Reinforcement Learning
Lecture 11

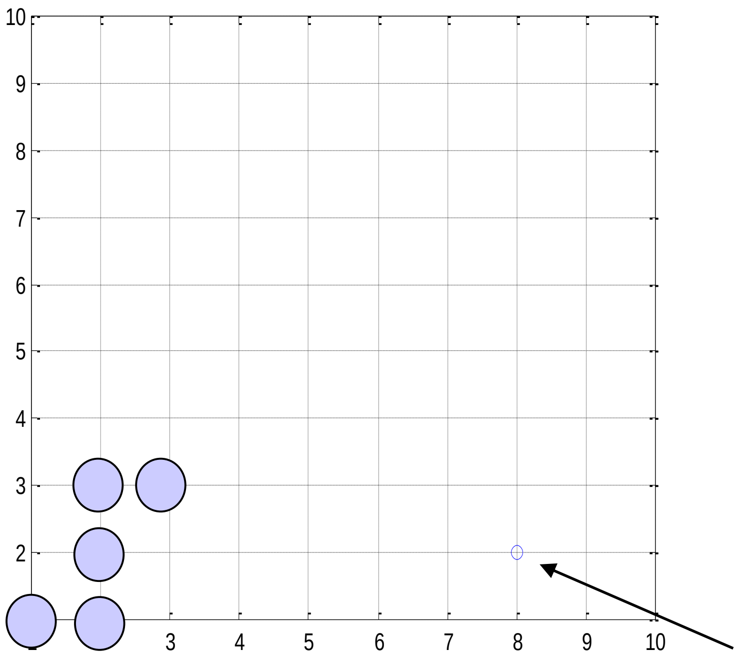
Jeong-Yean Yang

2020/12/10

5 Examples

State Value, V(s) in 2Dim Space

- l11mc.py and l11td.py
- Grid world (w=10, h=10)



Agent

Random movement
N, E, W,S

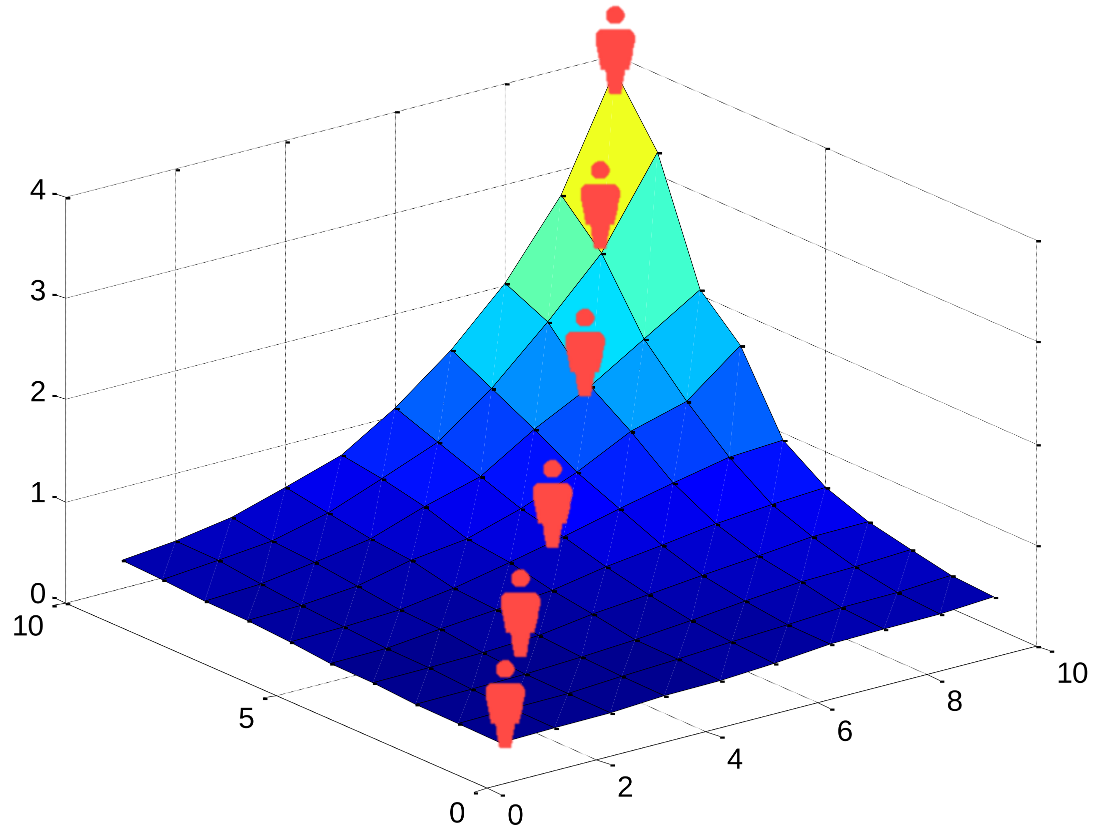
N: $y \leftarrow y+1$
S: $y \leftarrow y-1$
E: $x \leftarrow x+1$
W: $x \leftarrow x-1$

Initial position: (1,1)
Goal position : (10,10)

Reward at goal : r=+1
Others: r=0

State Value: V(s)

- Optimal path: Follows the Maximum State Value, V(s)

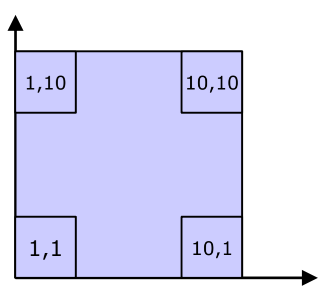


MC-based 2D Problem: l11mc.py

```
# initialization
n    = 10
s0   = array("1;1")
s    = array(2,1)
a    = array(2,1)
V    = array(n,n)
h    = array()
R    = 0
alpha=0.01;

def init():
    global s,h
    s    = s0.copy()
    h    = array()
```

- S0= initial position
  - (1,1) is the starting position
- V(s)= 2 dimension
  - V(1,1) ,v(1,2)… V(1,10)
  - V(2,1),V(2,2)…. V(2,10)
  - ….
  - V(10,1),V(10,2),…. V(10,10)
- History, h is also 2 Dim.

$$h = \begin{bmatrix} 1 \\ 1 \end{bmatrix}_{Initial}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \dots \begin{bmatrix} 10 \\ 10 \end{bmatrix}_{\text{Terminal}}$$



Monte Carlo Update

```
# II. Calculate MC method.
for i in range(0,h.c):
    s    = h[:,i+1]
    x    = int(s[1,1])
    y    = int(s[2,1])
    V[x,y]   = alpha*R + (1-alpha)*V[x,y]
clear(1)
surf(V)
```

