

MODEL-BASED REINFORCEMENT LEARNING

Scott O'Hara

Metrowest Boston Developers Machine Learning Group

02/26/2020

OVERVIEW

1. Introduction

- References
- Types of Machine Learning

2. Markov Decision Processes

- Definitions
- The Markov Property
- Examples

3. MDP Solutions

- The Bellman Equations
- Finite horizon techniques
- Infinite horizon techniques

4. Model-Based Reinforcement Learning

- The basic idea
- Model-Based RL
- Learning the reward and transition probabilities
- Credit assignment
- Exploration vs. exploitation

REFERENCES

The material for this talk is primarily drawn from the slides, notes and lectures of these courses with occasional reference to Sutton and Barto's book.

CS188 course at University of California, Berkeley:

- ▶ *CS188 – Introduction to Artificial Intelligence*, Profs. Dan Klein, Pieter Abbeel, et al. <http://ai.berkeley.edu/home.html>

CS181 course at Harvard University:

- ▶ *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Sarah Finney, Spring 2009
- ▶ *CS181 Intelligent Machines: Perception, Learning and Uncertainty*, Prof. David C Brooks, Spring 2011
- ▶ *CS181 – Machine Learning*, Prof. Ryan P. Adams, Spring 2014. <https://github.com/wihl/cs181-spring2014>
- ▶ *CS181 – Machine Learning*, Prof. David Parkes, Spring 2017. <https://harvard-ml-courses.github.io/cs181-web-2017/>

Stanford course CS229 :

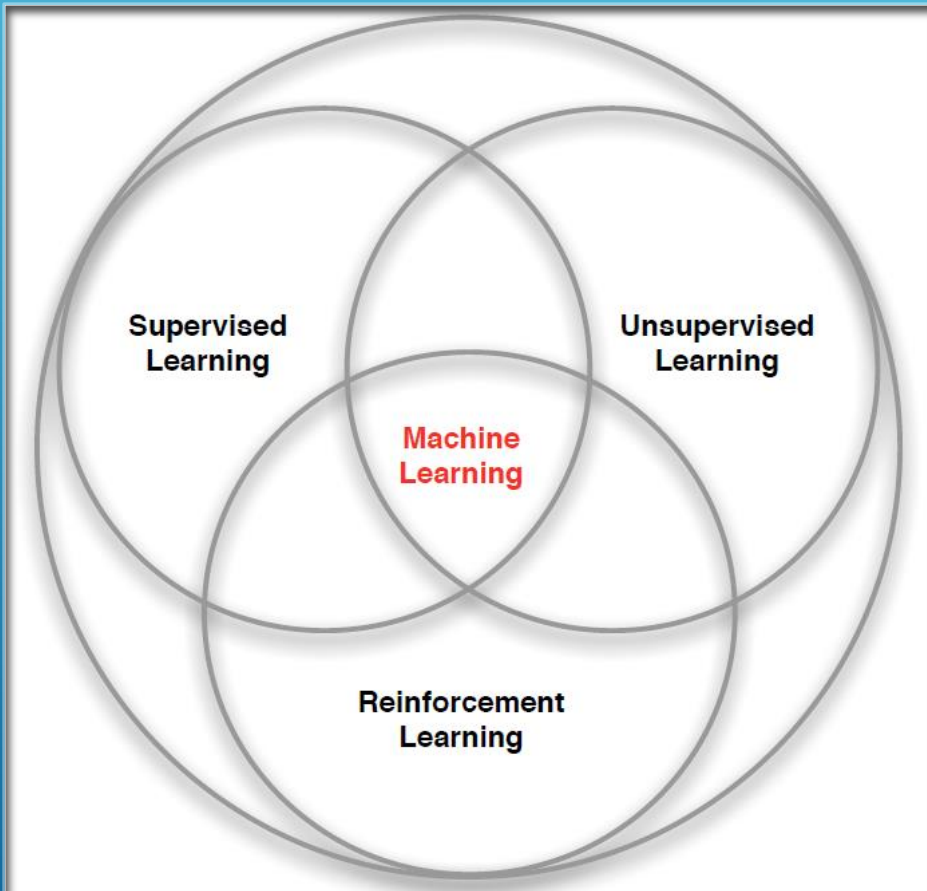
- ▶ *CS229 – Machine Learning*, Andrew Ng. <https://see.stanford.edu/Course/CS229>

Reinforcement learning: an introduction R. S. Sutton and A. G. Barto, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

UC BERKELEY CS188 IS A GREAT RESOURCE

- Websites:
 - <http://ai.berkeley.edu/home.html>
 - <http://gamescrafters.berkeley.edu/~cs188/sp20/>
 - <http://gamescrafters.berkeley.edu/~cs188/{sp|fa}<yr>/>
- Covers:
 - Search
 - Constraint Satisfaction
 - Games
 - Reinforcement Learning
 - Bayesian Networks
 - Surveys Advanced Topics
 - And more...
- Features:
 - Contains high quality YouTube videos, PowerPoint slides and homework.
 - Projects are based on the video game PacMan.
 - Material is used in many courses around the country.

3 TYPES OF MACHINE LEARNING



Supervised Learning – Learn a function from labeled data that maps input attributes to an output label e.g., linear regression, decision trees, SVMs.

Unsupervised Learning – Learn patterns in unlabeled data e.g., principle component analysis or clustering algorithms such as K-means, HAC, or Gaussian mixture models.

Reinforcement Learning – An agent learns to maximize rewards while acting in an uncertain environment.

SOME CHARACTERISTICS OF REINFORCEMENT LEARNING

- Learning happens as the agent interact with the world.
- There are no training or test sets. Training is guided by rewards and punishments obtained by acting in an environment.
- The amount of data an agent receives is not fixed. More information is acquired as you go.
- Actions are not always rewarded or punished immediately. “Delayed gratification” is possible.
- Agent actions can affect the subsequent data it receives. E.g., closing a door that can’t be opened again.

MARKOV DECISION PROCESSES



MARKOV DECISION PROCESSES

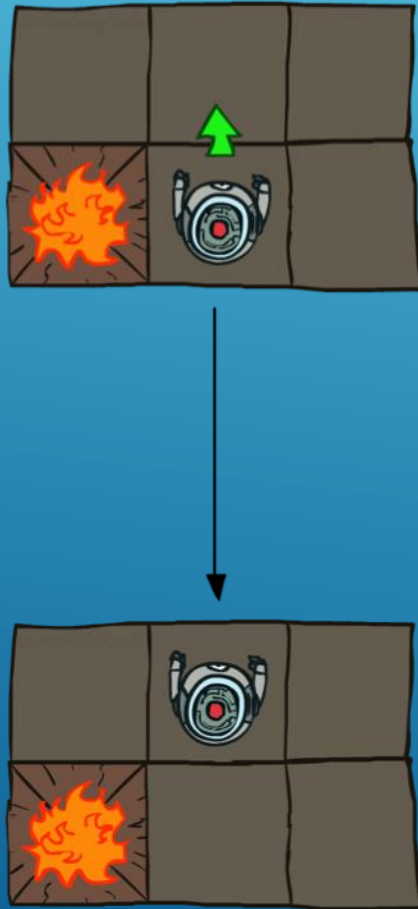
- The **Markov Decision Process** (MDP) provides a mathematical framework for reinforcement learning.
- An MDP is used to model optimal decision-making in situations where outcomes are uncertain.
- The initial analysis of MDPs assume **complete knowledge** of states, actions, rewards, transitions, and discounts.

