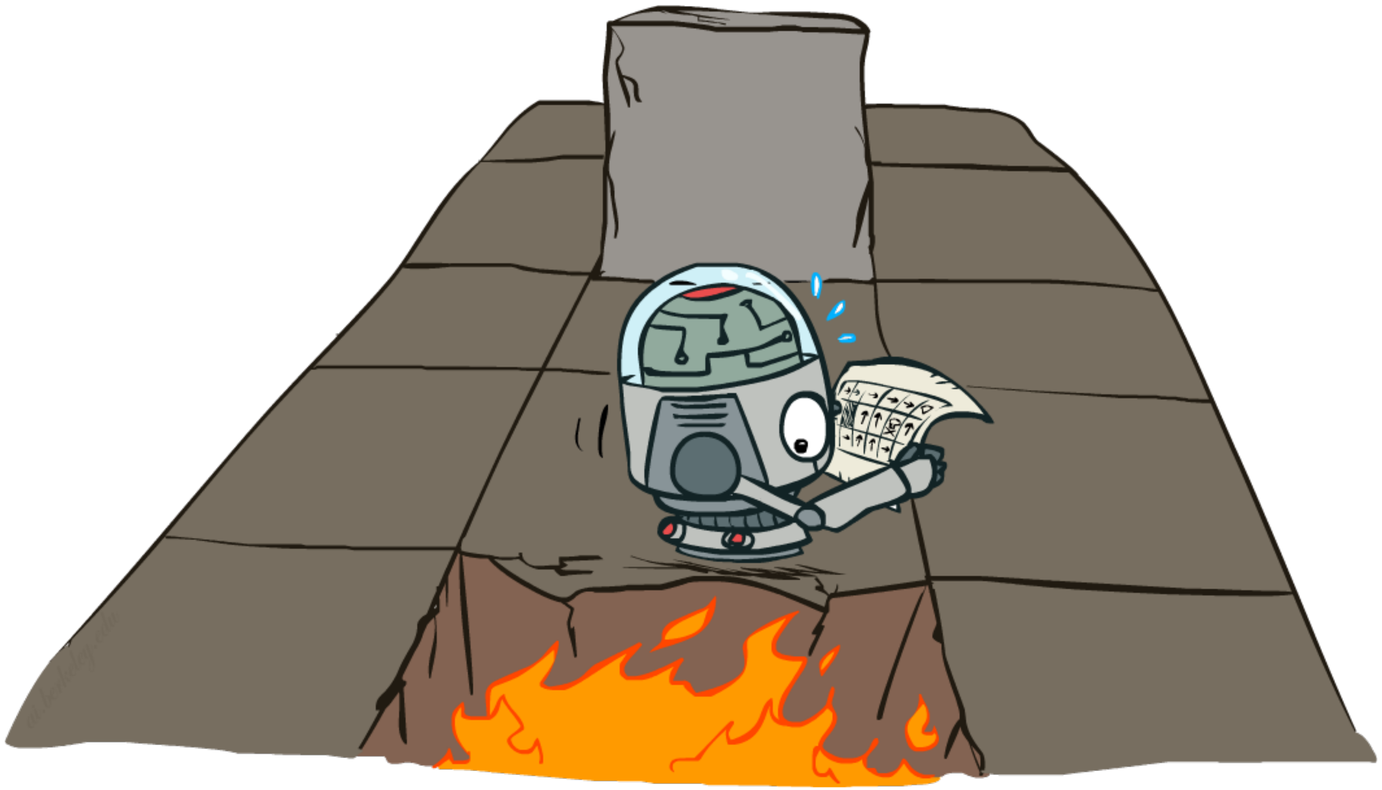
CS 188: Artificial Intelligence

Markov Decision Processes III + RL

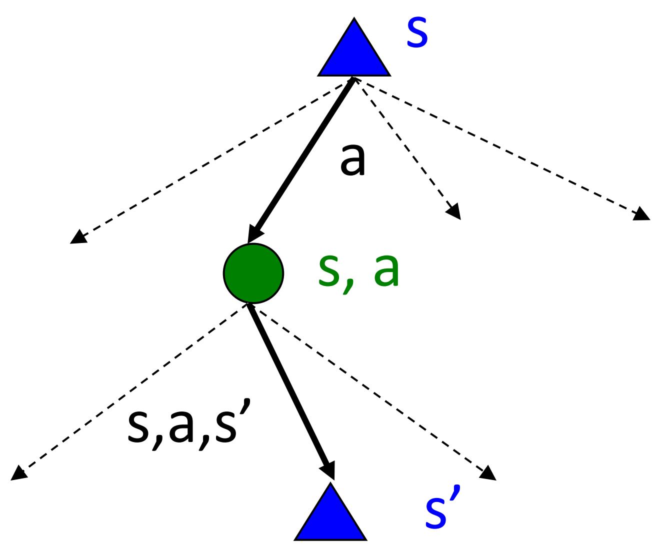


Instructor: Nathan Lambert

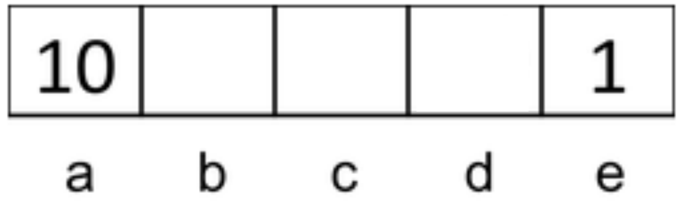
University of California, Berkeley

Recap: Defining MDPs

- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s' | s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards

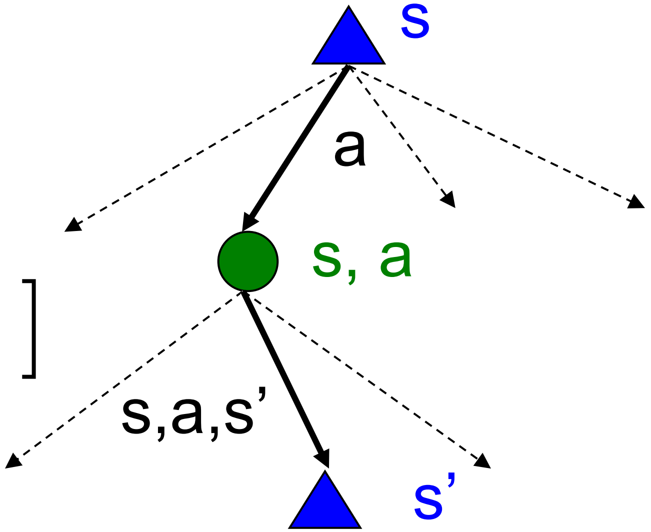


Values of States

- Recursive definition of value:

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

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

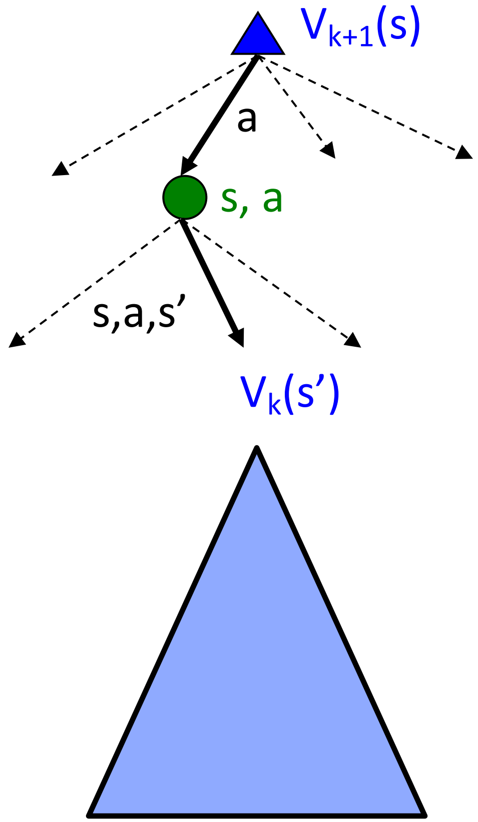


$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

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 of expectimax from each state:

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



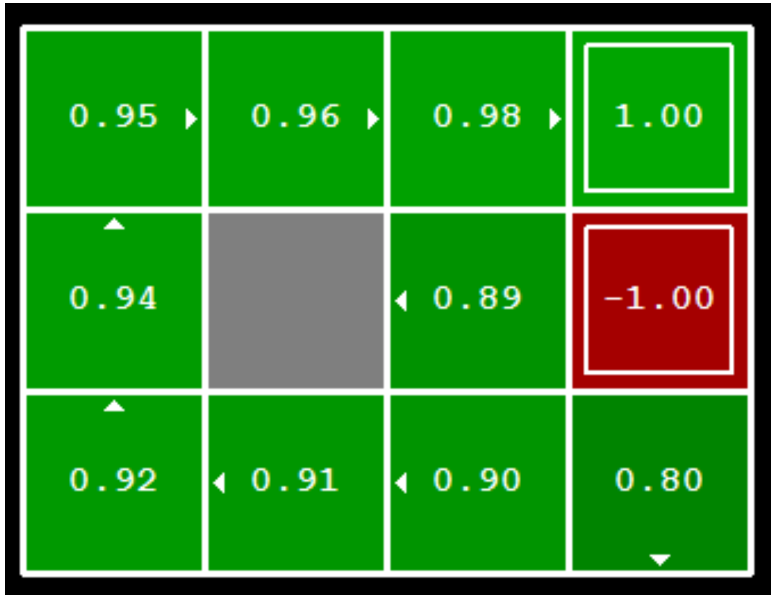
- Repeat until convergence
- (Complexity of each iteration: $O(S^2A)$)

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)

- How should we act?
  - It's not obvious!

- We need to do a mini-expectimax (one step)



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

- This is called policy extraction, since it gets the policy implied by the values

Policy Evaluation

- How do we calculate the V's for a fixed policy π?

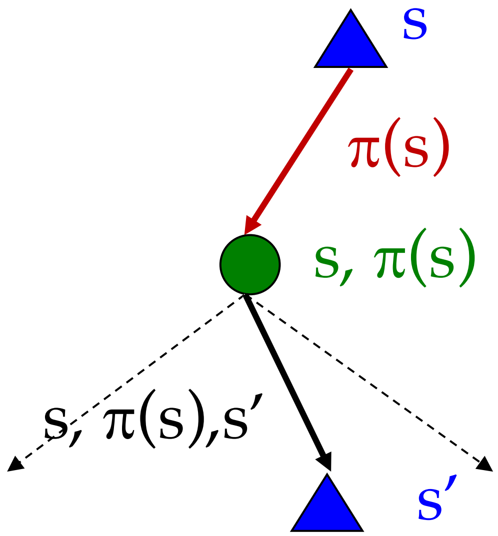
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

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

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

- Efficiency: O(S²) per iteration

- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)



Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

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

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

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

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs

Convergence when Solving MDPs

- Redefine value update as general Bellman Utility update
  - Recursive update or utility (sum of discounted reward)

$$V^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V^{\pi_i}_k(s')\right]$$

$$\downarrow$$

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s,a) U_i(s')$$

- How does this converge?
  - Assume fixed policy $\pi_i(s)$.
  - *R(s)* is the short term reward of being in *s*

Convergence when Solving MDPs

- How does this update rule converge?

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum P(s'|s,a) U_i(s')$$

- Re-write update: $U_{i+1} \leftarrow BU_i$ $\quad v' = Av$
  - B is a linear operator (like a matrix)
  - U is a vector
- Interested in delta between Utilities: $\|U_{i+1} - U_i\|$

$$\|BU_{i+1} - BU_i\| \leq \gamma \|U_{i+1} - U_i\|$$

Convergence when Solving MDPs

- How does this delta converge?

$$\|BU_{i+1} - BU_i\| \le \gamma \|U_{i+1} - U_i\|$$

  - Utility error estimate reduced by γ each iteration:
- Total Utilities are bounded,

$$\sum_{i=0}^{\infty} R_{max}\gamma^i \qquad \pm \frac{R_{max}}{(1-\gamma)}$$

- Consider minimum initial error: $\|U_0 - U\| \le \dfrac{2R_{max}}{(1-\gamma)}$
  - (Max norm)
- Max error: reduce by discount each step.

Utility Error Bound

- Error at step 0:

$$\|U_0 - U\| \leq \frac{2R_{max}}{(1-\gamma)}$$

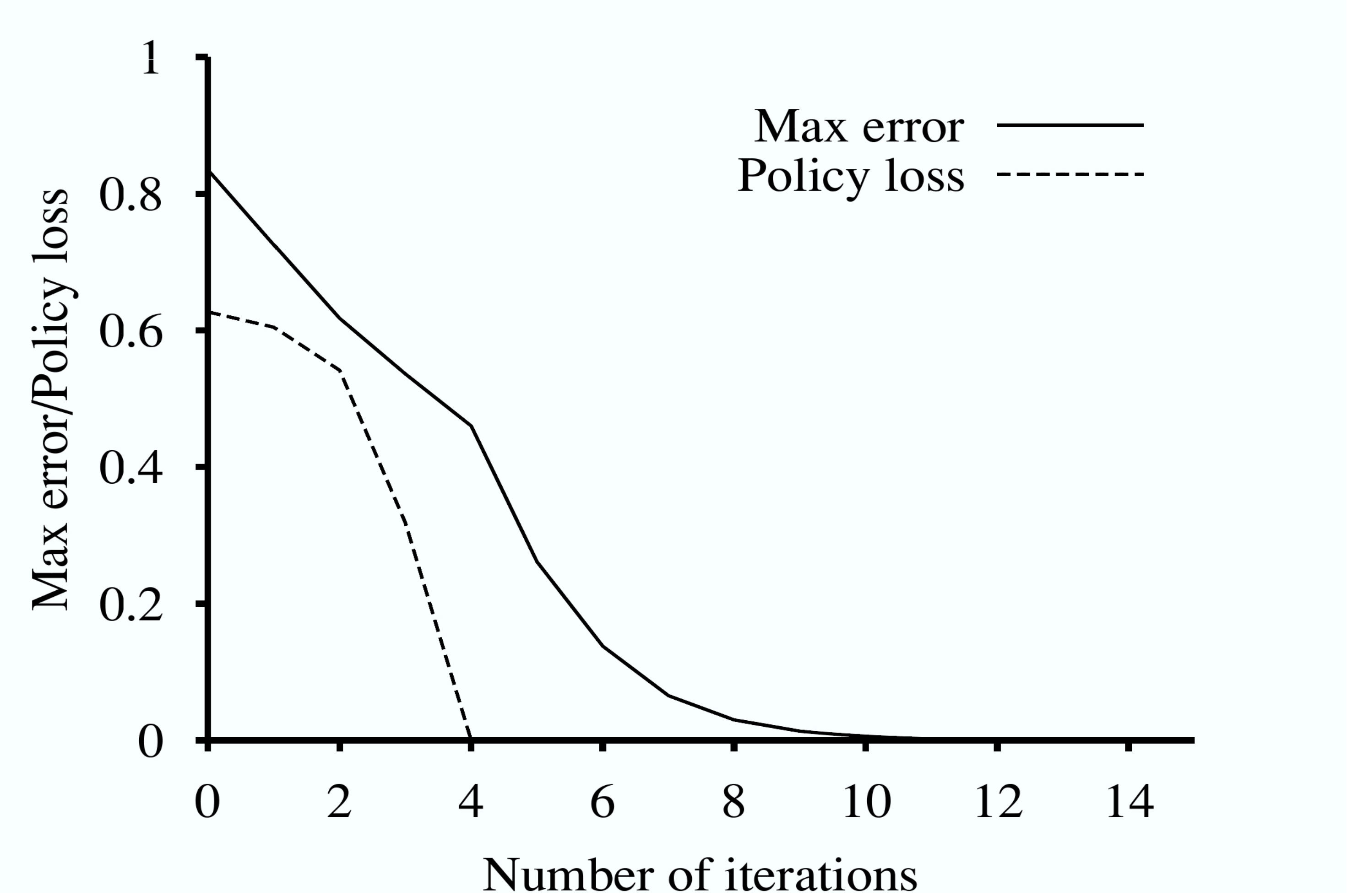
- Error at step N: $$\|U_N - U\| = \gamma^N \cdot \frac{2R_{max}}{(1-\gamma)} < \epsilon$$

