Introduction to Reinforcement Learning

Yin Li

yin.li@wisc.edu

University of Wisconsin, Madison

[Based on slides from Lana Lazebnik, Yingyu Liang, David Page, Mark Craven, Peter Abbeal, Daniel Klein]

Reinforcement Learning (RL)

Task of an agent embedded in an environment

repeat forever

1) sense world
2) reason
3) choose an action to perform
4) get feedback (usually reward = 0)
5) learn

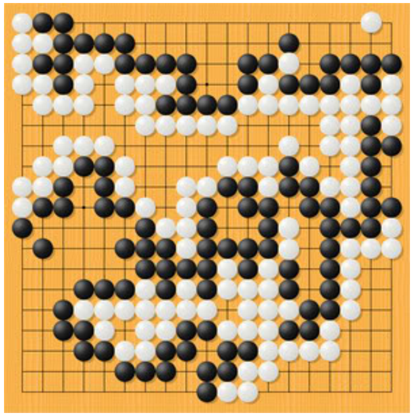
Reinforcement Learning (RL)

Task of an agent embedded in an environment

repeat forever

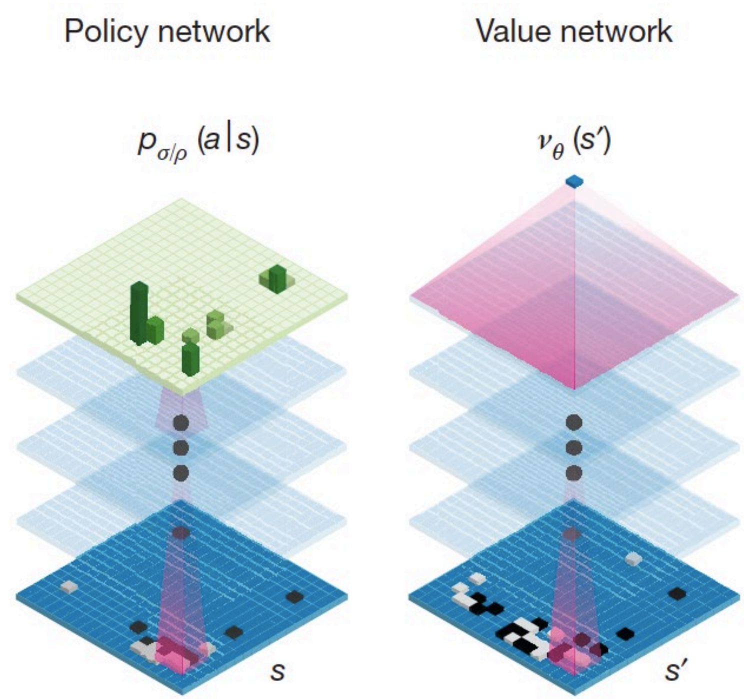
1) sense world
2) reason
3) choose an action to perform
4) get feedback (usually reward = 0)
5) learn

the environment may be the physical world or an artificial one



RL: Example

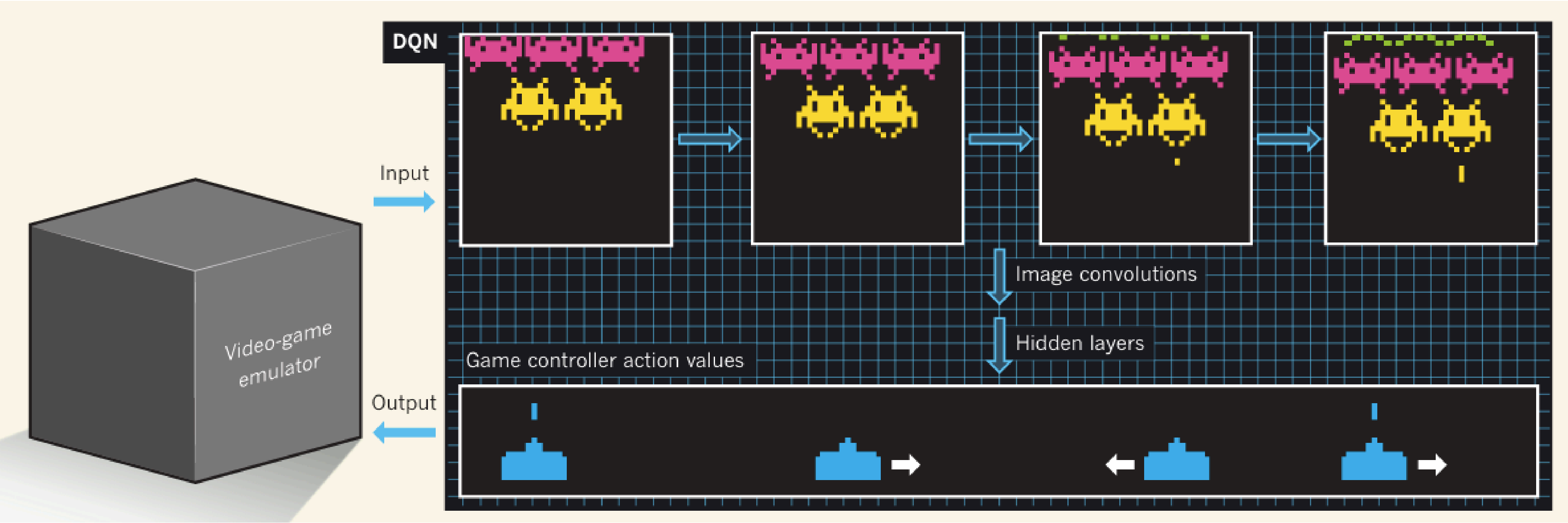
- AlphaGo and AlphaZero



https://deepmind.com/research/alphago/

RL: Example

- AI for video games

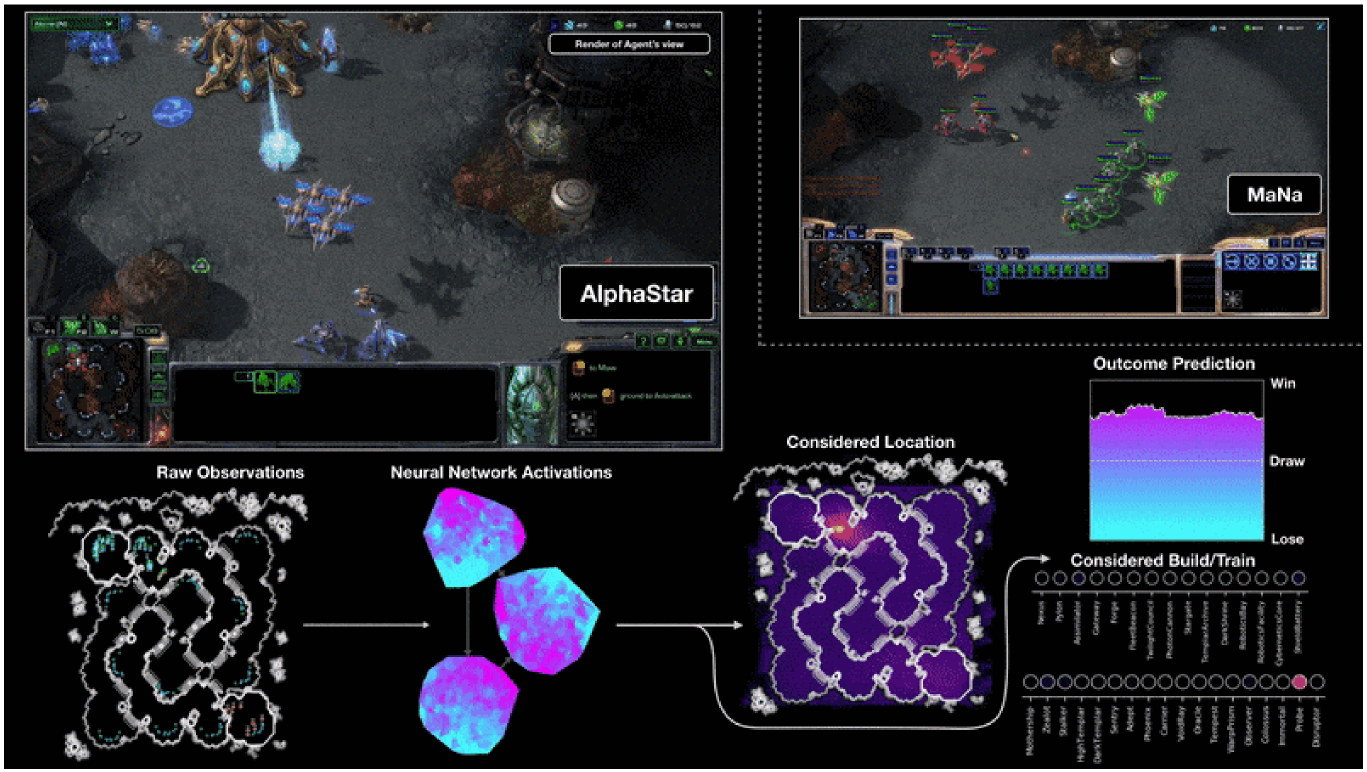


[Video]

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, [Human-level control through deep reinforcement learning], *Nature* 2015

