LNMIIT

Introduction to Data Science

Markov Decision Process and Reinforcement Learning

IDS

Sakthi Balan M

Some pictures and contents in the following slides are taken from AI course slides of edx. This is for education purpose only.

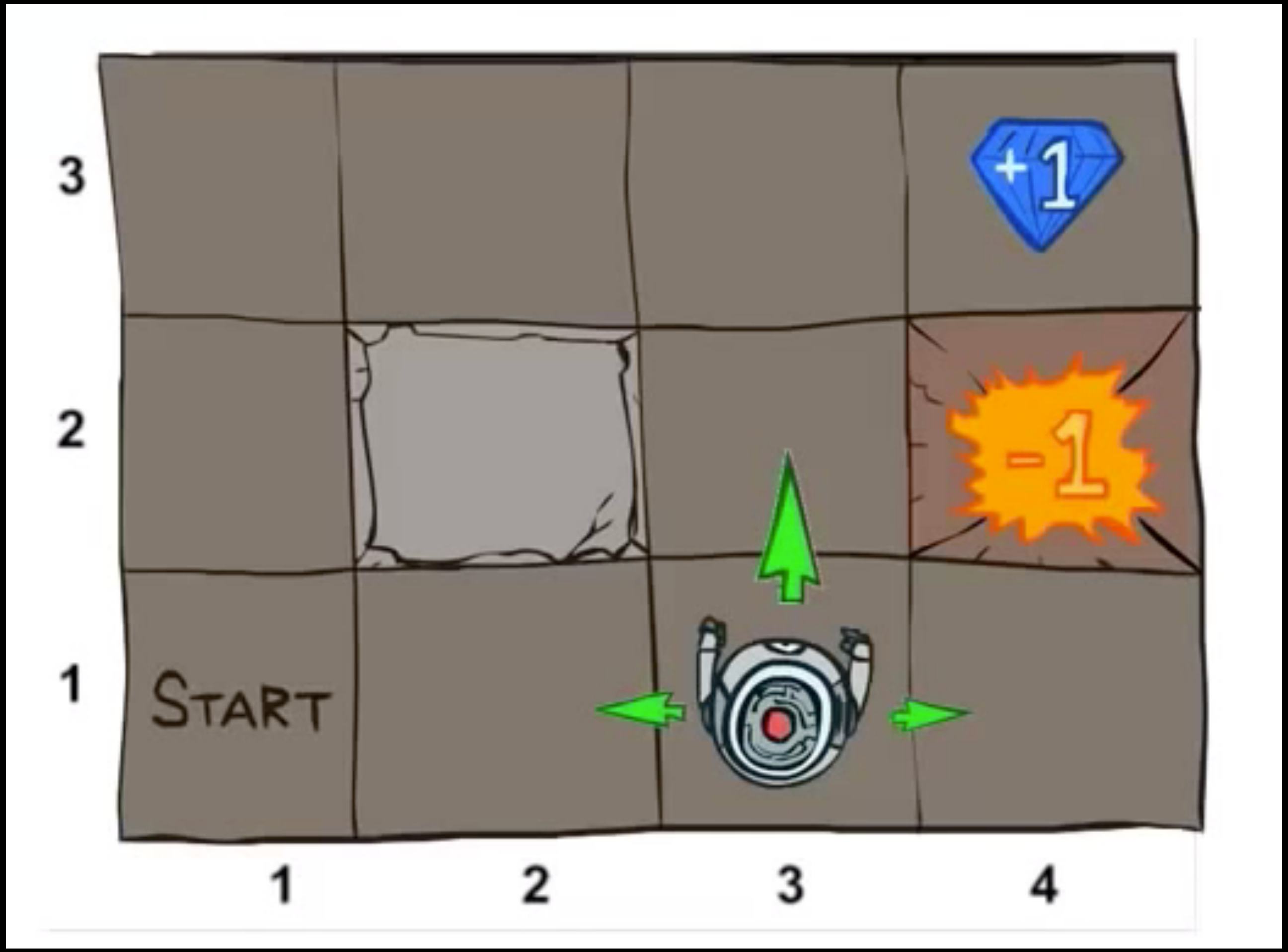
Markov Decision Process

- Andrey Markov came up with this model of a decision making when there is uncertainty in the outcome of the action
- Nondeterministic search problem
- Action need not result in the expected outcome
- Outcomes are probabilisitic in nature

LNMIIT

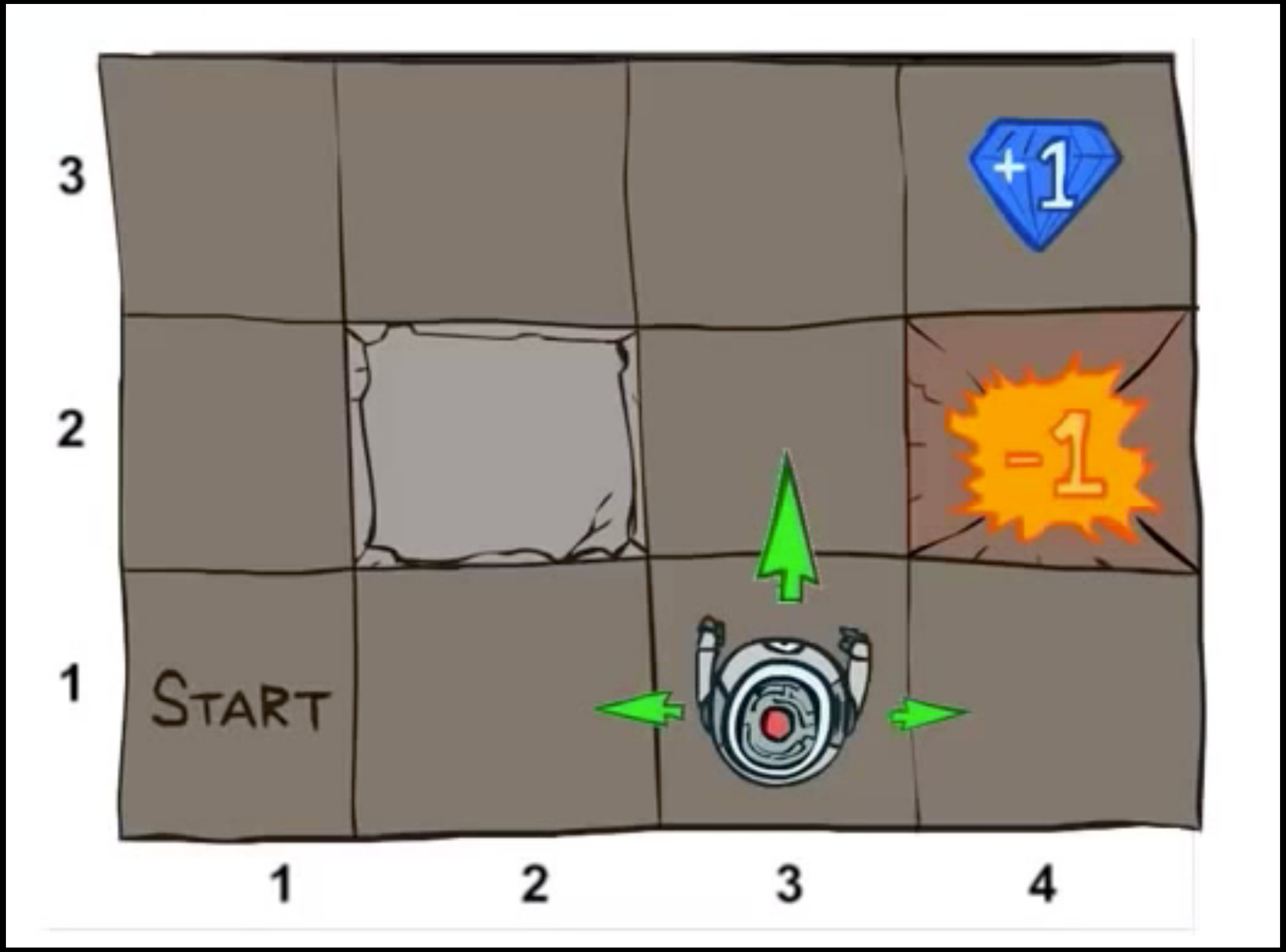
Markov Decision Problem

- Action may be move to North
- But it may result in North, East or West!
- Probability of going to N, W or E we already know
- Rewards associated with the outcomes
    - small living rewards (negative)
    - big reward or penalty
- Goal is to maximise the reward



Markov Decision Problem

- MDP is defined as
  - Set of states s in S
  - Set of actions a in A
  - Transition function T(s,a,s')
  - Reward function R(s,a,s')
  - Start state
  - Terminal state (not necessary)



Markov Decision Problem

- Action outcome depends on the present state only
- Action does not depend on all the previous states only the previous state (not the history)

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots S_0 = s_0)$$

$$=$$

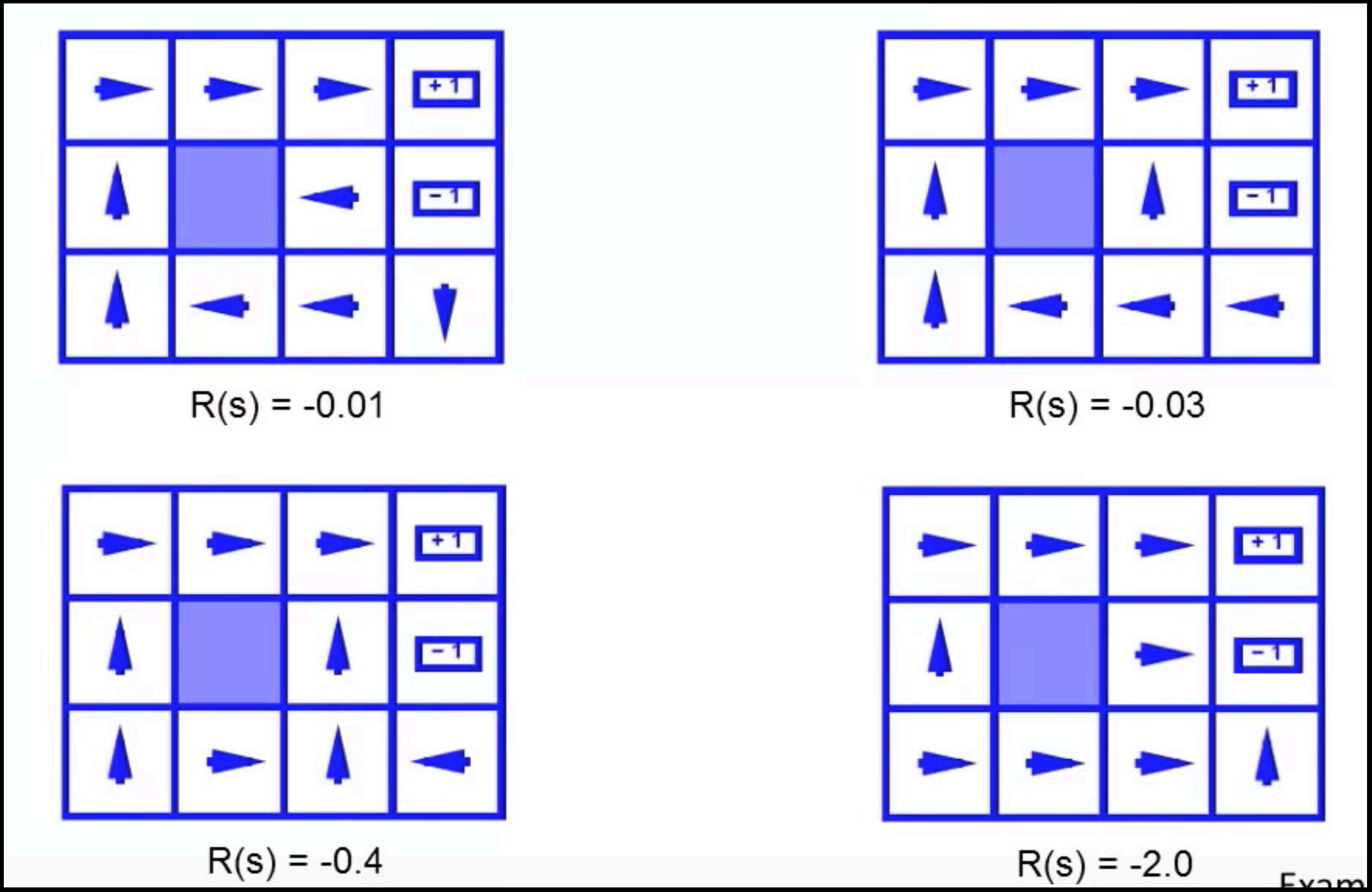
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Markov Decision Problem