- Steps for error below ϵ:

$$N = \frac{log\left(\frac{2R_{max}}{\epsilon(1-\gamma)}\right)}{log\left(\frac{1}{\gamma}\right)}$$

MDP Convergence Visualized

- Value iteration converges exponentially (with discount factor)
- Policy iteration will converge linearly to 0.



Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction (one-step lookahead)

- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

Partially Observed MDPs

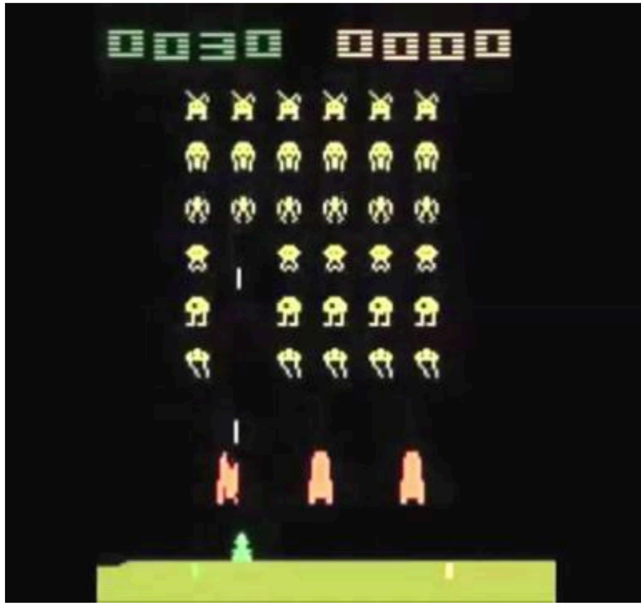
- How accurate is our model of MDPs?
  - Example: Robot
  - Example: Video game
- Do the pixels constitute all the information of a system?
  - Notion of observing some states!
  - Belief distribution
- Partially Observed MDP (POMDP)

Partially Observed MDPs

- How accurate is our model of MDPs?
  - Example: Robot
  - Example: Video game
- Do the pixels constitute all the information of a system?
  - Notion of observing some states!
  - Belief distribution over true states (from observations)

- Partially Observed MDP (POMDP)

