (not only from state 1) to the goal state using minimum path. Very impression solution, isn’t it?

To go from any state to the goal state, the agent only need to maximize the value of the arrow out of that state. Following these simple maximization procedures will eventually get the optimum solution! Q-Learning has transformed the global navigational values into local decision by the agent. In my scientific papers, I called these kinds of transformations as Sink Propagation Values (SPV).

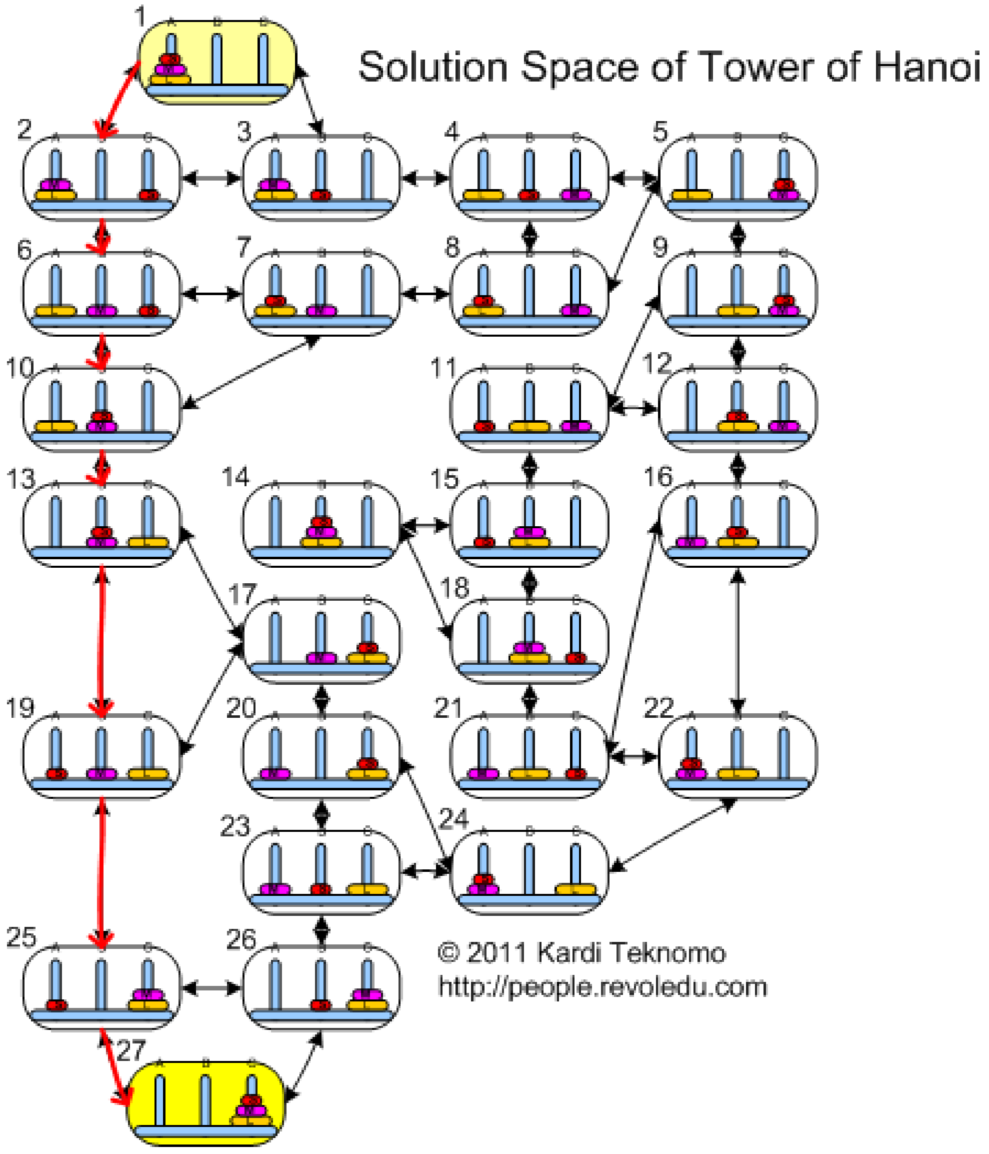
So, what is the solution of the tower of Hanoi?

The results is given as sequence of states is shown in the figure below



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	**Tower of Hanoi**																			
2																				
3	Controller		🟦	<---{delete controller to start computing; write any letter in the controller to reset}																
4				Press F9 to compute																
5											This Worksheet is companion of Kardi Teknomo's Tutoria									
6											**Q Learning by Example**									
7	gamma =		0.8	<---{fill with real number 0 to 1}							Copyright © 2005-2011 by Kardi Teknomo									
8	intial state		1	<---{fill with state 1 to 27}							Visit the complete version of this tutorial in									
9																				
10	result		1	-	2	-	6	-	10	-	13	-	19	-	25	-	27			
11																				

Better yet, graphical solution is always the best. Just follow the red arrows, you will get the optimum solution.



As summary, Q-Learning can help agent navigation and path motion planning such that the agent can find the optimum solution from any state to the goal state. Unless Dikjstra or A* method that find the optimum solution between two pairs of node, Q-learning will solve the optimum solution from any nodes to the goal node.