- Our goal is to maximise the reward this implies we need to find the movement so that it maximises the reward, i.e., we need to find the optimum action at each state!
- Finding the optimum action to be taken at each state is called as the optimum policy: $\prod^*: S \rightarrow A$

Markov Decision Problem

Optimum Policy



Taken from Stuart Russel's Example

LNMIIT

Example: Racing Car

- States: Cool, Warm, Overheated
- Action: Slow, Fast
- Going faster gets more reward - double reward
- Cool, Warm and Overheated are the states of this model
- Once we commit an action, either Slow or Fast then the pairs (Cool, Slow), (Cool, Fast), (Warm, Slow) and so on are called as q-states.
- When we are in a q-state we do not know what will be the outcome though because we have just committed not performed it.
- $T(s,a,s') = p(s'|s,a)$
- Based on state and a-state we can draw a tree structure

LNMIIT

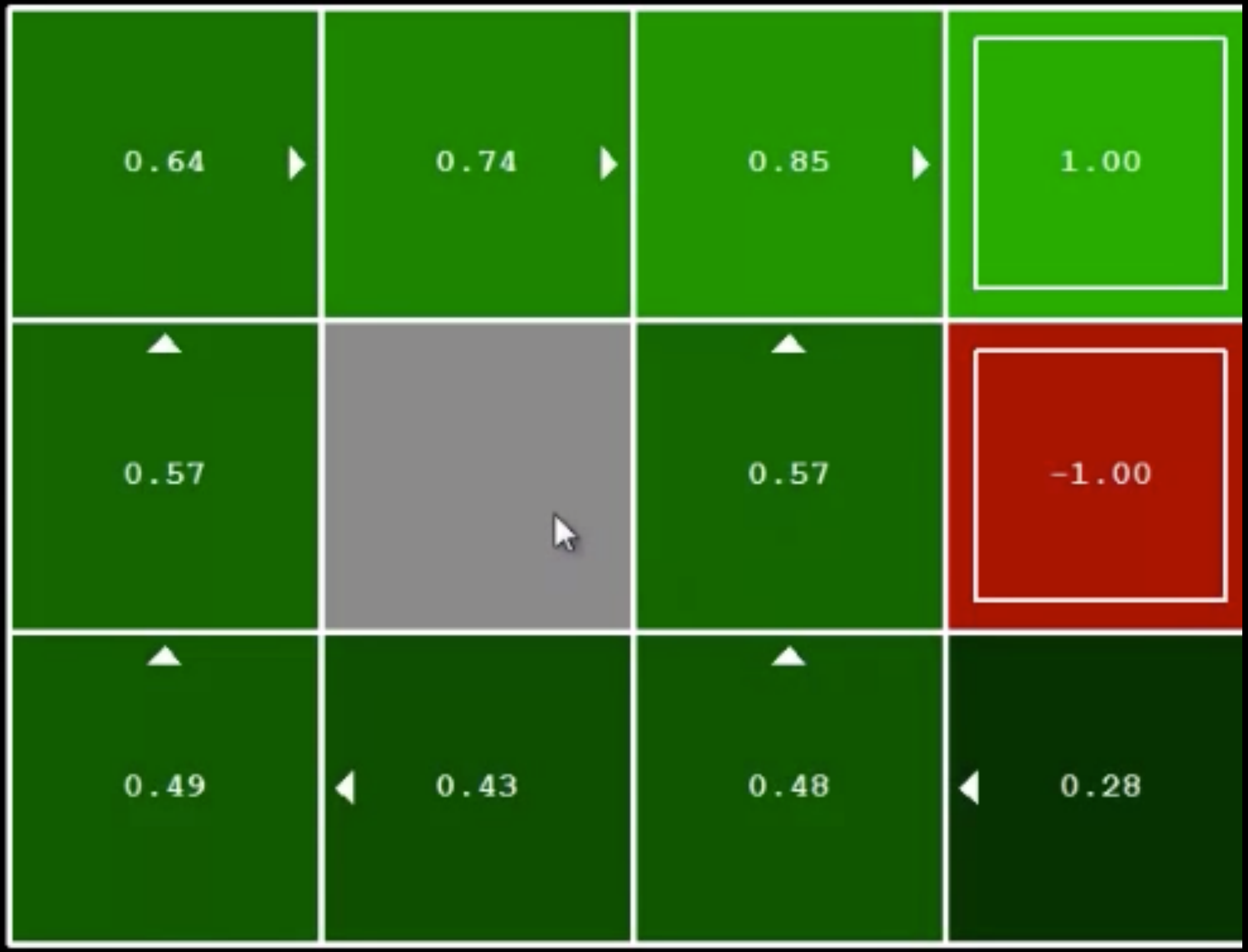
Utilities

- Maximise the reward
- Logic to go for reward immediately than waiting for it in future
- Impart a discouting factor where the rewards get discounted by a factor γ which we can define
- Example: Do you want to go for (1,0,0) or (0,0,1) if we have discounted factor γ then $(1,0,0) > (0,0,1)$

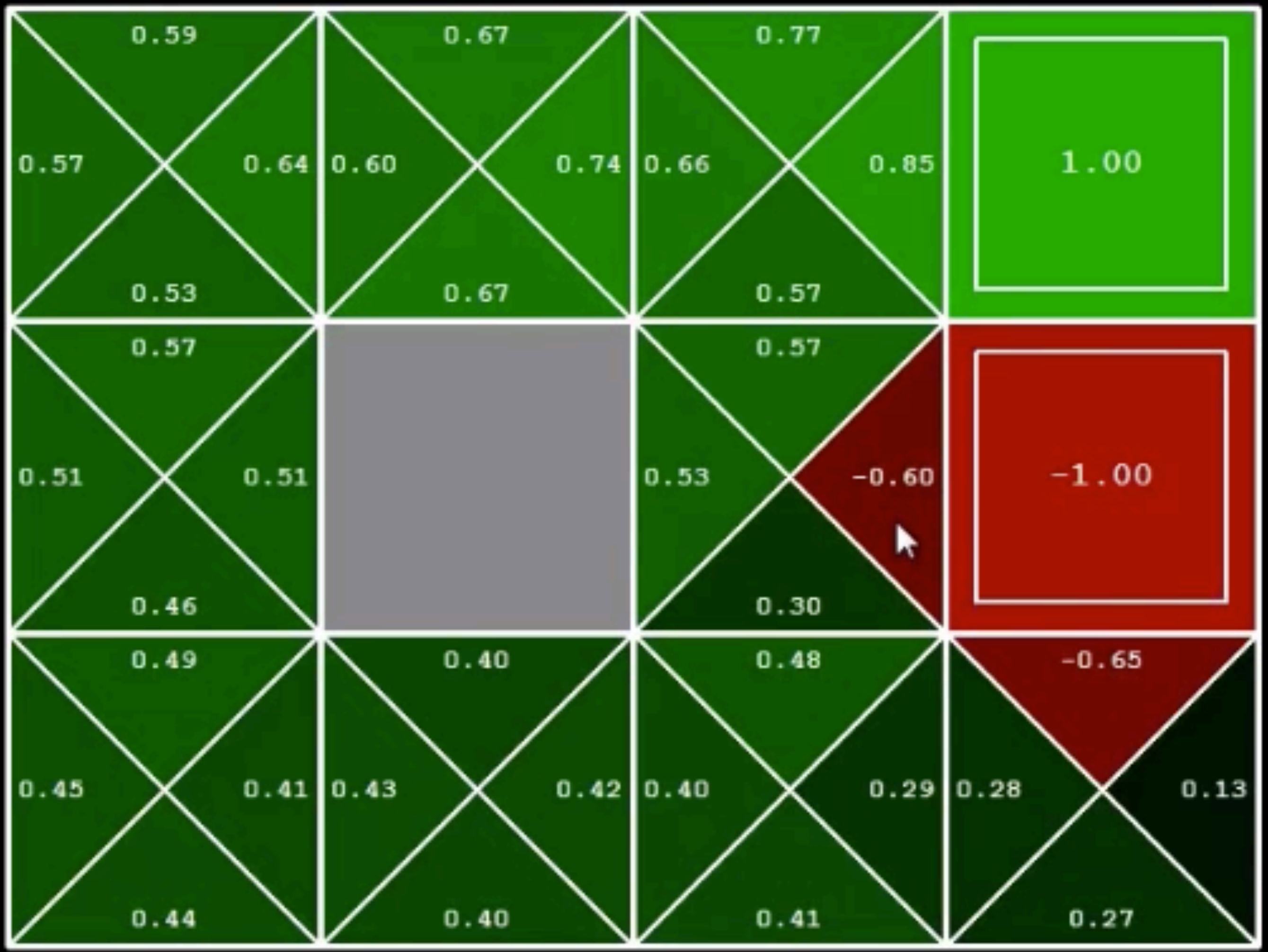
Utilities

- Utility of a state s - V*(s)
- Value of a q-state - Q*(s,a)
- Optimal policy - ∏*(s) - optimal action from s