THE MDP DECISION FRAMEWORK

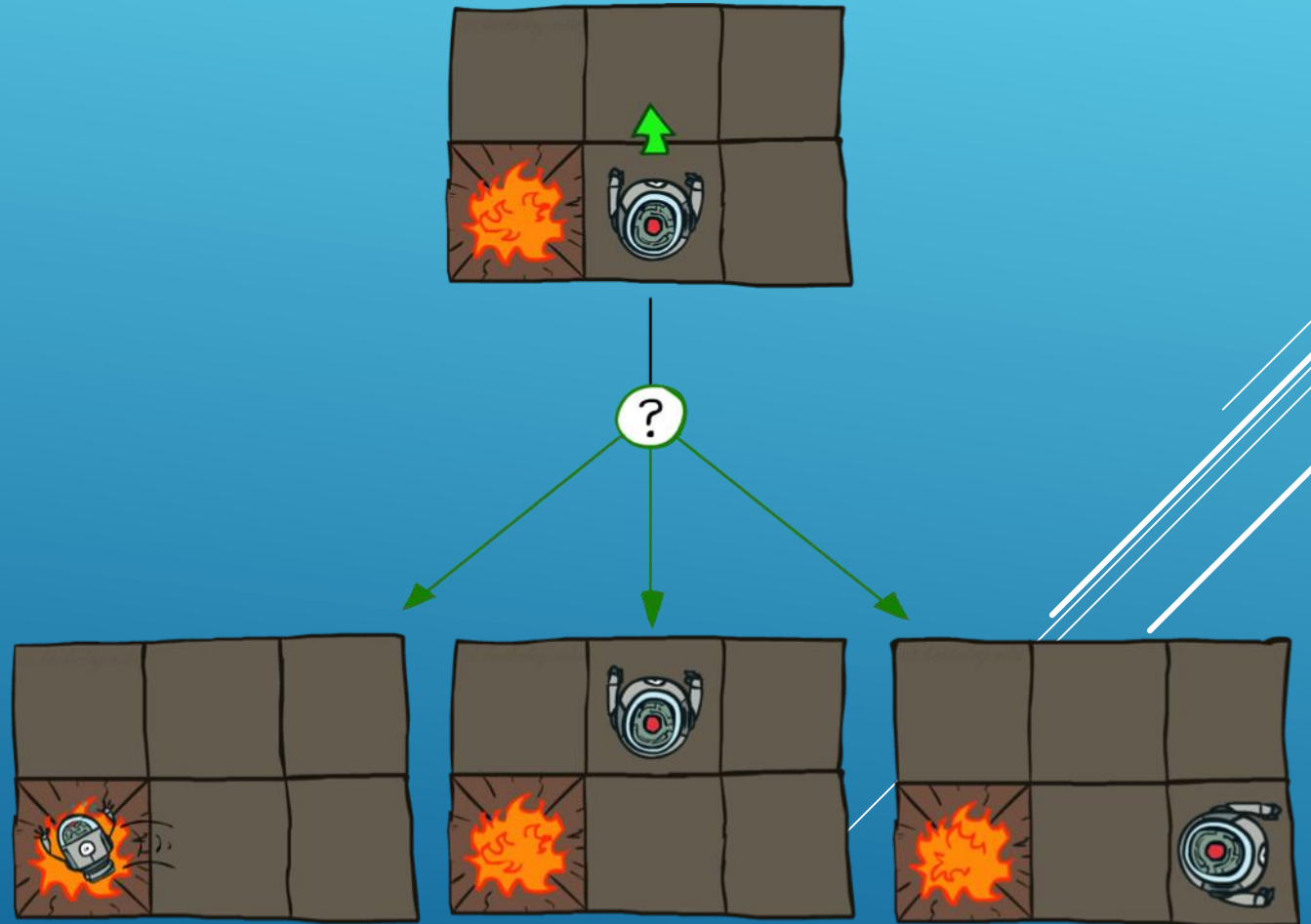
- Markov decision processes use probability to model uncertainty about the domain.
- Markov decision use utility to model an agent's objectives. The higher the utility, the “happier” your agent is.
- MDP algorithms discover an optimal decision policy π specifying how the agent should act in all possible states in order to maximize its expected utility.

EXAMPLE: GRID WORLD

Deterministic Grid World

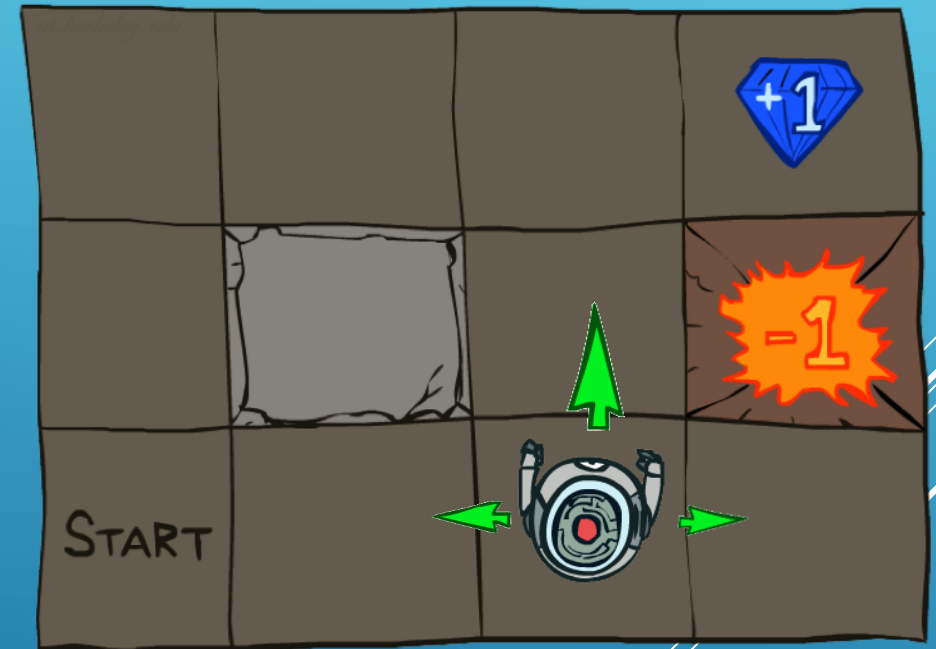


Stochastic Grid World



EXAMPLE: GRID WORLD

- A maze-like problem
 - The agent lives in a grid
 - Walls block the agent's path
- Noisy movement: actions do not always go as planned
 - 80% of the time, the action North takes the agent North (if there is no wall there)
 - 10% of the time, North takes the agent West; 10% East
 - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
 - Small "living" reward each step (can be negative)
 - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards



MARKOV DECISION PROCESSES

- **States:** s_1, \dots, s_n

- **Actions:** a_1, \dots, a_m

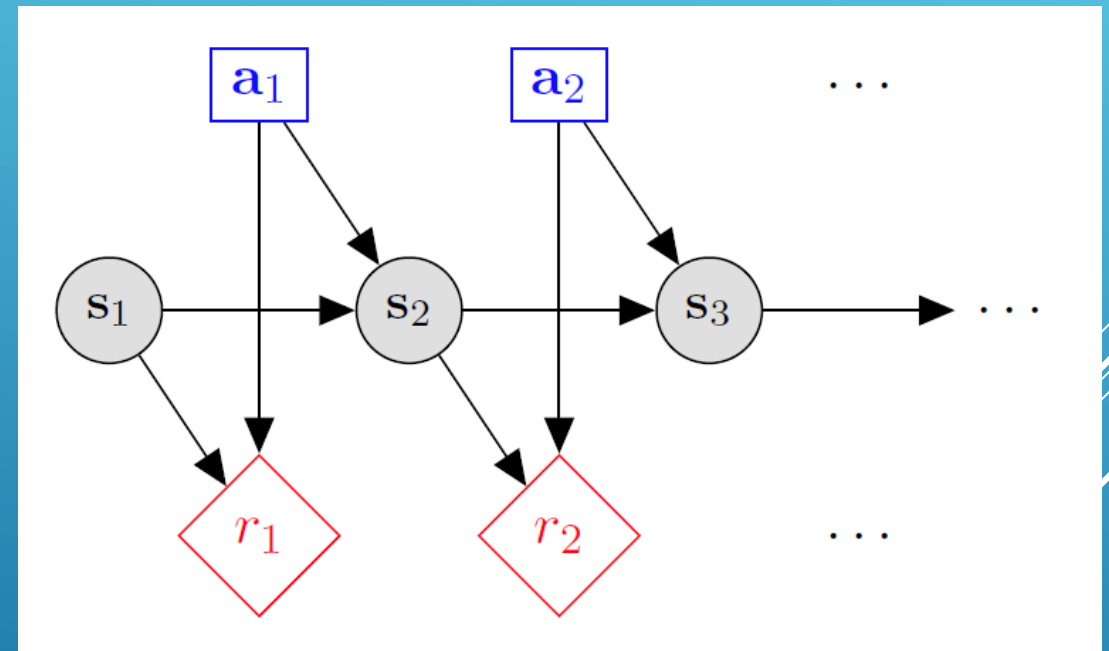
- **Reward Function:**

$$r(s, a, s') \in R$$

- **Transition model:**

$$T(s, a, s') = P(s' | s, a)$$

- **Discount factor:** $\gamma \in [0, 1]$

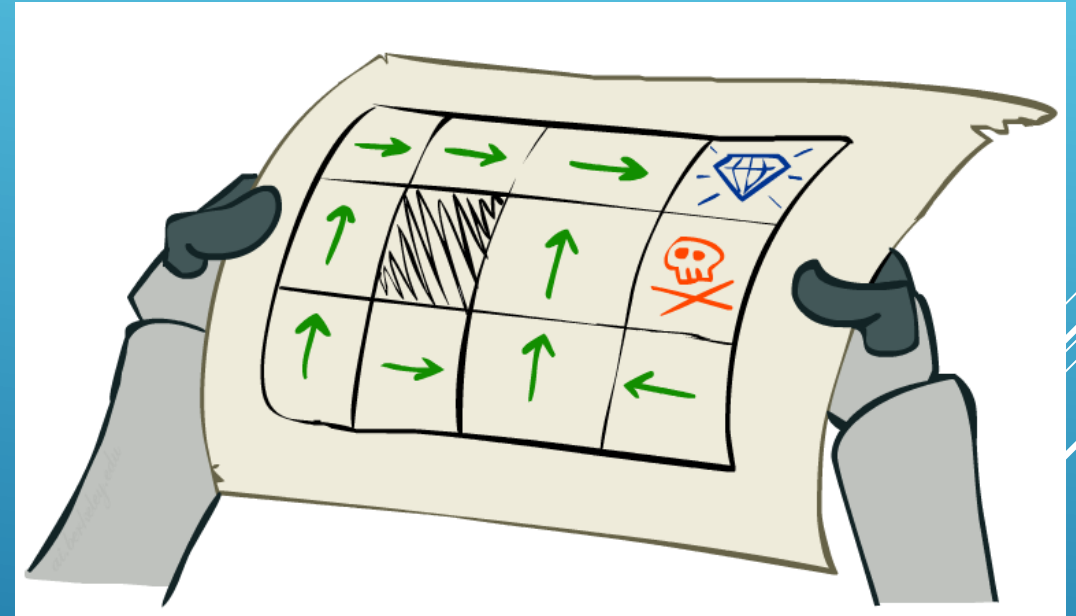


MDP GOAL: FIND AN
OPTIMAL POLICY π

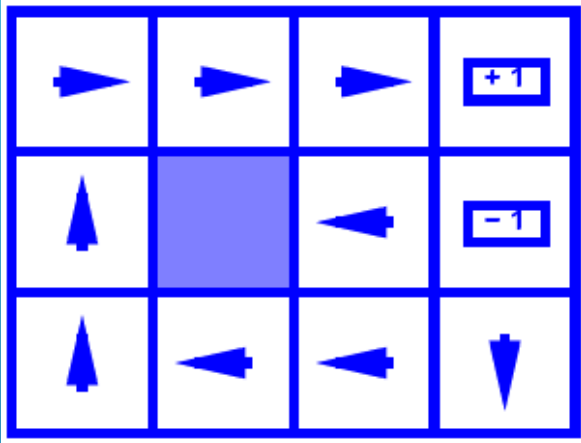


MDP GOAL: FIND AN OPTIMAL POLICY π

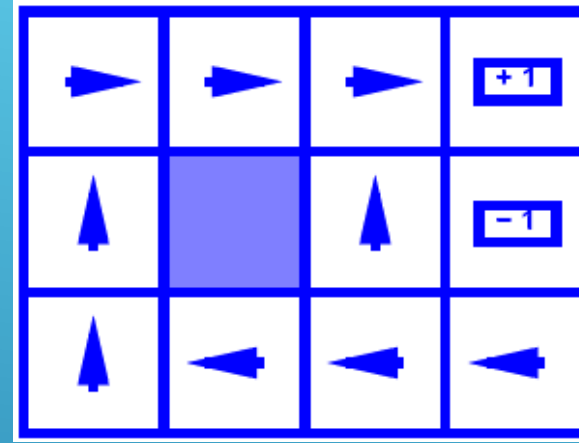
- ▶ In a state-space search, we look for an optimal **plan**, or sequence of actions, from a start state to a goal state.
- ▶ For MDPs, we look for an optimal **policy** $\pi^*: S \rightarrow A$ that completely specifies the best action to take at each state.
 - ▶ A policy π gives an action for each state.
 - ▶ An optimal policy π^* is one that maximizes the expected utility if followed.



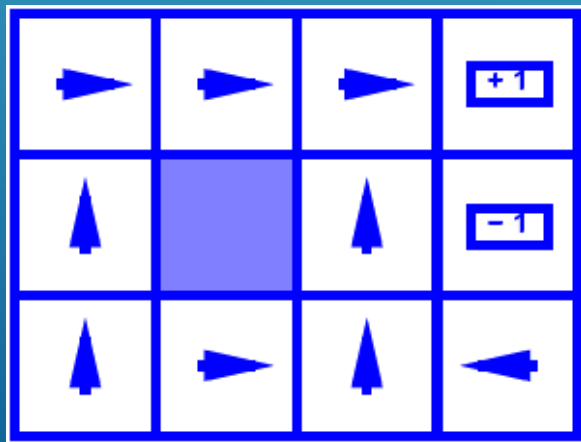
OPTIMAL POLICIES DEPEND ON THE DETAILS OF THE MDP



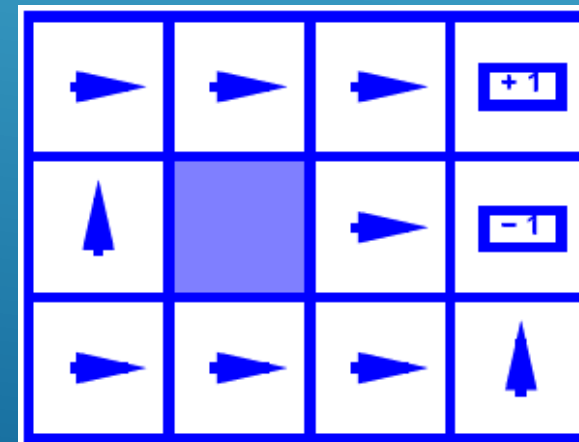
$$R(s) = -0.01$$



$$R(s) = -0.03$$



$$R(s) = -0.4$$



$$R(s) = -2.0$$

WHAT IS MARKOV ABOUT MDPS?

- ▶ “Markov” generally means that given the present state, the future and the past are independent of one another.
- ▶ For Markov decision processes, “Markov” means the next state resulting from an action depends only on the current state.

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

- ▶ This is just like exhaustive state-space search where the successor function only depends on the current state (not the history).

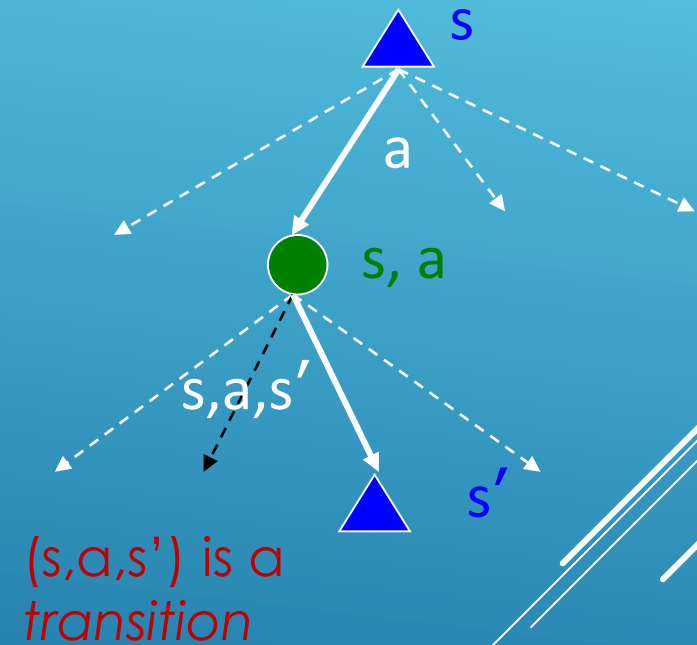


Andrey Markov
(1856-1922)

