

MORE Q-LEARNING

Scott O'Hara

Metrowest Developers Machine Learning Group

11/14/2018

REFERENCES

The material for this talk is primarily drawn from the slides, notes and lectures of these courses:

University of California, Berkeley CS188:

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

David Silver, DeepMind:

- ▶ *Introduction to Reinforcement Learning*
<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.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/>

OVERVIEW

1. Where We Have Been

- Reinforcement Learning Overview
- Markov Decision Processes (MDPs)
- 4 MDP Algorithms

2. Reinforcement Learning Algorithms

- Reinforcement Learning
- Model-based RL
- The Bellman Equations
- States and Q-States
- Temporal Difference (TD) Learning
- TD and Exponential Smoothing
- Q-Learning
- SARSA

OVERVIEW

1. → Where We Have Been ←

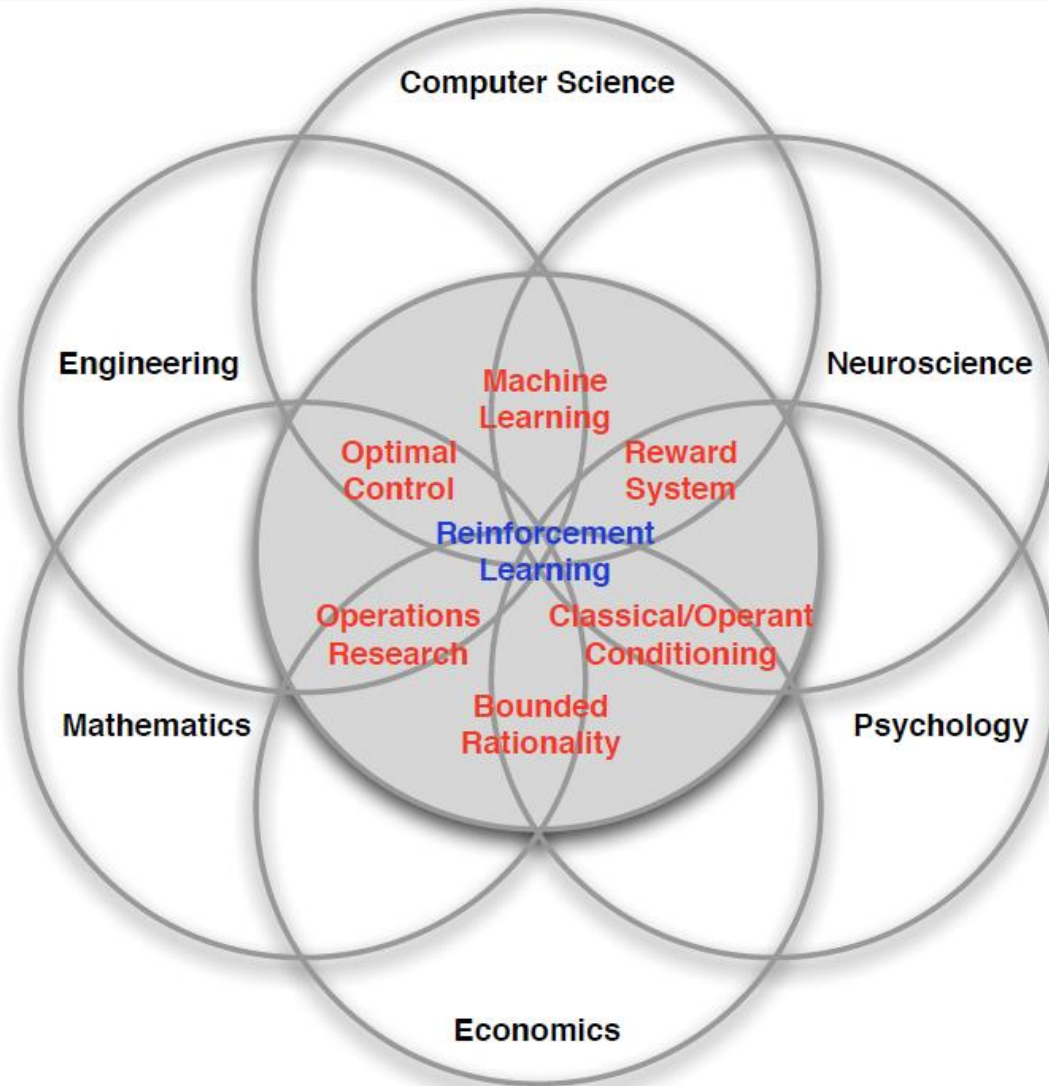
- Reinforcement Learning Overview
- Markov Decision Processes (MDPs)
- 4 MDP Algorithms

2. Reinforcement Learning Algorithms

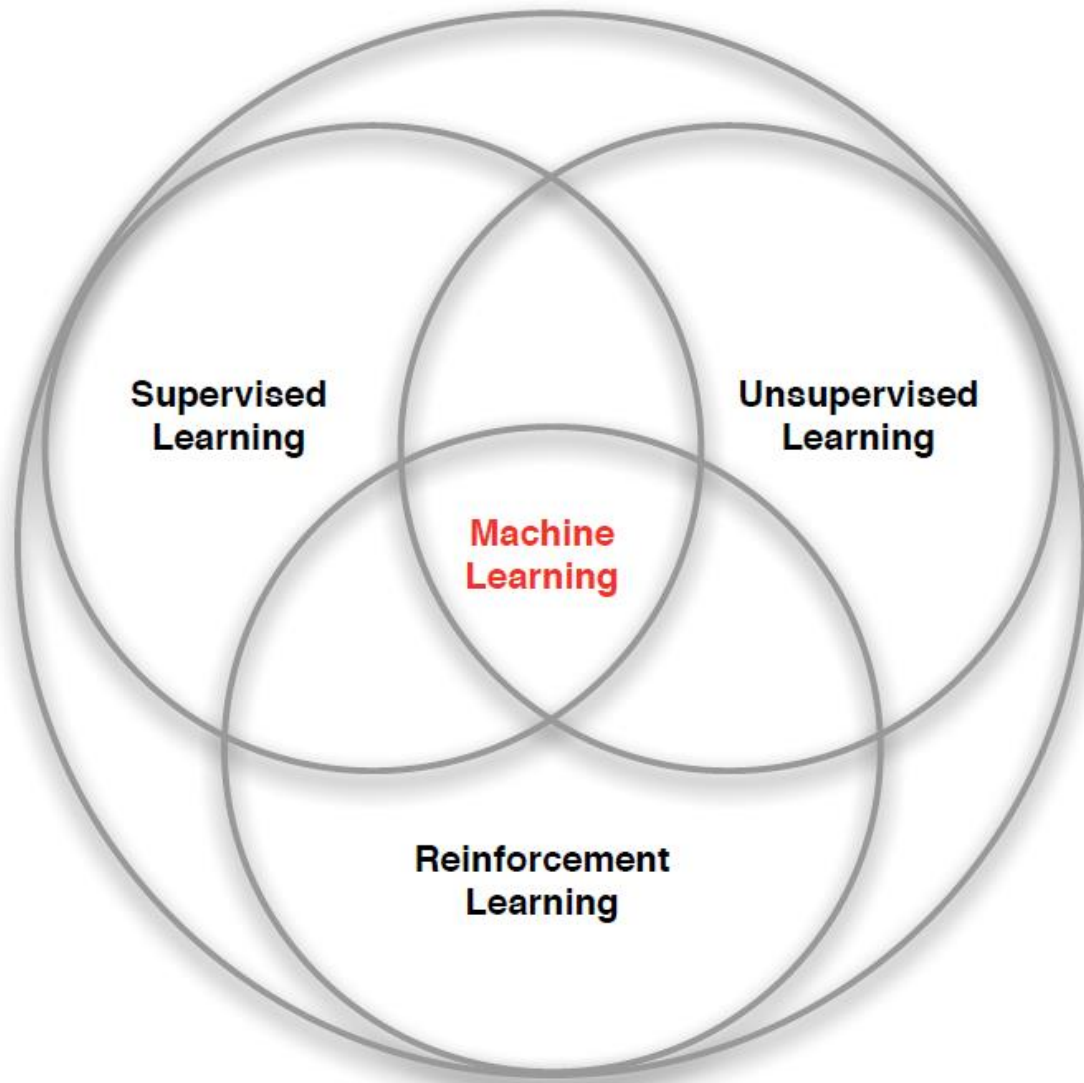
- Reinforcement Learning
- Model-based RL
- The Bellman Equations
- States and Q-States
- Temporal Difference (TD) Learning
- TD and Exponential Smoothing
- Q-Learning
- SARSA

REINFORCEMENT LEARNING OVERVIEW





MANY FACES OF REINFORCEMENT LEARNING



BRANCHES OF MACHINE LEARNING

Credit: adapted from lecture slides David Silver, DeepMind, "Introduction to Reinforcement Learning"

CHARACTERISTICS OF REINFORCEMENT LEARNING

- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, data is not i.i.d. – independent and identically distributed.)
- Agent's actions affect the subsequent data it receives.