Utilities

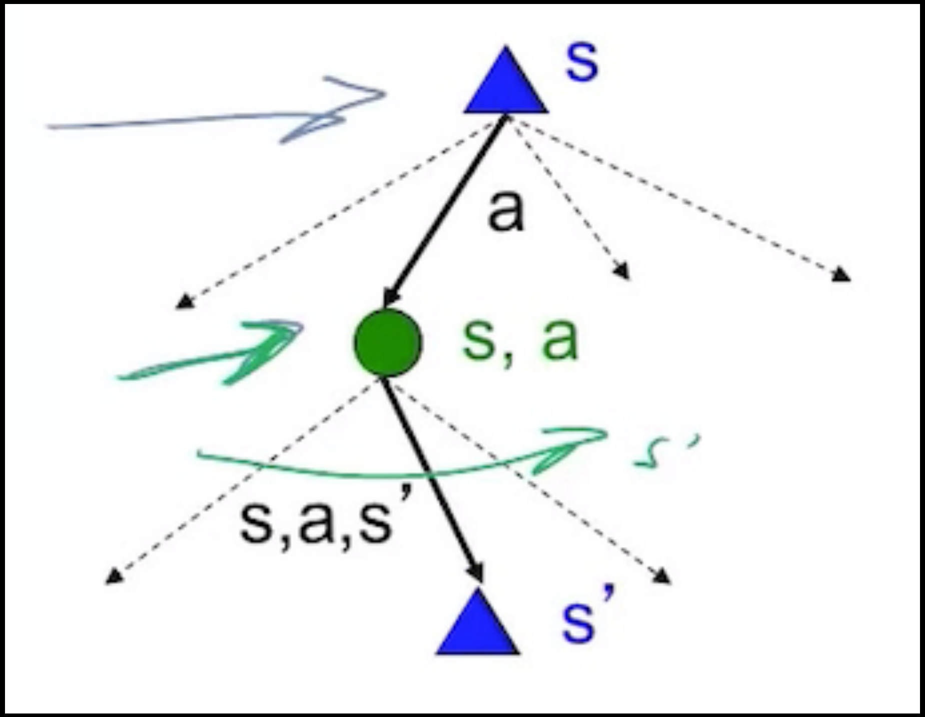


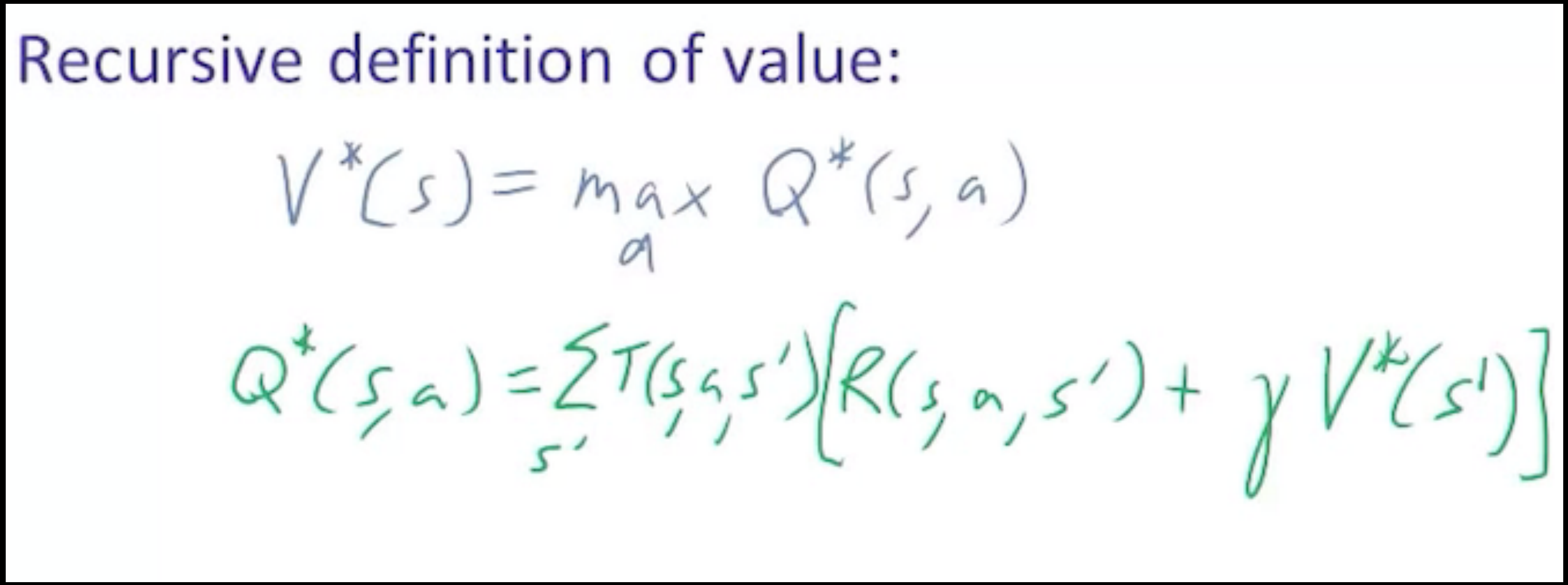
utility of states



utility of q-states

Utilities



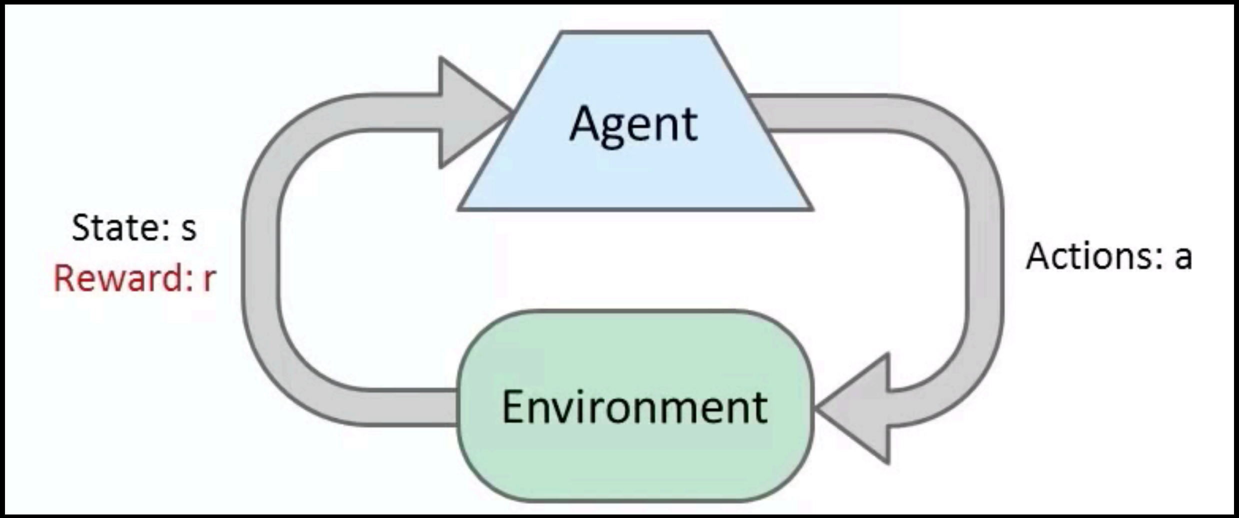


Recursive definition of value:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

Reinforcement Learning



- Behaviorial Psychology - reinforcement learning is studied heavily for a long time
- Receive feedback via reward points
- Utility function is defined interns of reward points
- Goal is to maximise the reward points
- Learning is completely based on the observed outcomes!

Reinforcement Learning

- MDP is used here
  - Set of states s in S
  - Set of actions a in A
  - Transition function T(s,a,s')
  - Reward function R(s,a,s')
  - Start state
  - Terminal state (not necessary)

- Main difference here are:
  - Transition function is not known
  - Reward function is not known

Reinforcement Learning

- Note that RL does not have a complete model of the environment and the reward functions. It assumes no prior knowledge of environment and the reward functions.

- RL uses the observed rewards and penalties to learn an optimal policy to face the environment

- RL is as simple as saying how an agent learns to behave itself when put in an environment

Reinforcement Learning

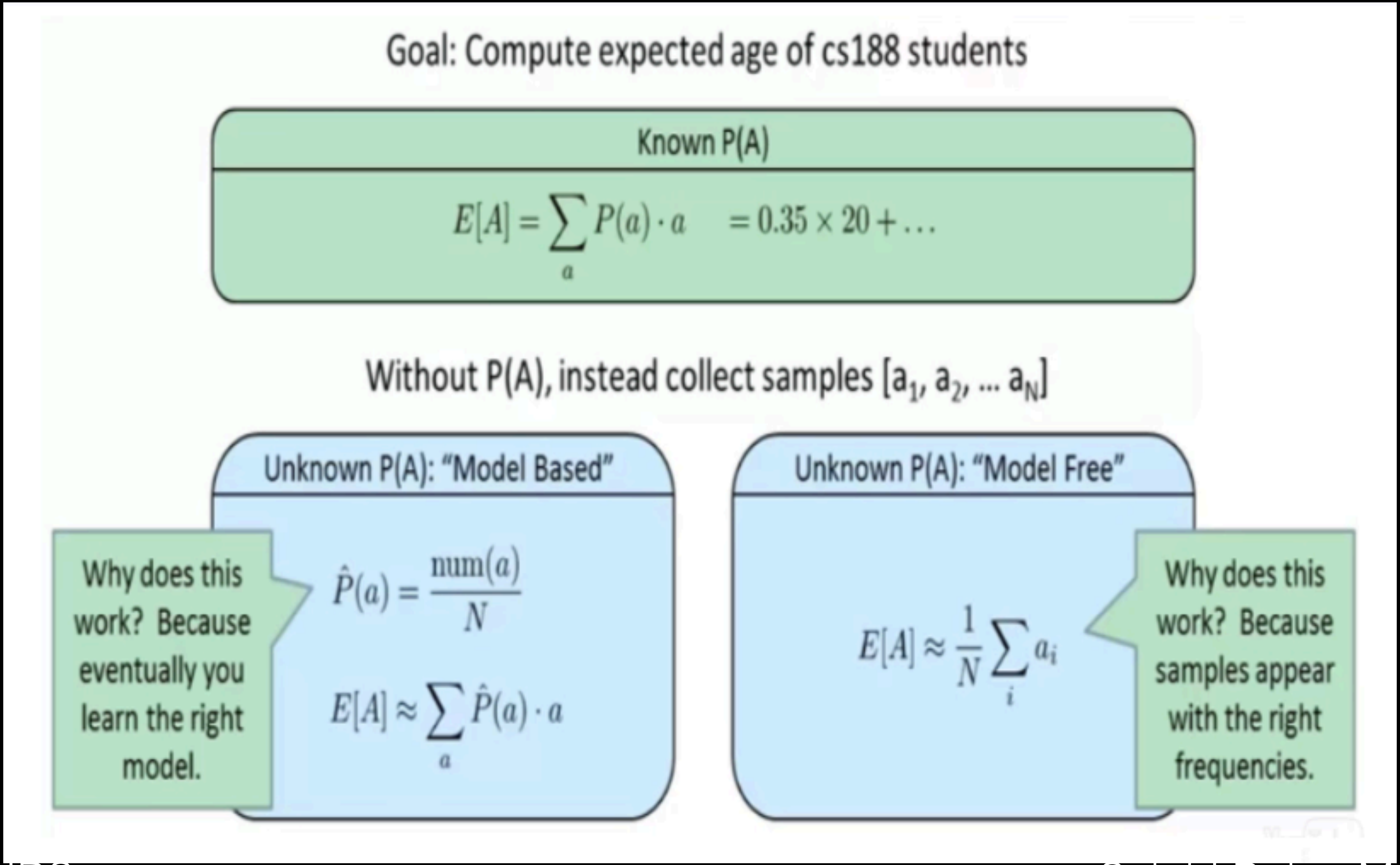
- Passive Learning: Agent’s policy is fixed. The task is to learn the utilities for the q-states
- Active Learning: Agent has to learn the policy in order to maximise the utility function. This done through exploration.