DATA STRUCTURES NEEDED FOR AN MDP ALGORITHM

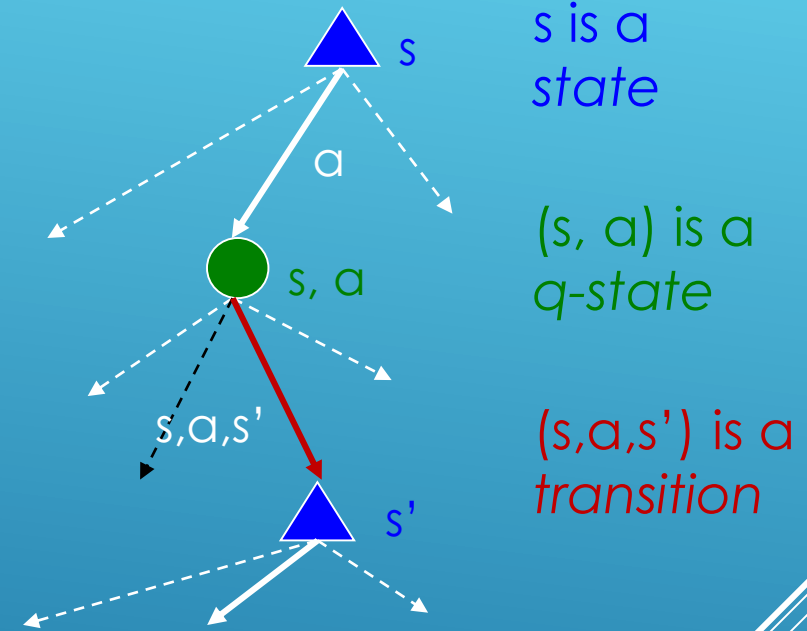
Representing the MDP:

- ▶ States S
- ▶ Actions A
- ▶ Transitions $P(s' | s, a)$ (or $T(s, a, s')$)
- ▶ Rewards $R(s, a, s')$
- ▶ Discount γ
- ▶ Start states: S
- ▶ Finish states: F



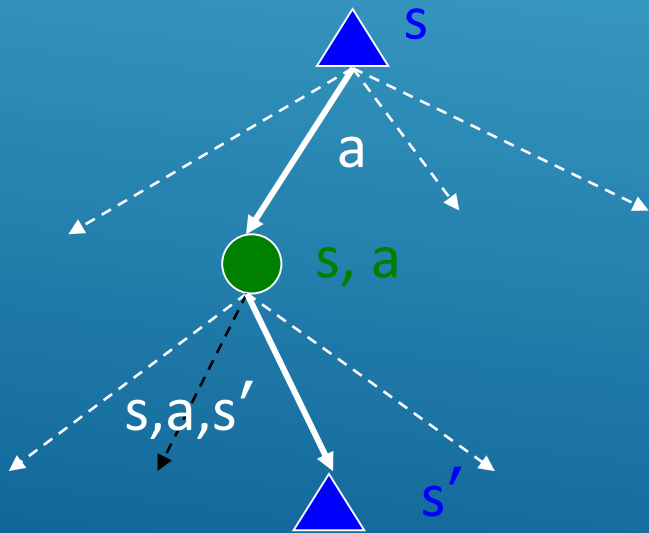
DATA STRUCTURES NEEDED FOR AN MDP ALGORITHM (2)

- **Utility** - the sum of discounted rewards received so far.
- The **value** (utility) of a **state s** :
 $V(s)$ = expected future utility starting in s and acting optimally,
- The **q-value** (utility) of a **q-state (s,a)** :
 $Q(s,a)$ = the future utility expected after taking action a from state s and acting optimally thereafter.
- The **policy**:
 $\pi(s)$ = best action to take from state s



THE BELLMAN EQUATIONS

- ▶ There is one equation $V^*(s)$ for each state s .
- ▶ There is one equation $Q^*(s, a)$ for each state s and action a .
- ▶ These are equations, not assignments. They define a relationship, which when satisfied guarantees that $V^*(s)$ and $Q^*(s, a)$ are optimal for each state and action.
- ▶ This in turn guarantees that the policy π^* is optimal.



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

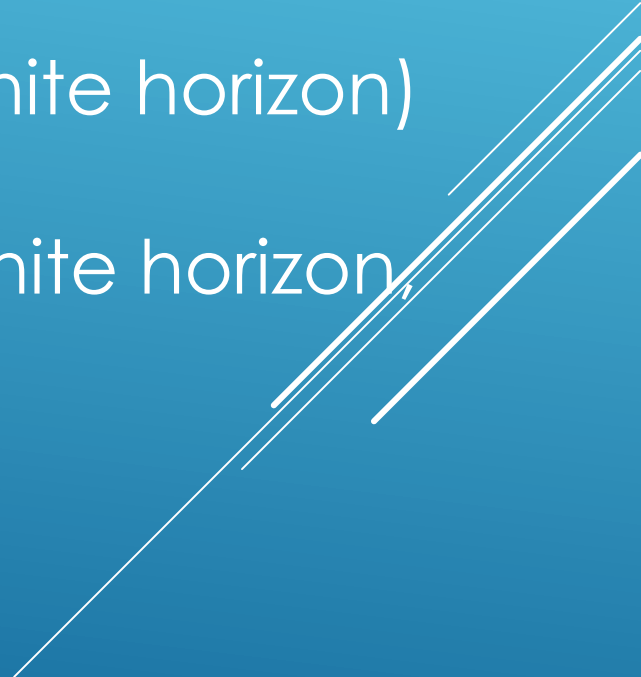
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