RL: Example

- AI for video games



O. Vinyals, I. Babuschkin, W.M. Czarnecki et al.
Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 2019

RL: Example

- End-to-end training of visuomotor policies

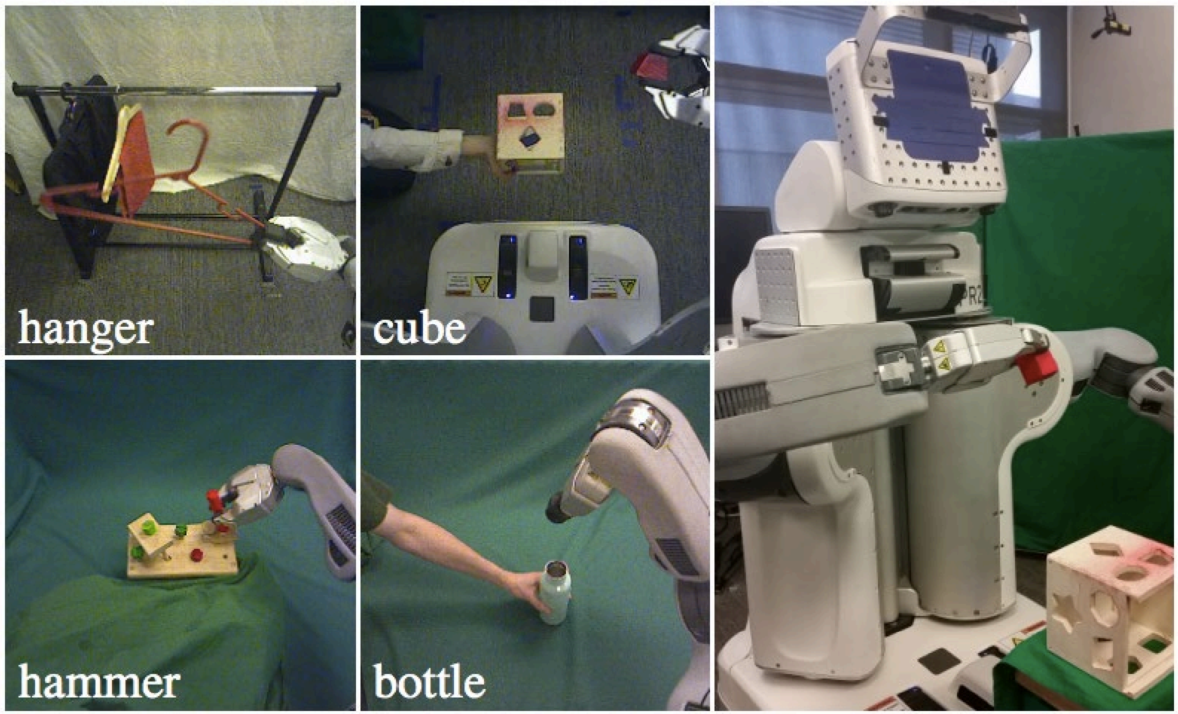


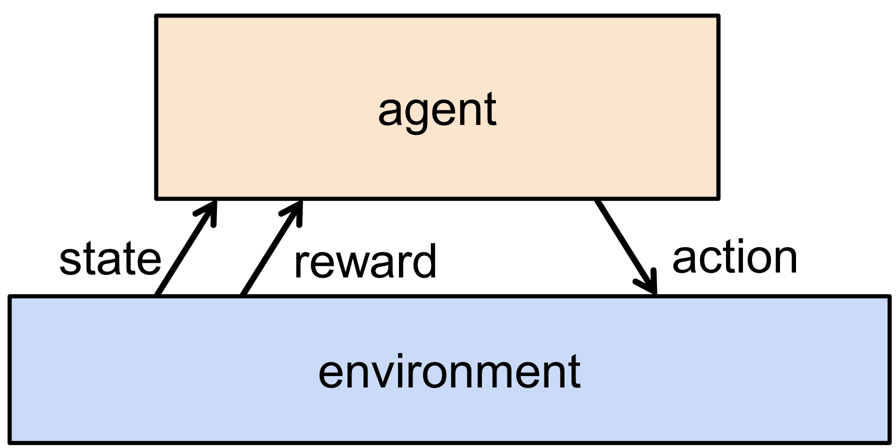
Fig. 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

Video

S. Levine, C. Finn, T. Darrell and P. Abbeel.
End-to-End Training of Deep Visuomotor Policies. JMLR 2016

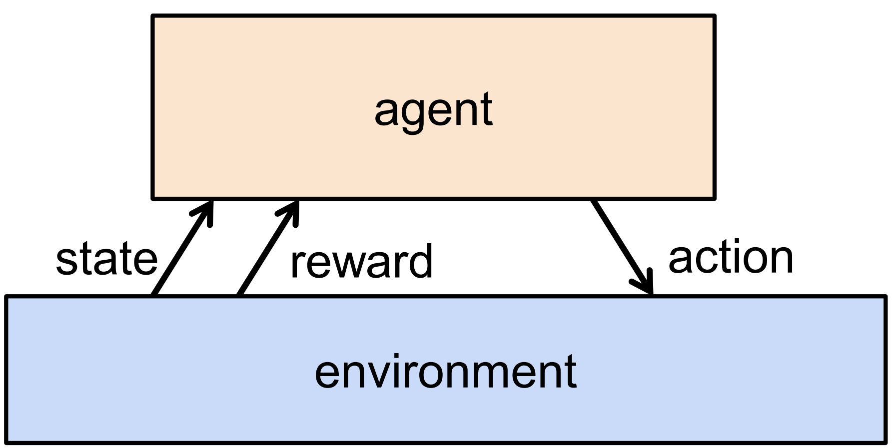
Reinforcement Learning (RL)

- Agent can take *actions* that affect the *state* of the environment and observe occasional *rewards* that depend on the state
  - set of states S
  - set of actions A
  - at each time t, agent observes state $s_t \in S$ and receives reward r_t
  - then chooses action $a_t \in A$ and changes to state s_{t+1}



Reinforcement Learning (RL)

- Agent can take *actions* that affect the *state* of the environment and observe occasional *rewards* that depend on the state
- The goal is to learn a mapping from states to actions (*policy*) to maximize expected reward over time



Formalism: Markov Decision Processes

- **States** S, beginning with initial state s_0
- **Actions** A
- **Transition model** $P(s_{t+1} \mid s_t, a_t)$
  - *Markov assumption*: the probability of going to s_{t+1} from s_t depends only on s_t and a_t and not on any other past actions or states
- **Reward function** $r(s_t)$

$$
\begin{array}{ccccccc}
s_0 & \xrightarrow{\quad a_0 \quad} & s_1 & \xrightarrow{\quad a_1 \quad} & s_2 & \xrightarrow{\quad a_2 \quad} & \\
r_0 & & r_1 & & r_2 & &
\end{array}
$$

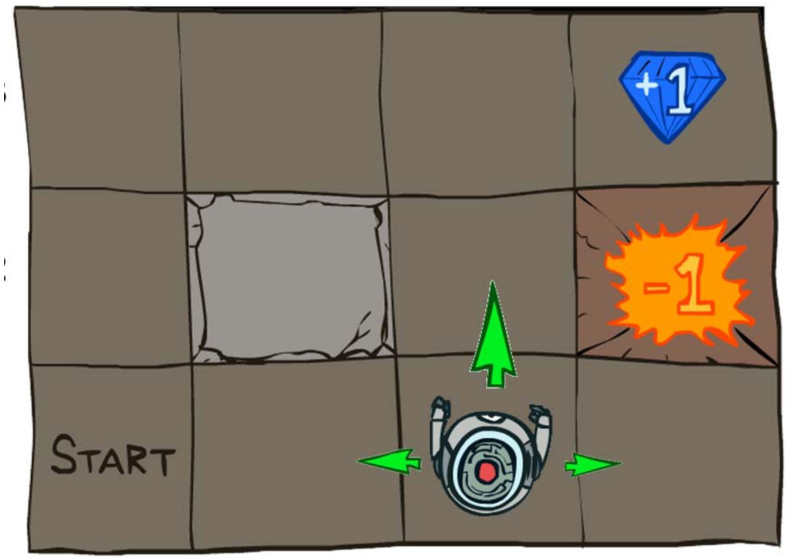
Formalism: Markov Decision Processes

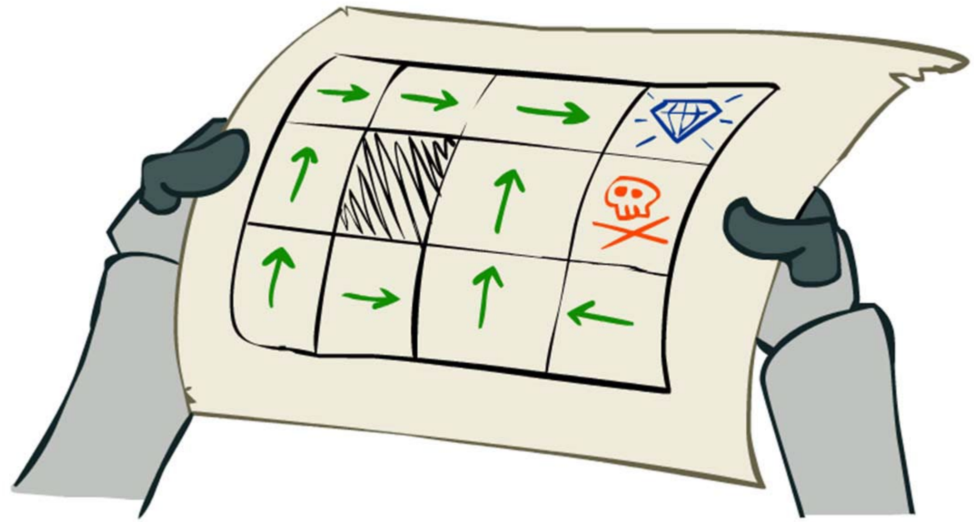
- **States** S, beginning with initial state s_0
- **Actions** A
- **Transition model** $P(s_{t+1} | s_t, a_t)$
  - *Markov assumption*: the probability of going to s_{t+1} from s_t depends only on s_t and a_t and not on any other past actions or states
- **Reward function** $r(s_t)$

- **Policy** $\pi(s) : S \rightarrow A$ the action that an agent takes in any given state
  - The "solution" to an MDP