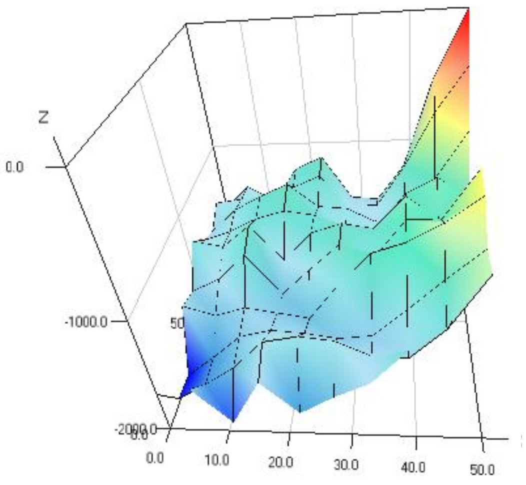
pp. 22 from lecture 10

History

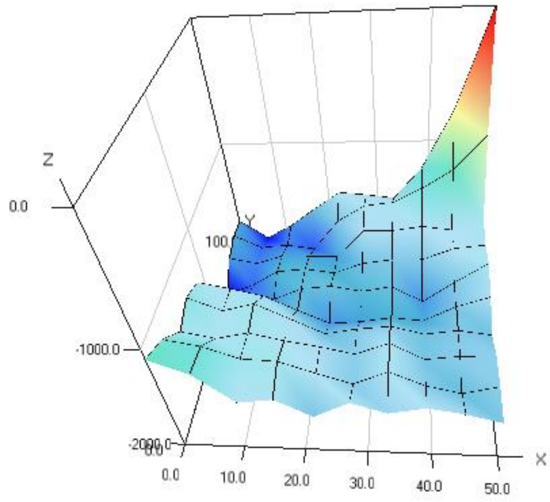
Return

- MC visits the past state
  - → The Agent may visit the state, s more than one.
  - If the number of visits increases, V(s) is updated by return R in many times.
  - With many visits, Return value is smoothly updated like color blurring by finger

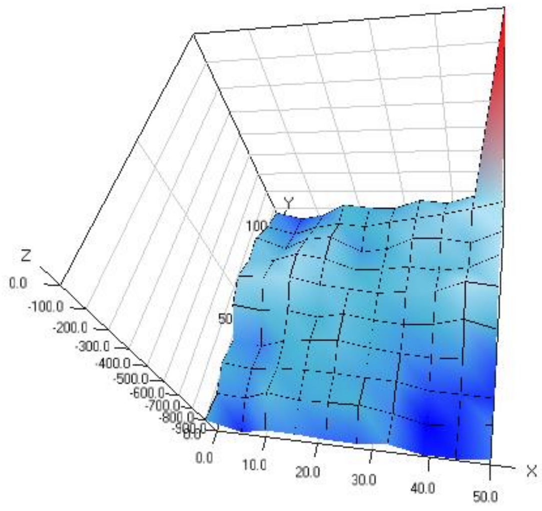
Monte Carlo Method
State Value l11mc.py



After 1 episode



After 50 episode



After 200 episode

Temporal Difference 2D problem: l11td.py

```
def episode():
    global s,V,alpha,g
    R   = 0
    init()

    # I. Exploration until an agent reaches at terminals
    while(True):
        # 1. save state,s as sold
        so  = s.copy()
```

SO is the old state
S is the new state
by action, a like
S = S0+a,

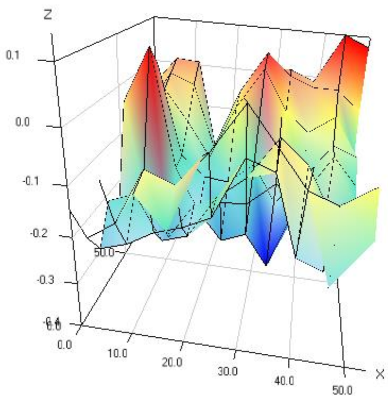
```
# 5. Update TD
x   = int(so[1,1])
y   = int(so[2,1])
xn  = int(s[1,1])
yn  = int(s[2,1])

V[x,y]  = alpha*(r+g*V[xn,yn]) + (1-alpha)*V[x,y];

# 6. Exit
if (stop==True):
    break;
```

$$V(s) = (1-\alpha)V(s) + \alpha\left(r(s) + \gamma V(s')\right)$$

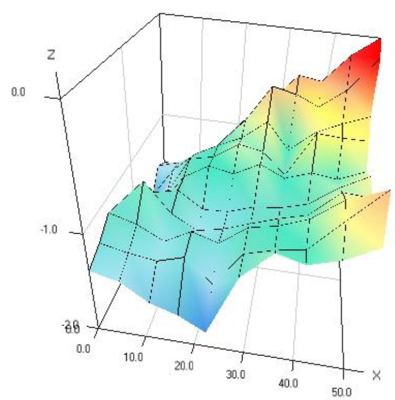
Value update by TD

TD learning in 2D : l11td.py

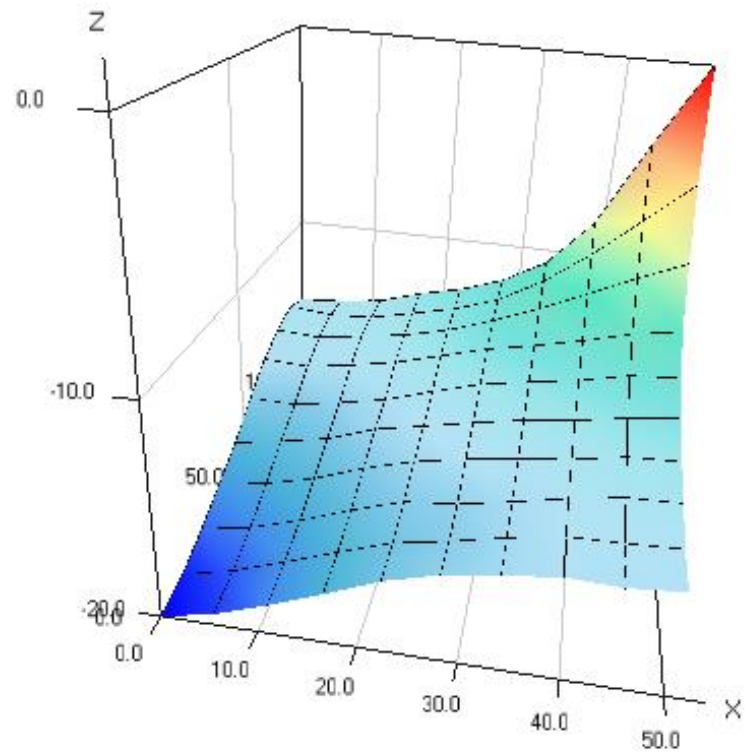
- Episodes : 100
- Alpha=0.01
- Gamma=0.99



1 episode



10 episode



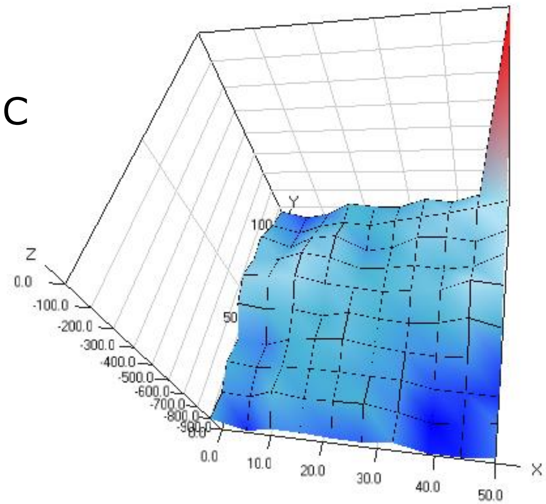
After 100 episode

Why TD is So Faster than MC?