Reinforcement Learning

- Model-Based Learning
- Model-Free Learning

LNMIIT

Model Free Vs Model Based - Analogy



Goal: Compute expected age of cs188 students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \quad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples $[a_1, a_2, \ldots a_N]$

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

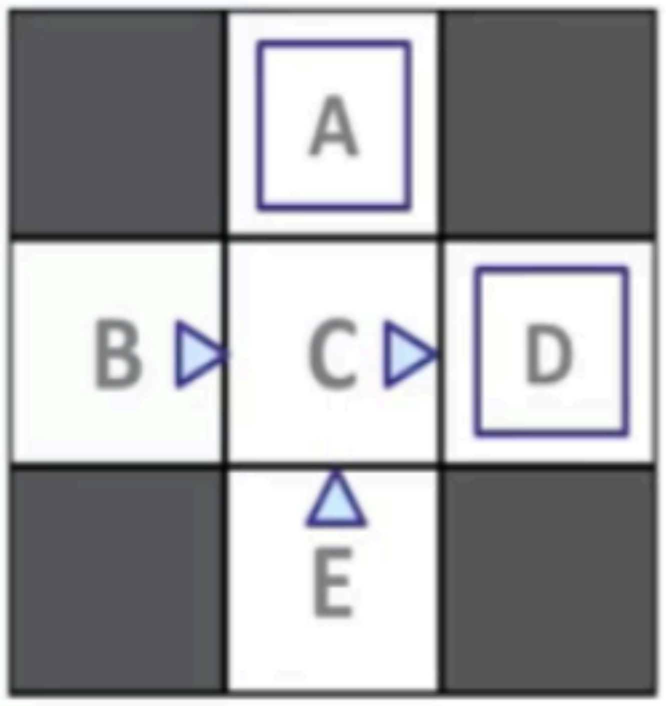
$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

LNMIIT

Passive Learning - Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

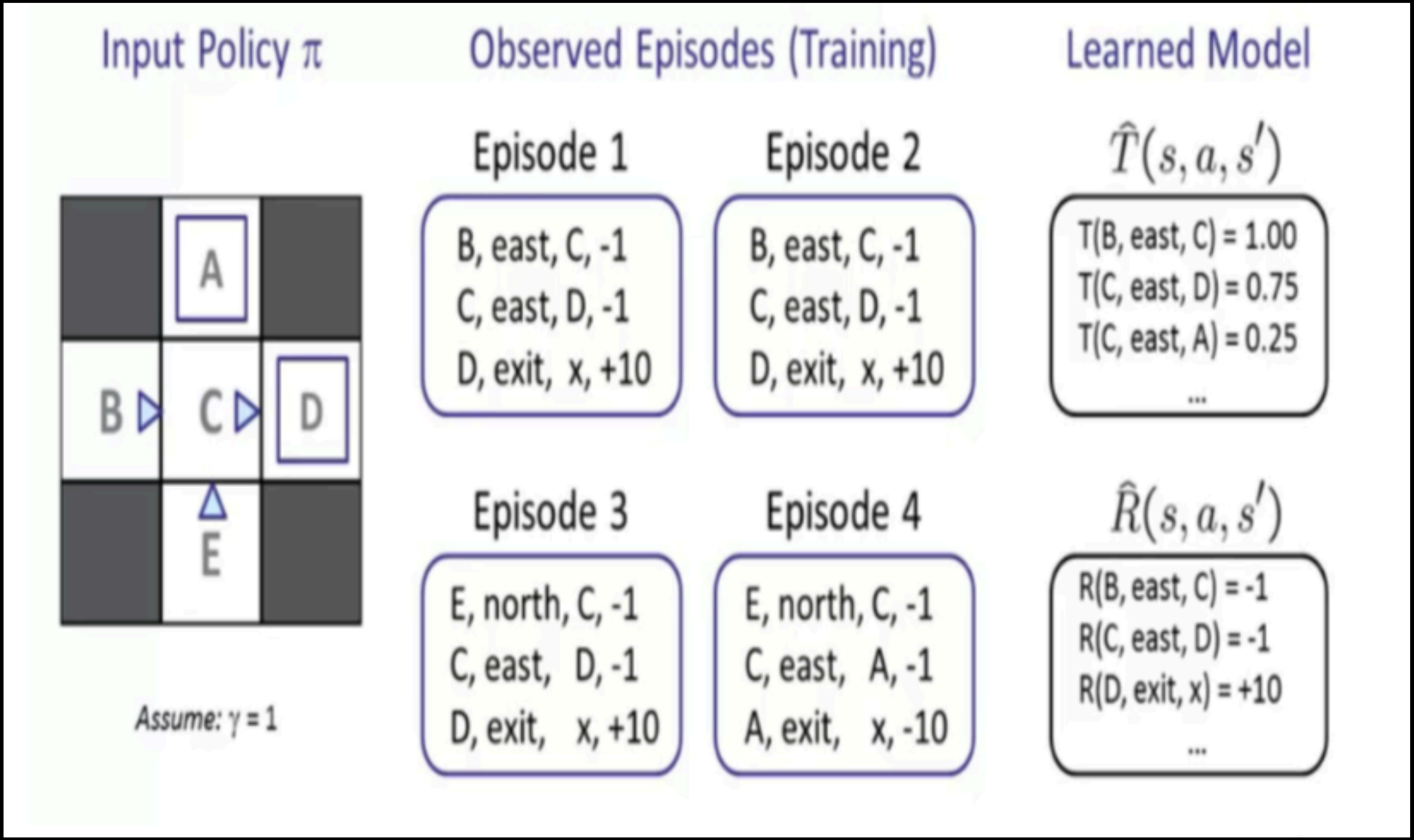
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

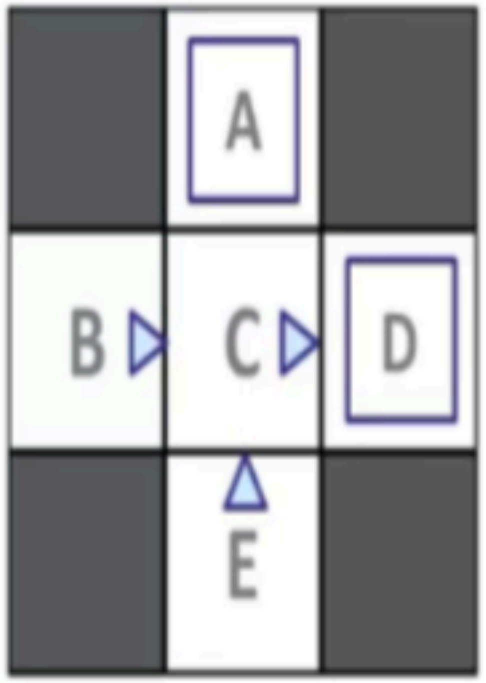
Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Passive Learning - Direct Evaluation



Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s,a,s')$

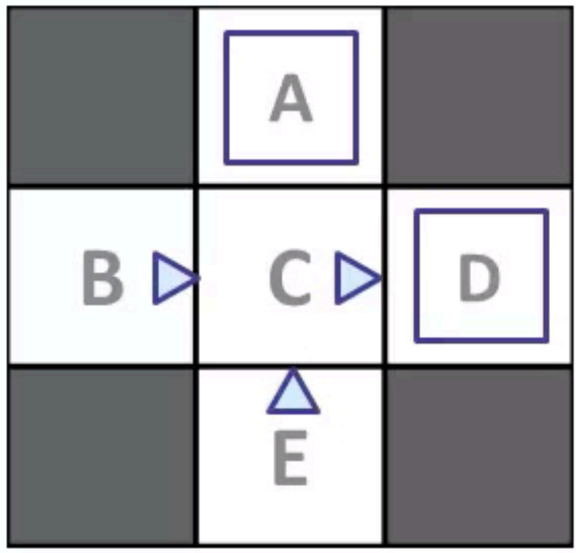
T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s,a,s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

Passive Learning - Direct Evaluation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

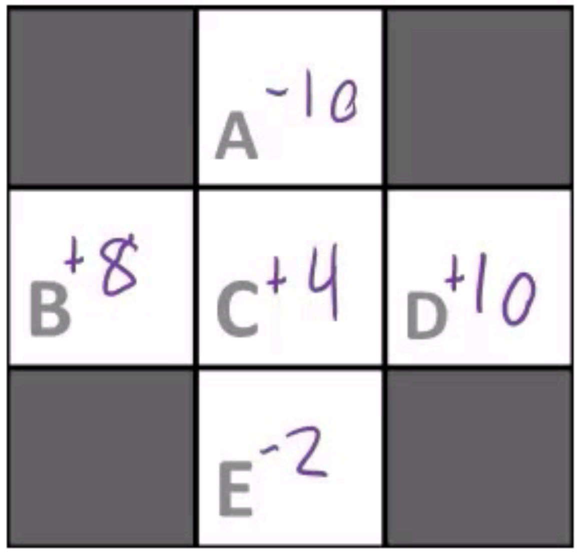
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

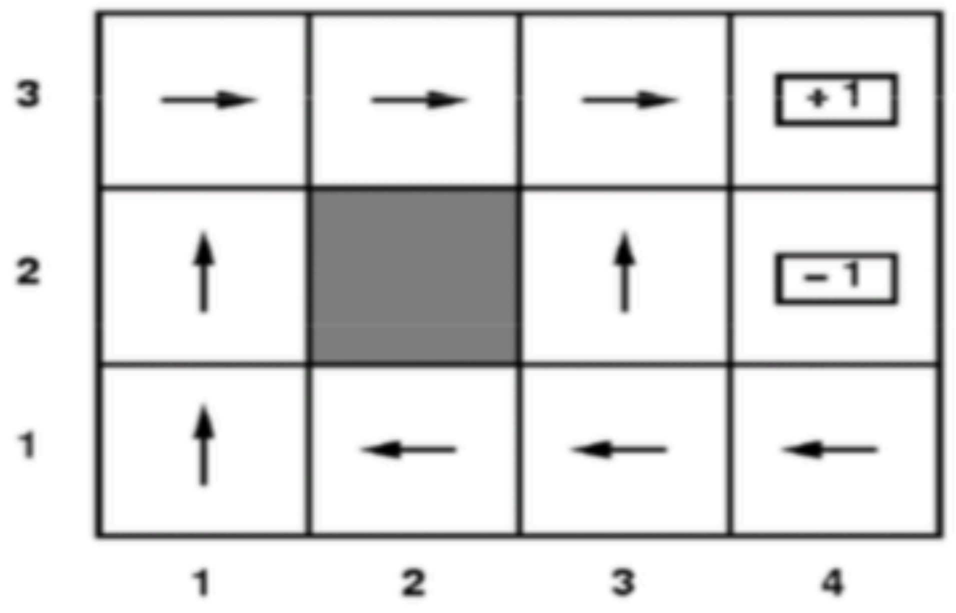
Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Output Values