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

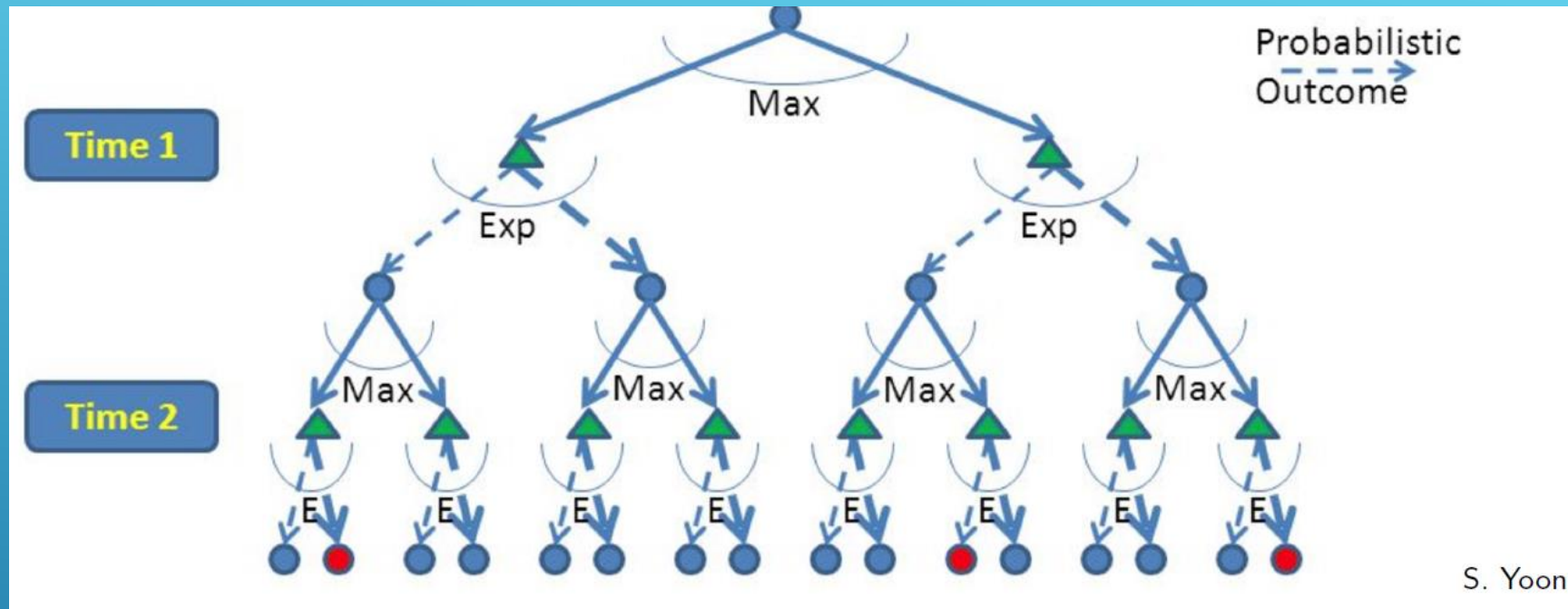
4 MDP ALGORITHMS



4 MDP ALGORITHMS

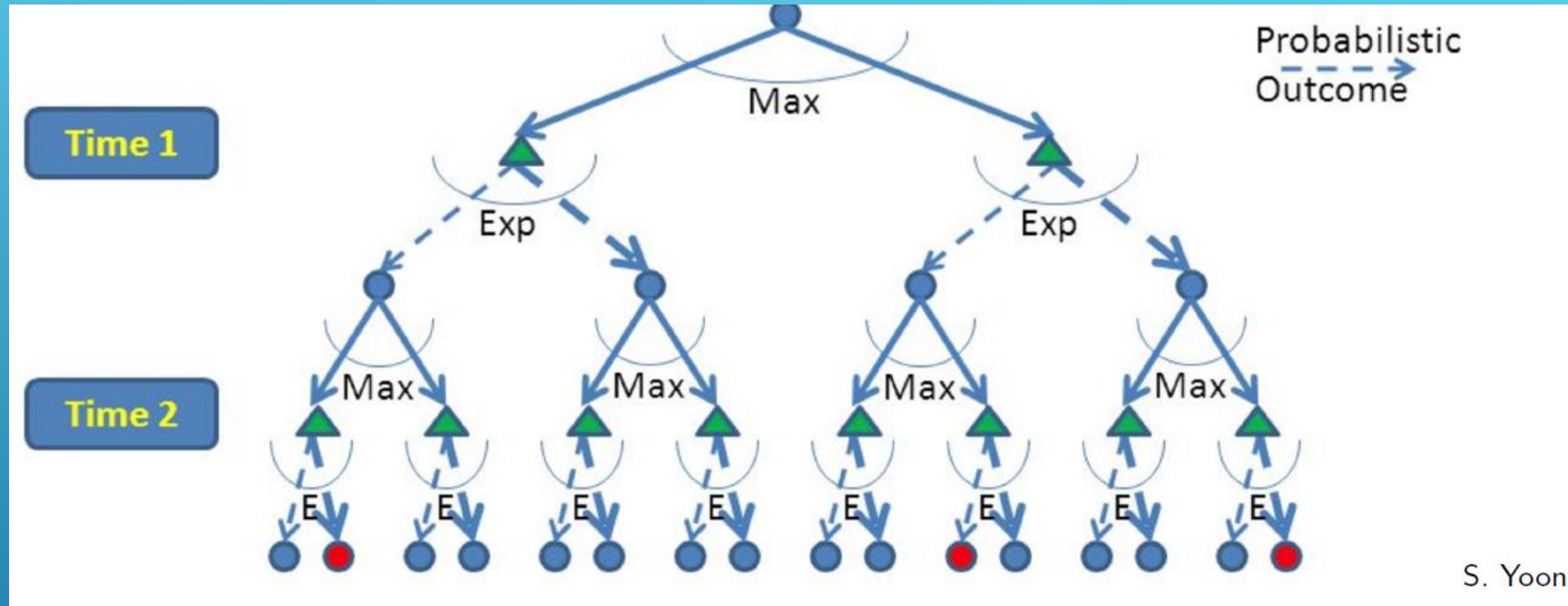
- **Expectimax** (recursive, finite horizon)
 - **Value Iteration** (dynamic programming, finite horizon)
 - **Value Iteration** (dynamic programming, infinite horizon)
 - **Policy Iteration** (dynamic programming, infinite horizon, optimize policy)
- 
- A series of white diagonal lines of varying lengths and thicknesses are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.

EXPECTIMAX: A GAME AGAINST NATURE



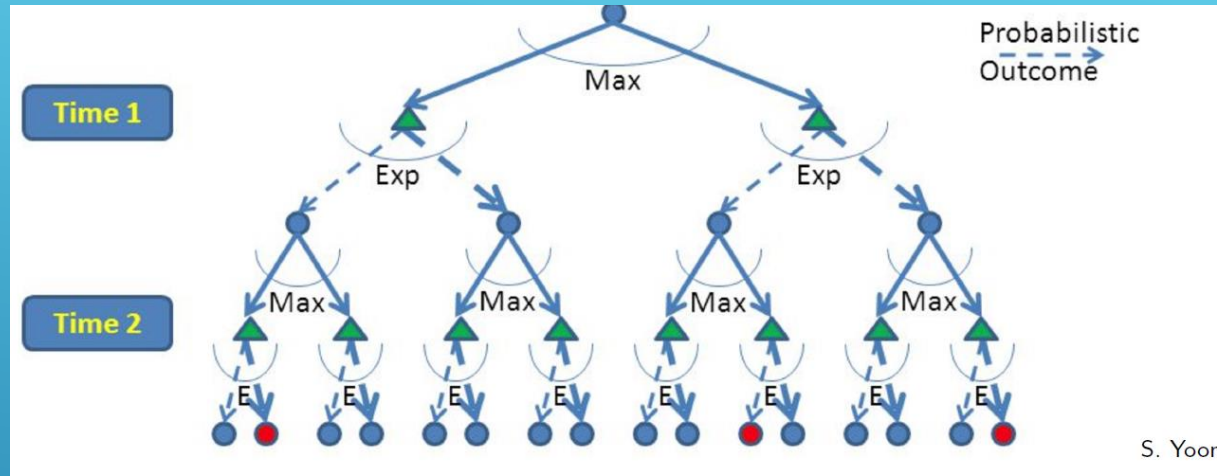
- Expectimax is like a game-playing algorithm except the opponent is nature.
- Expectimax is strongly related to the minmax algorithm used in game theory, but the response is probabilistic.
- Nodes where you move are called **states**: S (●)
- Nodes where nature moves are called **Q-states**: $\langle S, A \rangle$ (▲)

EXPECTIMAX: TOP-DOWN, RECURSIVE



- Expectimax is basically a planning algorithm.
- It finds the best move for 1 state.
- Build out a look-ahead tree to the decision horizon; take the max over actions, expectations over next states.
- Solve from the leaves, backing-up the expectimax values.