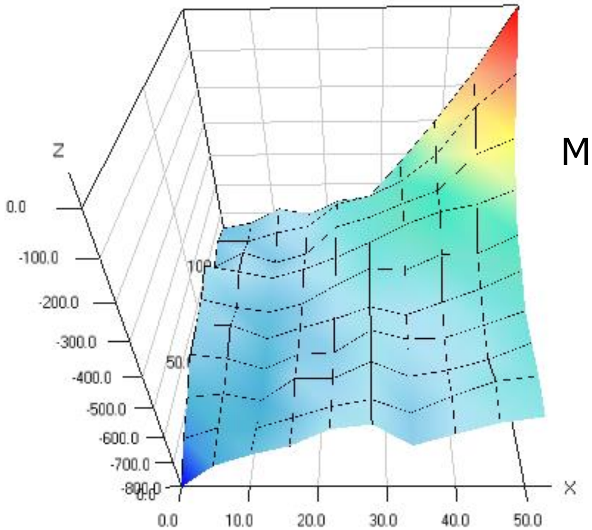
- In many cases, TD is faster than MC
- "But, it is NOT clearly Proven".

- We know that **MC is sensitive to alpha value**
- See next example

MC



Alpha=0.01

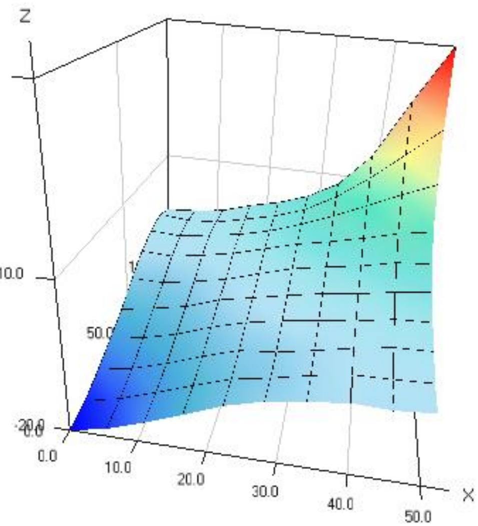


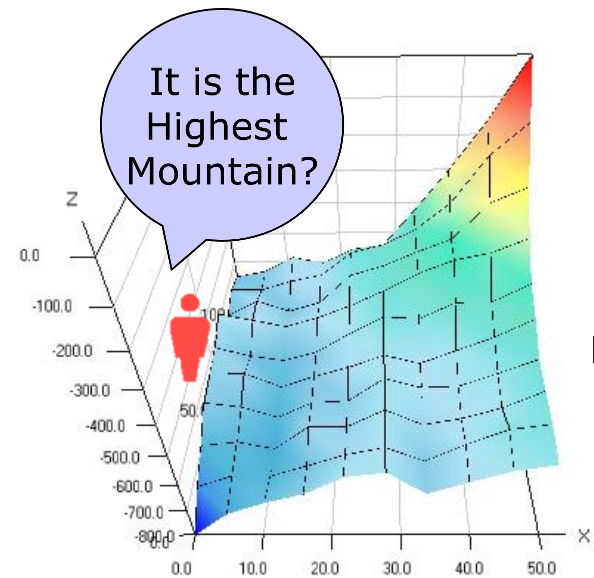
MC

Alpha=0.001

Why TD get so Smooth V(s)?

TD





MC

- TD is the function of (s, so)→ $V(s) = (1-\alpha)V(s) + \alpha\left(r(s) + \gamma V(s')\right)$
- MC is the function of (R, s)
- Which one is better?
  - TD looks better. Many local maxima is not good for climbing
  - However, Blurring with TD makes Biased V(s) sometimes.

6 Q-Learning

State Value Knows Where to GO, but does not teach Which Action to do.

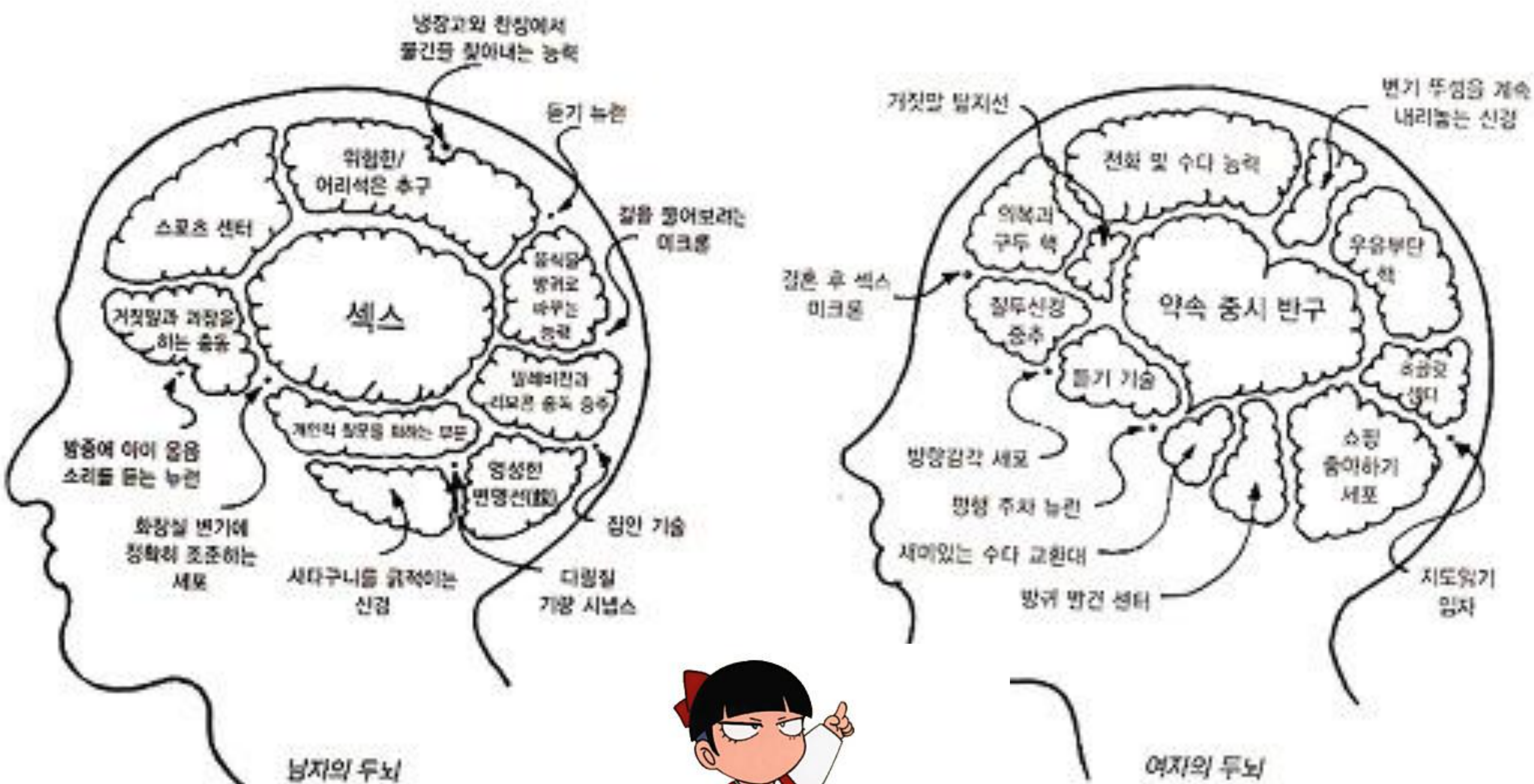
- State value, V(s): Expected returns of observed state

- From Sense(s)-and-Action(a),
  How we choose the best action?