EXAMPLES OF REINFORCEMENT LEARNING

- Fly stunt maneuvers in a helicopter
- Defeat the world champion at Backgammon
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play Atari games better than humans

EXAMPLES OF REINFORCEMENT LEARNING

RL Course by David Silver – Lecture 1: Introduction to Reinforcement Learning

<https://www.youtube.com/watch?v=2pWv7GOvuf0>

12:25 – 22.00

MARKOV DECISION PROCESSES



MARKOV DECISION PROCESSES

- Markov Decision Processes provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.
- The initial analysis of MDPs assume **complete knowledge** of states, actions, rewards, transitions, and discounts.

MARKOV DECISION PROCESSES

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

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

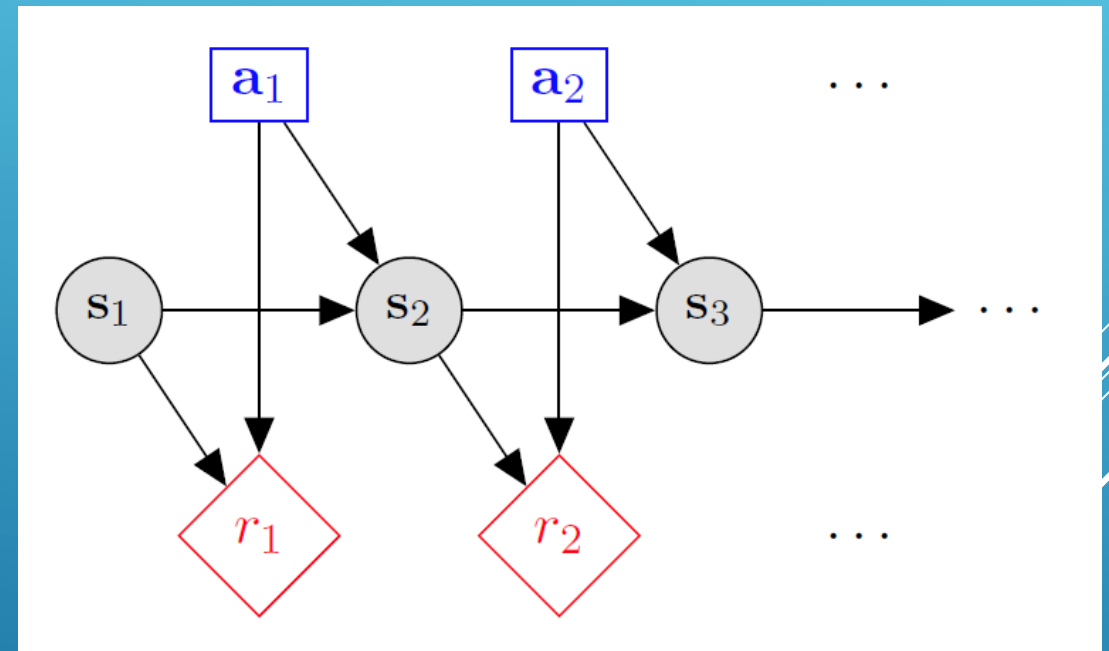
- **Reward Function:**

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

- **Transition model:**

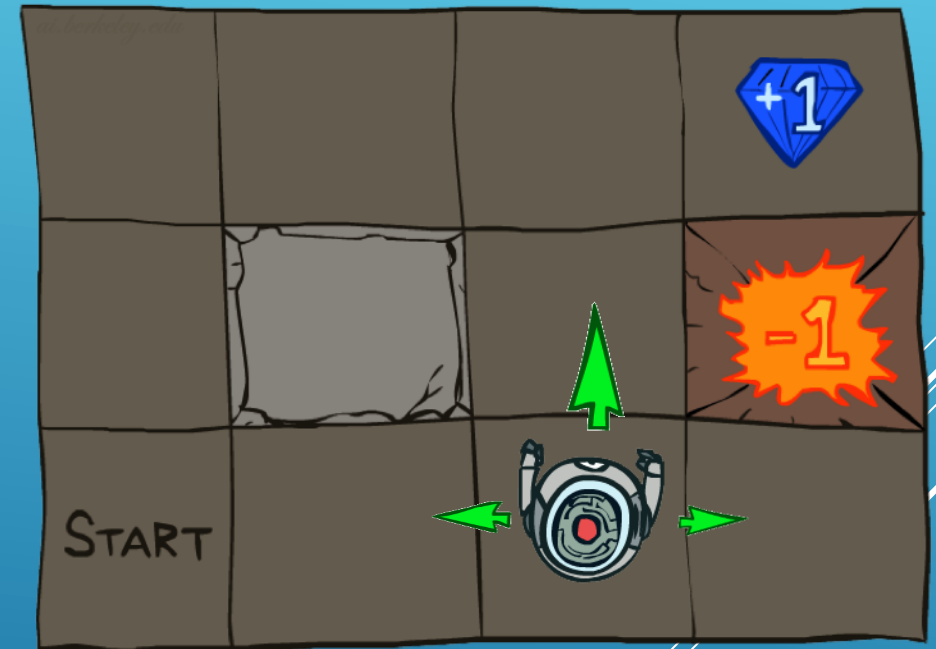
$$T(s, a, s') = P(s' | s, a)$$

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



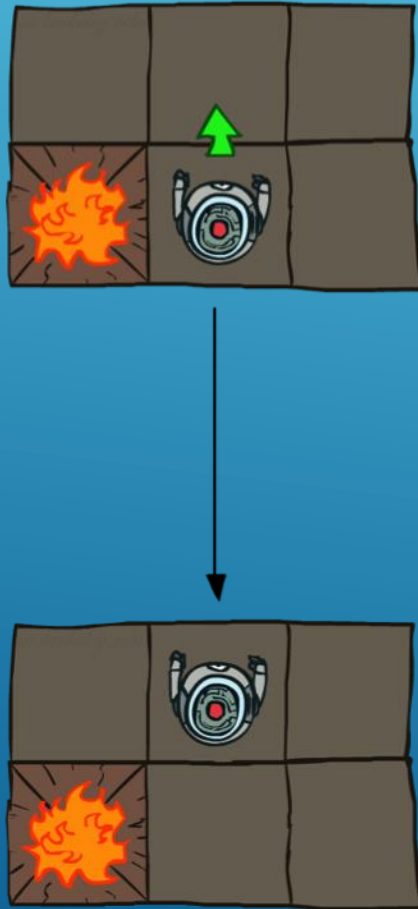