EXPECTIMAX: TOP-DOWN, RECURSIVE



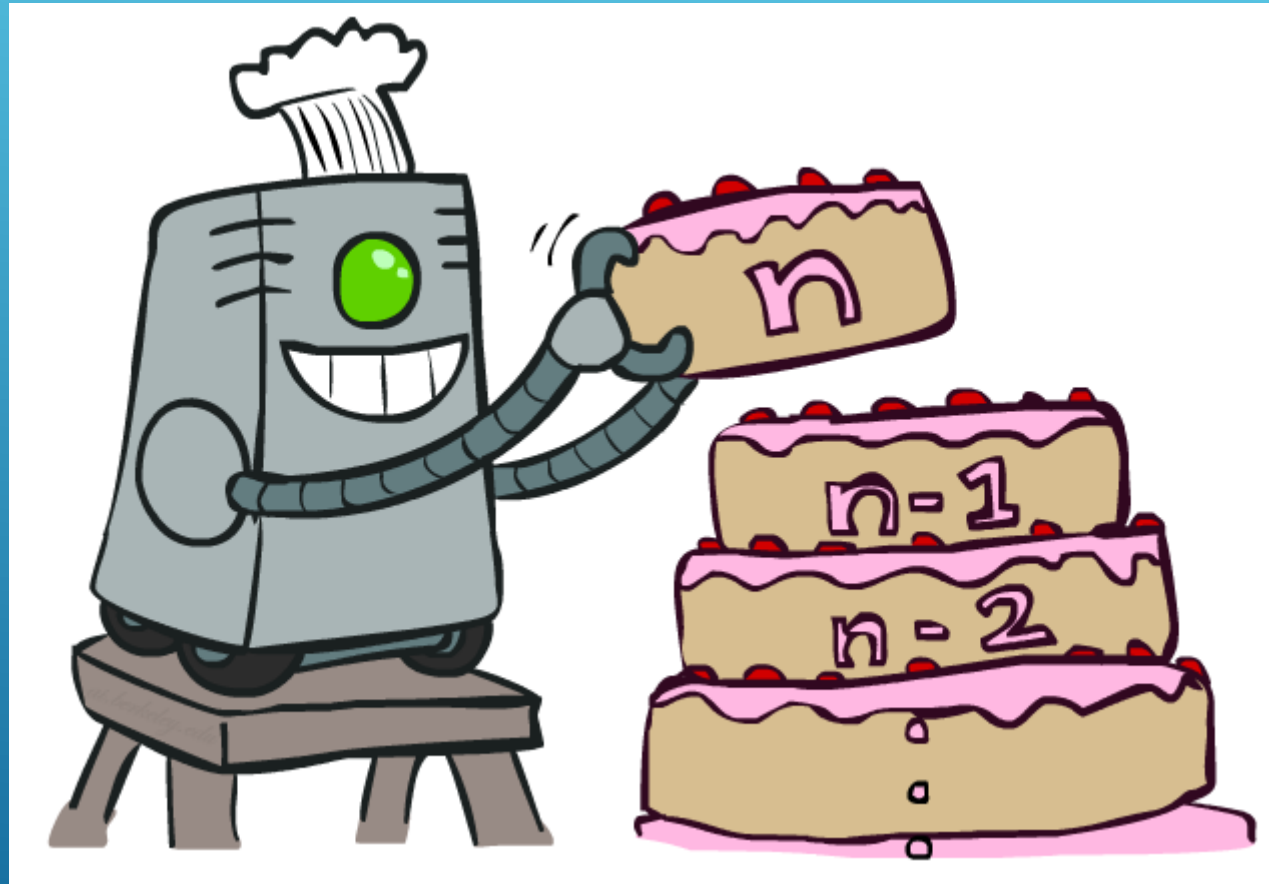
- Advantages:
 - Effective in rapidly changing environment where the MDP is known.
- Disadvantages:
 - computation is exponential in the horizon.
 - may expand the same subtree multiple times.

EXPECTIMAX ALGORITHM

Algorithm 1 Expectimax Search

```
1: function EXPECTIMAX(s)                                ▷ Takes a state as an input.
2:   if s is terminal then
3:     Return 0
4:   else
5:     for  $a \in \mathcal{A}$  do                                ▷ Look at all possible actions.
6:        $Q(s, a) \leftarrow R(s, a) + \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \text{EXPECTIMAX}(s')$   ▷ Compute expected value.
7:     end for
8:      $\pi^*(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q(s, a)$         ▷ Optimal policy is value-maximizing action.
9:     Return  $Q(s, \pi^*(s))$ 
10:   end if
11: end function
```

VALUE ITERATION USES DYNAMIC PROGRAMMING



VALUE ITERATION (FINITE HORIZON)

- ▶ Start with $V_0(s) = 0$ - no time steps left means an expected reward sum of zero
- ▶ Given vector of $V_k(s)$ values, do one ply from each state:

$$V_{k+1}(s) \leftarrow \max_a \left[\sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V_k(s')] \right]$$

- ▶ Repeat until $k == T$

-
- ▶ Complexity of each iteration: $O(S^2A)$

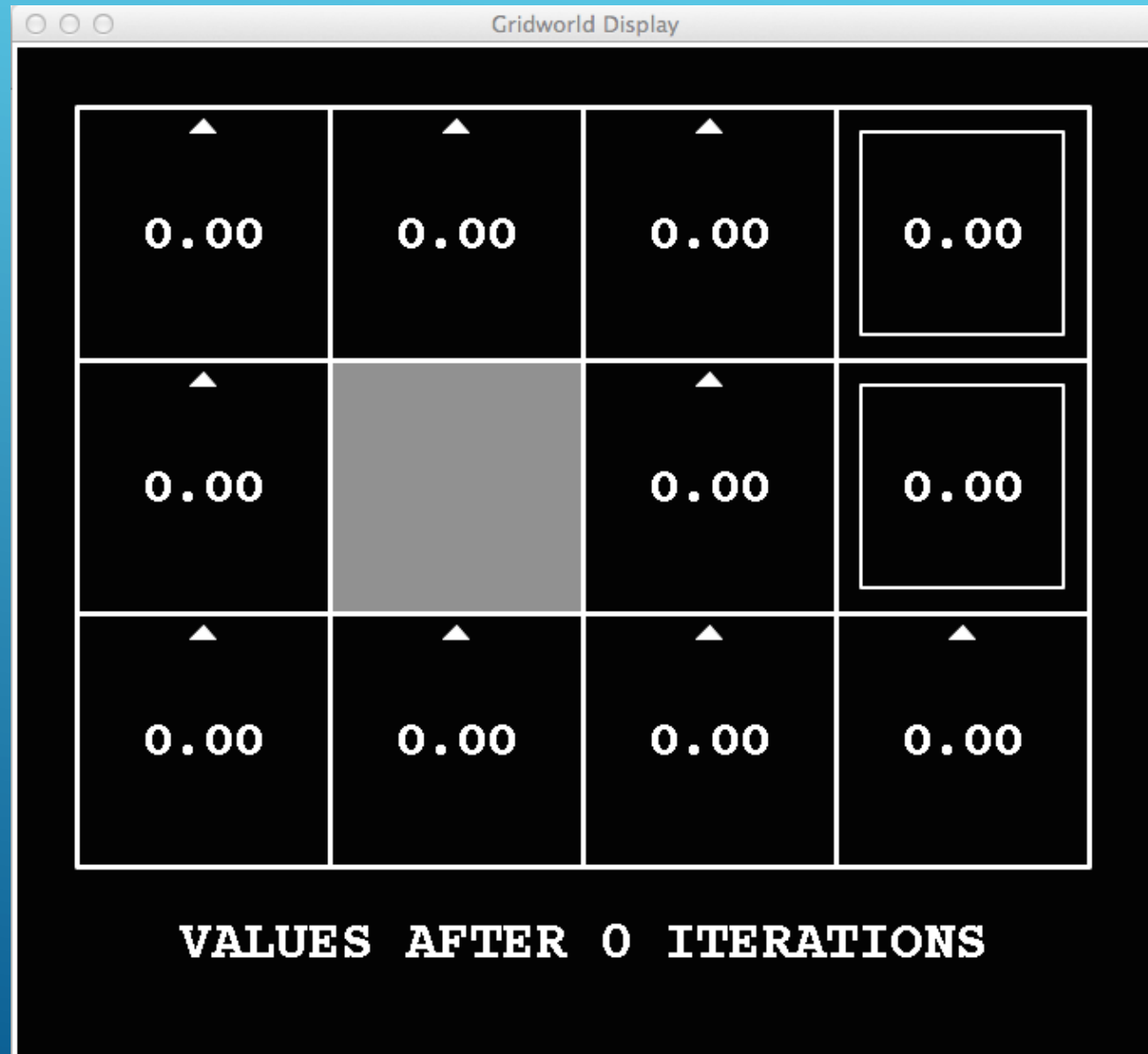
- ▶ For every state s , there are $|A|$ actions
- ▶ For every state s and action a , there are $|S|$ possible states s'

VALUE ITERATION (FINITE HORIZON)

Algorithm 2 Value Iteration

```
1: function VALUEITERATION( $T$ )                                     ▷ Takes a horizon as input.
2:   for  $s \in \mathcal{S}$  do                                             ▷ Loop over each possible ending state.
3:      $V_0(s) \leftarrow 0$                                          ▷ Horizon states have no value.
4:   end for
5:   for  $k \leftarrow 1 \dots T$  do                                   ▷ Loop backwards over time.
6:     for  $s \in \mathcal{S}$  do                                           ▷ Loop over possible states with  $k$  steps to go.
7:       for  $a \in \mathcal{A}$  do                                           ▷ Loop over possible actions.
8:          $Q_k(s, a) \leftarrow R(s, a) + \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V_{k-1}(s')$   ▷ Compute  $Q$ -function for  $k$ .
9:       end for
10:       $\pi_k^*(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q_k(s, a)$            ▷ Find best action with  $k$  to go in state  $s$ .
11:       $V_k(s) \leftarrow Q_k(s, \pi_k^*(s))$                          ▷ Compute value for state  $s$  with  $k$  steps to go.
12:    end for
13:  end for
14: end function
```