- State value is the INDIRECT legend.
- We want Direct Legend, that is the BEST action.

$$s \xrightarrow{Action(?)} s'$$

People Believes that I know my State, but it is NOT True



State Example (I)
You Cannot Observe Everything!

- State,s = Your consciousness

Observed State

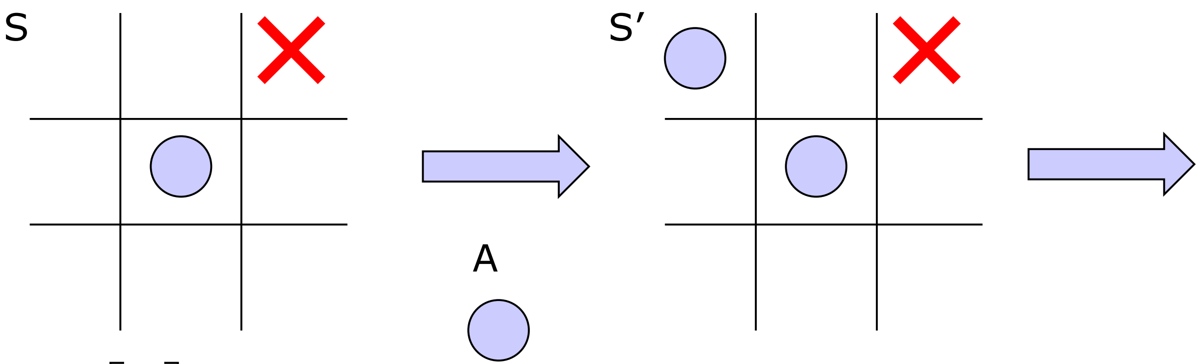
$$s = \begin{bmatrix} \boxed{\begin{matrix} Hunger \\ Money \\ Health \end{matrix}} \\ \cdots \\ \boxed{Grade} \end{bmatrix} \xrightarrow{\text{Action: Presentation}} s' = \begin{bmatrix} Hunger \\ Money \\ Health \\ \cdots \\ Grade^* \end{bmatrix}$$

Unobserved State

State Example (II)

It is NOT your Turn → Environment Dynamics

- Think Tic-Tac-Toe



Environment (opponent player) Do action of X mark

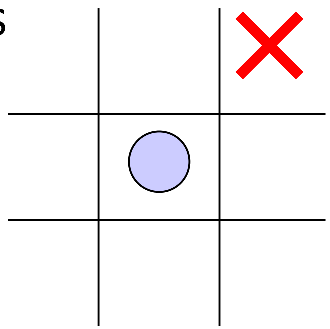
$$s=\begin{bmatrix} * \\ * \\ X \\ * \\ O \\ * \\ * \\ * \\ * \end{bmatrix}, 3^9=19683 \qquad s'=\begin{bmatrix} O \\ * \\ X \\ * \\ O \\ * \\ * \\ * \\ * \end{bmatrix}$$

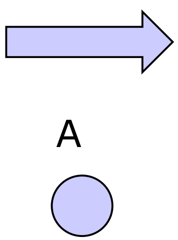
State Example (II)

It is NOT your Turn → Environment Dynamics

- Think Tic-Tac-Toe

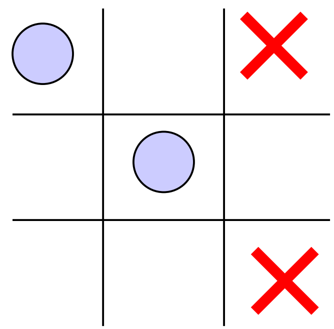
S





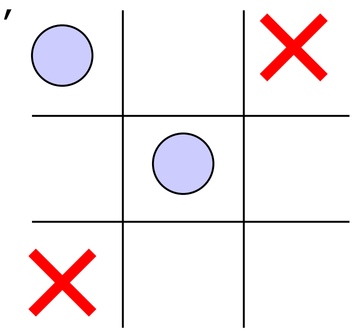
$$s=\begin{bmatrix} * \\ * \\ X \\ * \\ O \\ * \\ * \\ * \\ * \end{bmatrix}, 3^9 = 19683$$

S′



$$s'=\begin{bmatrix} O \\ * \\ X \\ * \\ O \\ * \\ * \\ * \\ X \end{bmatrix}$$

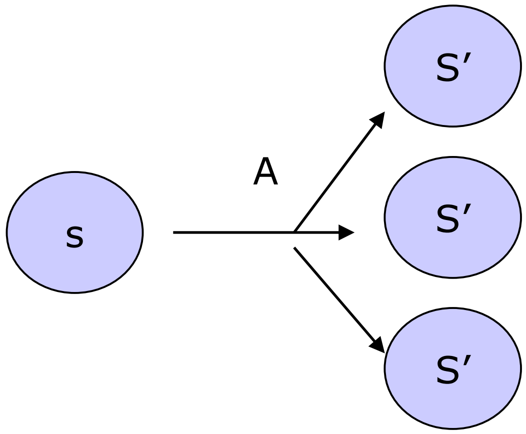
S′



$$s'=\begin{bmatrix} O \\ * \\ X \\ * \\ O \\ * \\ X \\ * \\ * \end{bmatrix}$$

Six Possible cases

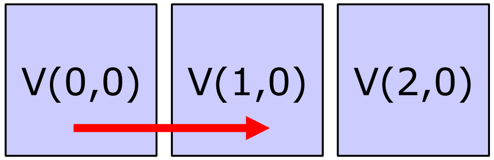
Environment Dynamics makes my prediction from S to S' to be Wrong!



Uncertainties

...

Action and Next state is NOT directly associated.



Which action is OK?