EXAMPLE: GRID WORLD

- 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

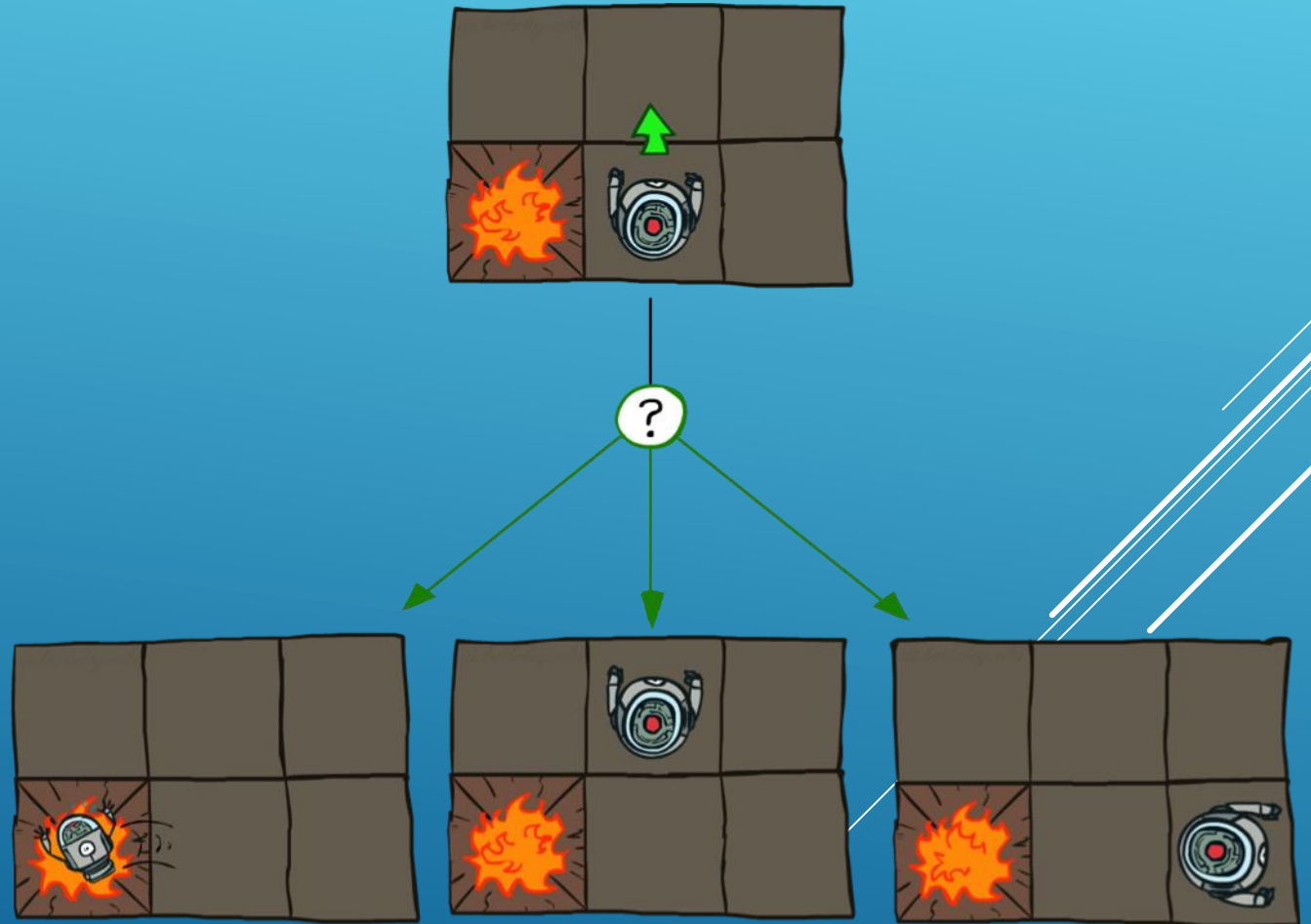


GRID WORLD ACTIONS

Deterministic Grid World



Stochastic Grid World



WHAT IS MARKOV ABOUT MDPS?

- ▶ “Markov” generally means that given the present state, the future and the past are independent
- ▶ For Markov decision processes, “Markov” means action outcomes depend only on the current state

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

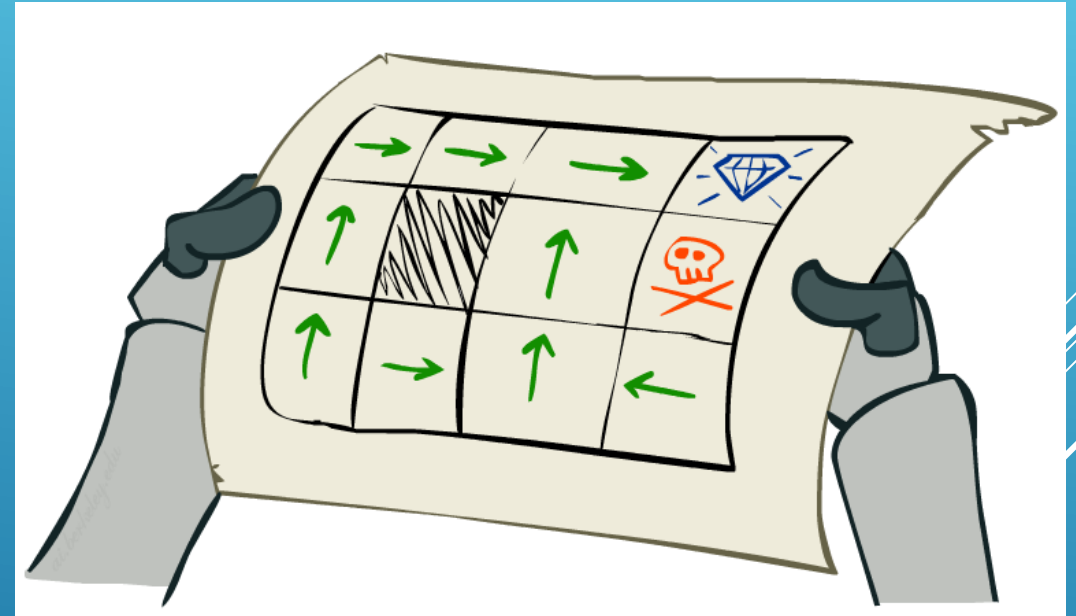
- ▶ This is just like search, where the successor function only depends on the current state (not the history)



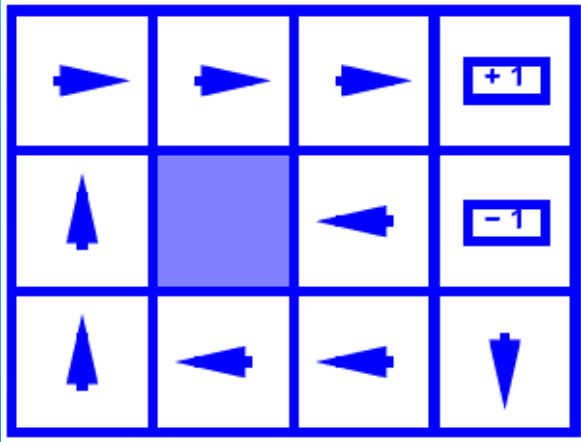
Andrey Markov
(1856-1922)

MDP GOAL: FIND AN OPTIMAL POLICY π

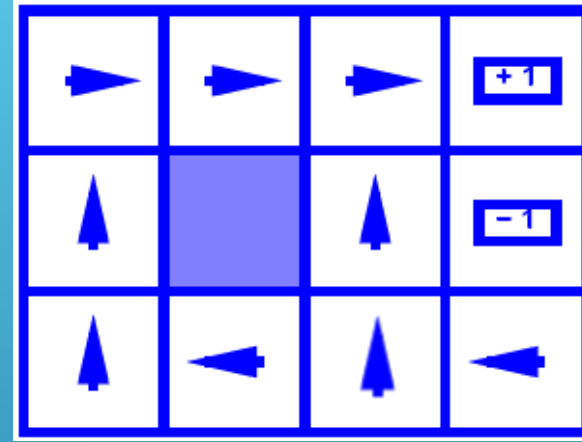
- ▶ In search problems, we look for an optimal **plan**, or sequence of actions, from start to a goal
- ▶ For MDPs, we want an optimal **policy** $\pi^*: S \rightarrow A$
 - ▶ A policy π gives an action for each state
 - ▶ An optimal policy is one that maximizes expected utility if followed