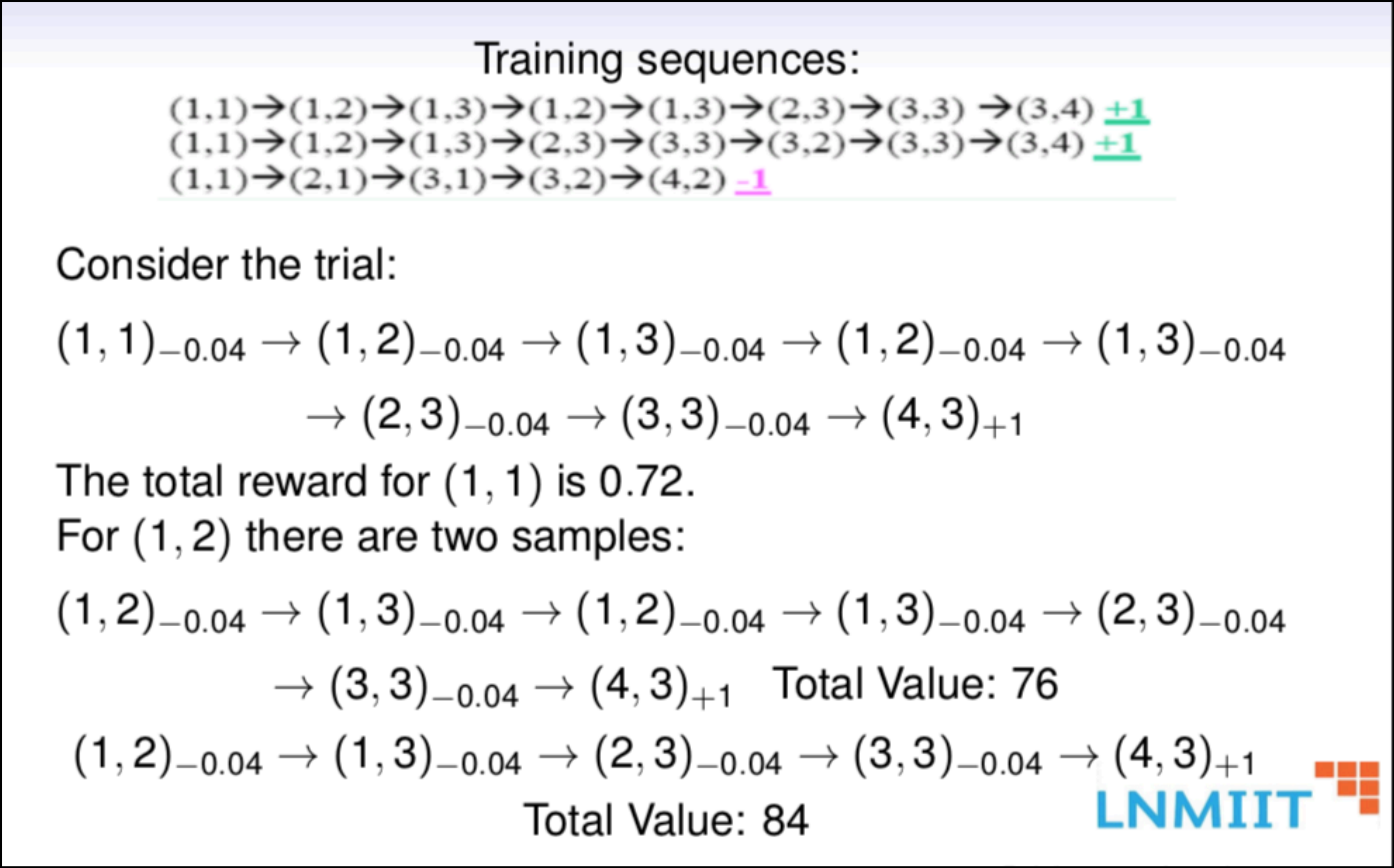
Passive Learning



Training sequences:

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

Passive Learning



Training sequences:

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

Consider the trial:

$$(1,1)_{-0.04} \to (1,2)_{-0.04} \to (1,3)_{-0.04} \to (1,2)_{-0.04} \to (1,3)_{-0.04} \to (2,3)_{-0.04} \to (3,3)_{-0.04} \to (4,3)_{+1}$$

The total reward for (1, 1) is 0.72.
For (1, 2) there are two samples:

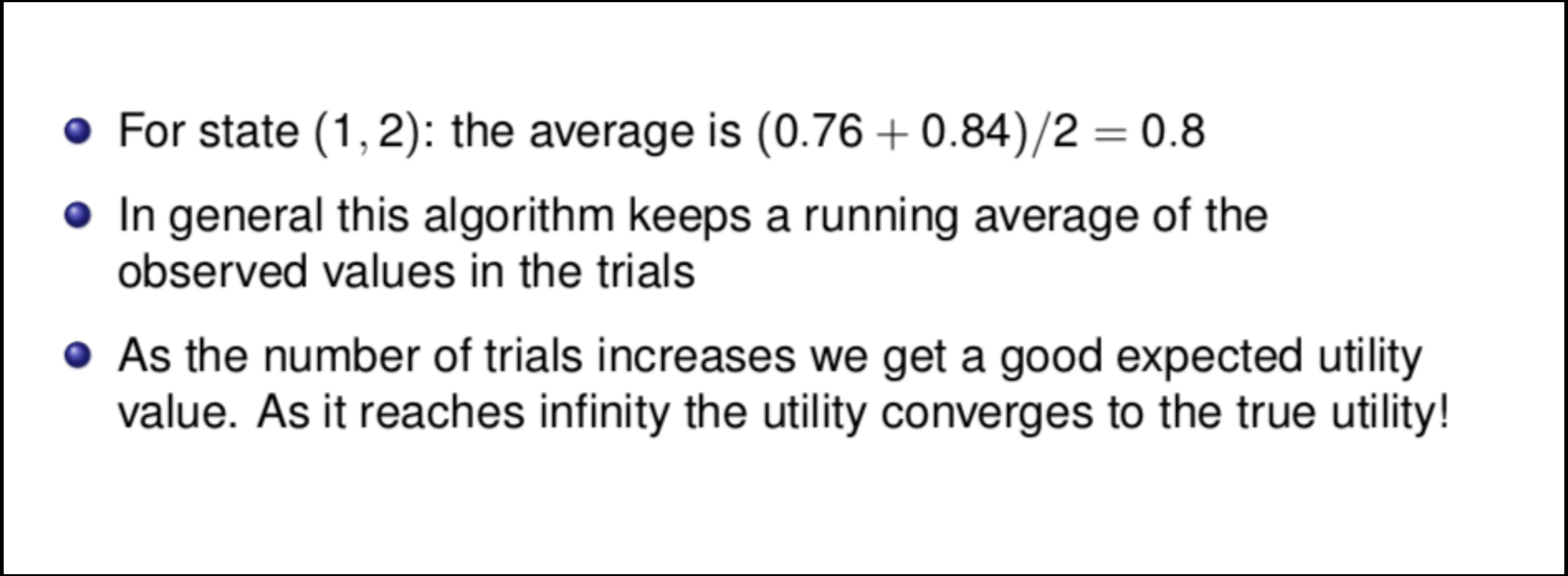
$$(1,2)_{-0.04} \to (1,3)_{-0.04} \to (1,2)_{-0.04} \to (1,3)_{-0.04} \to (2,3)_{-0.04} \to (3,3)_{-0.04} \to (4,3)_{+1}$$ Total Value: 76

$$(1,2)_{-0.04} \to (1,3)_{-0.04} \to (2,3)_{-0.04} \to (3,3)_{-0.04} \to (4,3)_{+1}$$

Total Value: 84

LNMIIT

Passive Learning



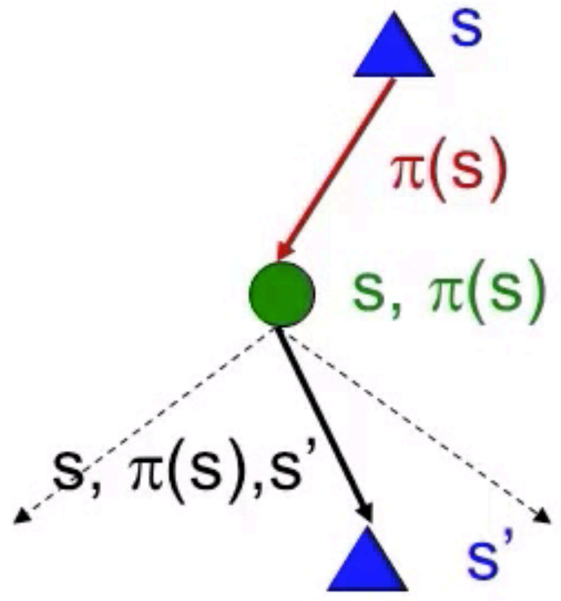
- For state $(1, 2)$: the average is $(0.76 + 0.84)/2 = 0.8$
- In general this algorithm keeps a running average of the observed values in the trials
- As the number of trials increases we get a good expected utility value. As it reaches infinity the utility converges to the true utility!

Temporal Difference Learning

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$



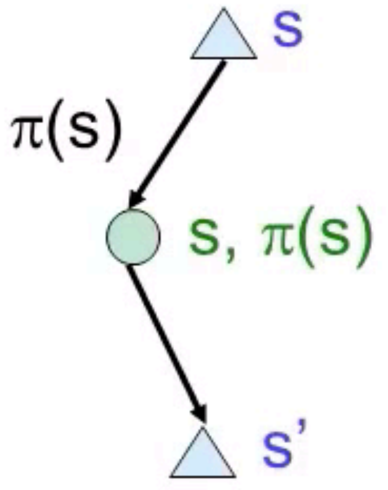
  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!

- Key question: how can we do this update to V without knowing T and R?
  - In other words, how to we take a weighted average without knowing the weights?