- Agent wants moves

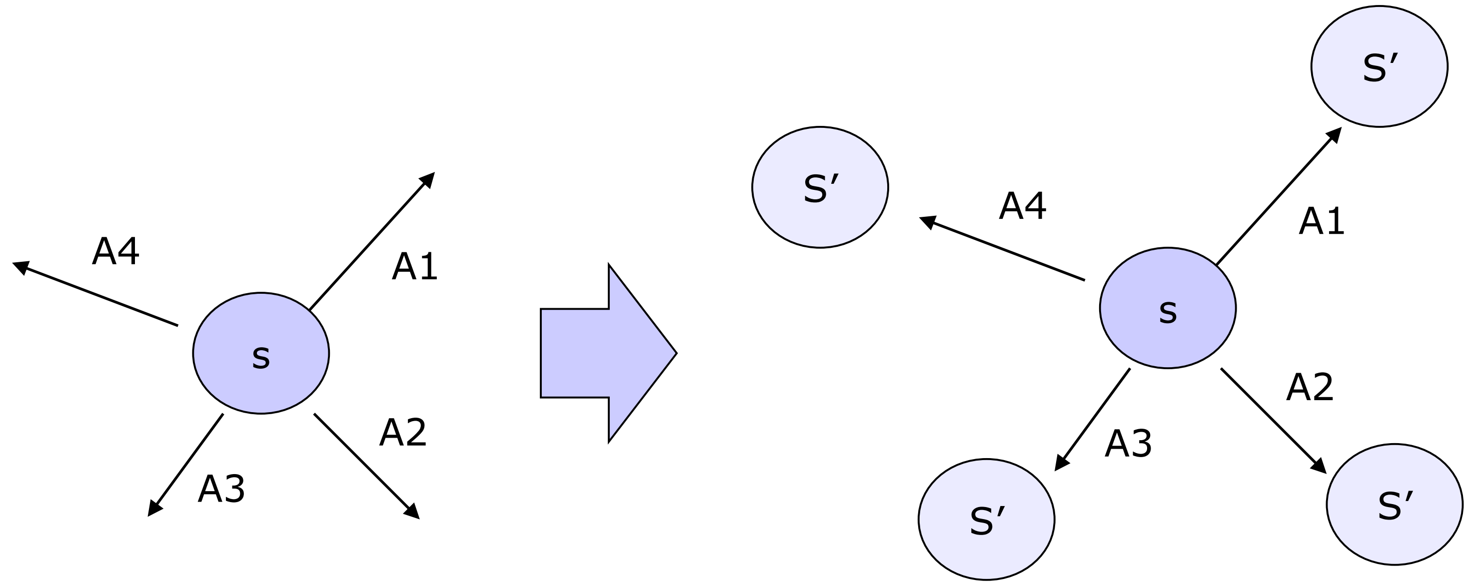
From S(0,0) to S(1,0).

But, what kinds of action can do this?

+1 or right move is NOT the Answer in stochastic world

Q(s,a) space instead of State Value,V(s)

- Q space : State-and-Action Space (S-A space)



- In Q space, all possible actions are considered with a given

Q-Learning

- Instead of TD-based learning with State Value, V(s),
- Q-learning uses Q space, Q(s,a)

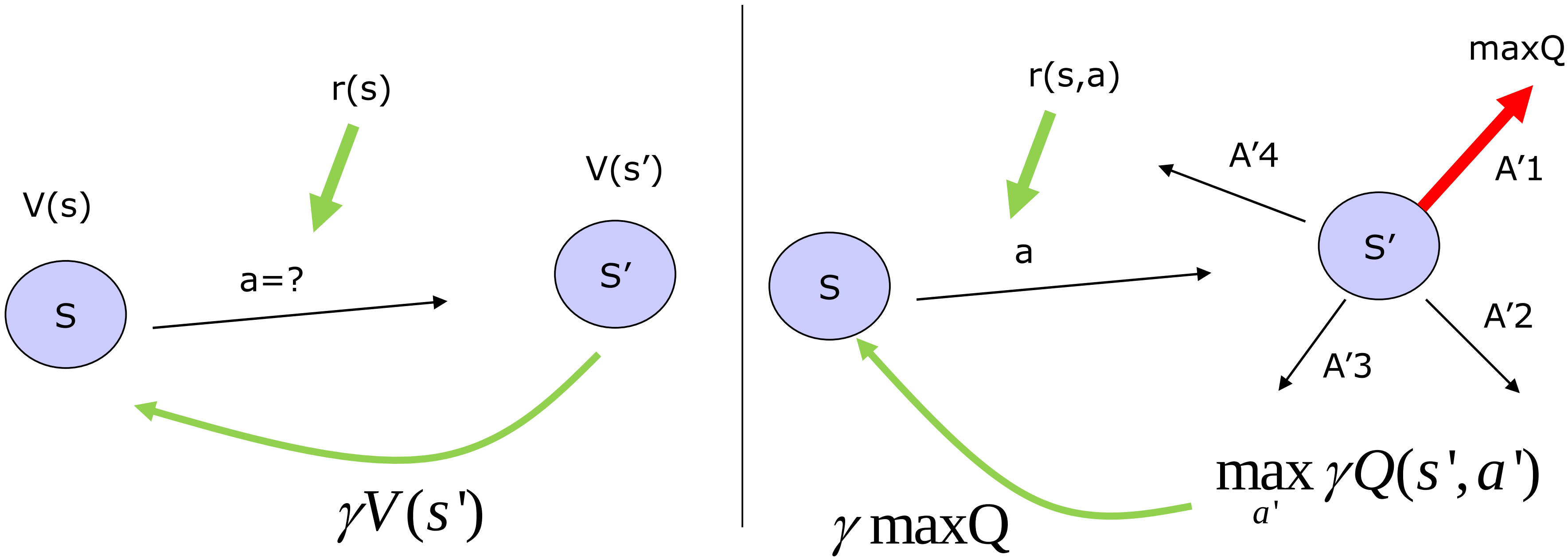
$$TD : V(s) = r(s) + \gamma \underline{V(s')}$$

$$Q - Learning : Q(s,a) = r(s,a) + \gamma \underline{\max_{a'} Q(s',a')}$$

- Think Expectation,

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \max_{a'} Q(s',a') \right]$$

Update Rule of TD- VS. Q-Learning



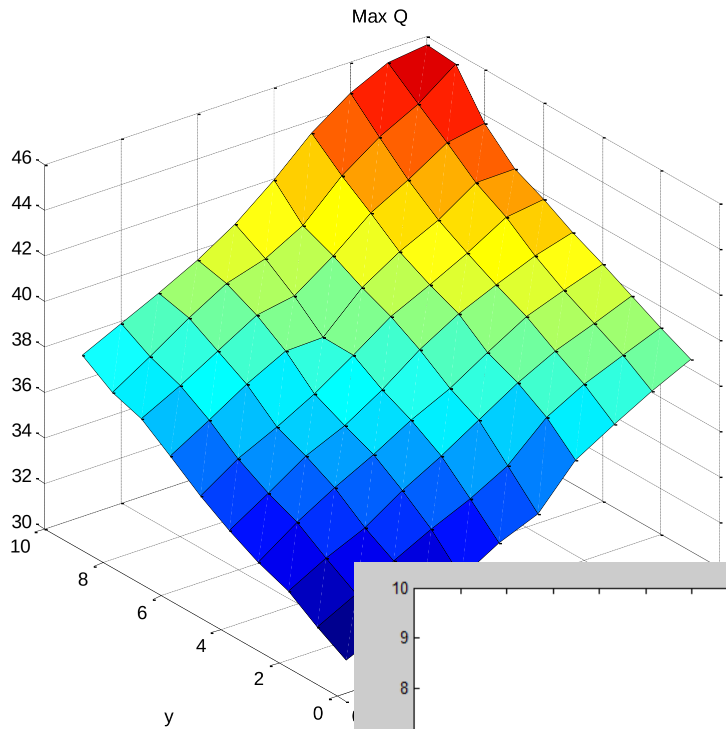
- Q learning in State-and-Action space
- V(s') is not defined in SA space.
- The discounted maximum Q is updated for state S.