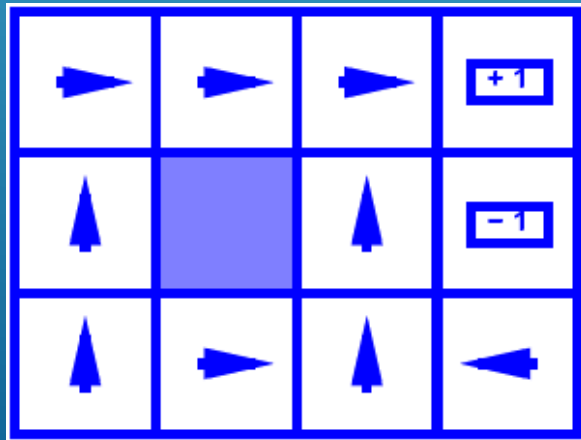
OPTIMAL POLICIES



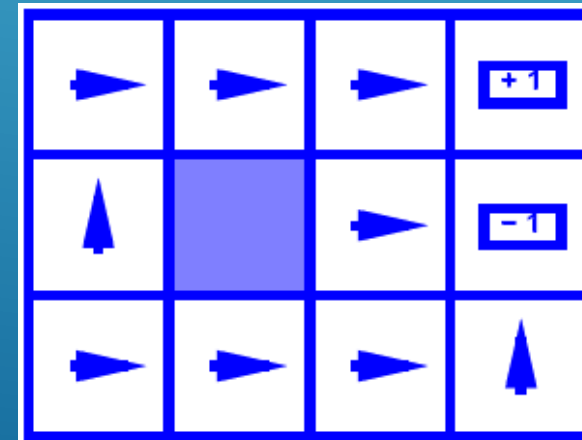
$$R(s) = 0.05$$



$$R(s) = 0.0$$



$$R(s) = -0.01$$



$$R(s) = -2.0$$

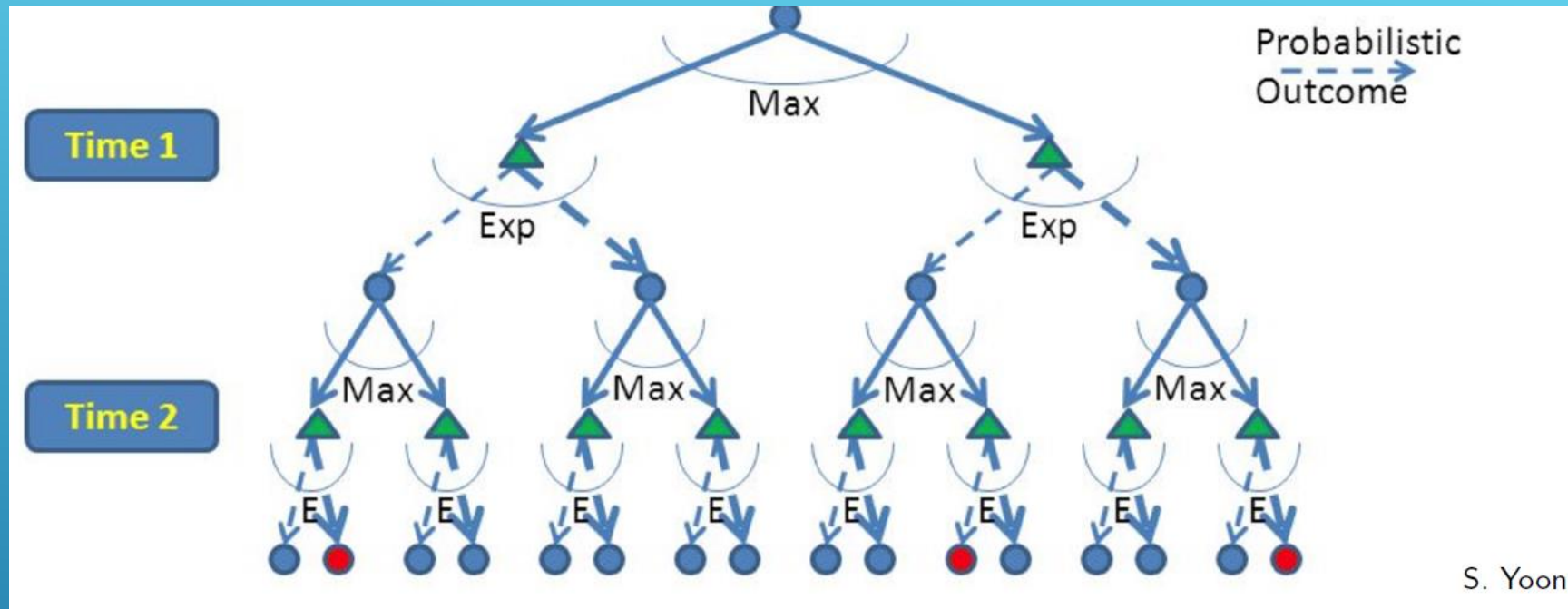
4 MDP ALGORITHMS





4 MDP ALGORITHMS

- Assume complete knowledge of the MDP
- Make use of the Bellman Equations
- **Expectimax** (recursive, finite horizon)
- **Value Iteration** (dynamic programming, finite horizon)
- **Value Iteration** (dynamic programming, infinite horizon)
- **Policy Iteration** (dynamic programming, infinite horizon, optimize policy)

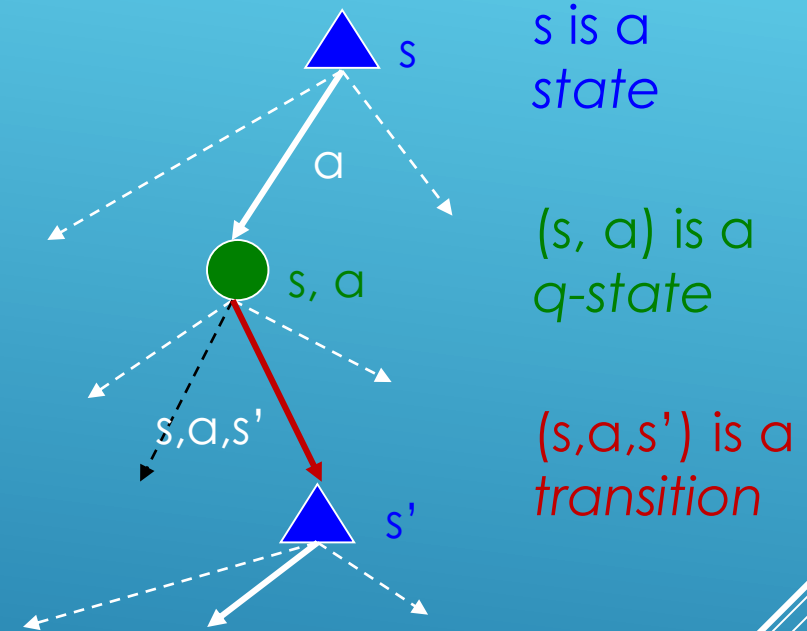
EXPECTIMAX: A GAME AGAINST NATURE



- 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**: $\langle S, A \rangle$ ()

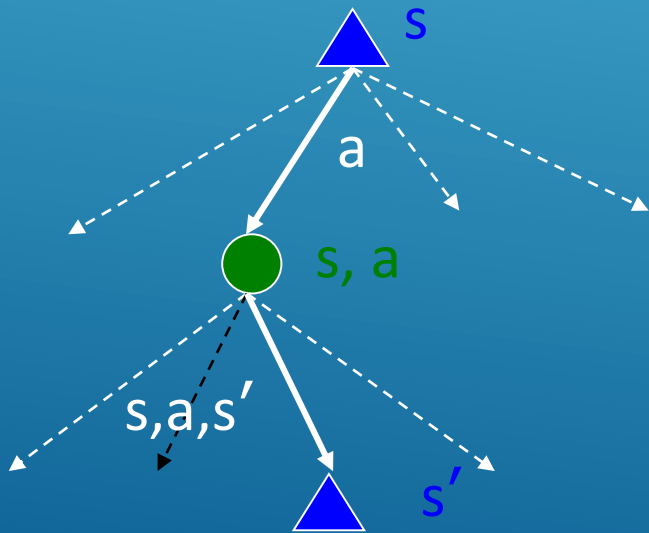
OPTIMAL QUANTITIES

- The value (utility) of a **state s** :
 $V^*(s)$ = expected utility starting in s and acting optimally
- The value (utility) of a **q-state (s,a)** :
 $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
 $\pi^*(s)$ = optimal action 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.