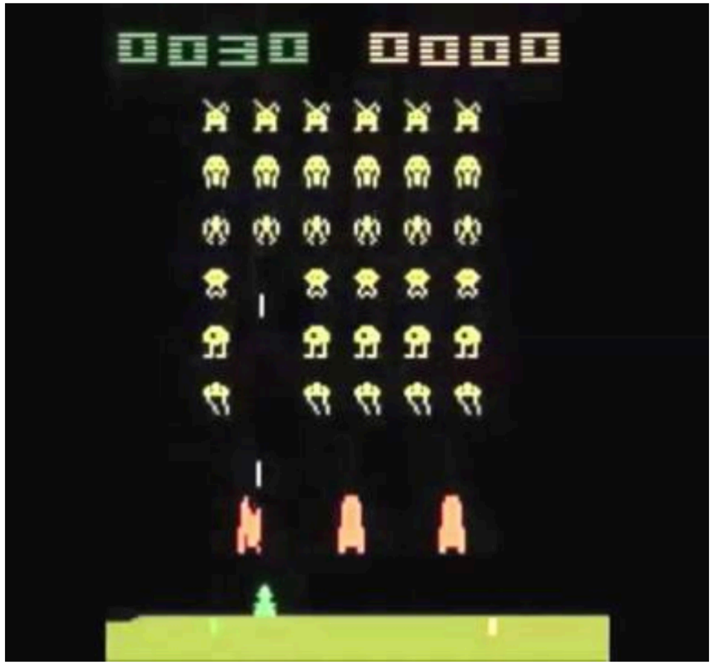
Partially Observed MDPs

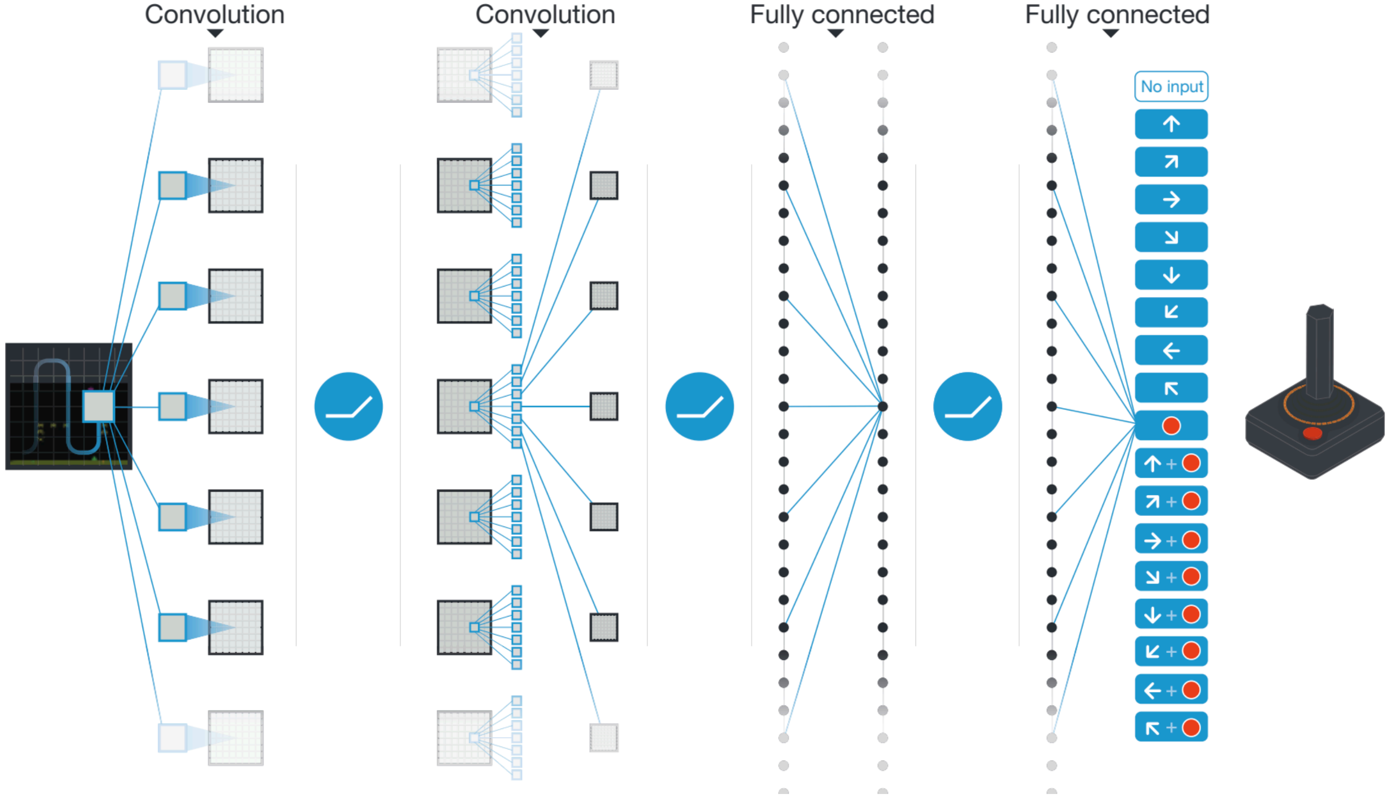


- Consider an example:
  - Finite number of pixels
  - Finite number of actions
  - Finite number of timesteps
  - Huge state-space.
- Can we solve an MDP with observations instead of states?
  - How is this solved?
  - Deep Q-learning (approximate Q values)
  - Are the transitions and reward functions known?

Seminal Paper

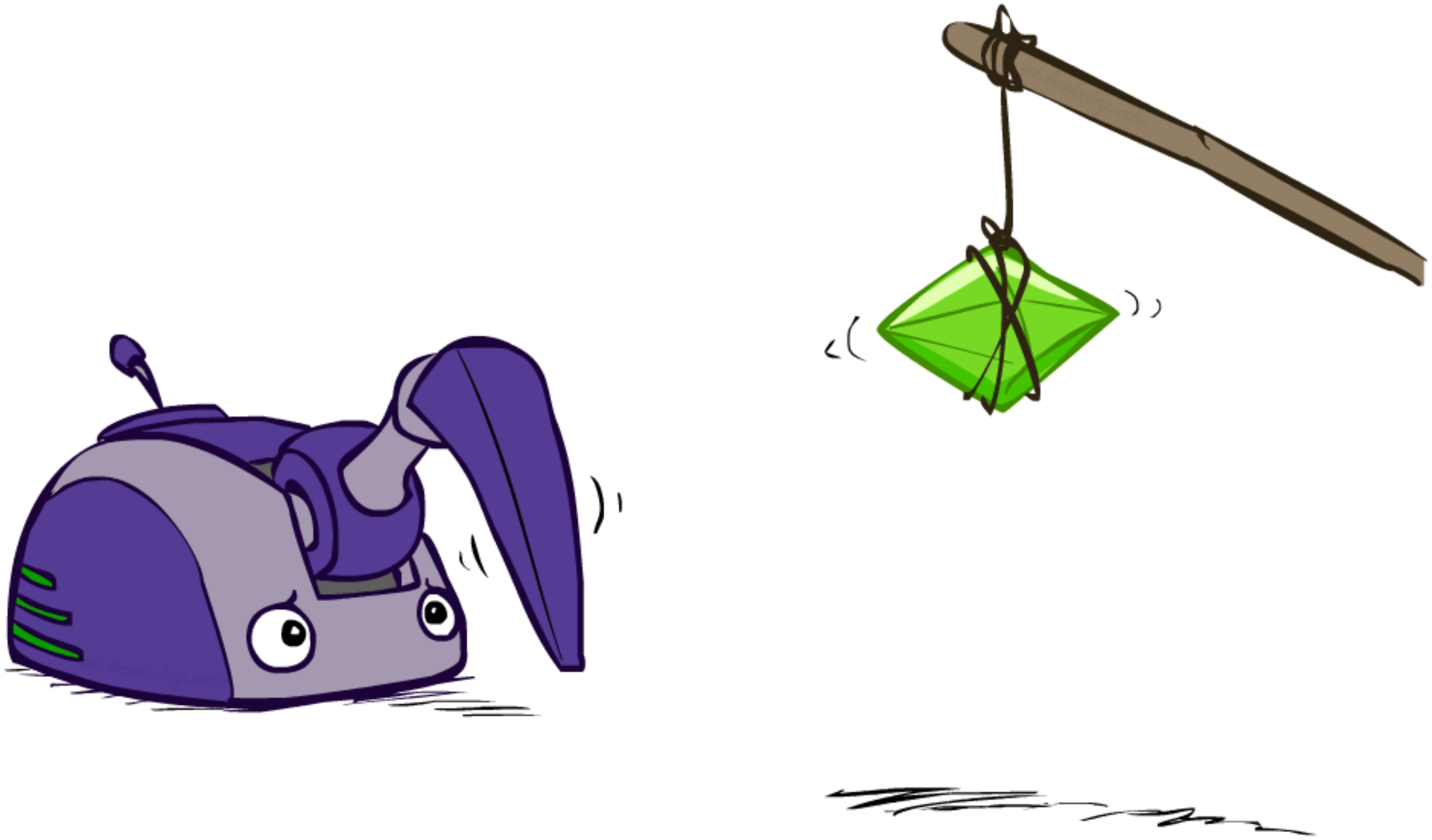
- Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.



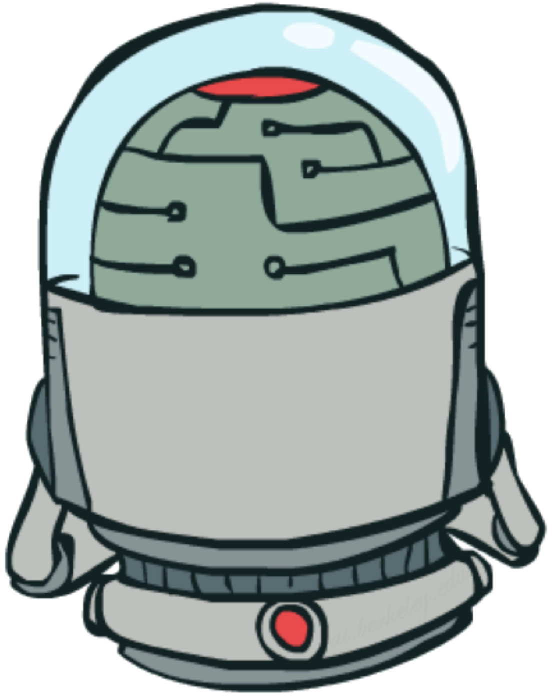


DeepMind Atari (©Two Minute Lectures)