Q-Learning's Two Stages.

- 1. Exploration
  - Exploration is based on an agent's Experience.
  - It is episodic memory.
  - An agent tries to explore the target space in a random way.
  - All Returns and Actions are stored into Q values.

- 2. Exploitation
  - get the best actions
  - Using episodic memory during exploration, an agent tries to find the best(or optimized) actions in every steps.
- Question :What is the goal of Exploitation?
  - It is not to reach goals, but an agent tries to **get more rewards**

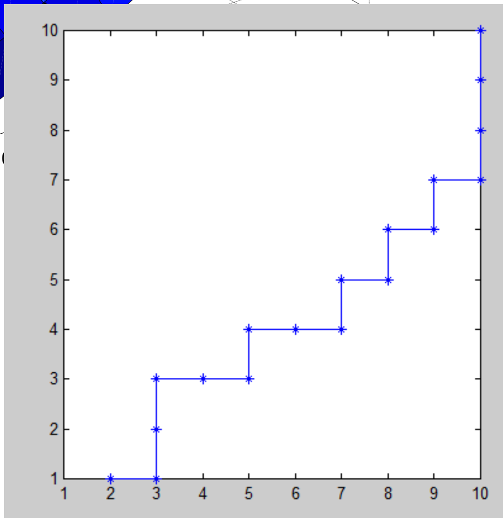
Grid World Test
test4 or ex/ml/l11q1



- Plot MaxQ in each state.
- It is faster than TD.

- Q(s,a) indicated which way an agent goes!
- Find the best way

(find the max Q)



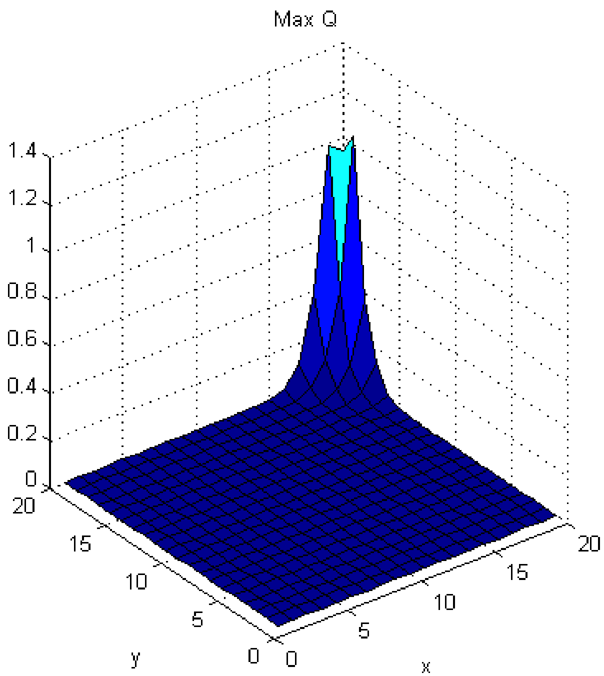
Q(s,Left)=0.1, Q(s,Right)=0.3

Q(s,Up)=0.8, Q(s, Down)=0.01

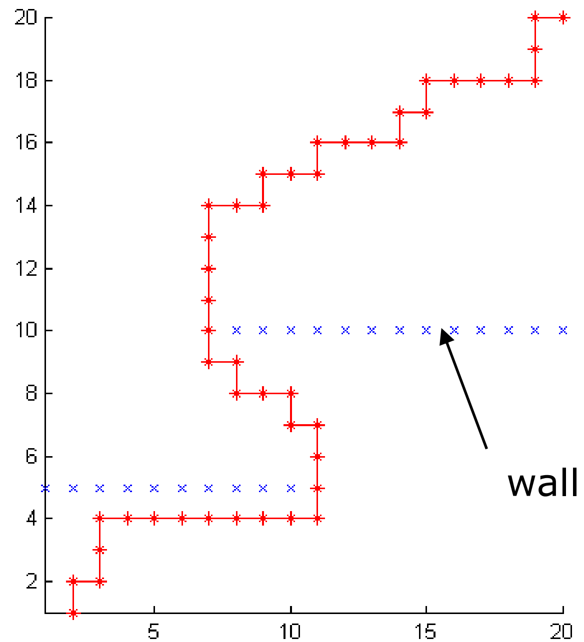
→ Now, action is Up!

Labyrinth Test

- Test5
- Map data: 0 for empty, 1 for wall



Exploration Result



Exploitation

7 HW. Q-Learning

Q-Learning : l11q1.py

- Q-learning has two modes.
- 1. Exploration: random searching for update Q value

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \max_{a'} Q(s',a') \right]$$

- 2. Exploitation: Following Maximum Q value
  - An agent follows Maximum Q value
  - Argmax(Q(s,a)) = a* → Best policy(action)

Q-Learning with Q-value class

```
class QS:
    v    = None;

    def __init__(self):
        self.v   = [0]*maxa

    def Print(self):
        print(self.v)

    def Max(self):
        return max(self.v)

    def ArgMax(self):
        n    = len(self.v)
        m    = -100000;
        mi   = 0;
        for i in range(0,n):
            if (self.v[i]>m):
                mi   = i;
                m    = self.v[i]
        return mi
```

- Q-value also has actions

$$s = (x, y)$$

if max of action, a= 4

$$Q(s,a) = v = [0,0,0,0]$$

$$\max_{a=1\sim4} Q(s,a)$$

$$a^* = \arg\max_{a=4} Q(s,a)$$

$$mi = a^* (0 \sim 3)$$

Exploration

```
# I. Exploration until an agent reaches at terminals
while(True):
    # 1. save state,s as sold
    xo  = int(x)     # avoid reference call
    yo  = int(y)

    # 2. Do random action,a
    a   = randint(4)
    if (a==0):  # west
        x   = x-1;
    elif(a==1):# east
        x   = x+1;
    elif(a==2):# north
        y   = y+1;
    elif(a==3):# south
        y   = y-1

    # 3. check if state is out of area, 0<x<w-1, 0<y<h-1
    if (x<0):
        x   = 0
    if (x>=w):
        x   = w-1;
    if (y<0):
        y   = 0
    if (y>=h):
        y   = h-1
```

$$s' = (x, y)$$

$$s = (xo, yo)$$

```
    # 4. check if state,s is on terminal, and obtain a reward
    r = 0
    stop = False
    if (x==xg and y==yg):
        r   = 1
        stop    = True
    else:
        r   = -1

    # 5. Update Q
    #loop.io.print(xo,yo,x,y)
    Q[xo][yo].v[a]  = alpha*(r + g*Q[x][y].Max()) + (1-alpha)*Q[xo][yo].v[a]

    # 6. Exit
    if (stop==True):
        break;
display()
```