$K=0$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=1$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=2$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=3$



Noise = 0.2
Discount = 0.9
Living reward = 0

VALUE ITERATION (INFINITE HORIZON)

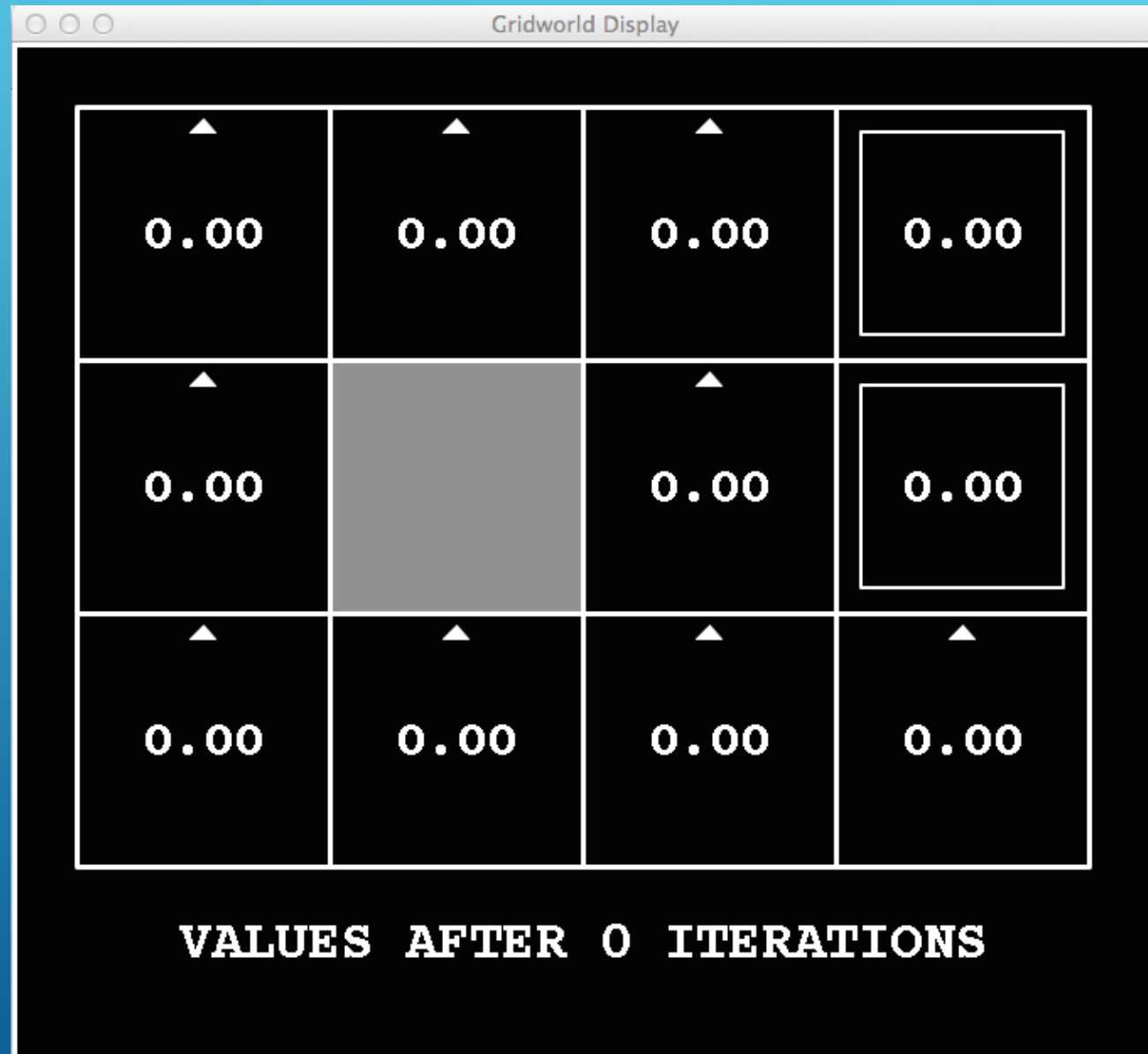
- ▶ Start with $V_0(s) = 0$ - no time steps left means an expected reward sum of zero
- ▶ Given vector of $V_k(s)$ values, do one ply from each state:

$$V_{k+1}(s) \leftarrow \max_a \left[\sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma V_k(s')] \right]$$

- ▶ Repeat until convergence
-

- ▶ Theorem: will converge to unique optimal values
 - ▶ Basic idea: approximations get refined towards optimal values
 - ▶ Policy may converge long before values do

$K=0$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=1$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=2$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=3$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=4$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=5$



Noise = 0.2
Discount = 0.9
Living reward = 0

K=6



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=7$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=8$



Noise = 0.2
Discount = 0.9
Living reward = 0

$K=9$



Noise = 0.2
Discount = 0.9
Living reward = 0

K=10



Noise = 0.2
Discount = 0.9
Living reward = 0

K=11



Noise = 0.2
Discount = 0.9
Living reward = 0

K=12



Noise = 0.2
Discount = 0.9
Living reward = 0

K=100



Noise = 0.2
Discount = 0.9
Living reward = 0

VALUE ITERATION (INFINITE HORIZON)

Algorithm 1 Infinite Horizon Value Iteration

```

1: function VALUEITERATION( $\gamma$ )
2:   for  $s \in \mathcal{S}$  do
3:      $V(s) \leftarrow 0$ 
4:   end for
5:   repeat
6:      $V_{\text{old}}(\cdot) \leftarrow V(\cdot)$ 
7:     for  $s \in \mathcal{S}$  do
8:       for  $a \in \mathcal{A}$  do
9:          $Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V_{\text{old}}(s')$ 
10:       end for
11:        $\pi^*(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q(s, a)$ 
12:        $V(s) \leftarrow Q(s, \pi^*(s))$ 
13:     end for
14:   until  $|V(s) - V_{\text{old}}(s)| < \epsilon, \forall s \in \mathcal{S}$ 
15:   Return  $\pi^*(\cdot)$ 
16: end function

```


POLICY ITERATION

- ▶ Alternative approach for optimal values:
 - ▶ **Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
 - ▶ **Step 2: Policy improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
 - ▶ **Repeat** steps until new policy is the same as the old policy.
- ▶ This is **policy iteration**
 - ▶ It's still optimal!
 - ▶ Can converge (much) faster under some conditions

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

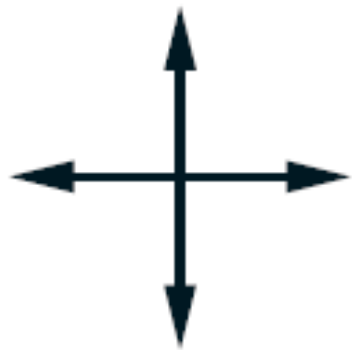
old-action $\leftarrow \pi(s)$

$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

If *old-action* $\neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

POLICY ITERATION EXAMPLE



actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R_t = -1$
on all transitions