Reinforcement Learning

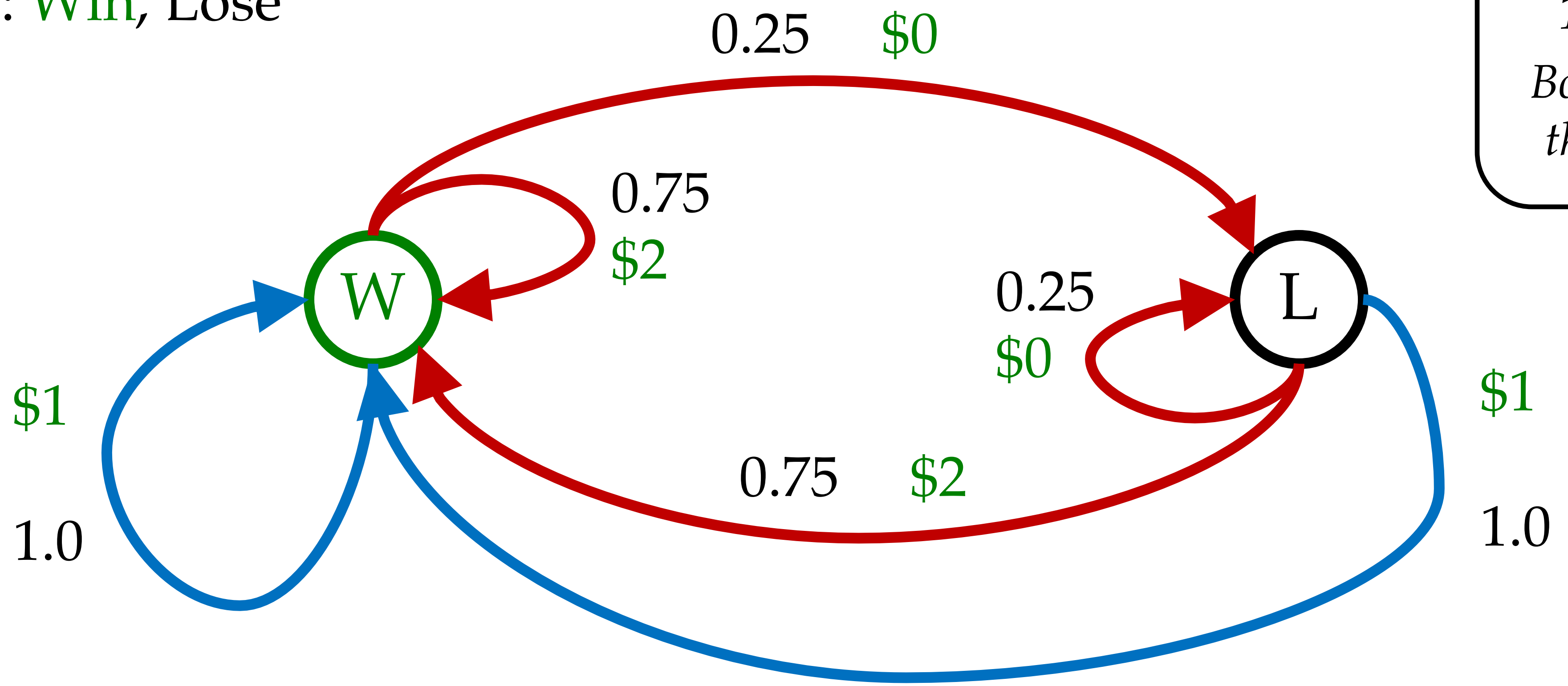


Double Bandits



Double-Bandit MDP

- Actions: *Blue*, *Red*
- States: Win, Lose



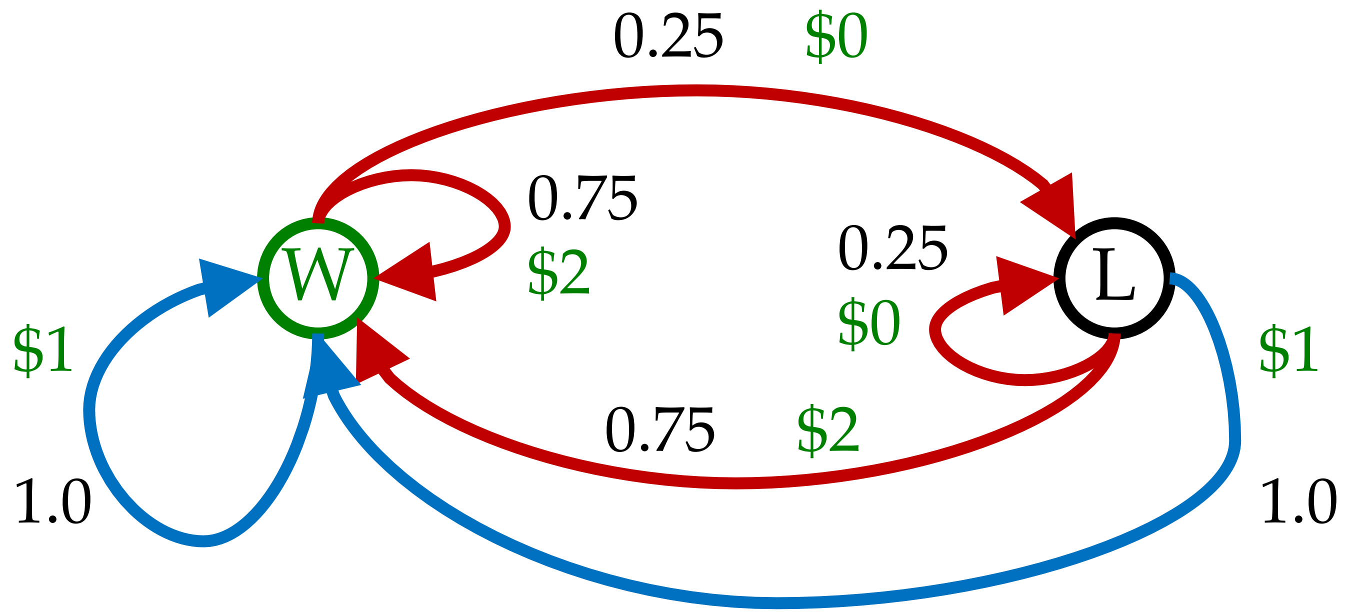
No discount
10 time steps
Both states have the same value