LNMIIT

Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

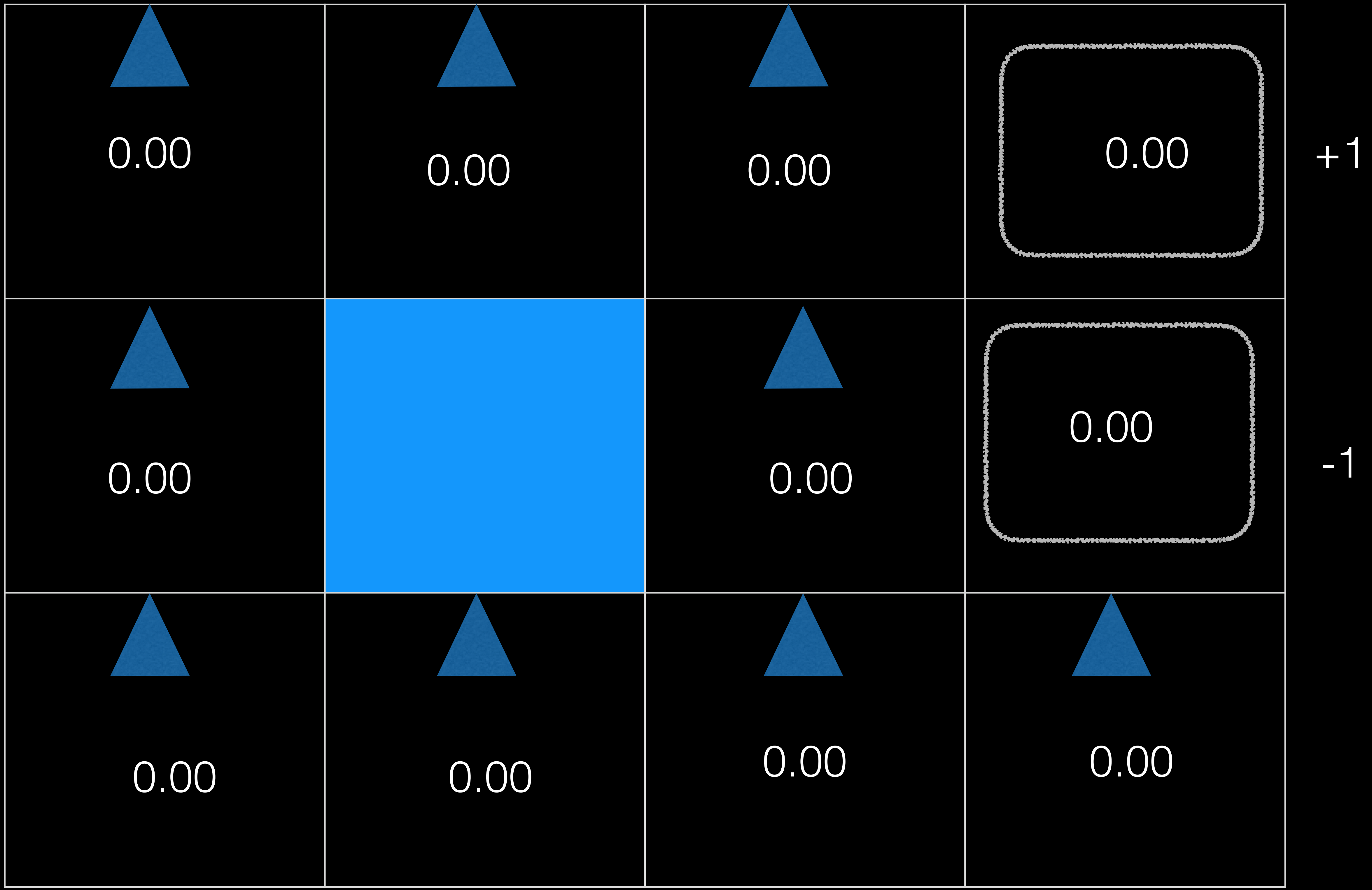


Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

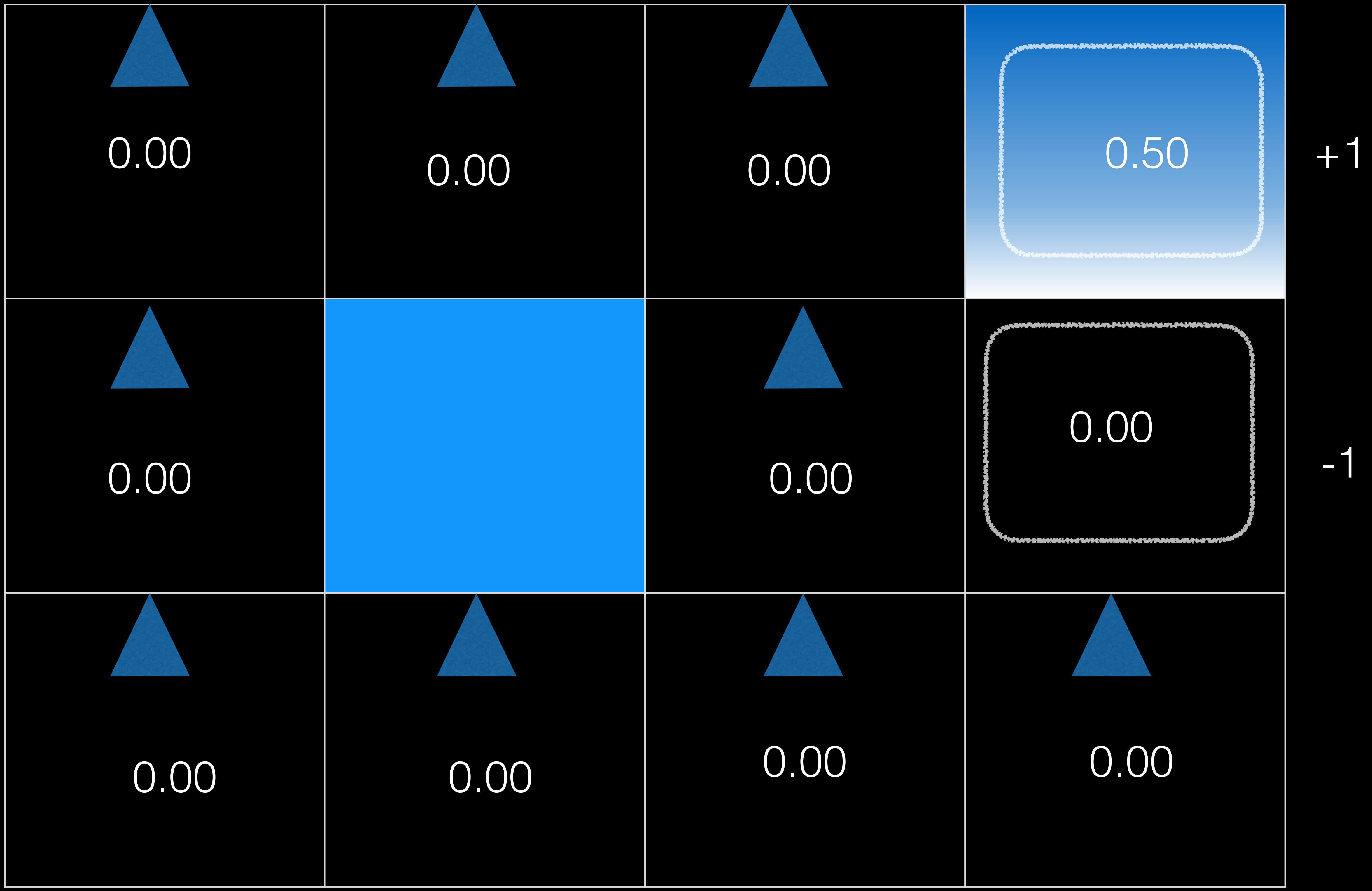
Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