Example MDP: Grid world

- Goal: find the best policy



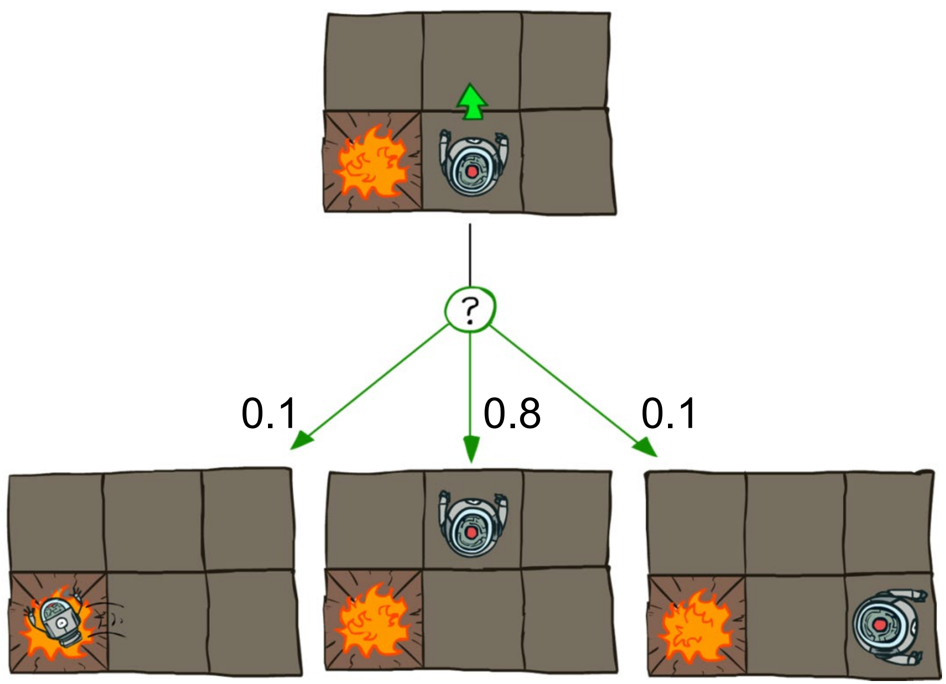


Source: P. Abbeel and D. Klein

Example MDP: Grid world

- With an unreliable robot

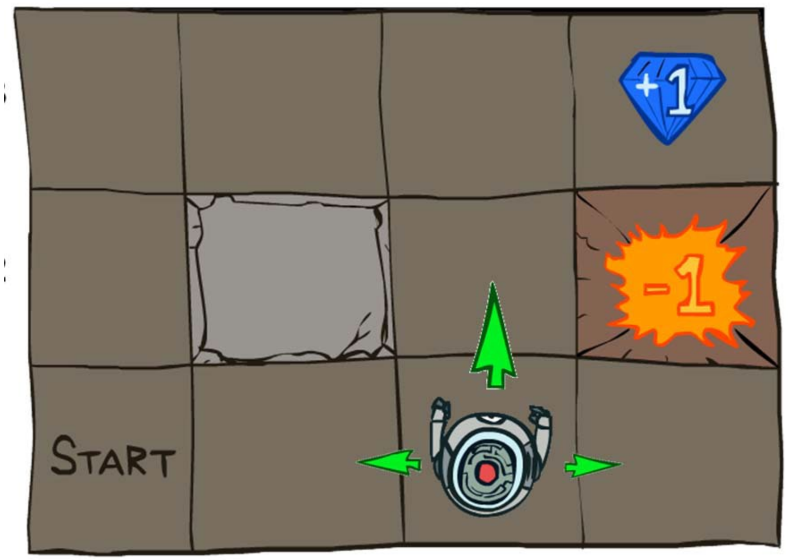
Transition model:



Source: P. Abbeel and D. Klein

Example MDP: Grid world

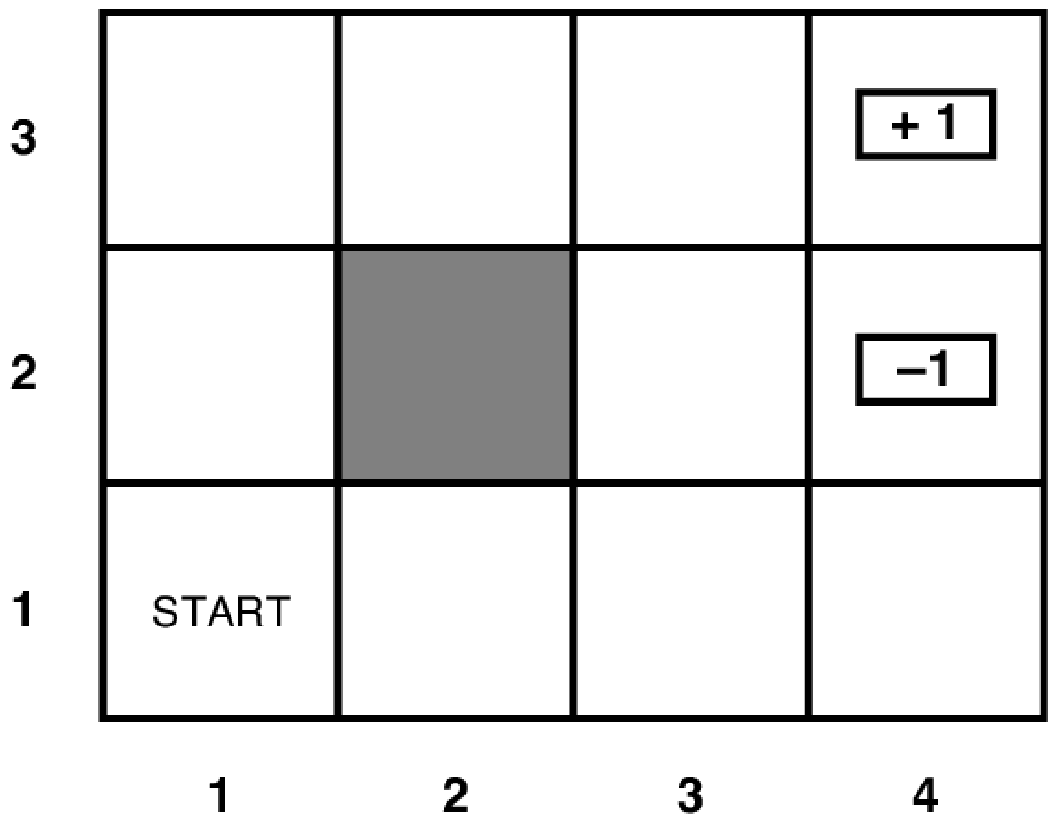
- Reach the target quickly



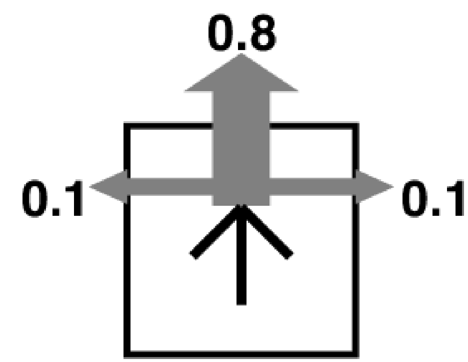
$r(s) = -0.04$ for every non-terminal state

Source: P. Abbeel and D. Klein

Example MDP: Grid world

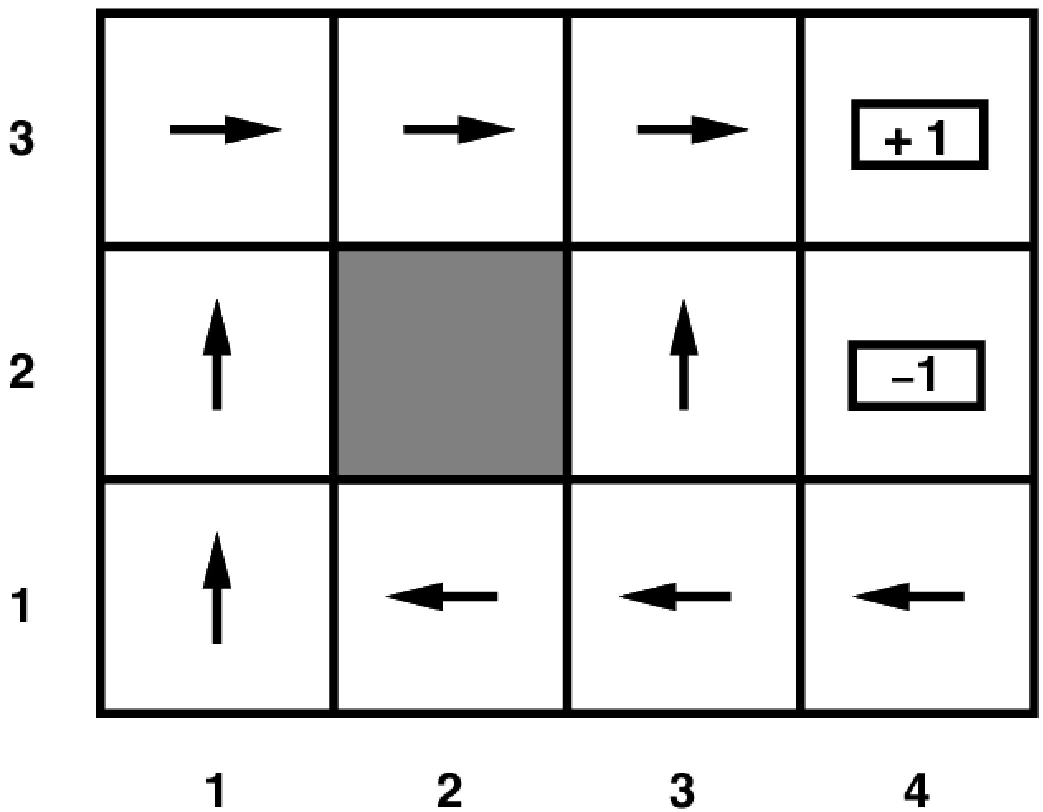


Transition model:



r(s) = -0.04 for every non-terminal state

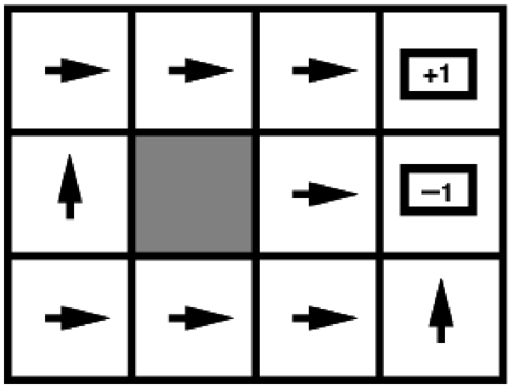
Example MDP: Grid world



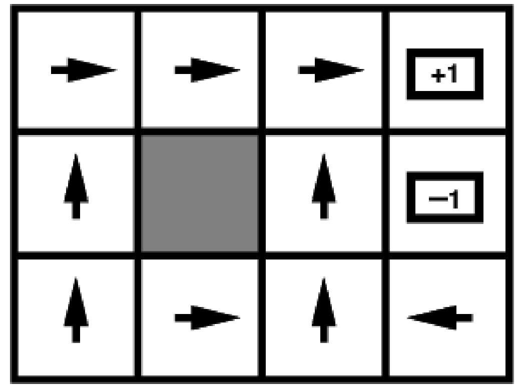
Optimal policy when r(s) = -0.04 for every non-terminal state

Example MDP: Grid world

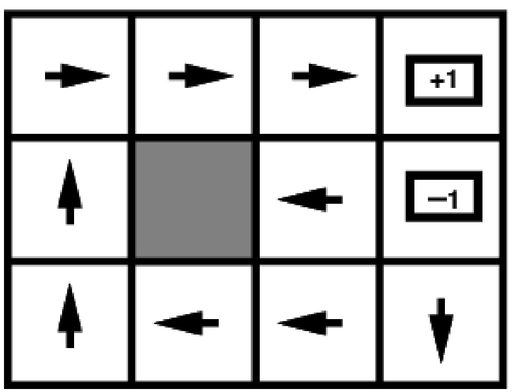
- Optimal policies for various values of $r(s)$:



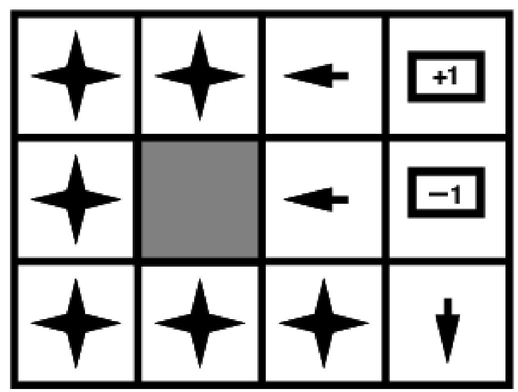
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

Formalism: Markov Decision Processes

- **States** S, beginning with initial state s_0
- **Actions** A
- **Transition model** $P(s_{t+1} \mid s_t, a_t)$
  - *Markov assumption*: the probability of going to s_{t+1} from s_t depends only on s_t and a_t and not on any other past actions or states
- **Reward function** $r(s_t)$

- **Policy** $\pi(s) : S \rightarrow A$ the action that an agent takes in any given state
  - The "solution" to an MDP

Formalism: Markov Decision Processes

- **States** S, beginning with initial state s_0
- **Actions** A
- **Transition model** $P(s_{t+1} | s_t, a_t)$
  - *Markov assumption*: the probability of going to s_{t+1} from s_t depends only on s_t and a_t and not on any other past actions or states
- **Reward function** $r(s_t)$

- **Policy** $\pi(s) : S \rightarrow A$ the action that an agent takes in any given state
  - The "solution" to an MDP
  - How to find the best policy?

Defining the optimal policy

- Given a policy π, we can define the *expected utility* over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(sequence)U(sequence)$$

- The value function of s_0 w.r.t. policy π
- The optimal policy should maximize this utility

Defining the optimal policy

- Given a policy π, we can define the *expected utility* over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(sequence)U(sequence)$$

- The value function of s_0 w.r.t. policy π
- The optimal policy should maximize this utility
- How to define the utility of a state sequence?
  - Sum of rewards of individual states
  - Problem: infinite state sequences

Discounted rewards

- To define the utility of a state sequence, *discount* the individual state rewards by a factor γ between 0 and 1:

$$U(s_0, s_1, \ldots) = r(s_0) + \gamma\, r(s_1) + \gamma^2 r(s_2) + \cdots$$

$$= \sum_{t \geq 0} \gamma^t\, r(s_t)$$

Image source: P. Abbeel and D. Klein

Discounted rewards

- To define the utility of a state sequence, *discount* the individual state rewards by a factor γ between 0 and 1:

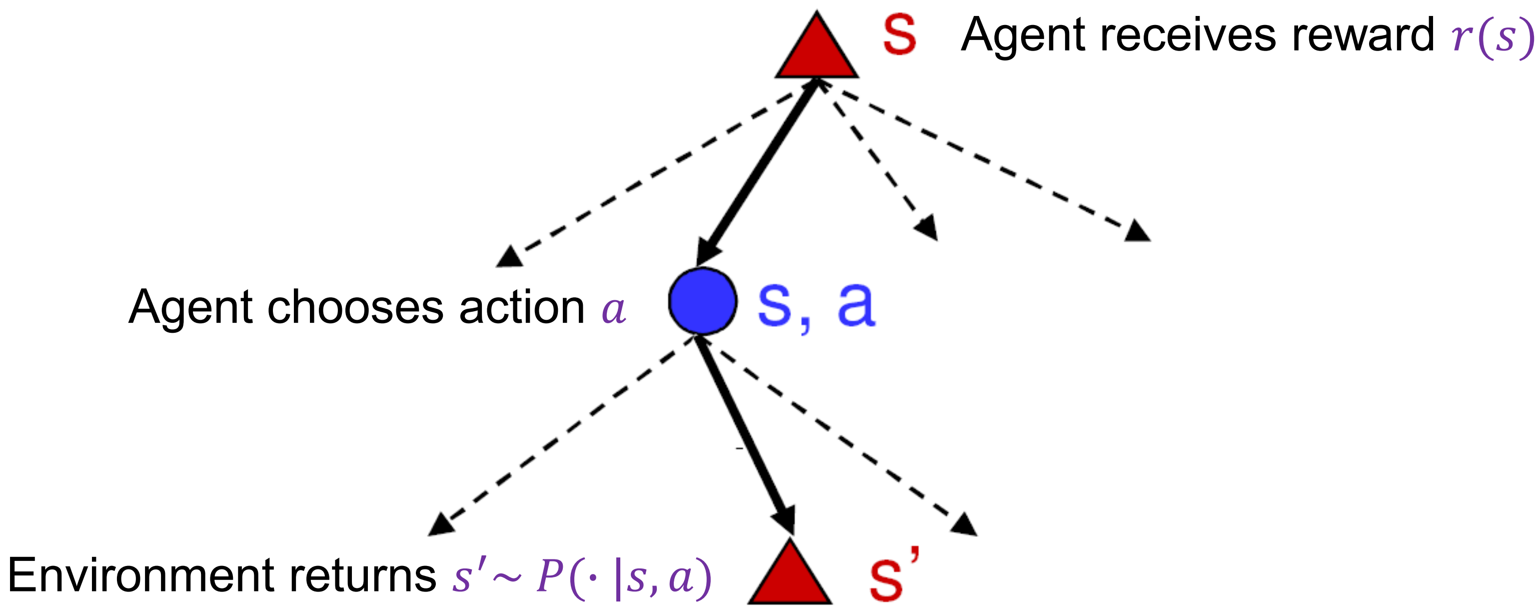
$$U(s_0, s_1, \ldots) = r(s_0) + \gamma\, r(s_1) + \gamma^2 r(s_2) + \cdots$$

$$= \sum_{t \geq 0} \gamma^t\, r(s_t) \leq \frac{r_{max}}{1-\gamma}$$

  - Sooner rewards count more than later rewards
  - Makes sure the total utility stays bounded

- How to find the policy maximizing the value function – the expected sum of discounted rewards?

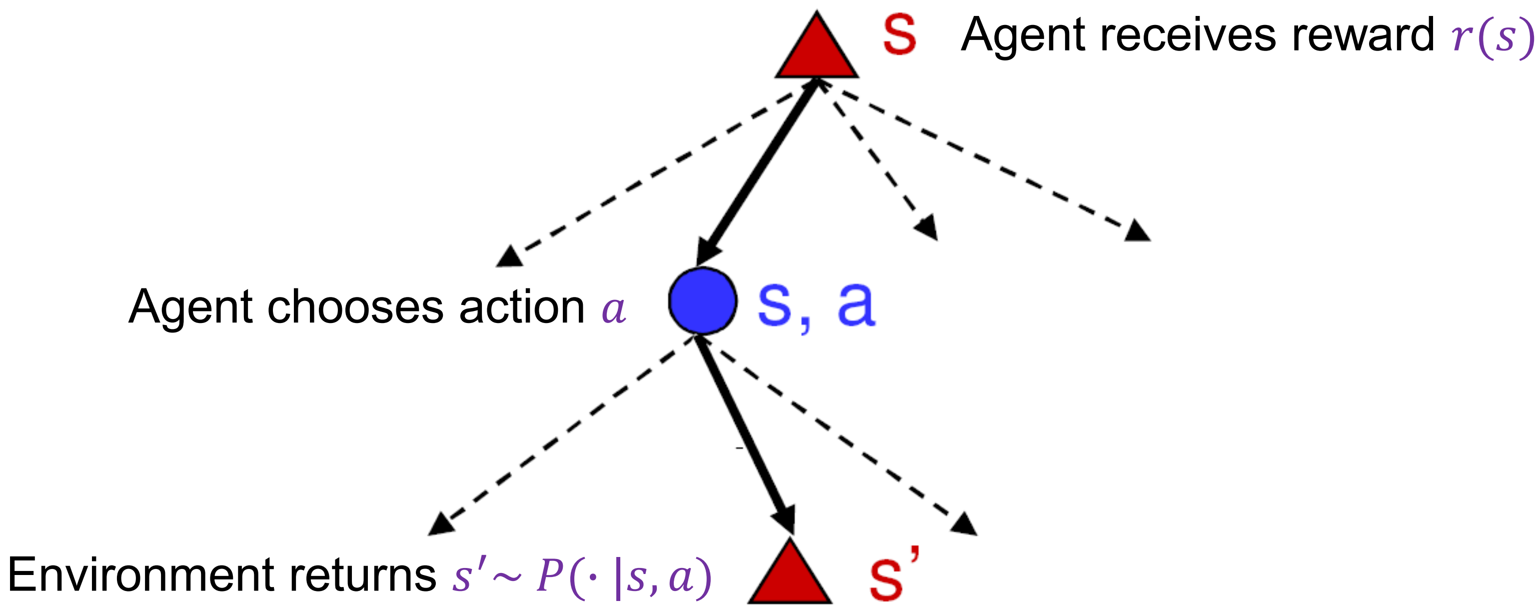
The Bellman equation



- Define state utility $V^*(s)$ as the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state s