Q Learning using Matlab

I have made simple Matlab Code for this example and you can modify it for your need.

To model the environment you need to make the instant reward matrix R. Put zero for any door that is not directly to the goal and put value 100 to the door that lead directly to the goal. For unconnected states, use minus Infinity (-Inf) so that it become very negative number. We want to maximize the Q values, thus very negative number will not be considered at all.

The state is numbered 1 to N (in our example N = 6). The result of the code is only normalized **Q** matrix.

You may experiment in the effect of parameter gamma to see how it influences the results.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Q learning of single agent move in N rooms
% Matlab Code companion of
% Q Learning by Example, by Kardi Teknomo
% (http://people.revoledu.com/kardi/)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function q=ReinforcementLearning
clc;
format short
format compact

% Two input: R and gamma
% immediate reward matrix; row and column = states; -Inf = no door between room
R=[-inf,-inf,-inf,-inf,   0, -inf;
   -inf,-inf,-inf,   0,-inf,  100;
   -inf,-inf,-inf,   0,-inf, -inf;
   -inf,   0,    0,-inf,   0, -inf;
      0,-inf,-inf,   0,-inf,  100;
    -inf,  0,-inf,-inf,   0,  100];
gamma=0.80;             % learning parameter



q=zeros(size(R));        % initialize Q as zero
q1=ones(size(R))*inf;    % initialize previous Q as big number
count=0;                 % counter

for episode=0:50000
    % random initial state
    y=randperm(size(R,1));
    state=y(1);

    % select any action from this state
    x=find(R(state,:)>=0);        % find possible action of this state
    if size(x,1)>0,
       x1=RandomPermutation(x);   % randomize the possible action
       x1=x1(1);                  % select an action (only the first element of random sequence)
    end

    qMax=max(q,[],2);
    q(state,x1)= R(state,x1)+gamma*qMax(x1);     % get max of all actions from the next state for Q of
current state
    state=x1;

    % break if convergence: small deviation on q for 1000 consecutive
    if sum(sum(abs(q1-q)))<0.0001 & sum(sum(q >0))
       if count>1000,
          episode  % report last episode
          break % for
       else
          count=count+1; % set counter if deviation of q is small
       end
    else
       q1=q;
       count=0;  % reset counter when deviation of q from previous q is large
    end
end

%normalize q
g=max(max(q));
if g>0,
    q=100*q/g;
end
```

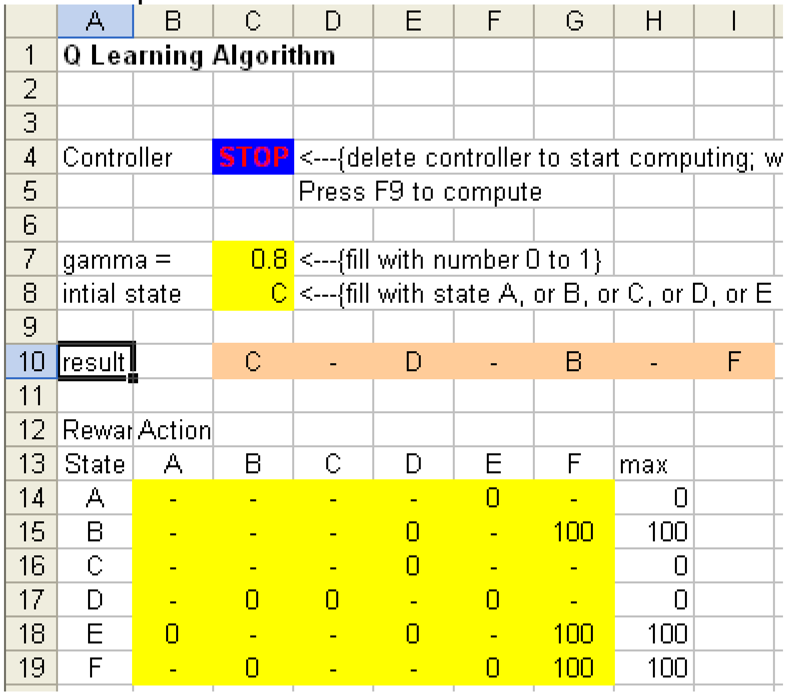
The code above is using basic library RandomPermutation