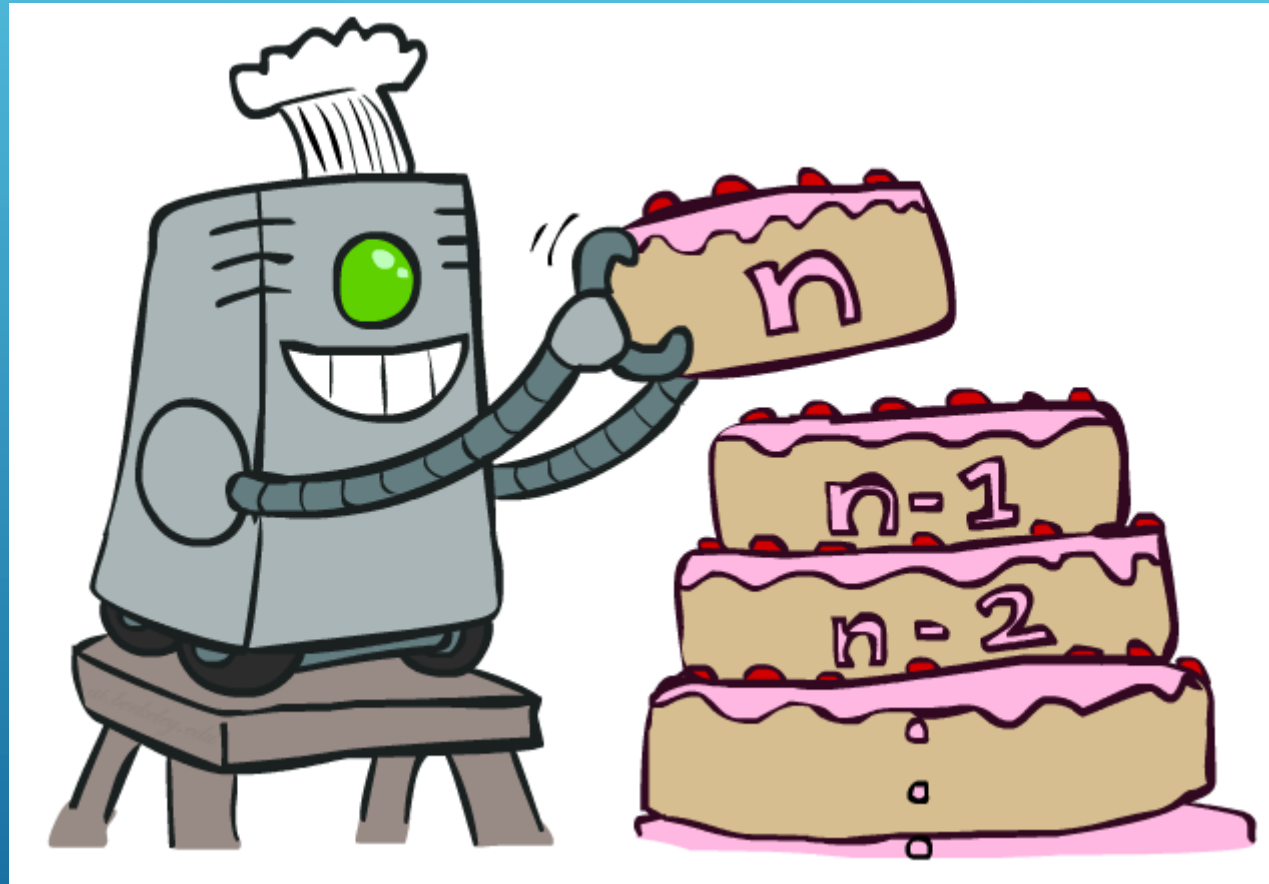


$$V^*(s) = \max_a Q^*(s, a)$$

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

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

VALUE ITERATION USES DYNAMIC PROGRAMMING

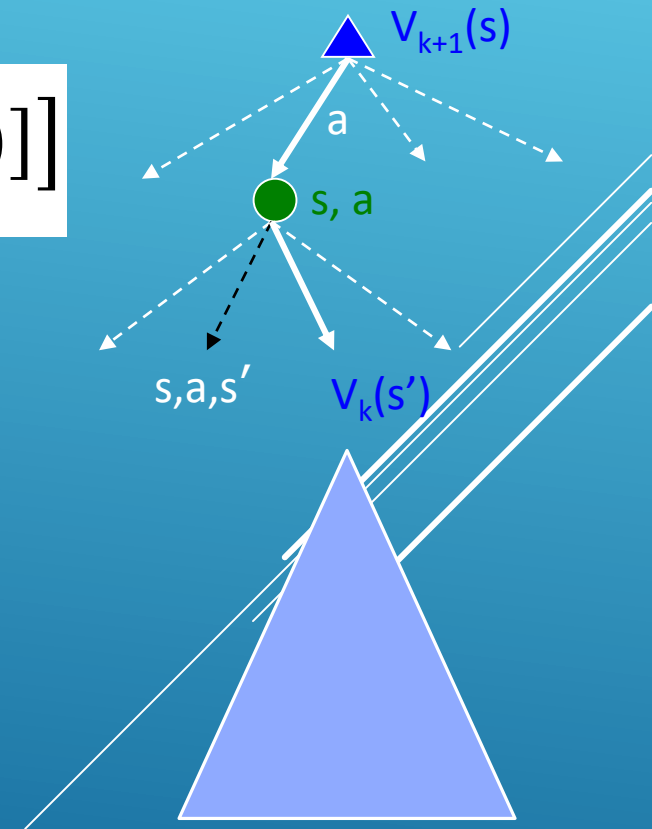


VALUE ITERATION

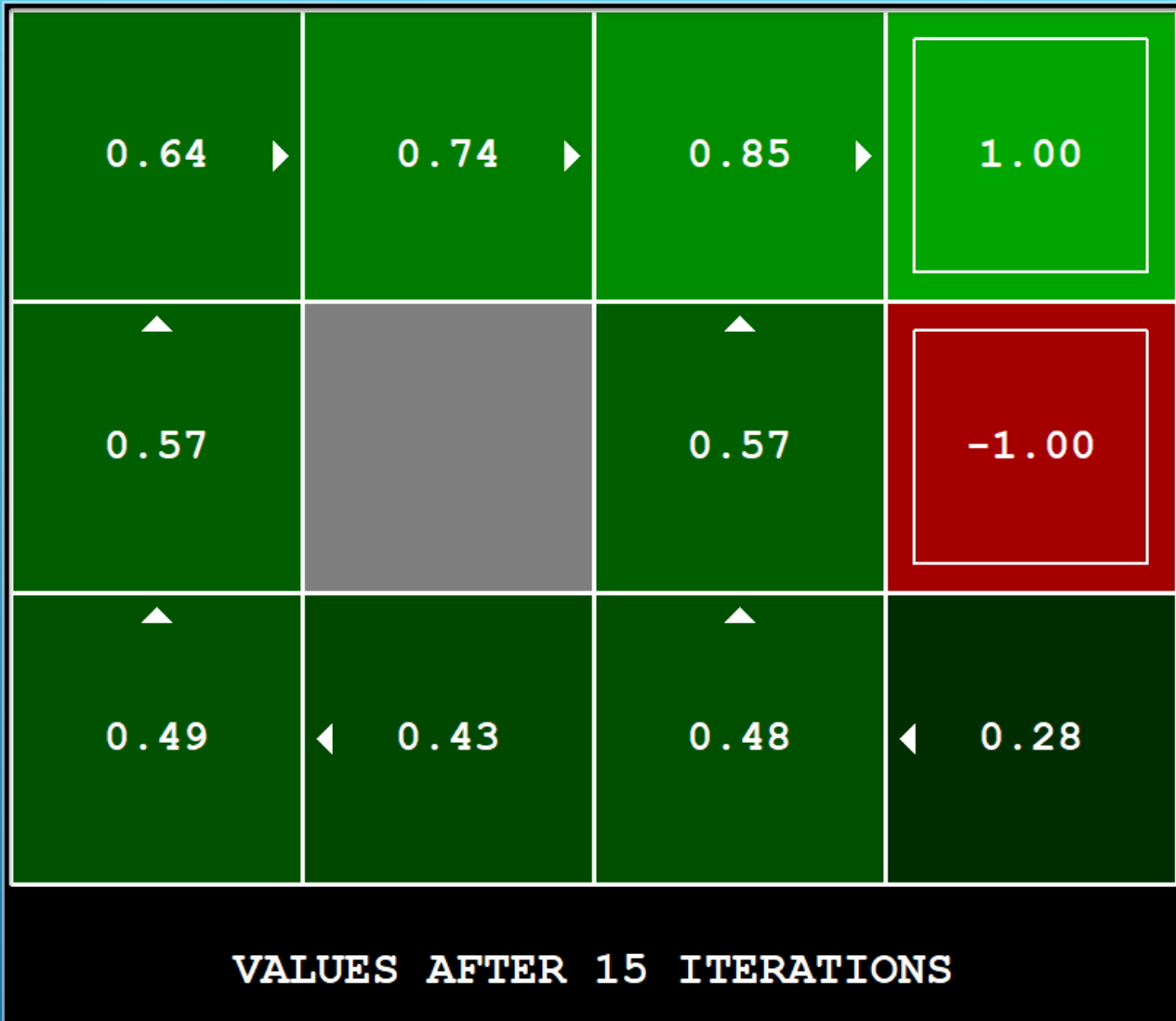
- ▶ Start with $V_0(s) = 0$ - no time steps left means an expected reward sum of zero
- ▶ Given vector of $V_k(s)$ values, do one ply from each state:

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

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



VALUE ITERATION EXAMPLE: GRIDWORLD



OPTIONS:

- h show this help message and exit
- d D discount on future (default 0.9)
- r reward for living for a time step (default 0.0)
- n P how often action results in unintended direction (default 0.2)
- e E chance of taking a random action in q-learning (default 0.3)
- l P TD learning rate (default 0.5)
- i R number of rounds of value iteration (default 10)
- k K number of episodes of the MDP to run (default 1)
- g G grid to use (case sensitive; options are: BookGrid, BridgeGrid, CliffGrid, CliffGrid2, MazeGrid, SmallGrid, MediumGrid, default BookGrid)
- w X request a window width of X pixels *per grid cell* (default 150)
- a A agent type (options are 'random', 'value' and 'q', default random)
- t use text-only ASCII display
- p pause GUI after each time step when running the MDP
- q skip display of any learning episodes
- s S speed of animation, $S > 1.0$ is faster, $0.0 < S < 1.0$ is slower (default 1.0)
- m manually control agent
- v display each step of value iteration

```
python gridworld.py -k 40 -w 200 -s 0.25 -i 15 -a value -p -v
```

POLICY ITERATION

- ▶ **Step 1: Policy Evaluation:** For fixed current policy π , find values with policy evaluation:

- ▶ Iterate until values converge:

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

- ▶ **Step 2: Improvement:** For fixed values, get a better policy using policy extraction:


- ▶ One-step look-ahead:

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

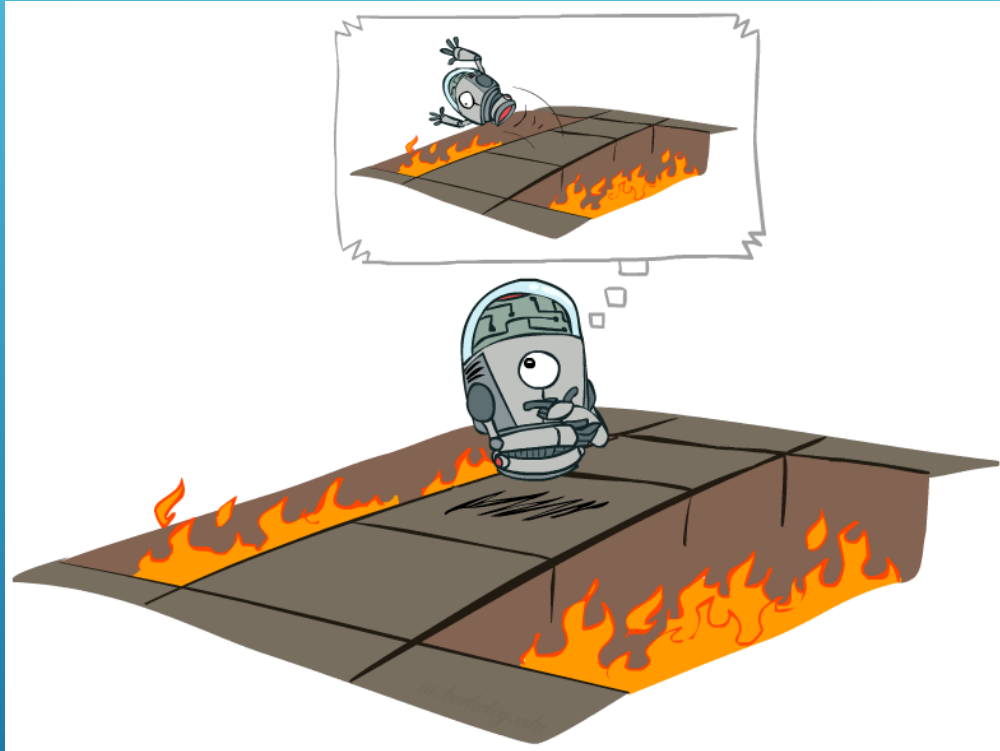
REINFORCEMENT LEARNING



REINFORCEMENT LEARNING: THE BASIC IDEA

- Select an action
 - If action leads to reward, reinforce that action
 - If action leads to punishment, avoid that action
 - Basically, a computational form of Behaviorism (Pavlov, B. F. Skinner)
- 
- A series of white lines of varying lengths and orientations are positioned on the right side of the slide, extending from the middle to the bottom right corner.

OFFLINE VS. ONLINE



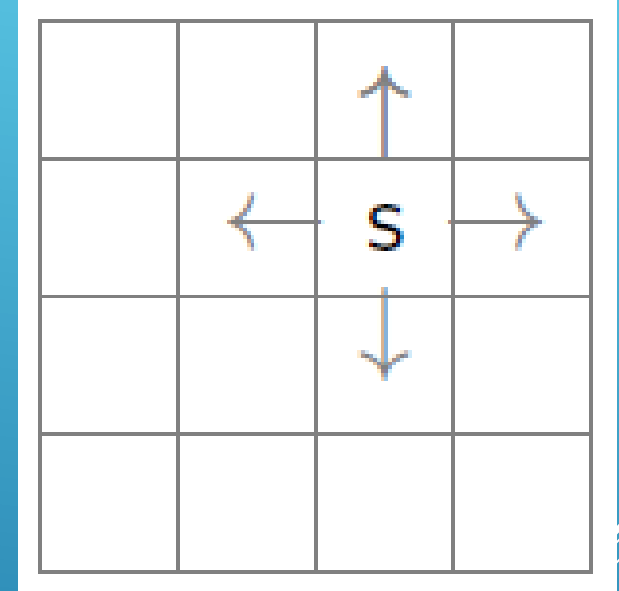
Offline Learning



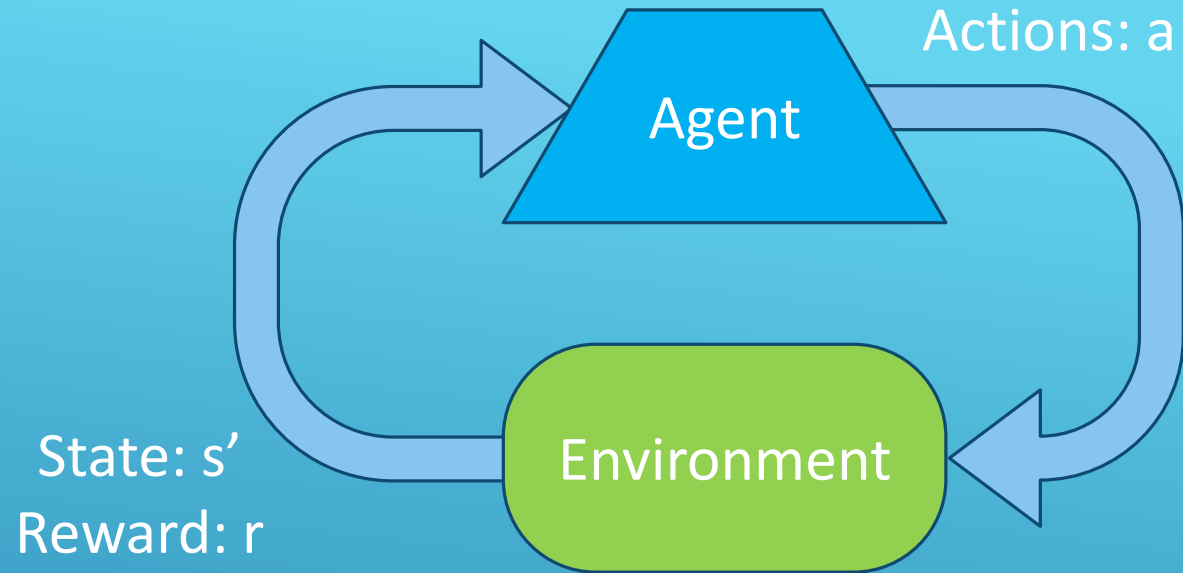
Online Learning

THE LEARNING FRAMEWORK

- Learning is performed **online**, learn as we interact with the world
- In contrast with supervised learning, there are no training or test sets. The reward is accumulated over interactions with the environment.
- Data is not fixed, more information is acquired as you go.
- The training distribution can be influenced by action decisions.



REINFORCEMENT LEARNING



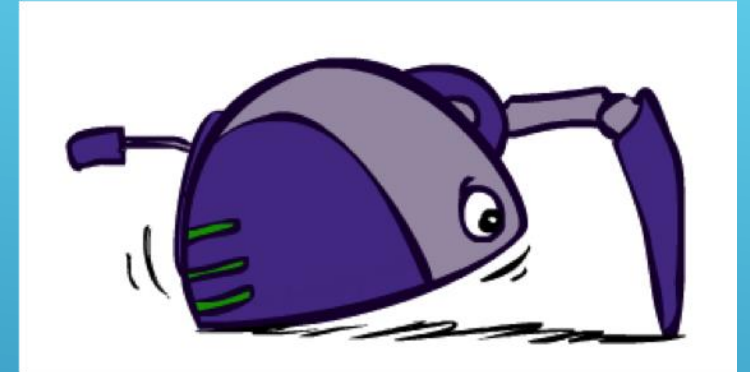
- Agent knows the current state s , takes action a , receives a reward r and observes the next state s'
- Agent has **no access** to reward model $r(s,a,s')$ or the transition model $p(s' | s,a)$
- Agent learn to act so as to maximize expected rewards.
- All learning is based on observed samples of outcomes.

MODEL-BASED REINFORCEMENT LEARNING



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 or policy iteration
- ▶ Repeat



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: PROS AND CONS

○ **Pros:**

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


○ **Cons:**

- Assumes model is small enough to solve
 - Requires expensive solution procedure
- 
- A series of white diagonal lines of varying lengths and thicknesses, located in the bottom right corner of the slide.

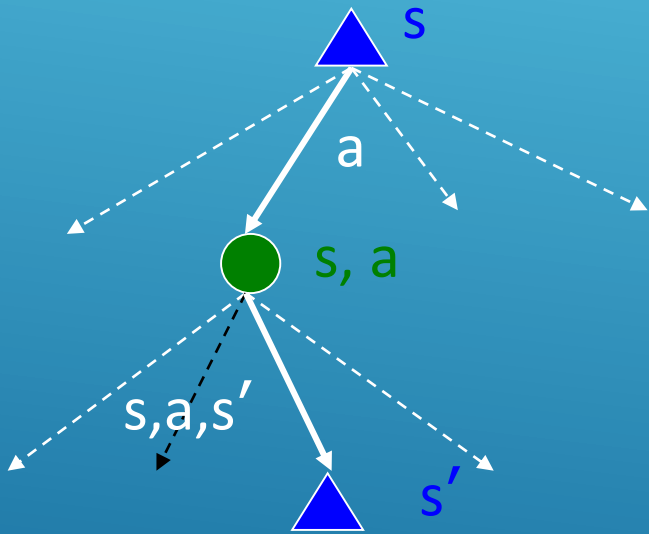
MODEL-FREE REINFORCEMENT LEARNING: Q-LEARNING



Q-LEARNING

- Don't learn a model, learn the Q function directly
 - Appropriate when model is too large to store, solve or learn
 - size of transition of model: $O(|S|^2)$
 - value iteration cost: $O(|A||S|^2)$
 - size of Q function $O(|A||S|)$
- 
- A series of three parallel white diagonal lines are located in the bottom right corner of the slide, extending from the middle of the right edge towards the bottom left.