Image source: L. Lazbenik

The Bellman equation

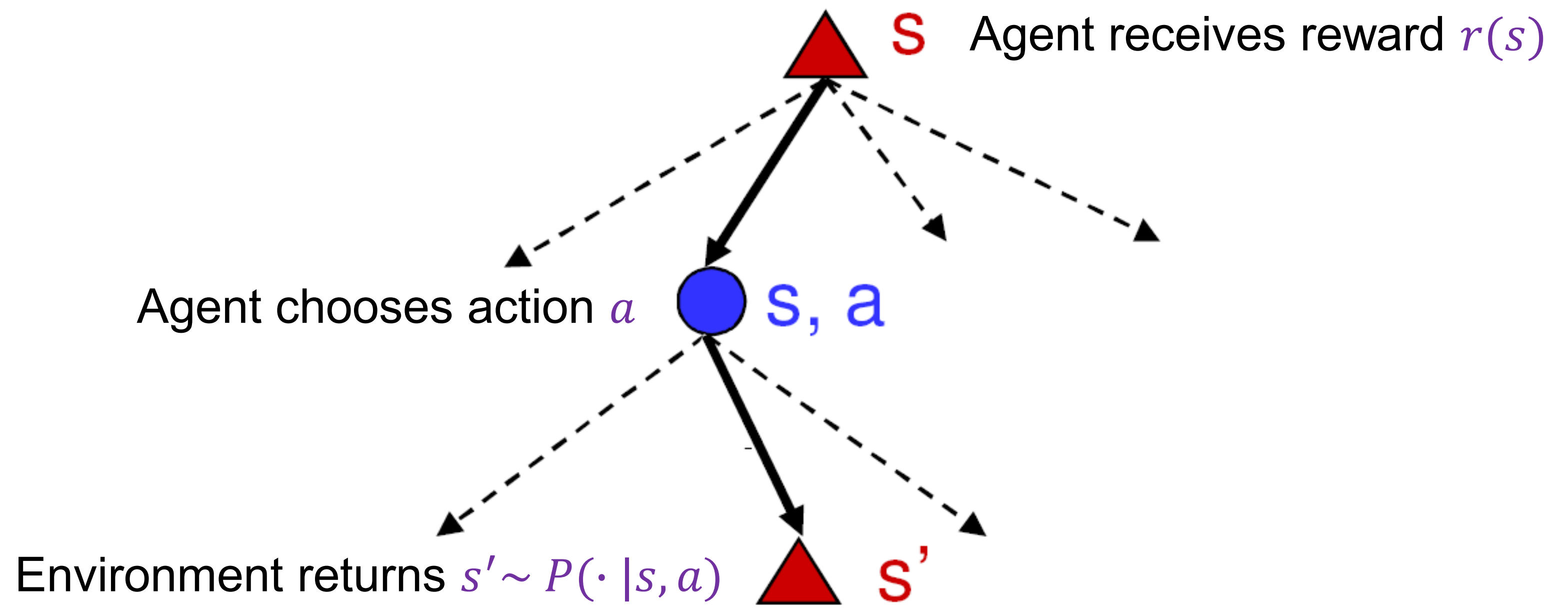


- What is the expected utility of taking action **a** in state **s**?

$$\sum_{s'} P(s' \mid s, a) V^*(s')$$

Image source: L. Lazbenik

The Bellman equation

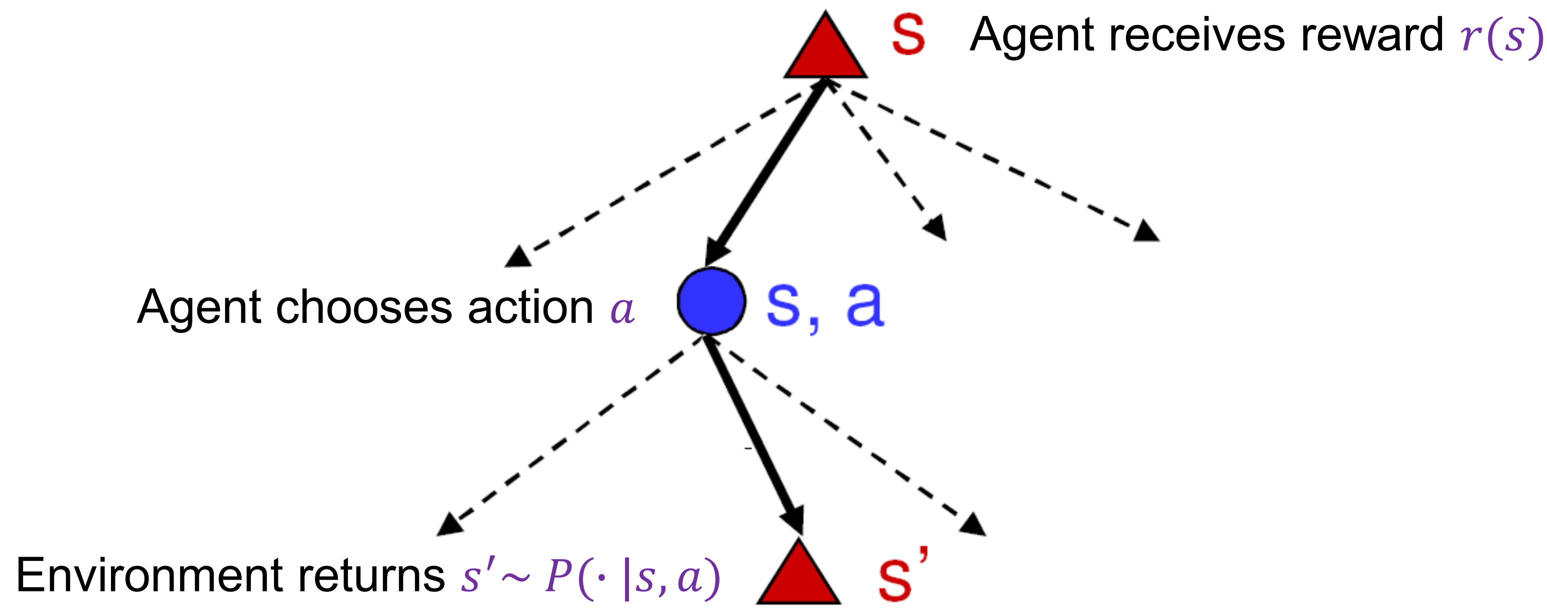


- How do we choose the optimal action?

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s')$$

Image source: L. Lazbenik

The Bellman equation

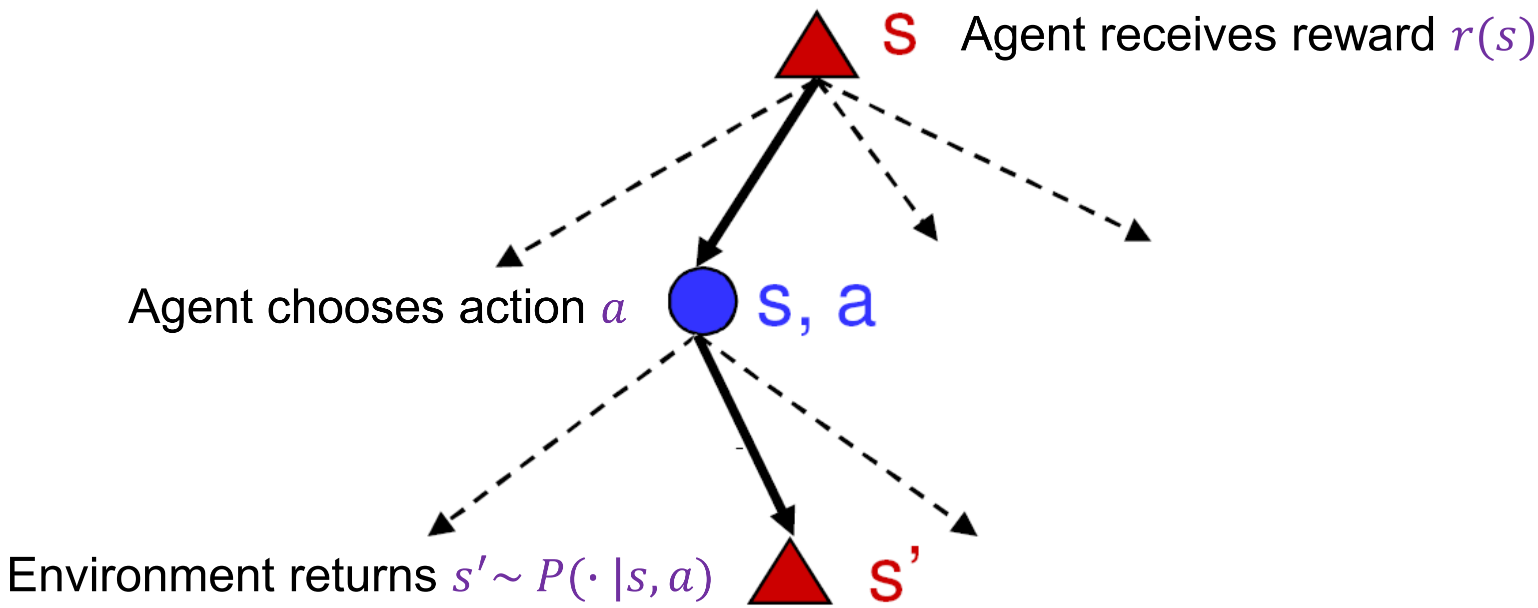


- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

$$V^*(s) = r(s) + \gamma \sum_{s'} P(s'|s, \pi^*(s)\,) V^*(s')$$

Image source: L. Lazbenik

The Bellman equation

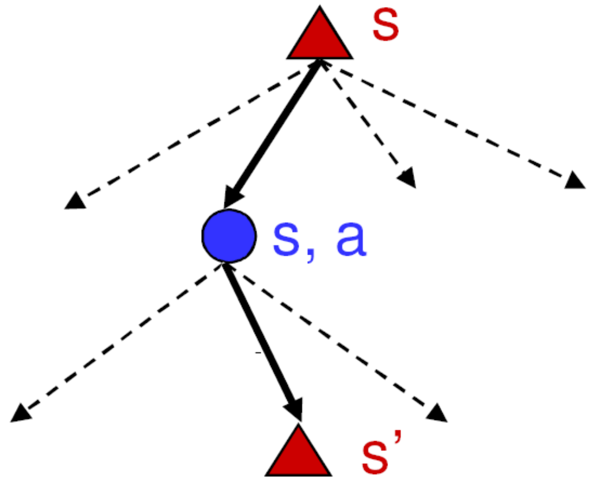


- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')$$

Image source: L. Lazbenik

The Bellman equation



- Recursive relationship between optimal values of successive states:

$$V^*(s) = \underbrace{r(s)}_{\text{Reward in current state}} + \underbrace{\gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')}_{\text{Discounted expected future reward assuming agent follows the optimal policy}}$$

Image source: L. Lazbenik

The Bellman equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

- The best policy to the MDP from s_0 is given by $V^*(s)$
- The solution is $\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a) V^*(s')$

The Bellman equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

- The best policy to the MDP from s_0 is given by $V^*(s)$
- The solution is $\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a) V^*(s')$

- If we know $r(s)$ and $P(s'|s,a)$, how can we compute $V^*(s)$?