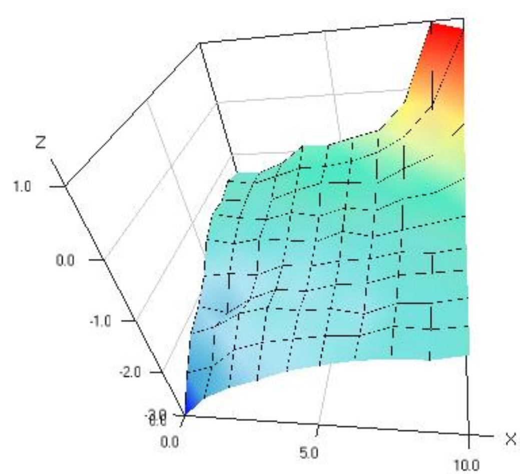
$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \max_{a'} Q(s',a')\right]$$

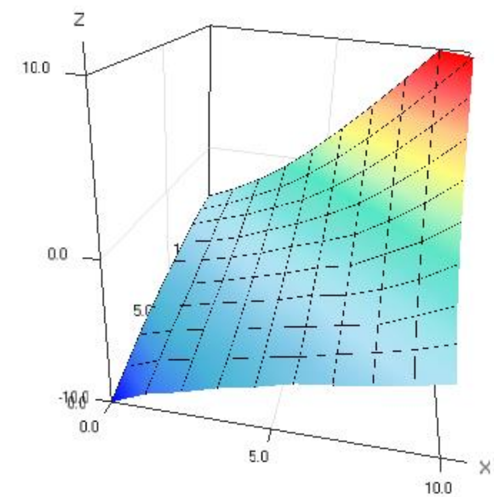
Result of l11q1.py

- Exploration with 100 episodes

```
# Learning for Update Q value
def explore():
    initenv()
    for i in range(0,100):
        episode()
        loop.io.print(i)
```

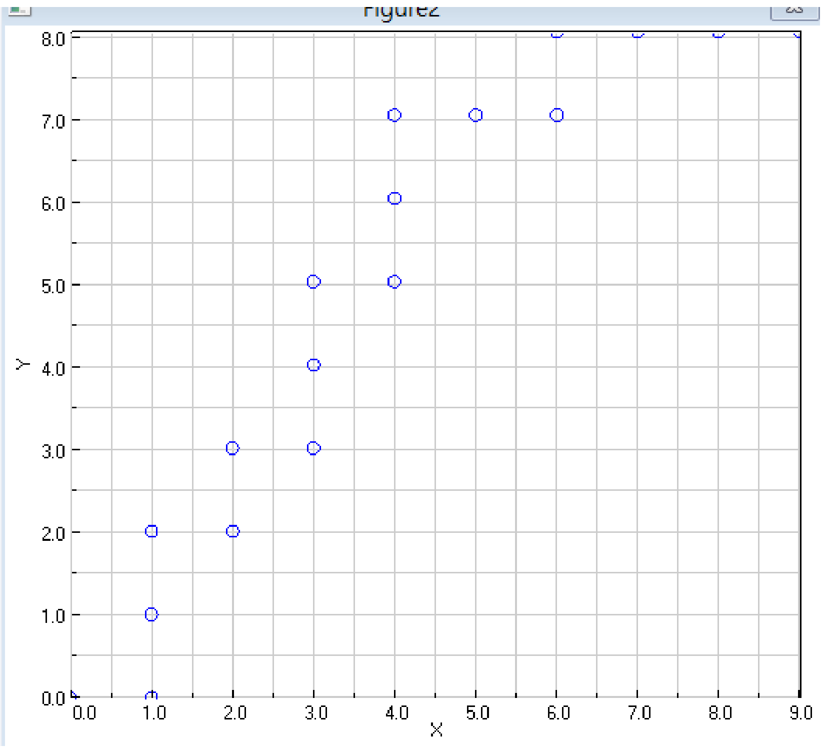
- Draw Qmax value





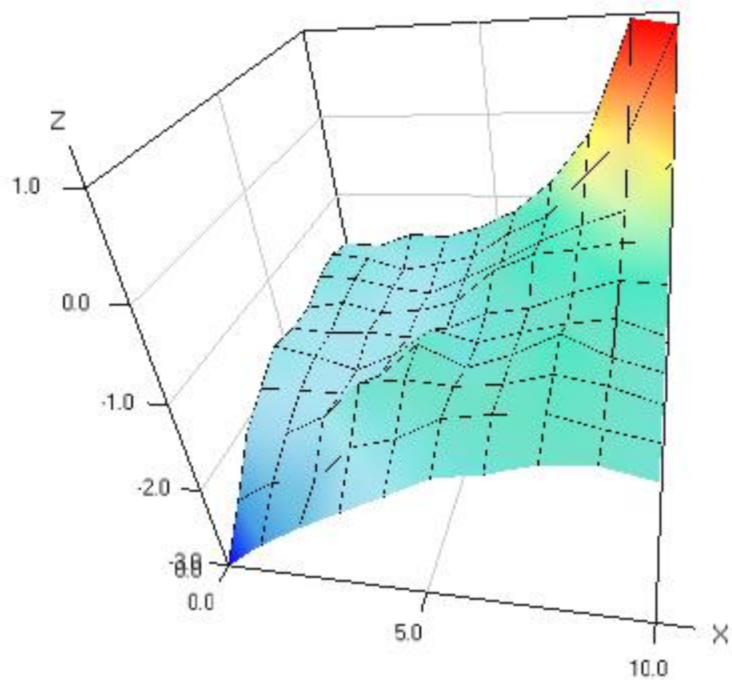
Exploitation with l11q1.py

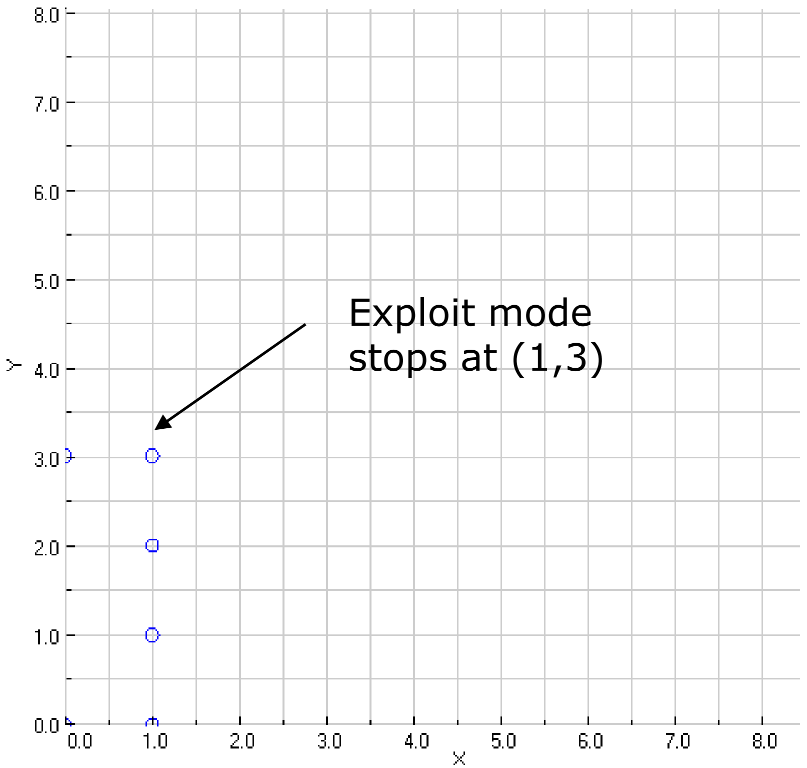
- Start s=si(0,0)
- Repeat:
  - Find a*= argmax(Q(s,a))
  - Do the best action, a*
  - Then, we get S'
  - If S' is terminal , then stops
  - S← S'



Complete your Q-Learning

- With a given l11q1.py, exploration result is like this
  - Exploit mode stops at s=(1,3)





From l11q1.py, Answer the questions

- Prob. 1. Why an agent stops at s=(1,3)
  - Hint) see the Qmax picture. You see local maxima.