Offline Planning

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

No discount
10 time steps
Both states have the same value

	Value
Play Red	15
Play Blue	10



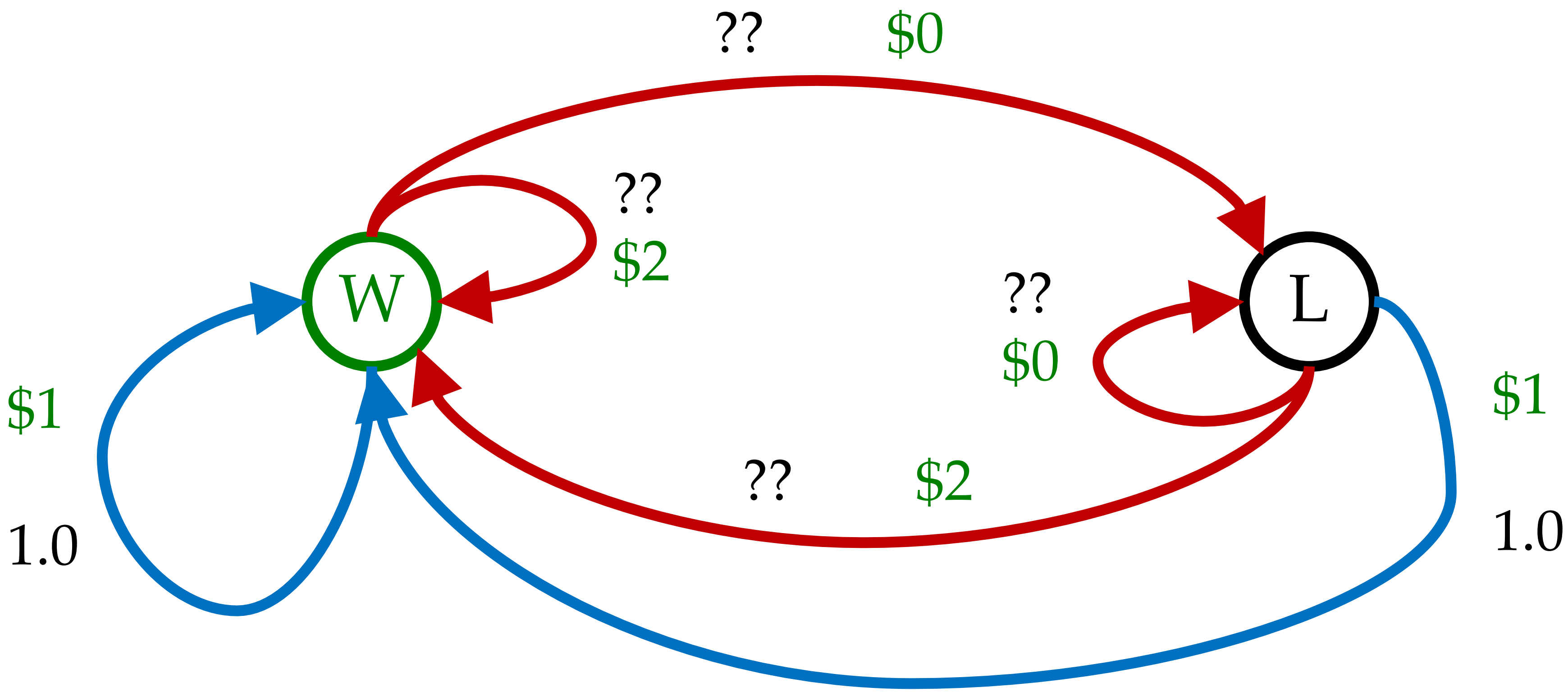
Let's Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Online Planning

- Rules changed! Red's win chance is different.



Let's Play!

$2 $2 $2 $0 $0
$2

$0 $0 $0 $0

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computa
  - You needed to actually act to figure it out

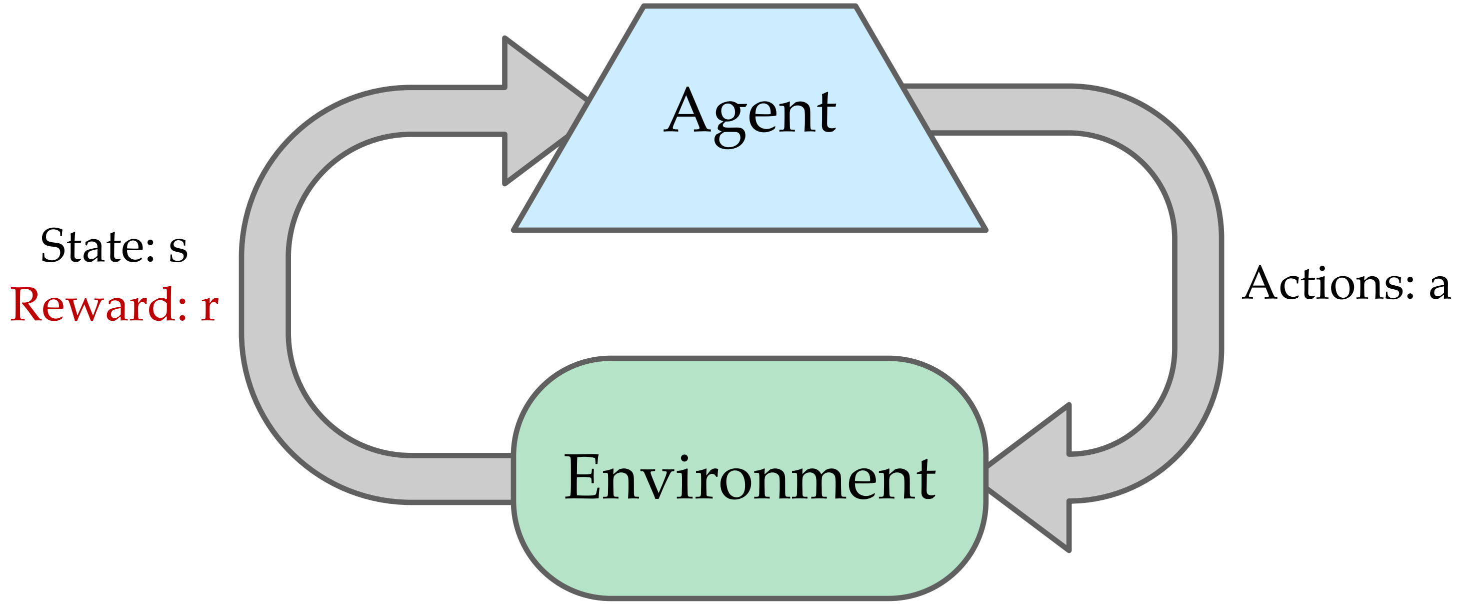
- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states s ∈ S
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy π(s)

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Reinforcement Learning



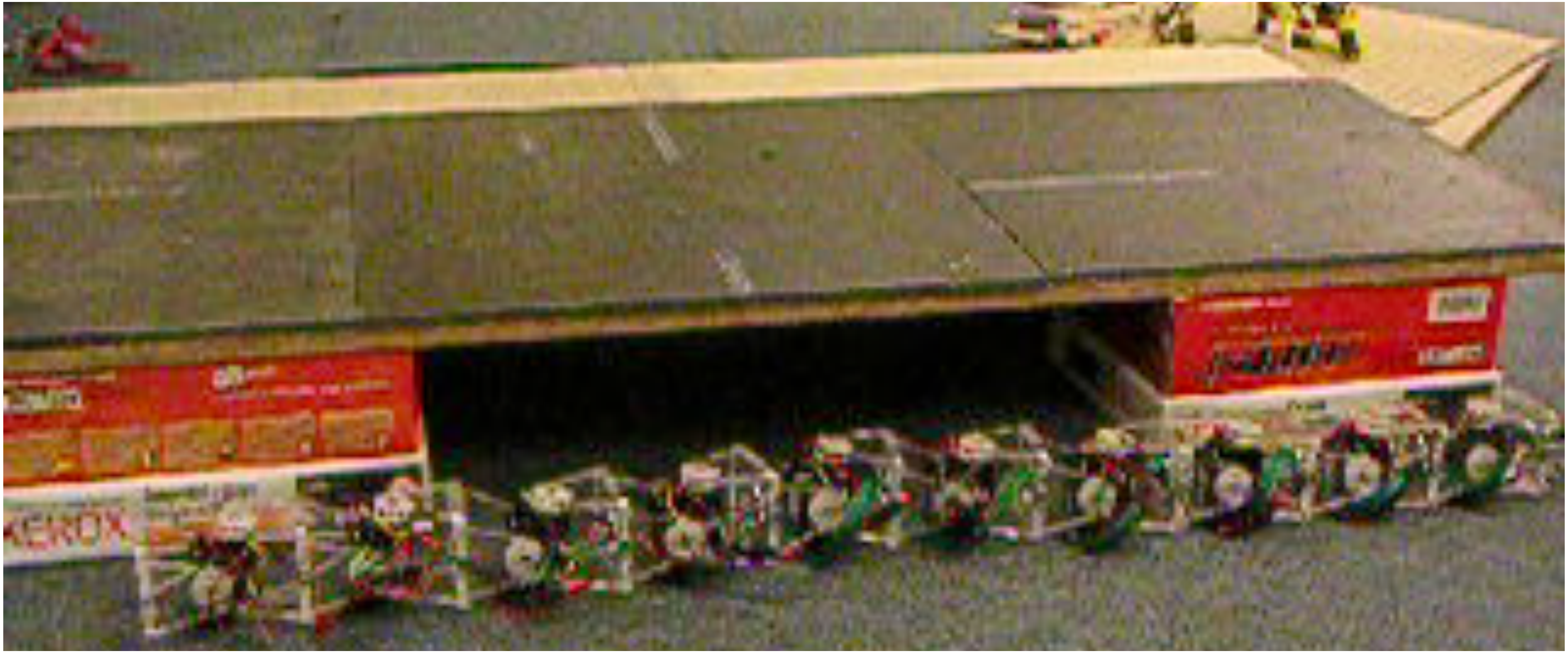
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Example: Learning to Fly