```
function y=RandomPermutation(A)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% return random permutation of matrix A
% unlike randperm(n) that give permutation of integer 1:n only,
% RandomPermutation rearrange member of matrix A randomly
% This function is useful for MonteCarlo Simulation, Bootstrap sampling, game, etc.
%
% Copyright Kardi Teknomo(c) 2005
% (http://people.revoledu.com/kardi/)
%
% example: A = [ 2, 1, 5, 3]
%   RandomPermutation(A) may produce [ 1, 5, 3, 2] or [ 5, 3, 2, 3]
%
% example:
%   A=magic(3)
%   RandomPermutation(A)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[r,c]=size(A);
b=reshape(A,r*c,1);         % convert to column vector
x=randperm(r*c);            % make integer permutation of similar array as key
w=[b,x'];                   % combine matrix and key
d=sortrows(w,2);            % sort according to key
y=reshape(d(:,1),r,c);      % return back the matrix
```

Q Learning Using MS Excel

The spreadsheet companion of this tutorial contains two worksheets. No macro or any script included. The first worksheet (RL1) is only using spreadsheet to compute the Q matrix from R matrix (up to 5 iterations). The purpose is to give you idea on how it works. The second worksheet (RL1 Auto) is the main programs that use MS excel iteration. It can compute Q matrix until convergence value.

To compute it automatically, first please make sure that the Tools-Options-Calculation Tab – Iteration checkbox is ON. Then delete STOP cell and press F9.



	A	B	C	D	E	F	G	H	I
1	**Q Learning Algorithm**								
2									
3									
4	Controller		STOP	<---{delete controller to start computing; w					
5				Press F9 to compute					
6									
7	gamma =		0.8	<---{fill with number 0 to 1}					
8	intial state		C	<---{fill with state A, or B, or C, or D, or E					
9									
10	result		C	-	D	-	B	-	F
11									
12	Rewar	Action							
13	State	A	B	C	D	E	F	max	
14	A	-	-	-	-	0	-	0	
15	B	-	-	-	0	-	100	100	
16	C	-	-	-	0	-	-	0	
17	D	-	0	0	-	0	-	0	
18	E	0	-	-	0	-	100	100	
19	F	-	0	-	-	0	100	100	

To reset the controller, type any letter (e.g. STOP) in the blue cell.

Yellow cells are input cells you can modify them without affecting the program.
The results are in orange.

How the MS Excel Q learning works?

Unlike pure Q learning algorithm I explained in previous sections, Q learning in MS excel is using iteration. The result is the same but the model is much more simplified without any random number involve (thus my methods in Excel is deterministic, not stochastic).

If you read through all this tutorial, you will find that Q learning find the value of Q from initial state until the goal state. In this MS Excel program, however, I use the dual of this Q learning algorithm. Instead of finding one state until goal, in each step of the iteration, I am finding the Q value of all states. Eventually, it will reach the same result because of the convergence of Q learning.

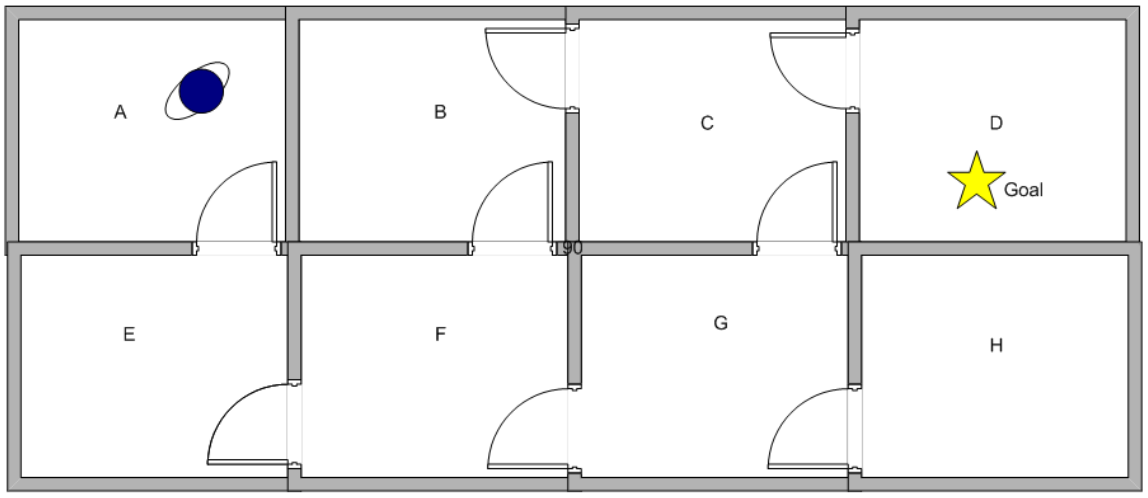
What the dual of Q learning algorithm perform is to redistribute the value of R matrix into optimal values that can guide the agent from any state to the goal state. This redistribution is very useful especially when the agent face new or unknown environment.

The dual technique of Q-learning comes with its strength and weakness. The strength is in the simplification of computation. The stochastic search through many episodes (as in the original Q-learning algorithm) becomes deterministic search through simple iterations. The dual technique has its own weakness that it requires you to know all the states in advanced before you can iterate. The pure Q-Learning algorithm does not have this weakness because the state can be created during the stochastic learning.

I assume you know how to use MS Excel function VLOOKUP, INDEX, MATCH and IF. Refer to the help of Microsoft Excel if you are not sure about those internal functions.

Practice Make Perfect

For your own practice, try by yourselves to model the following rooms into R matrix and get the Q matrix. Then try any initial room and let your agent move to the target room



As a clue, you have now 8 interconnected rooms, thus you should have 8 nodes. The matrix would have 8 rows by 8 columns.

References

- Mitchell, T. M. (1997) Machine Learning, McGrawHill
- Sutton, R.S. and Barto, A.G. (1998), Reinforcement Learning – an Introduction, MIT Press