RECALL THE BELLMAN EQUATIONS



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

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

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

FROM VALUE ITERATION TO Q-VALUE ITERATION

- ▶ Value iteration: find successive (depth-limited) values

- ▶ Start with $V_0(s) = 0$, which we know is right
- ▶ Given V_k , calculate the depth $k+1$ values for all states:

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

- ▶ But Q-values are more useful, so compute them instead

- ▶ Start with $Q_0(s,a) = 0$, which we know is right
- ▶ Given Q_k , calculate the depth $k+1$ q-values for all q-states:

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

Q-LEARNING

- ▶ Q-Learning: sample-based Q-value iteration

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

- ▶ Learn $Q(s,a)$ values as you go

- ▶ Receive a sample transition (s,a,r,s')
- ▶ Consider your old estimate: $Q(s, a)$
- ▶ Consider your new sample estimate:

$$Q(s, a) \approx r + \gamma \max_{a'} Q(s', a')$$

- ▶ Incorporate the new estimate into a running average:

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

Q-LEARNING UPDATE RULE

- ▶ On transitioning from state s to state s' on action a , and receiving reward r , update:

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

- ▶ α is the **learning rate**
- ▶ a large α results in quicker learning, but may not converge.
- ▶ α is often decreased as learning goes on.

TEMPORAL DIFFERENCE (TD) LEARNING

- ▶ Q-Learning is an example of a more general approach to learning called **Temporal Difference Learning**

- ▶ In TD learning, the general update is:

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

- ▶ Rewrite Q-Learning update:

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

$$Q(s, a) \leftarrow Q(s, a) + \alpha[(r + \gamma \max_{a'} Q(s', a')) - Q(s, a)]$$

- ▶ The idea is to repeatedly nudge the new estimate towards its learning target in each time period.
- ▶ Another example of TD Learning is TD-Value Learning, which learns the value function $V(s)$ instead of $Q(s, a)$.

TD LEARNING AND EXPONENTIAL SMOOTHING

- ▶ The TD Learning/Q-Learning update rule is similar to a times series technique called *exponential smoothing*.
- ▶ the simplest form of exponential smoothing is given by the formulas:

$$s_0 = x_0$$

$$s_t = \alpha x_t + (1 - \alpha)s_{t-1}, t > 0$$

where α is the **decay rate** (as opposed to the **step size** or **learning rate**)

EXPONENTIAL SMOOTHING (2)

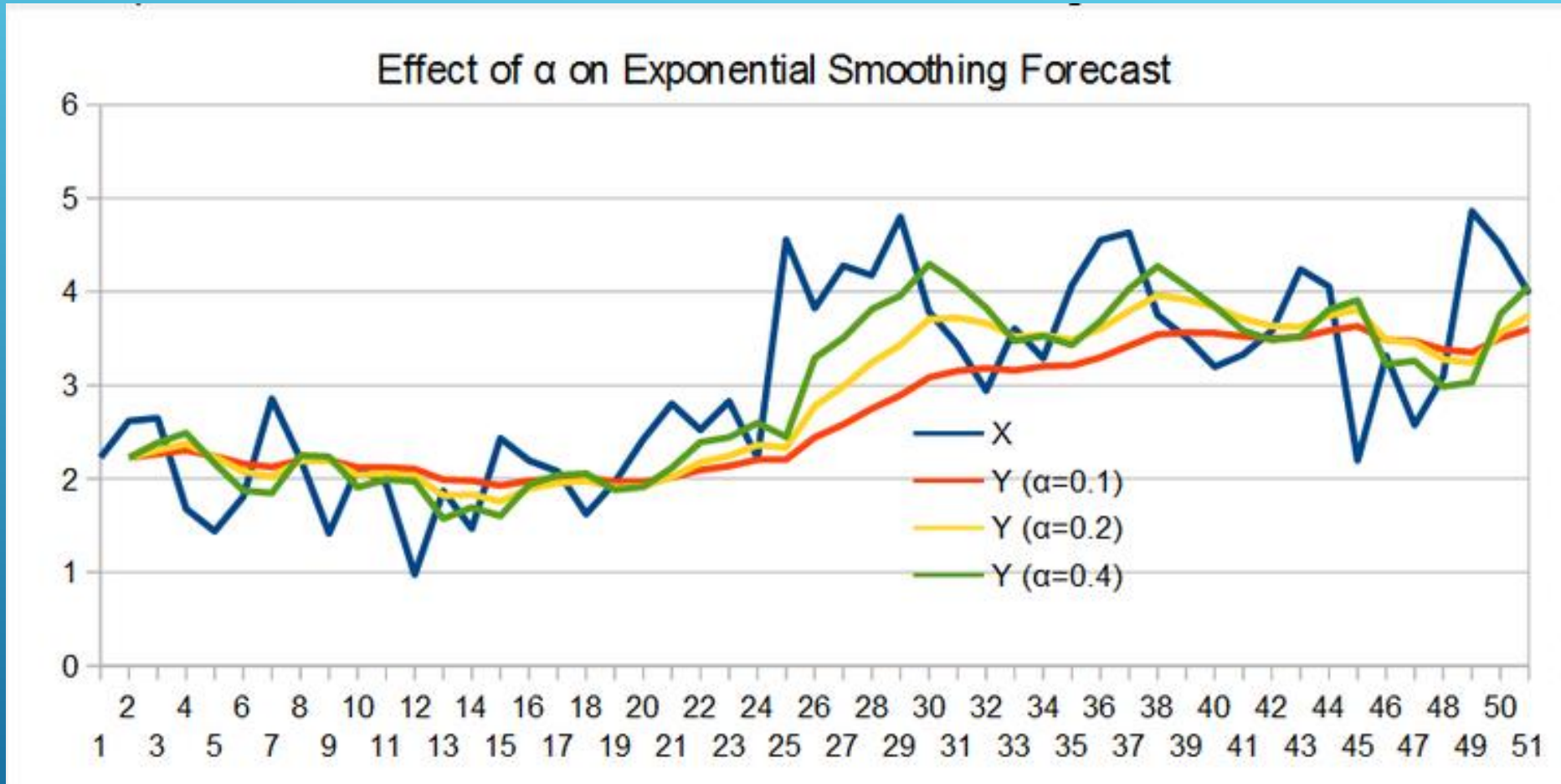
As time progresses, the affect on s_t of more remote terms decay exponentially as they recede into the past.

$$\begin{aligned}s_0 &= x_0 \\ s_t &= \alpha x_t + (1 - \alpha)s_{t-1}, \quad t > 0\end{aligned}$$

The above equations can be expanded thus:

$$\begin{aligned}s_t &= \alpha x_t + (1 - \alpha)s_{t-1} \\ &= \alpha x_t + \alpha(1 - \alpha)x_{t-1} + (1 - \alpha)^2 s_{t-2} \\ &= \alpha [x_t + (1 - \alpha)x_{t-1} + (1 - \alpha)^2 x_{t-2} + (1 - \alpha)^3 x_{t-3} + \cdots + (1 - \alpha)^{t-1} x_1] + (1 - \alpha)^t x_0.\end{aligned}$$

EXPONENTIAL SMOOTHING EXAMPLE



Notice how Curves become more “wiggly” as α increases.

Q-LEARNING ALGORITHM

For each state s and action a :

$$Q(s, a) \leftarrow 0$$

Begin in state s :

Repeat:

For all actions associated with state s ,
→ **CHOOSE ACTION a** ← based on the
 Q values for state s

Receive reward r and transition to s'

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

$$s \leftarrow s'$$

CHOOSING THE ACTION

- Learned Q function determines the policy
 - in state s , choose action with largest $Q(s,a)$
- Still have to worry about exploration vs. exploitation.
 - use techniques we discussed previously.

CHOOSING AN ACTION: EXPLORATION VS EXPLOITATION

- **Exploit:** use your current model to maximize the expected utility now.
- **Explore:** choose an action that will help you improve your model.
- **How to Exploit?** use the current policy.
- **How to Explore?**
 - choose an action randomly
 - choose an action you haven't chosen yet
 - choose an action that will take you to an unexplored state.

EXPLORATION STRATEGY: ϵ -GREEDY

- Explore with probability ϵ . Exploit with probability $1 - \epsilon$.
- Weaknesses:
 - Does not exploit when learning has converged.
- Uses:
 - appropriate if the world is changing.