[Lambert et. al, RA-L 2019]

[Video: quad-fly]

Example: Sidewinding



[Andrew Ng]

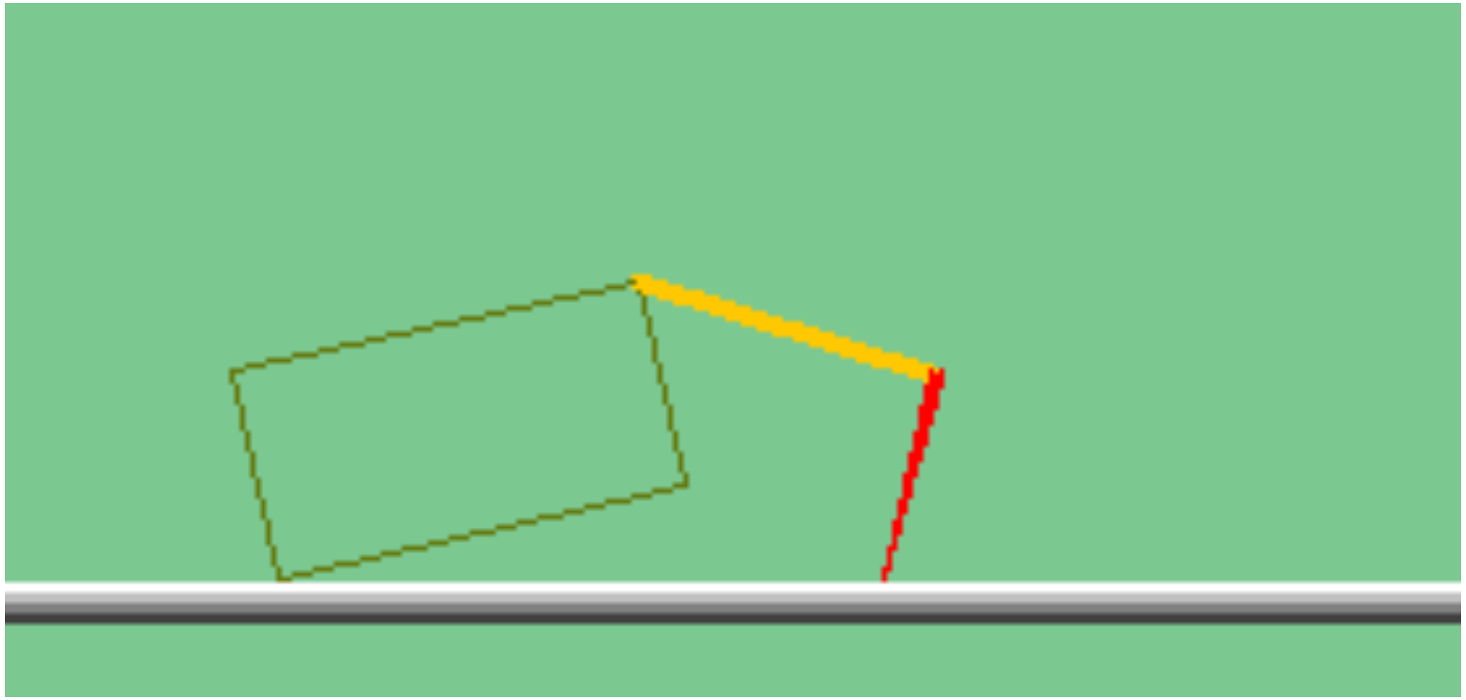
[Video: SNAKE – climbStep+sidewinding]

Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

The Crawler!



[Demo: Crawler Bot (L10D1)] [You, in Project 3]

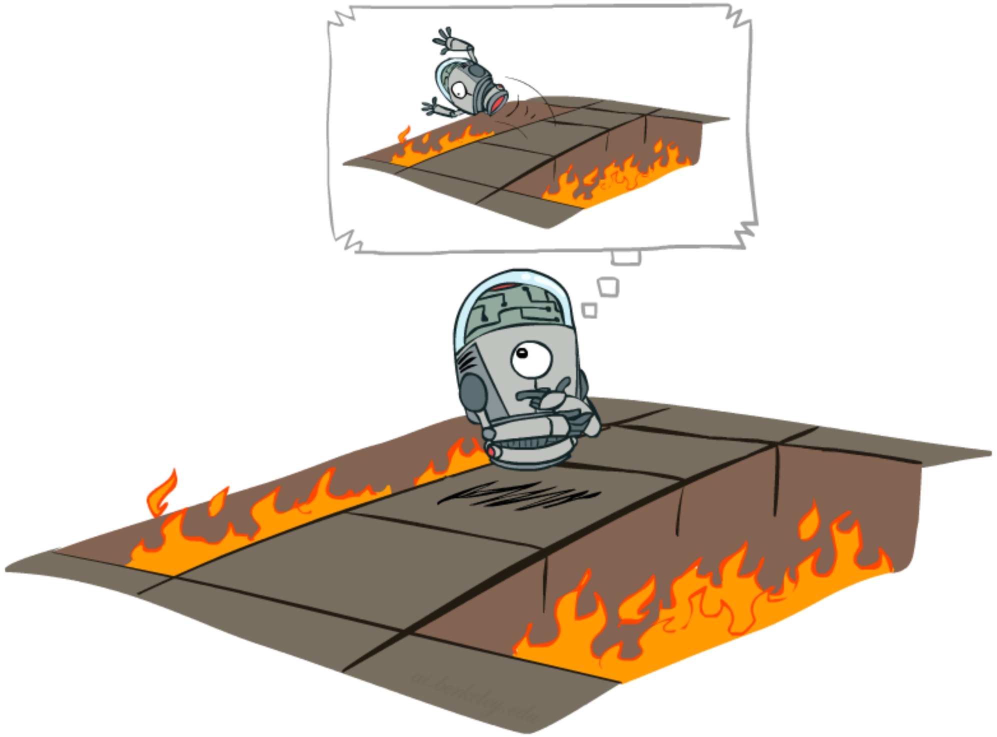
Video of Demo Crawler Bot

Reinforcement Learning

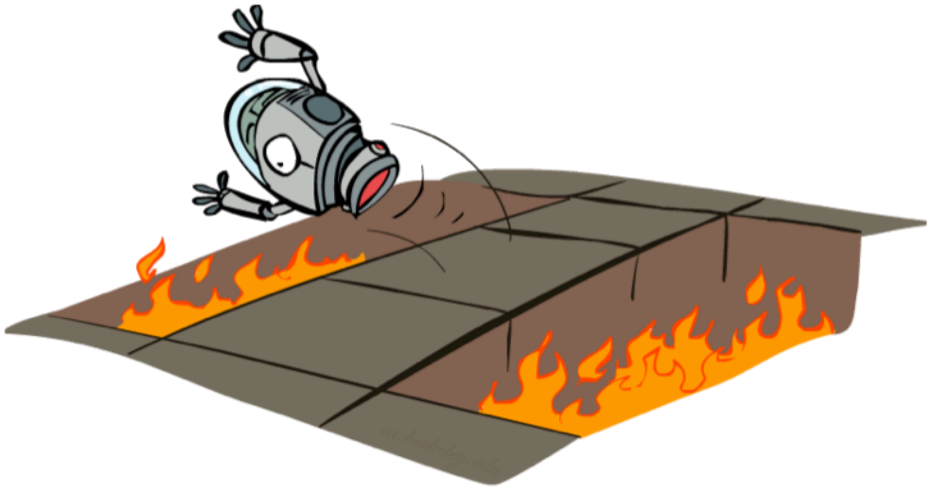
- Still assume a Markov decision process (MDP):
  - A set of states s ∈ S
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy π(s)