POLICY ITERATION EXAMPLE (K=0)

v_k for the
random policy

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

greedy policy
w.r.t. v_k

	↕↔↕	↕↔↕	↕↔↕
↕↔↕	↕↔↕	↕↔↕	↕↔↕
↕↔↕	↕↔↕	↕↔↕	↕↔↕
↕↔↕	↕↔↕	↕↔↕	

POLICY ITERATION EXAMPLE (K=1)

v_k for the
random policy

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

greedy policy
w.r.t. v_k

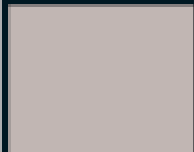

	←	↕	↕
↑	↕	↕	↕
↕	↕	↕	↓
↕	↕	→	

POLICY ITERATION EXAMPLE (K=2)

v_k for the
random policy

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

greedy policy
w.r.t. v_k

	←	←	↕
↑	↖	↕	↓
↑	↕	↘	↓
↕	→	→	

POLICY ITERATION EXAMPLE (K=3, OPTIMAL)

v_k for the
random policy

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

greedy policy
w.r.t. v_k

	←	←	↙
↑	↖	↙	↓
↑	↖	↘	↓
↖	→	→	

POLICY ITERATION EXAMPLE (K=10, OPTIMAL)

v_k for the
random policy

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

greedy policy
w.r.t. v_k

	←	←	↙
↑	↖	↙	↓
↑	↖	↘	↓
↖	→	→	

POLICY ITERATION EXAMPLE ($K=\infty$, OPTIMAL)

v_k for the
random policy

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

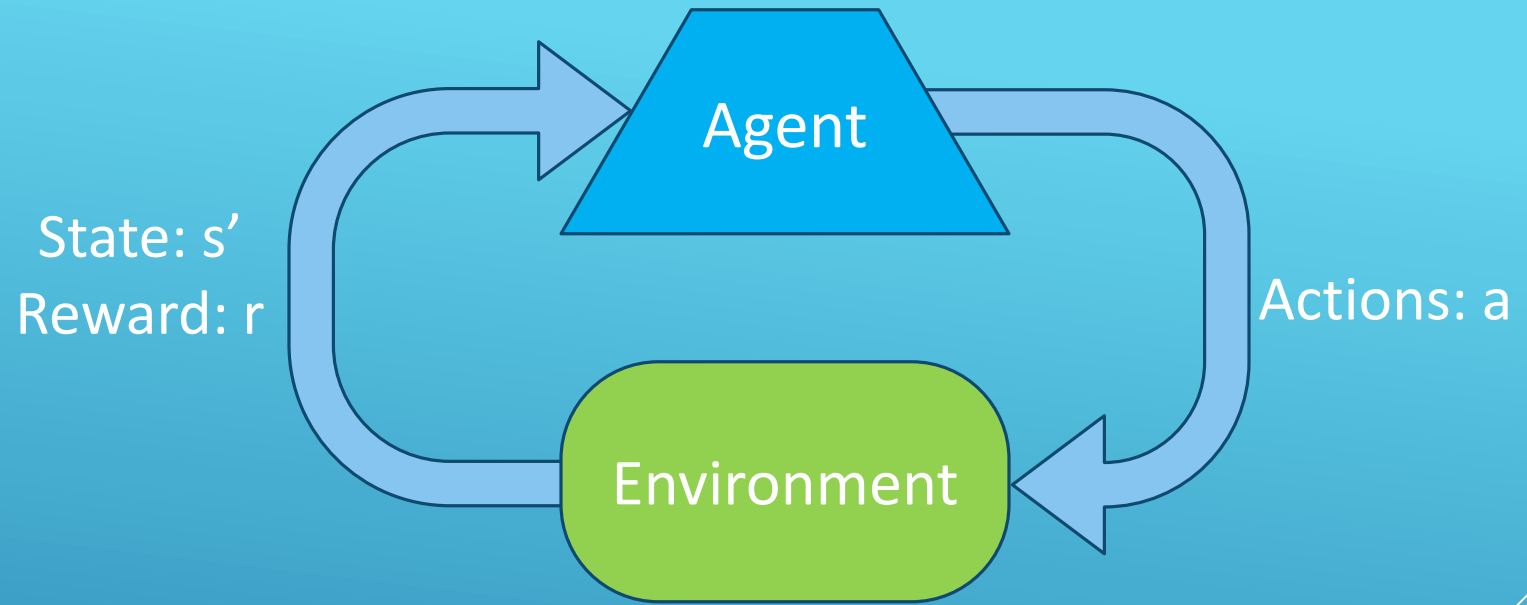
greedy policy
w.r.t. v_k

	←	←	↙
↑	↖	↙	↓
↑	↖	↘	↓
↖	→	→	

MODEL-BASED REINFORCEMENT LEARNING



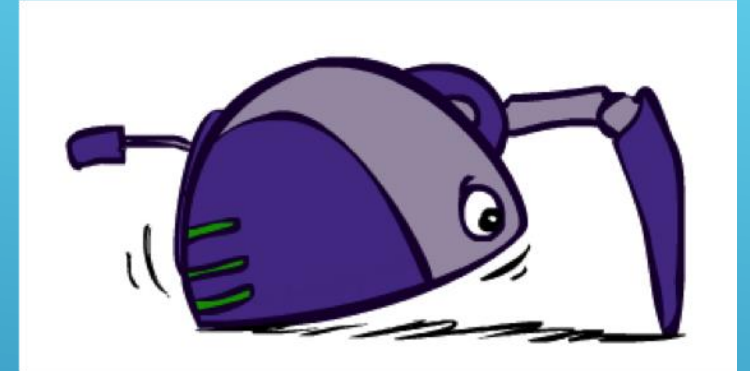
REINFORCEMENT LEARNING



- Agent knows the current state s , takes action a , receives a reward r and observes the next state s'
- Agent has **no access** to reward model $r(s,a)$ or transition model $p(s' | s,a)$
- Agent must learn to act so as to maximize expected rewards.
- All learning is based on observed samples of outcomes!
- Under these conditions, it is a very challenging problem to learn the policy π .

MODEL-BASED LEARNING

- ▶ Model-Based Idea:
 - ▶ Learn an approximate model based on experiences
 - ▶ Solve for values as if the learned model were correct
- ▶ Step 1: Learn empirical MDP model
 - ▶ Count outcomes s' for each s, a
 - ▶ Normalize to give an estimate of $\hat{T}(s, a, s')$
 - ▶ Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')
- ▶ Step 2: Solve the learned MDP
 - ▶ For example, use value iteration, as before



LEARN THE REWARD AND TRANSITION DISTRIBUTIONS

- Try every action in each state a number of times
- $RTotal(s, a, s')$ = total reward for taking action a in state s and transitioning to state s'
- $N(a, s)$ = number of times action a is taken in state s
- $N(s, a, s')$ = number of times s transitions to s' on action a
- $\hat{R}(s, a, s') = RTotal(s, a, s')/N(s, a, s')$
- $\hat{T}(s, a, s') = N(s, a, s')/N(a, s)$

TRANSITION/REWARD PARAMETER TABLE

For every state s :

State s'

Action a

$\hat{T}(s, a0, s0)$ $\hat{R}(s, a0, s0)$	$\hat{T}(s, a0, s1)$ $\hat{R}(s, a0, s1)$	$\hat{T}(s, a0, s2)$ $\hat{R}(s, a0, s2)$	$\hat{T}(s, a0, s3)$ $\hat{R}(s, a0, s3)$
$\hat{T}(s, a1, s0)$ $\hat{R}(s, a1, s0)$	$\hat{T}(s, a1, s1)$ $\hat{R}(s, a1, s1)$	$\hat{T}(s, a1, s2)$ $\hat{R}(s, a1, s2)$	$\hat{T}(s, a1, s3)$ $\hat{R}(s, a1, s3)$
$\hat{T}(s, a2, s0)$ $\hat{R}(s, a2, s0)$	$\hat{T}(s, a2, s1)$ $\hat{R}(s, a2, s1)$	$\hat{T}(s, a2, s2)$ $\hat{R}(s, a2, s2)$	$\hat{T}(s, a2, s3)$ $\hat{R}(s, a2, s3)$

MODEL-BASED RL

Let π^0 be arbitrary

$k \leftarrow 0$

Experience $\leftarrow \emptyset$

Repeat

$k \leftarrow k + 1$

 Begin in state i

 For a while:

 Choose action a based on π^{k-1}

 Receive reward r and transition to j

 Experience \leftarrow Experience $\cup \langle i, a, r, j \rangle$

$i \leftarrow j$

 Learn MDP M from Experience

 Solve M to obtain π^k

MODEL-BASED RL: PROS AND CONS

○ **Pros:**

- Makes maximal use of experience
- Solves model optimally, given enough experience

○ **Cons:**

- Model must be small enough to solve.
- Solution procedure is computationally expensive.
- Learning is episodic, not continuous, so agent does not benefit immediately from experience.

○ **Conclusion:**

- Model-free approaches are used for most real-world problems.
- Examples: Q-Learning, Monte Carlo, SARSA, TD-Learning, Deep QL, etc.