EXPLORATION STRATEGY: BOLTZMANN

- In state s , choose action a with probability p :

$$p = \frac{e^{\frac{Q(s,a)}{t}}}{\sum_{a'} e^{\frac{Q(s,a')}{t}}}$$

- Simulated annealing: t is a “temperature”
- High temperature means more exploration
- Over time, t cools, reducing exploration
- Sensitive to cooling schedule.

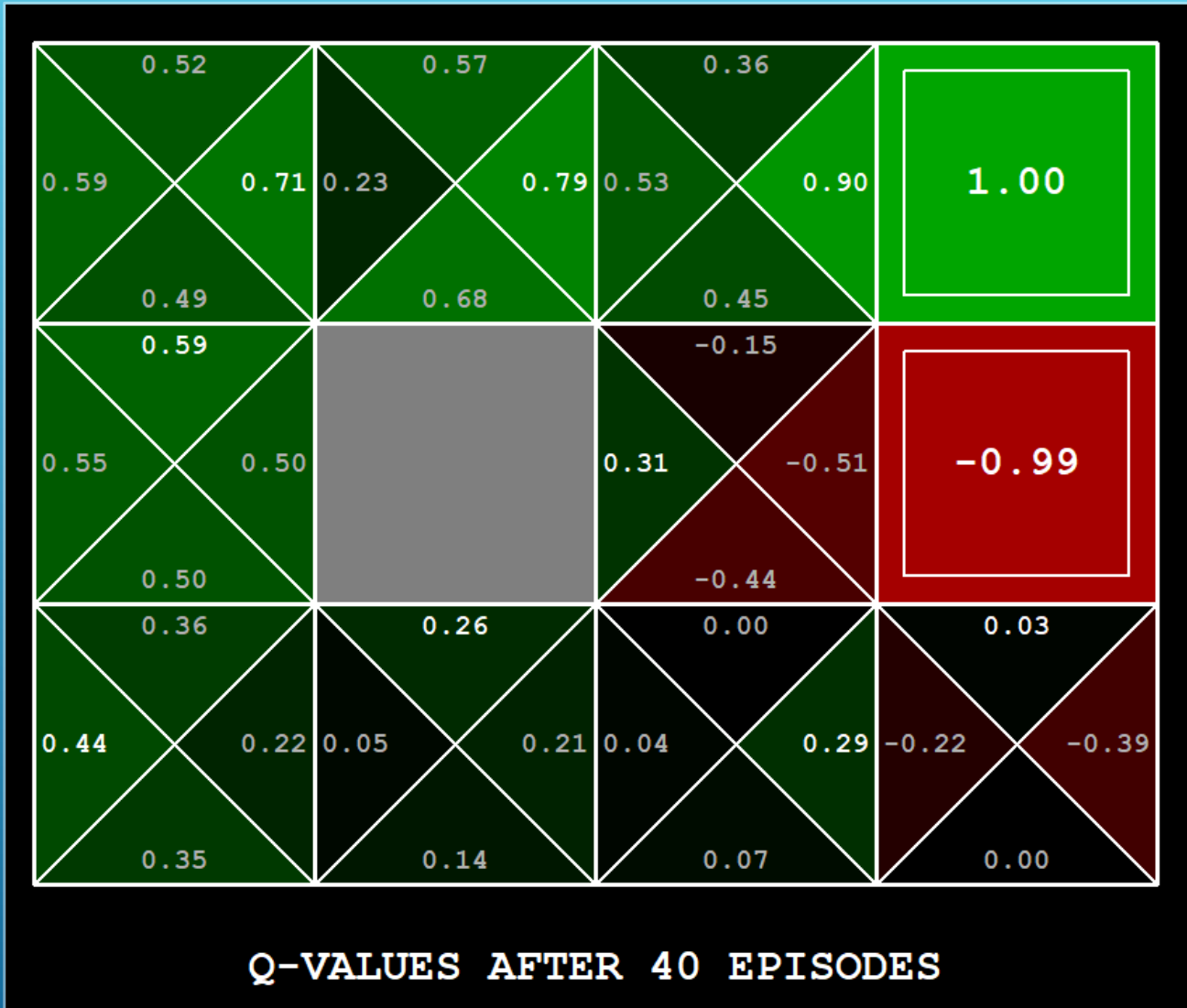
EXPLORATION STRATEGY: R-MAX

- Initialize reward for each state to R_{max} , the largest reward possible
- Keep track of the number of times each state has been visited
- After c visits, mark the state as known and update and update reward and transition probabilities.
- Need to know R_{max}



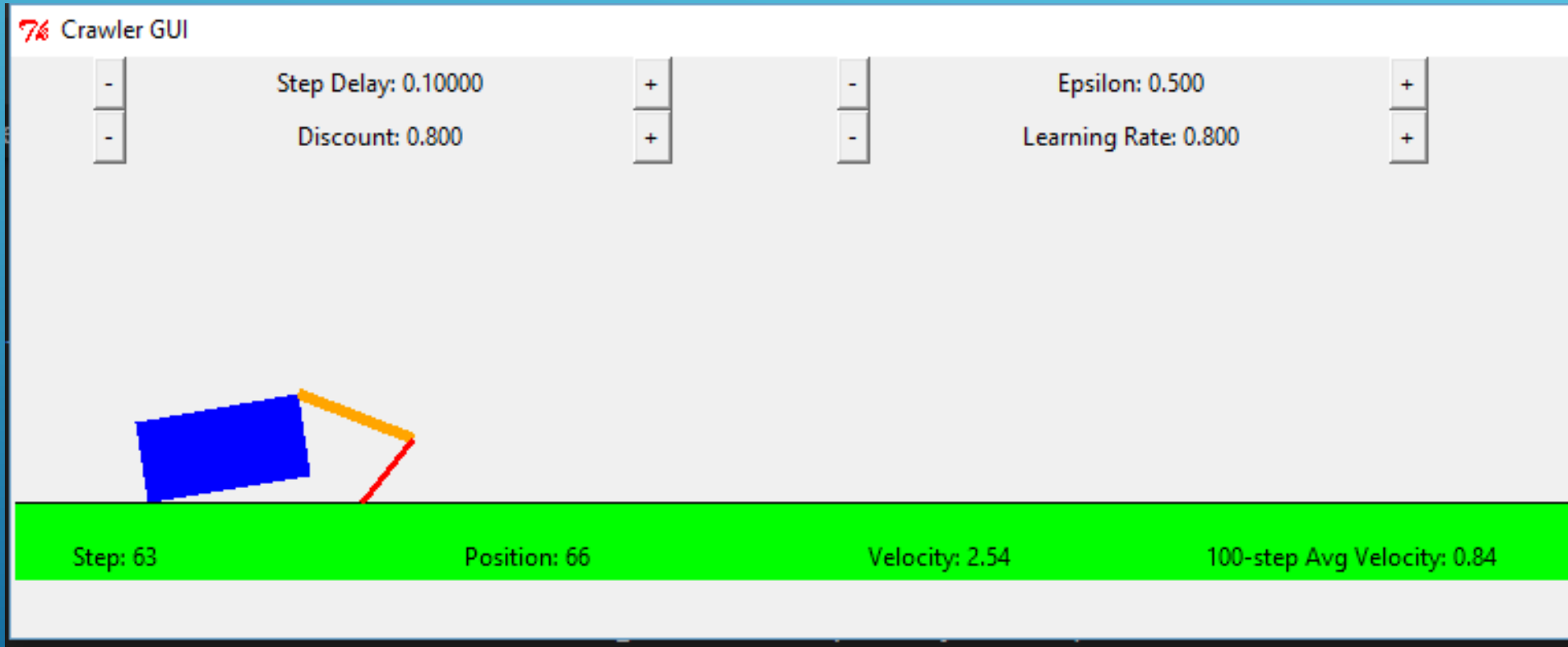
VIDEO OF DEMO Q-LEARNING -- GRIDWORLD

Q-LEARNING EXAMPLE: GRIDWORLD



```
python gridworld.py -k 40 -w 200 -s 2.0 -a q -v [-p]
```

Q-LEARNING EXAMPLE: CRAWLER ROBOT




Q-LEARNING EXAMPLE: CRAWLER ROBOT

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

Description	Training Steps	Discount (γ)	Learning Rate (α)	Avg Velocity
Default	~100K	0.8	0.8	~1.73
Low γ	~100K	0.5	0.8	0.0
High γ	~100K	0.919	0.8	~3.33
Low α	~100K	0.8	0.2	~1.73
High α	~100K	0.8	0.9	~1.73
High γ , Low α	~100K	0.919	0.2	~3.33

NEXT TIME: MORE RL

- Possible Topics:
 - SARSA – Q-Learning alternative
 - More about Generalization
 - Partially Observable Markov Decision Processes (POMDPs)
 - Deep Q-Learning
- 
- A series of white diagonal lines of varying lengths and thicknesses, located in the bottom right corner of the slide, creating a modern, abstract graphic element.