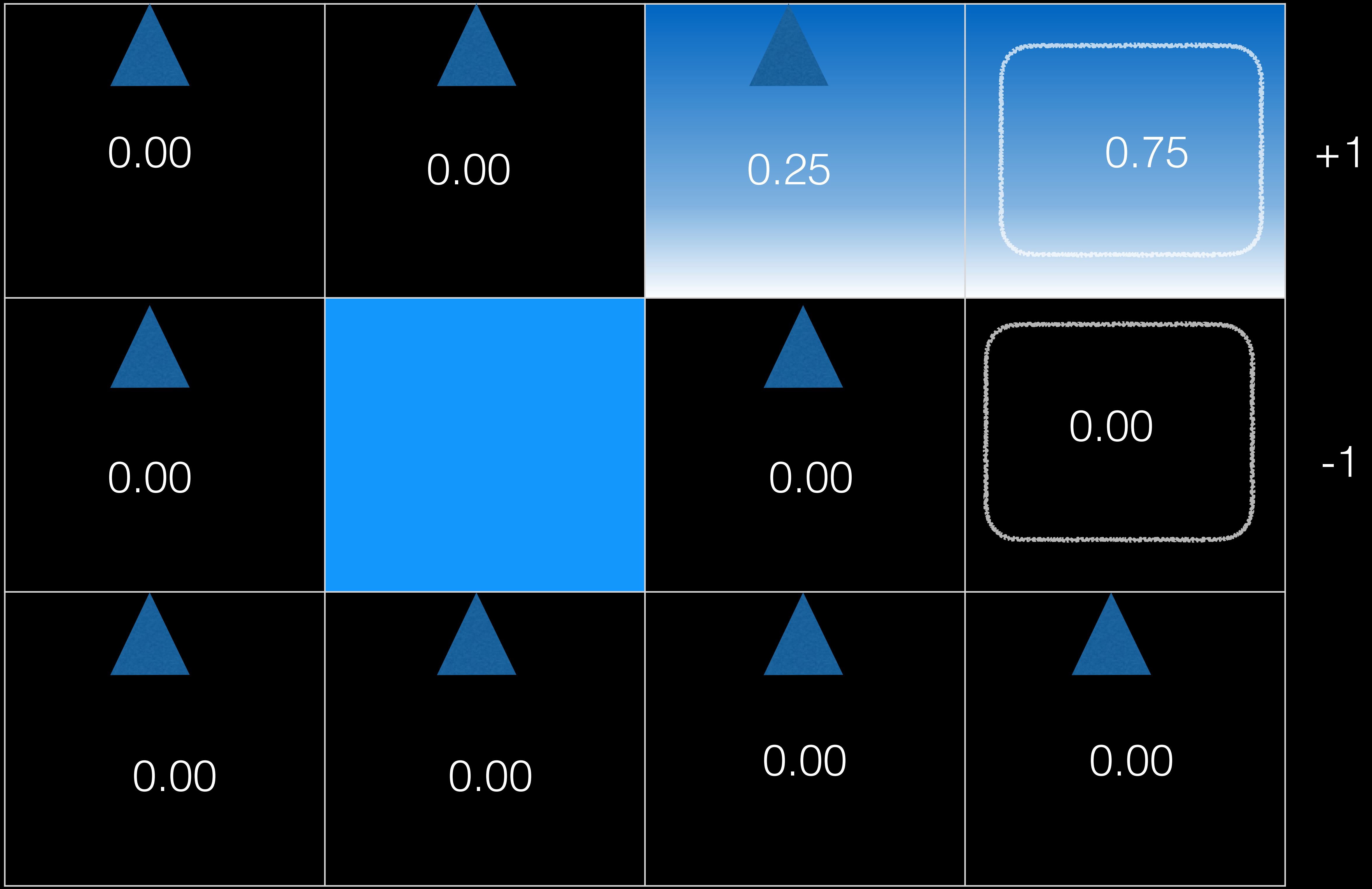
Temporal Difference Learning



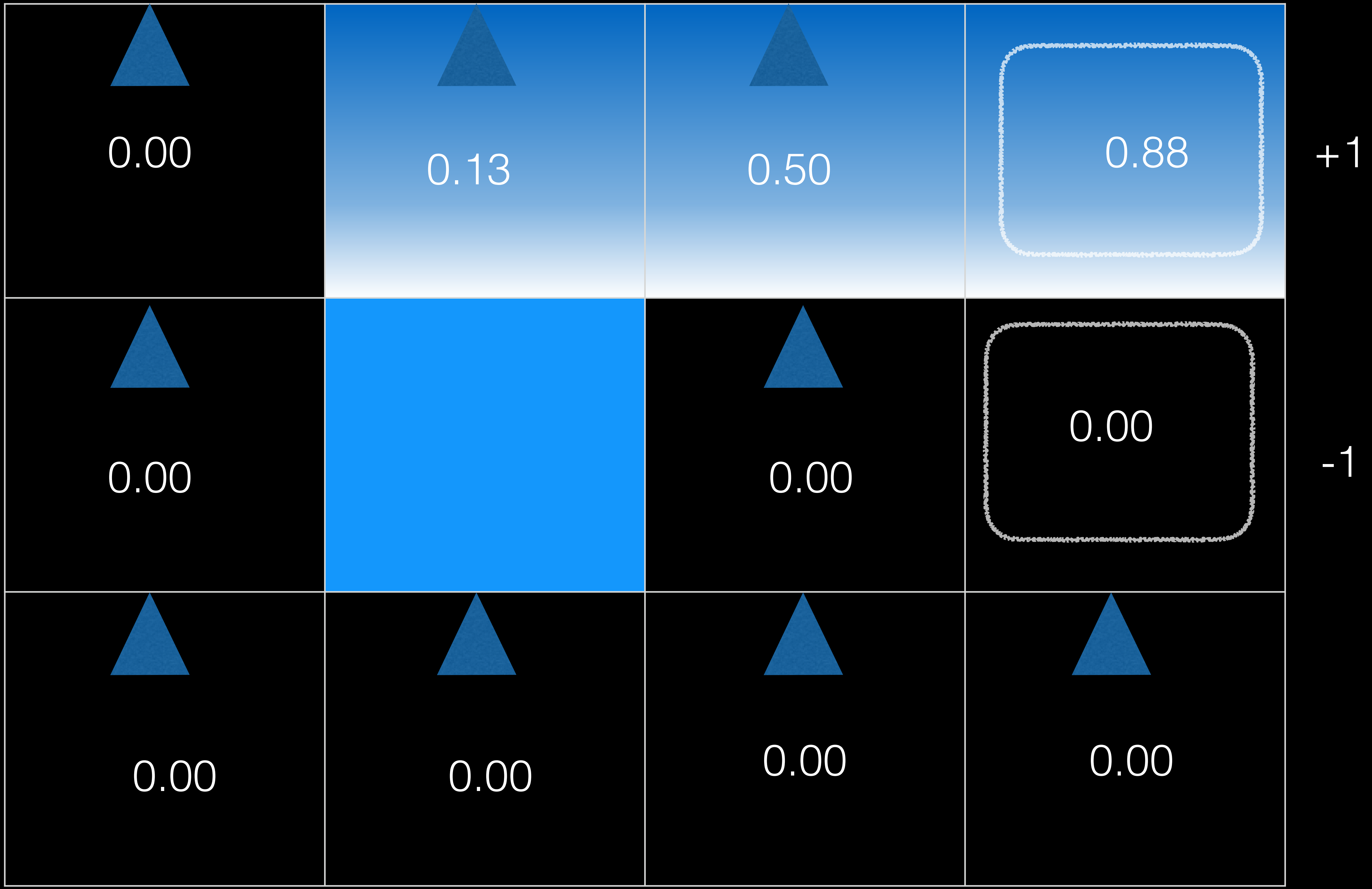
Temporal Difference Learning



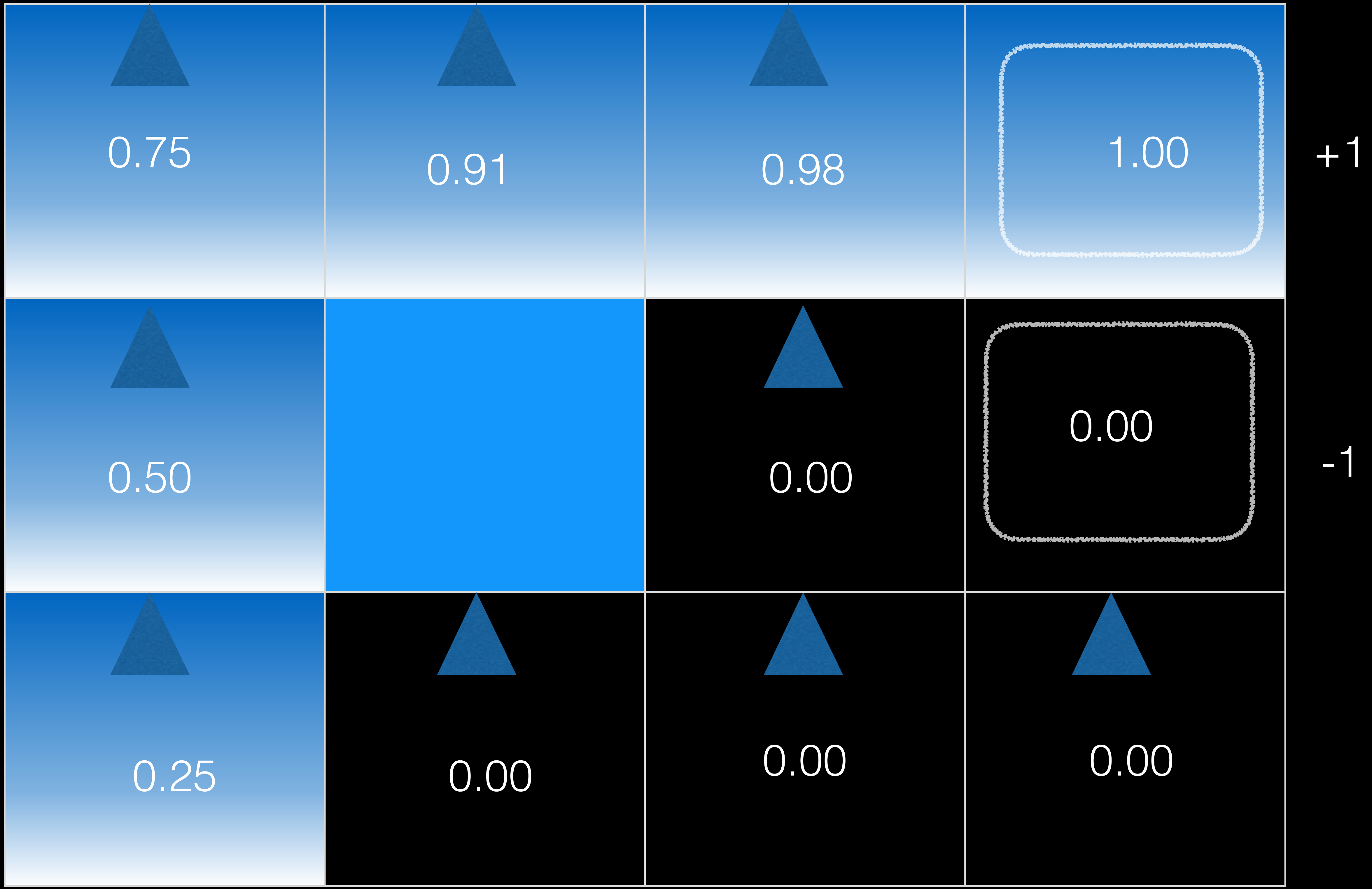
Temporal Difference Learning



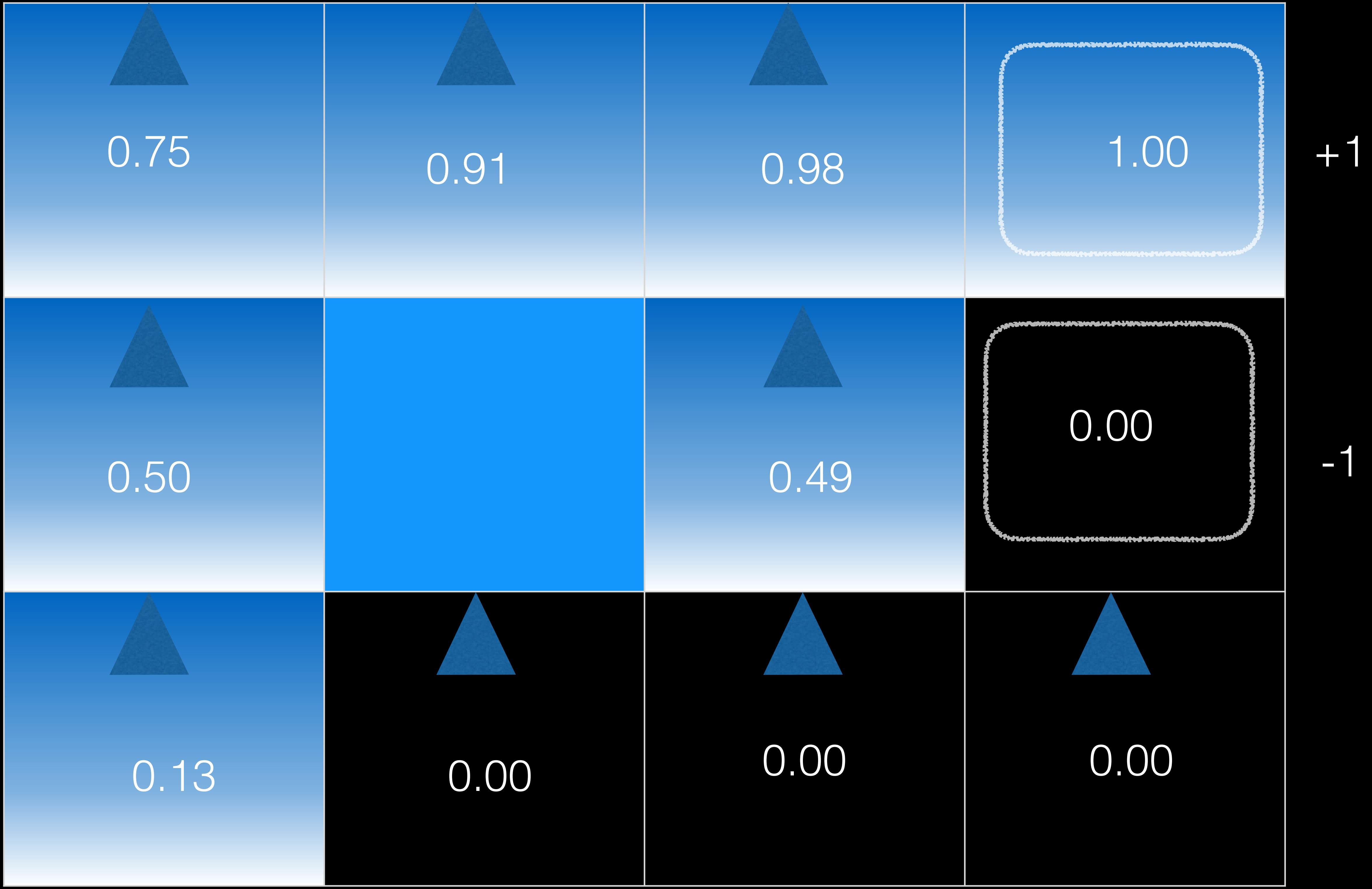
Temporal Difference Learning



Temporal Difference Learning (After few more steps)