Value iteration

- Start out with every $\mathrm{V}_0(s) = 0$
- Iterate until convergence
  - During the *i*th iteration, update the utility of each state according to the equation:

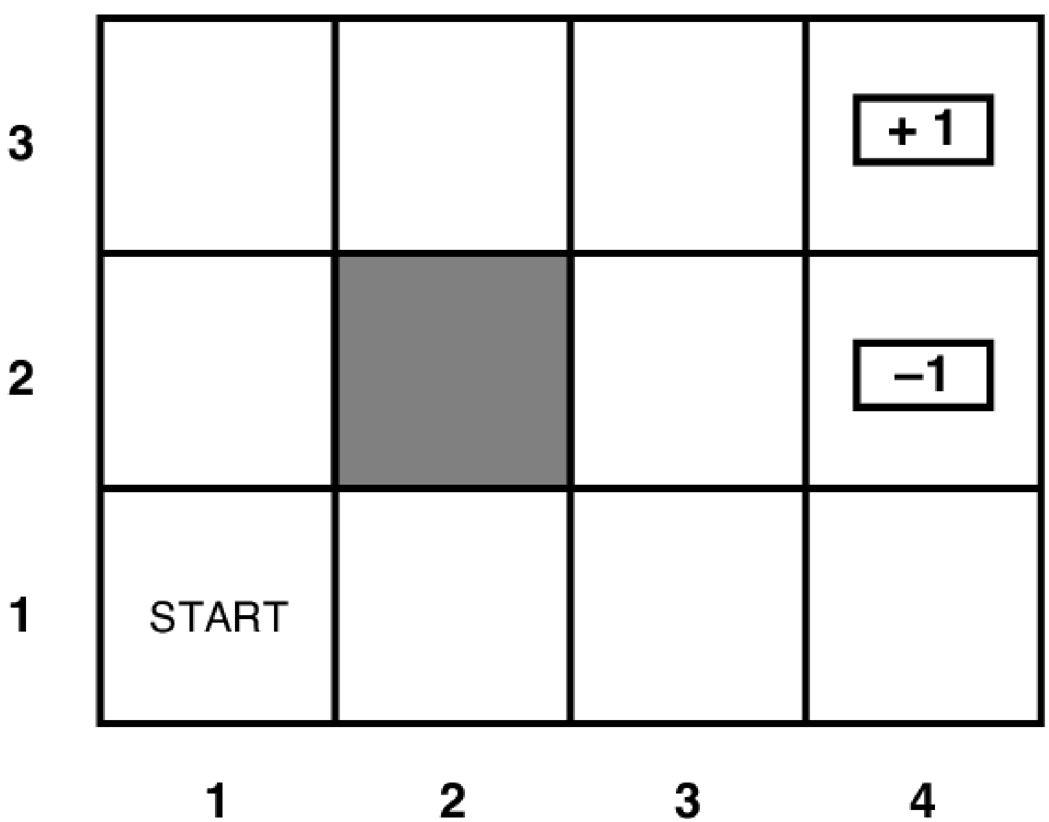
$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

- With infinitely many iterations, guaranteed to find the correct utility values $V^*(s)$
  - Even if we randomly traverse environment instead of looping through each state and action
  - In practice, don't need infinitely many iterations…

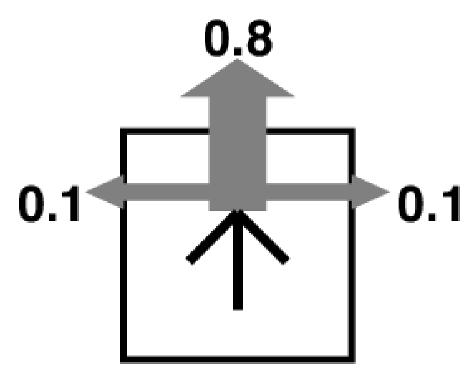
Value iteration

What effect does the update have?

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

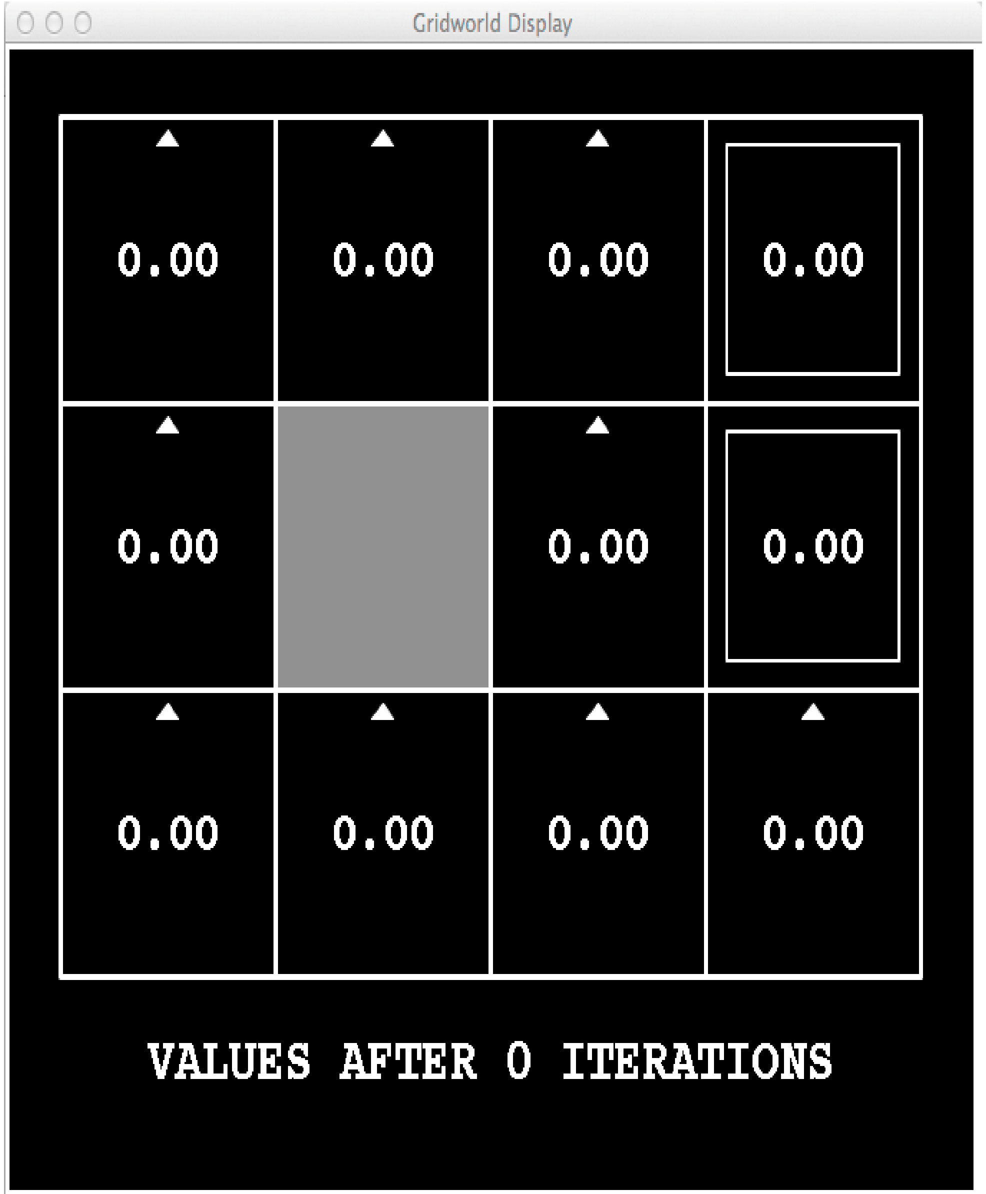


Transition model:



r(s) = 0 for every non-terminal state

Value iteration demo

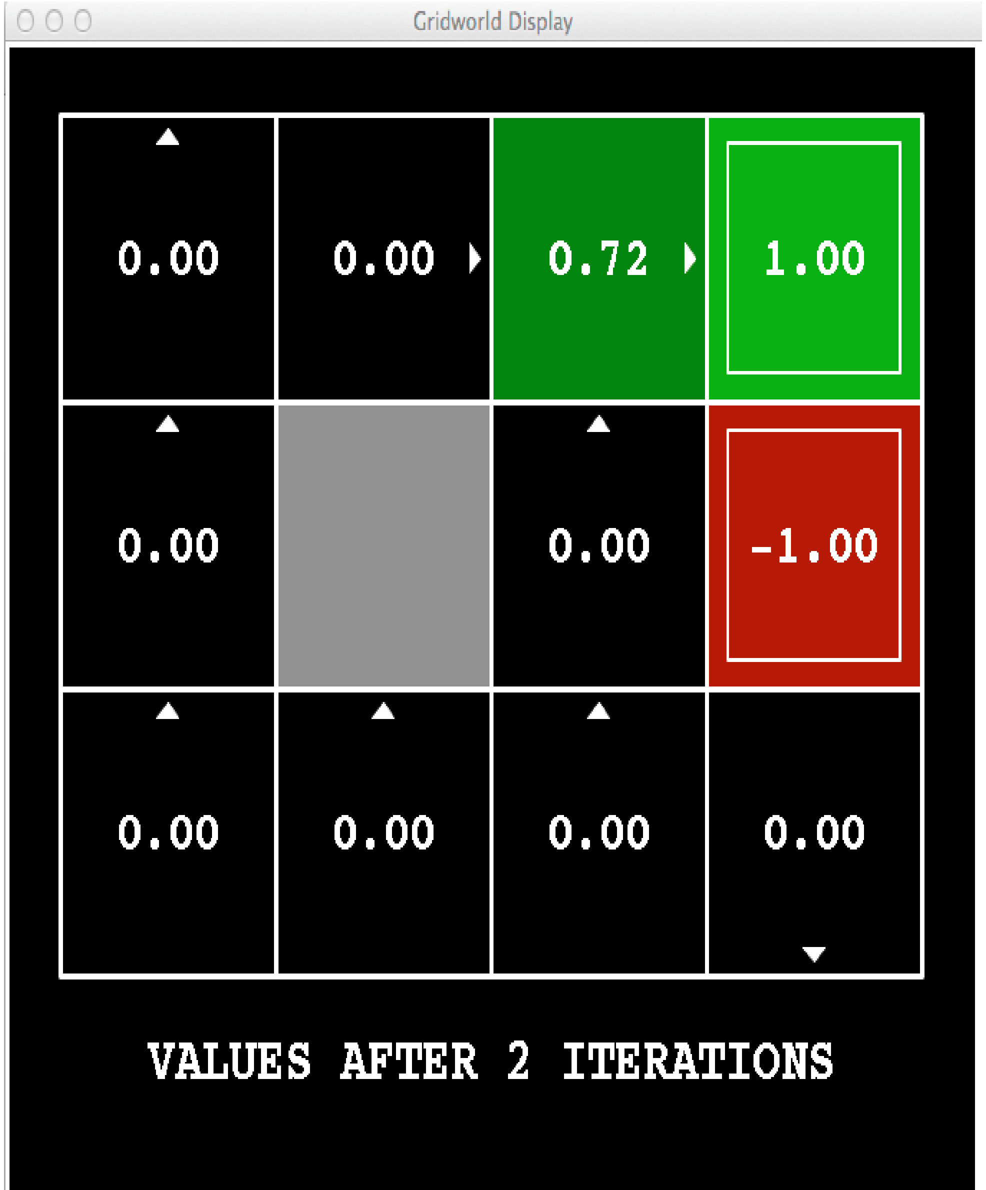


Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

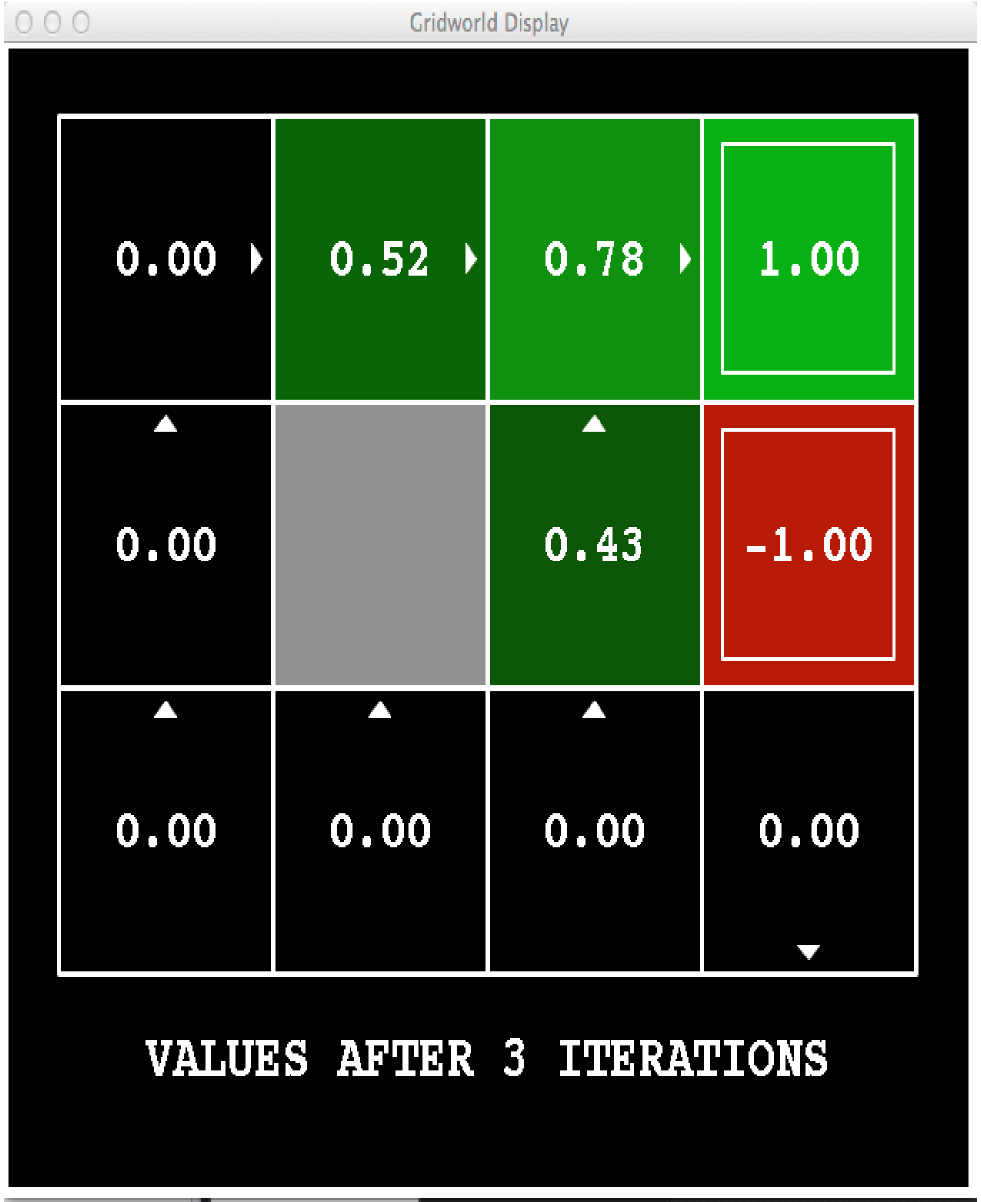
Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo



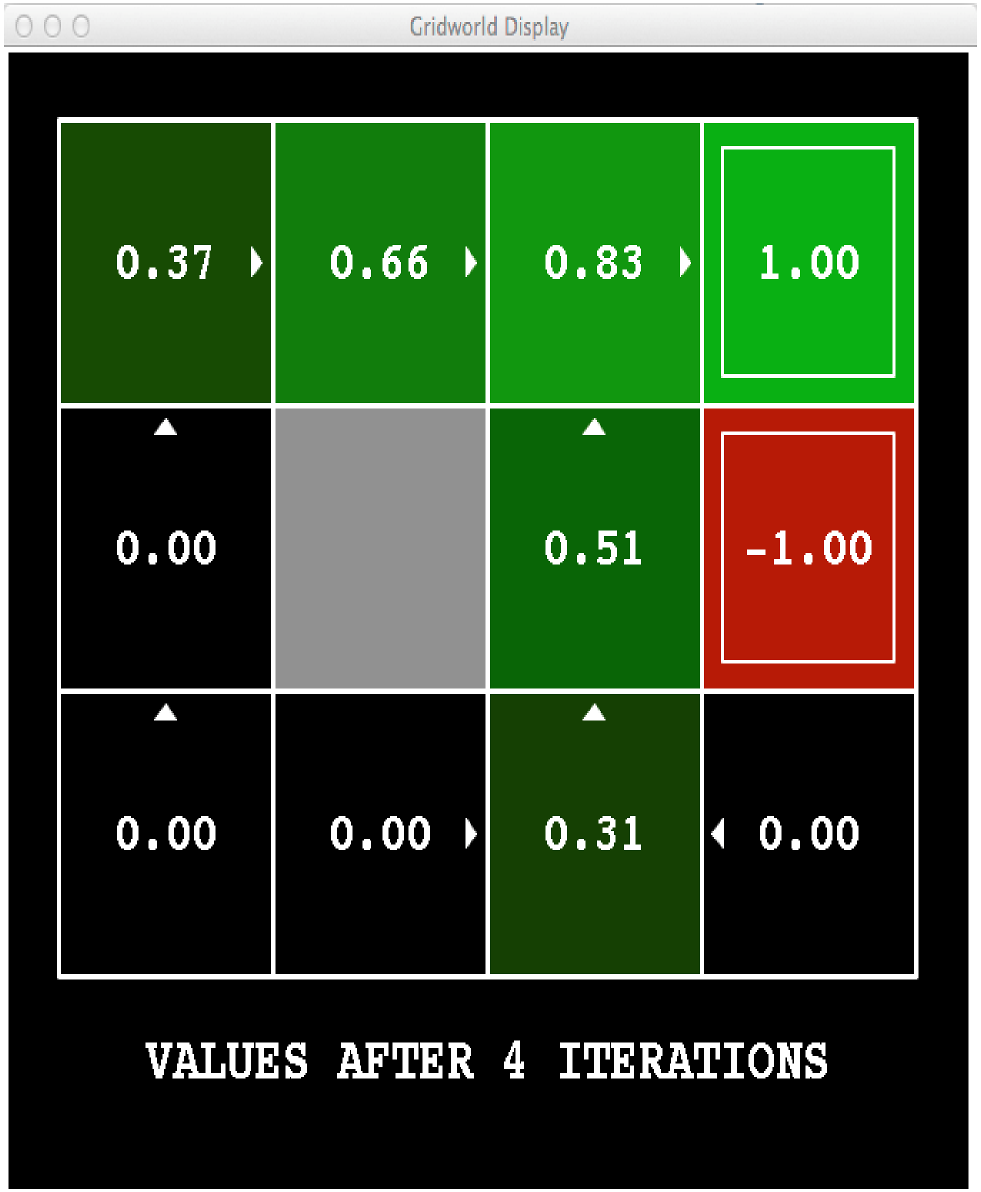
Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo



Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo



Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

Noise = 0.2
Discount = 0.9
Living reward = 0

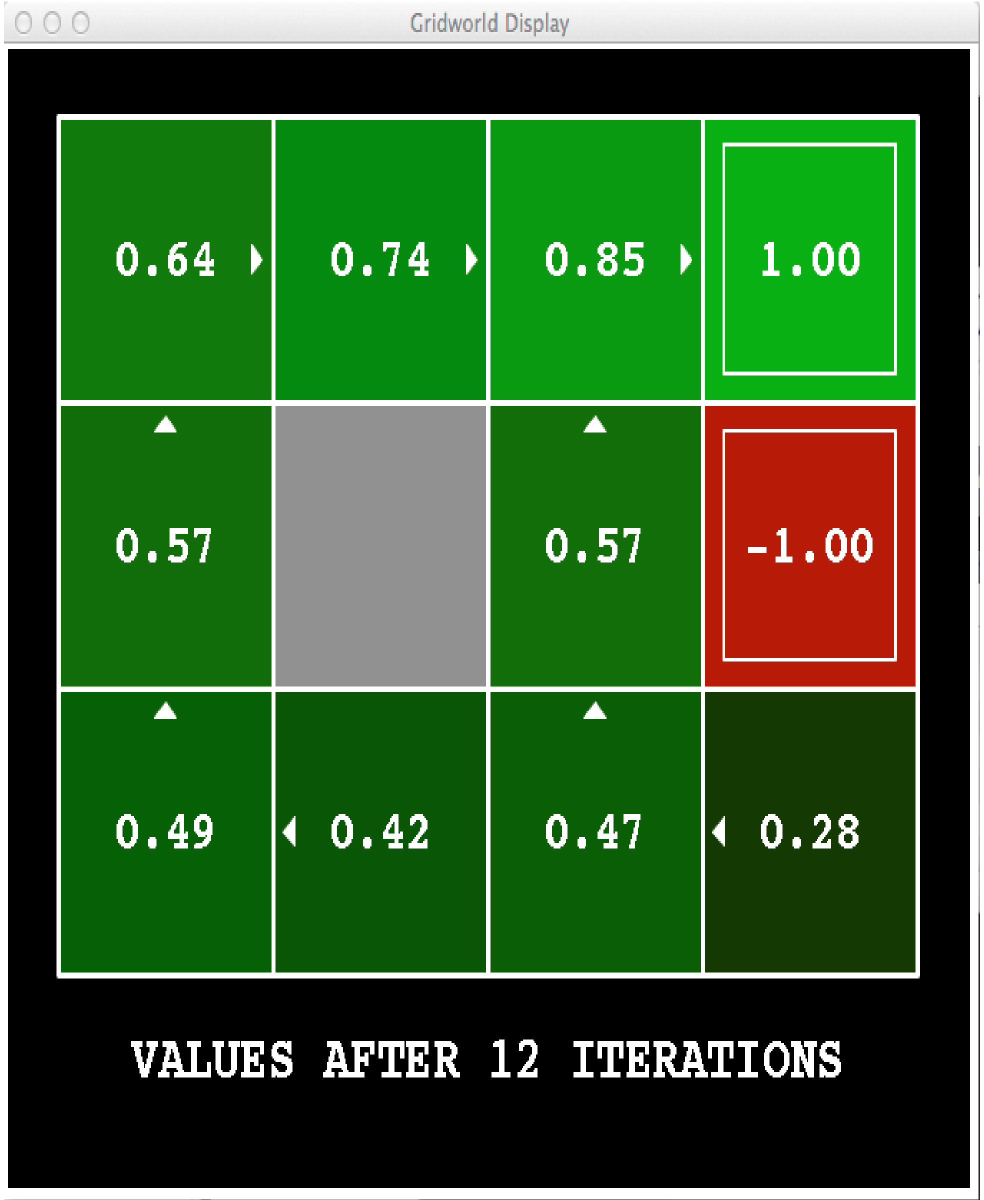
Value iteration demo

Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo

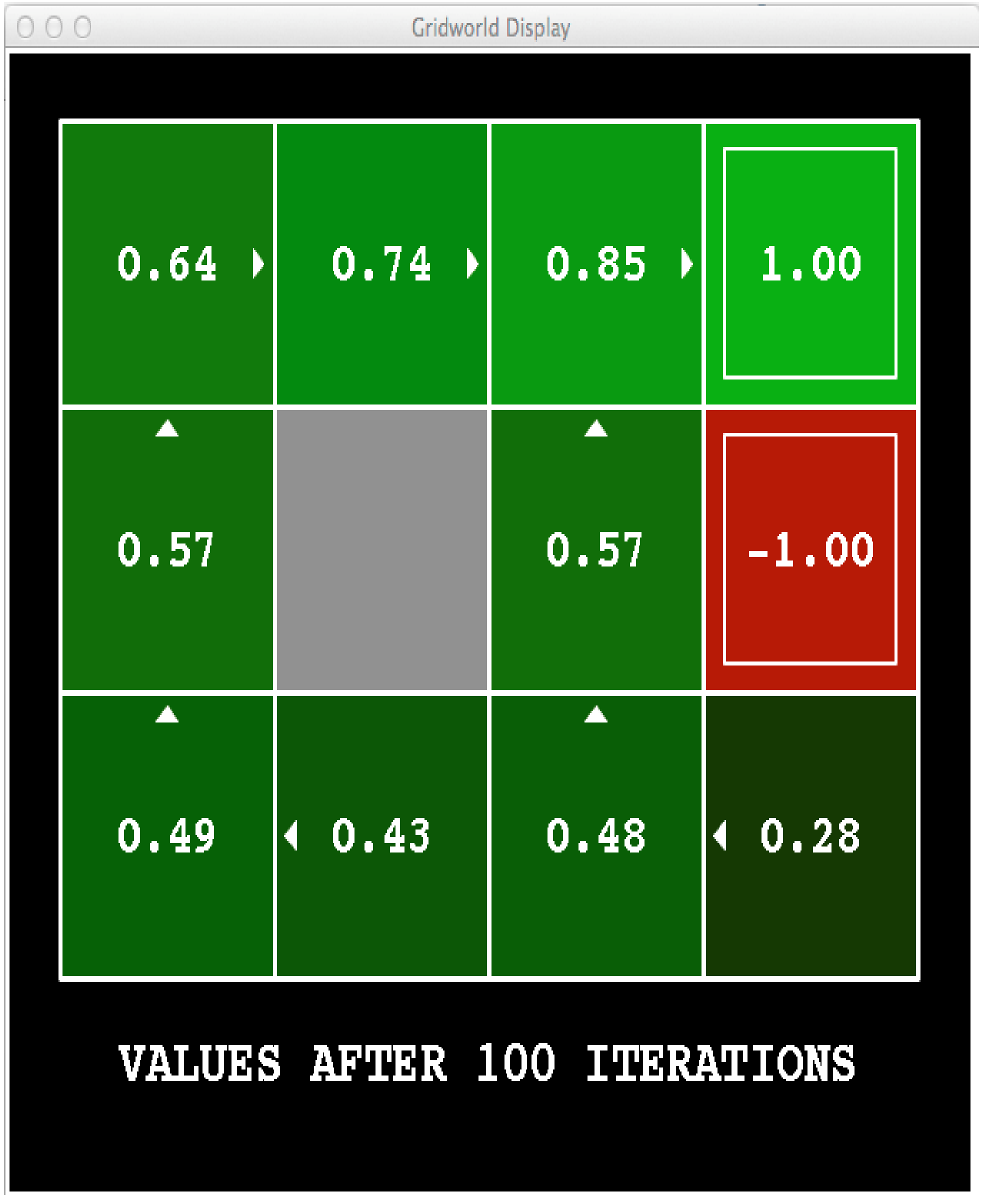
Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo



Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration demo



Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration: Recap

- Start out with every $\mathrm{V}_0(s)$ = 0
- Iterate until convergence
  - During the *i*th iteration, update the utility of each state according to the Bellman equation:

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

- Assuming that we know the model of the world $P(s'|s,a)$
- What if we don't?

Q-learning: a sketch

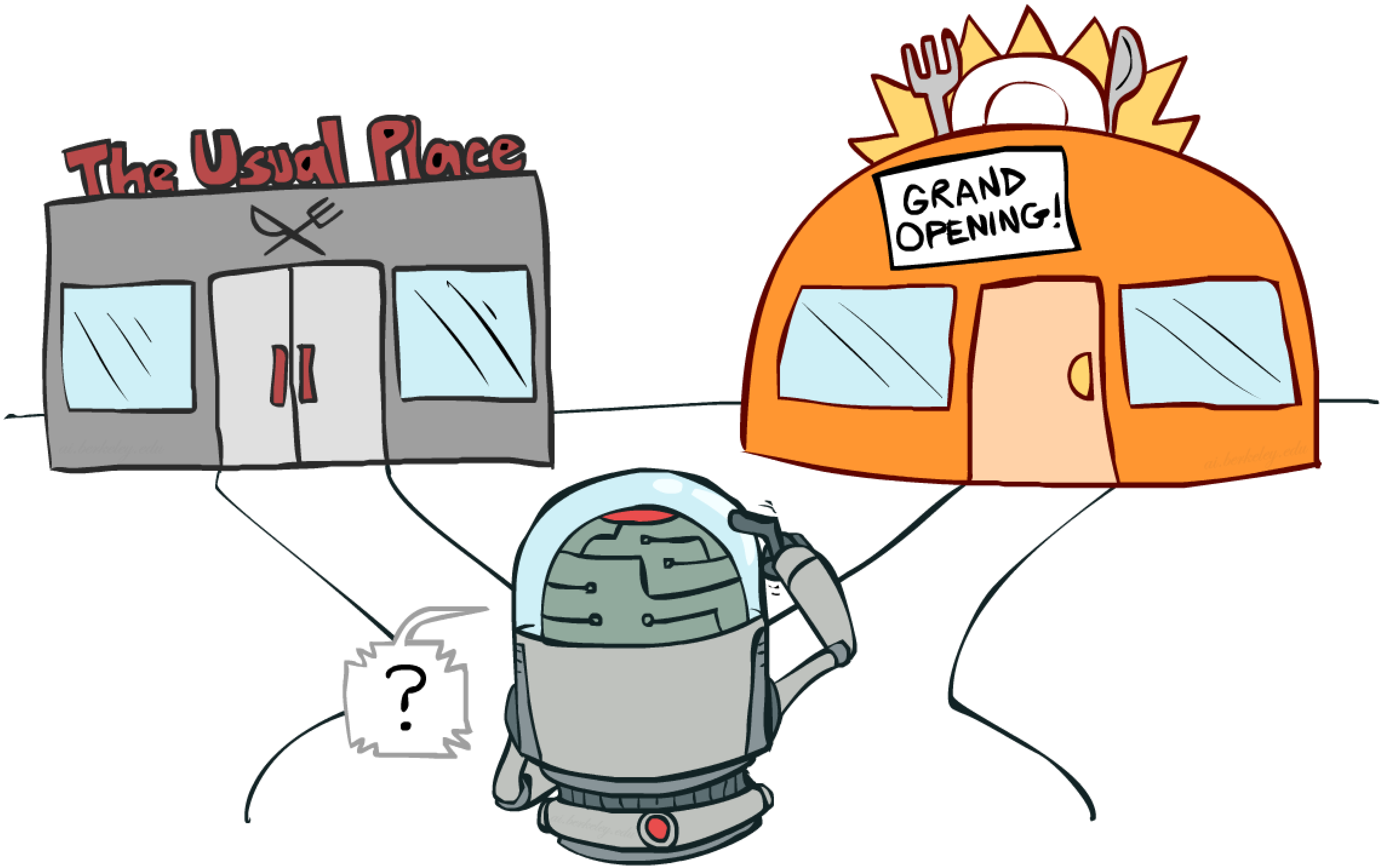
- **Idea:** learn how to act without explicitly learning the transition probabilities P(s' | s, a)
- **Q-learning:** learn an *action-utility function* Q(s,a) that tells us the value of doing action a in state s
- Relationship between Q-values and utilities:

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

- With Q-values, you don't need the transition model to select the next action:

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

Exploration vs. exploitation



Source: Berkeley CS188

Exploration vs. exploitation

Exploration: take an action with unknown consequences

- Pros:
  - Get a more accurate model of the environment
  - Discover higher-reward states than the ones found so far
- Cons:
  - When you're exploring, you're not maximizing your utility
  - Something bad might happen

Exploitation: go with the best strategy found so far

- Pros:
  - Maximize reward as reflected in the current utility estimates
  - Avoid bad stuff
- Cons:
  - Might also prevent you from discovering the true optimal strategy

Summary

- Reinforcement learning task
- Markov decision process
- Value functions & Bellman equation
- Value iteration
- Optional: Q-learning idea