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn
  - Get 'measurement' of R at each step

Offline (MDPs) vs. Online (RL)

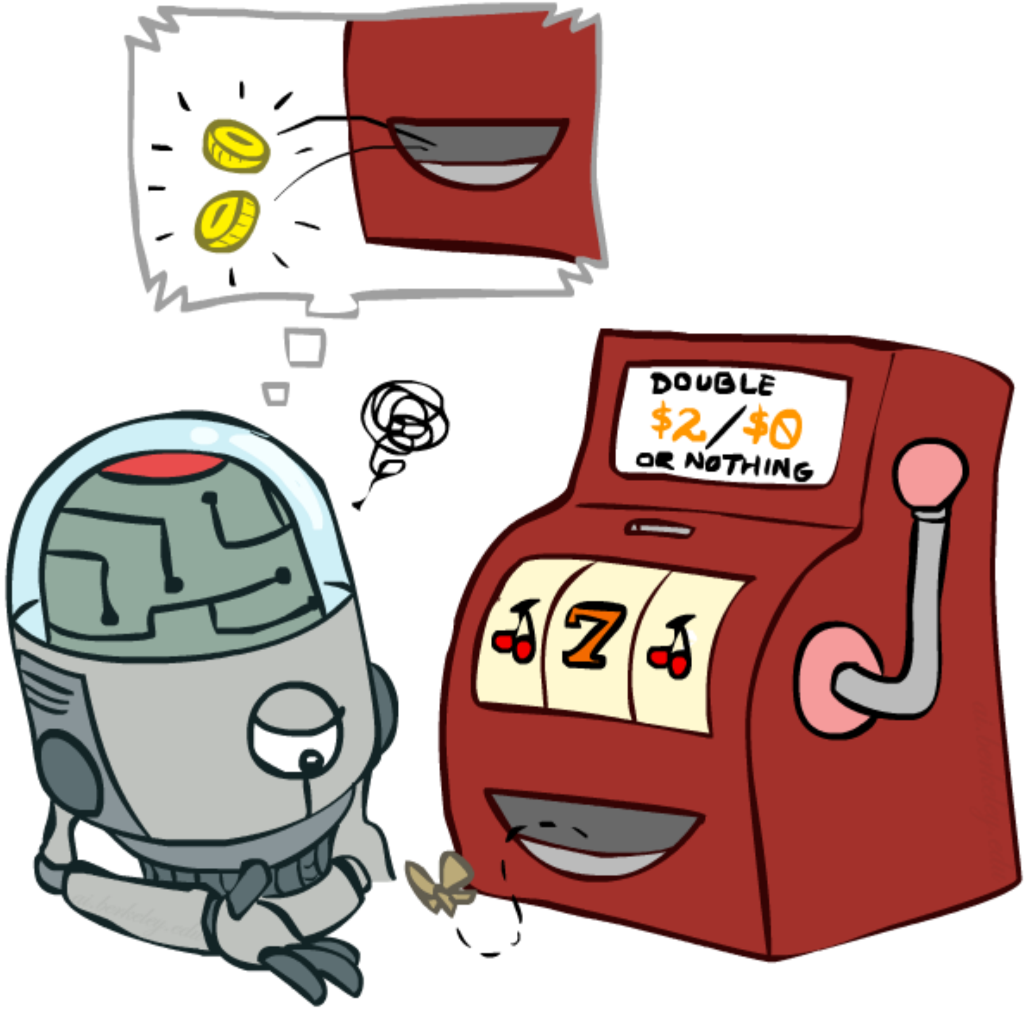


Offline Solution

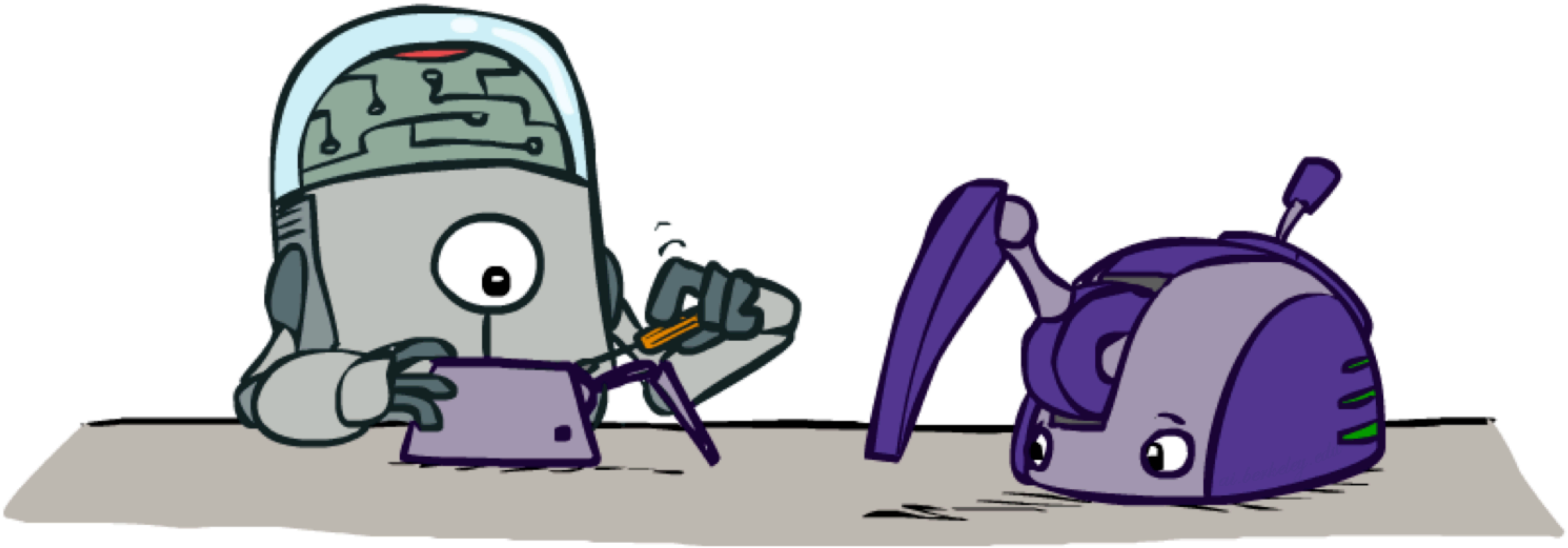


Online Learning

Model-Free Learning



Model-Based Learning



Passive Reinforcement Learning