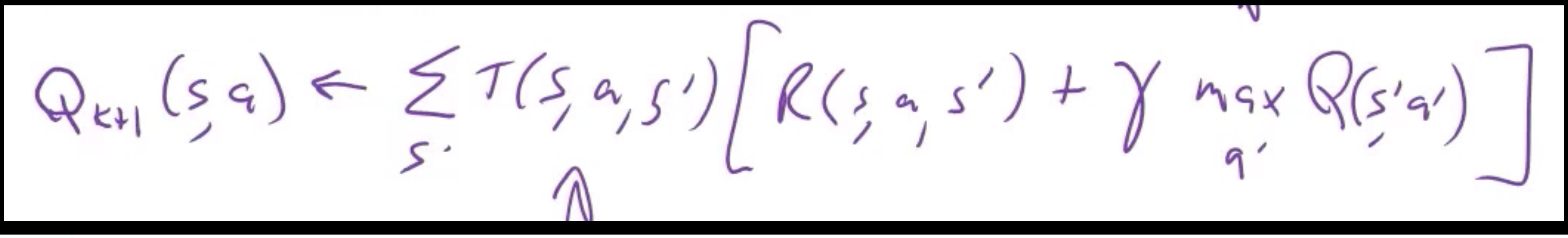
Temporal Difference Learning



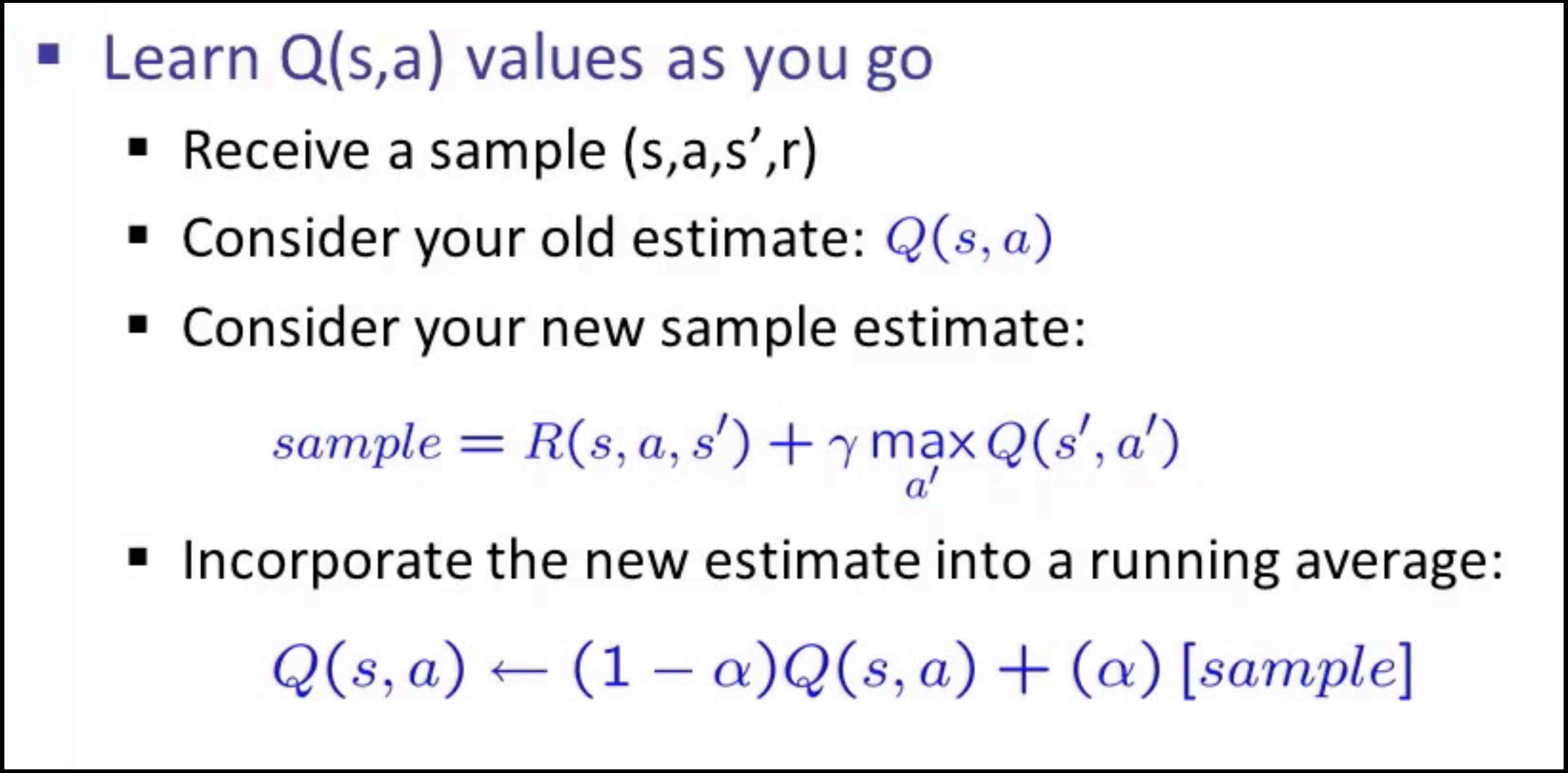
Active Reinforcement Learning

- Agents choose the actions now
- Goal is to learn an optimal policy
- Don't know about transitions or reward points
- Agents makes choices and then learns
- Tradeoff: exploration or exploitation
- Not offline - real-time actions and learning!

Active Reinforcement Learning - Q-Value Iteration



$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$



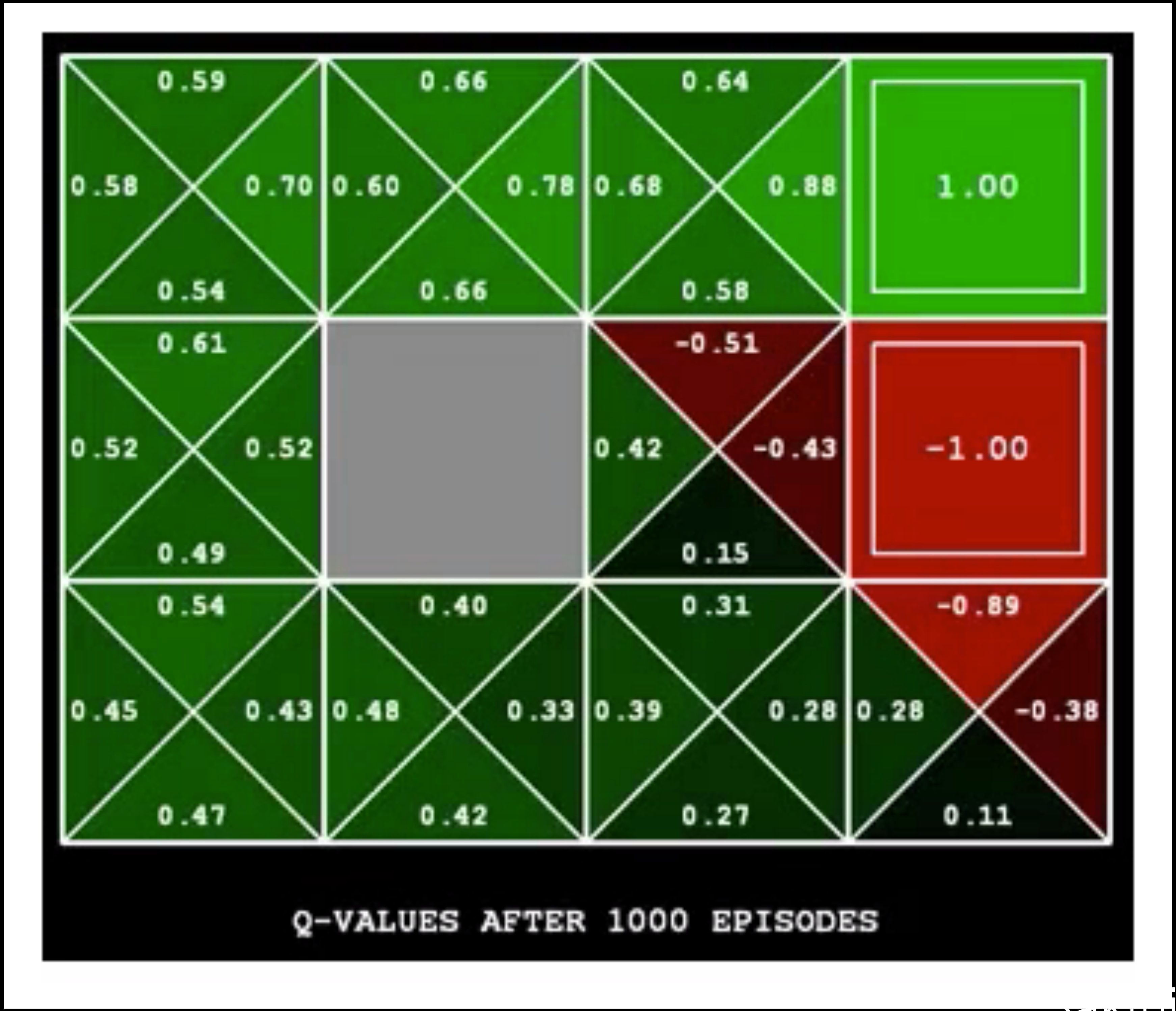
- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

Active Reinforcement Learning - Q-Value Iteration



Active Reinforcement Learning - Q-Value Iteration

- Q-Value Iteration always converges to optimal policy even if he agent is acting sub-optimally!
- Agent has to explore "enough"
- Learning rate should be low enough
- Agent eventually learns the optimal policy irrespective of what action the agent selects for learning
- This is called as off-policy learning