- Prob. 2. Complete your Q-learning
  - Exploitation MUST stop at s=(9,9)
  - What code should be changed in ‘l11q1.py’..
  - Hint) If you understand Q learning, It is not so hard..

Prob. 3. Add Noise(l11q2.py)

```
# 2. Do random action,a
a     = randint(4)
acopy     = a;

if  (randint(100)>70):
        a = randint(4)

if  (a==0):   # west
        x     = x-1;
elif(a==1):# east
        x     = x+1;
elif(a==2):# north
        y     = y+1;
elif(a==3):# south
        y     = y-1
a     = acopy
```

Right action

Corrupted action

- Probability of 70%, action works good.
- Otherwise, action is corrupted
- Prob. 3.1: Complete your Q-learning
- Prob. 3.2: What happens on Qmax graph?
- Prob. 3.3: What happens on Exploit Mode?
- Prob. 3.4: if we increase corruption percentage with 70%, what happens?

Prob. 3.5: Explain why RL is good in this hard noisy environment

Prob.4 Add Noise on Exploration and Exploitation. (l11q4.py)

```
# 2. Find the best optimal policy by Argmax
a    = Q[x][y].ArgMax()
acopy = a

if (randint(100)>Noise):
    a    = randint(4):

if (a==0):  # west
    x    = x-1;
elif(a==1):# east
    x    = x+1;
elif(a==2):# north
    y    = y+1;
elif(a==3):# south
    y    = y-1
a=acopy
```

- Prob. 4.1 “Complete your Learning”

  Explain the exploitation results

- Prob. 4.2 If we increase noise,

What happens?

Noises on Exploitation.
→ Noise corrupts the best optimal action.

Prob. 5. with l11q5.py If an agent does not stop at Terminal, What happens at Qmax graph?

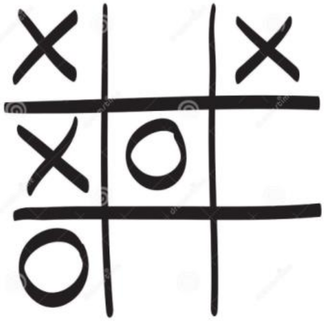
- 1. Add Noises on Action.
- 2. Agent does NOT stop at Terminal.
- 3. After 500000 actions, STOP the episode.

- What happens?
- What is the difference with the result of Prob. 1
  - Hint) See the maximum Qmax value
- **Why the maximum Qmax value is so different?**

8 Tic-Tac-Toe

Tic-Tac-Toe

- How many states is in Tic-Tac-Toe?
- The number of End-Game is 958.
- The First Offence wins game with 626



```
x,x,x,x,o,o,x,o,o,positive
x,x,x,x,o,o,o,x,o,positive
x,x,x,x,o,o,o,o,x,positive
x,x,x,x,o,o,o,b,b,positive
x,x,x,x,o,o,b,o,b,positive
x,x,x,x,o,o,b,b,o,positive
x,x,x,x,o,b,o,o,b,positive
x,x,x,x,o,b,o,b,o,positive
x,x,x,x,o,b,b,o,o,positive
x,x,x,x,b,o,o,o,b,positive
x,x,x,x,b,o,o,b,o,positive
x,x,x,x,b,o,b,o,o,positive
x,x,x,o,x,o,x,o,o,positive
x,x,x,o,x,o,o,x,o,positive
x,x,x,o,x,o,o,o,x,positive
x,x,x,o,x,o,o,b,b,positive
x,x,x,o,x,o,b,o,b,positive
x,x,x,o,x,o,b,b,o,positive
x,x,x,o,x,b,o,o,b,positive
```

First offence = 626/958
Second offence = 332/958

```
x,x,o,x,x,o,o,b,o,negative
x,x,o,x,x,o,b,o,o,negative
x,x,o,x,x,b,o,o,o,negative
x,x,o,x,o,x,o,o,b,negative
x,x,o,x,o,x,o,b,o,negative
x,x,o,x,o,o,o,x,b,negative
x,x,o,x,o,o,o,b,x,negative
x,x,o,x,o,o,b,x,o,negative
x,x,o,x,o,b,o,x,o,negative
x,x,o,x,o,b,o,o,x,negative
```

Tic-Tac-Toe in Q-learning

0	1	2
3	4	5
6	7	8

- State
  - S=[0,0,0,0,0,0,0,0,0]
- RL agent takes 'o=2' and Human does 'x=1', and blank is 0
- Q(s,a)
  - Possible actions are also 0~8
- Example)
  - 1. s=[0,0,0,0,0,0,0,0,0]
  - 2. RL does action=4
  - 3. then s* =[0,0,0,0,2,0,0,0,0]
  - 4. Human does action =0 ➔ Environmental changes
  - 5. finally, s' = [1,0,0,0,2,0,0,0,0]

How we determine Reward?

- If RL(o) wins a game, then obtains reward, r = 1
- If RL(o) loses a game, then obtains reward, r= -1
- Otherwise, r=0

- How it works?
  - Agent attempts to WIN a game,
  - No defense..

- If RL wins a game, r = 1
- If RL loses a game, r= -10

How to determine Q Space?

- Q space is very complex and high dimensions
- Every turns Q space is added
  - Check if there is same Q?
    - Update Q
  - Otherwise,
    - Create a new Q

```
def AddState(s,a):
    global Q,o,x
    n = len(Q)

    for i in range(0,n):
        if (Q[i].a!=a):
            continue;
        t       = Q[i]

        bsame   = True
        for j in range(0,9):
            if (s[j]!=t.s[j]):
                bsame   = False
                break;
        if (bsame==True):
            return t

    # add new Q(s,a)
    newq    = QS()
    newq.s  = s.copy()
    newq.a  = a
    Q.append(newq)
    return Q[len(Q)-1]
```

```
class QS:
    def __init__(self):
        self.a  = 0;
        self.s  = [0,0,0,0,0,0,0,0,0];
        self.v  = 0;
```

See Example
ex/ml/l11ttt

- All learned Q space has number of 8618
- Learning by explore()
- In each step, you can check which action is the best

$$s = [s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, s_{31}, s_{32}, s_{33}]$$
$$s_{ij} = 0 \; for \;\; empty$$
$$s_{ij} = 1 \; for \;\; X$$
$$s_{ij} = 2 \; for \;\; O$$

```
ttt.ArgMaxQ([0,0,0,0,0,0,0,0,0])
4
ttt.ArgMaxQ([1,0,0,0,2,0,0,0,0])
1
ttt.ArgMaxQ([1,2,0,0,2,0,0,1,0])
3
```