CPSC 4420/6420

Artificial Intelligence

07 – Markov Decision Processes II

September 10, 2020

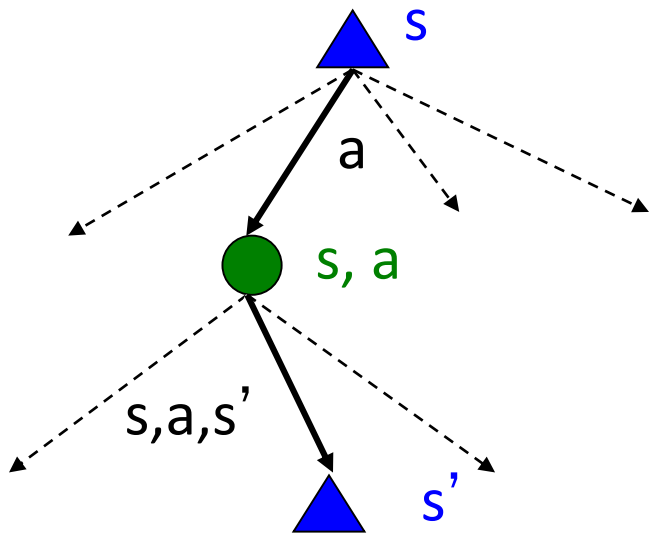
Announcements

- Project 2 will be released later today
  - Deadline is on 9/24
- Quiz 3 will be released later today
  - Deadline is next Thursday

Quick recap: MDPs

- Markov decision processes:
  - States S
  - Actions A
  - Transitions $P(s'|\, s, a)$
  - Rewards $R(s, a, s')$ (and discount γ)
  - Start state s_0
  - Horizon H

- Quantities so far:
  - Policy: map of states to actions
  - Utility: sum of (discounted) rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state



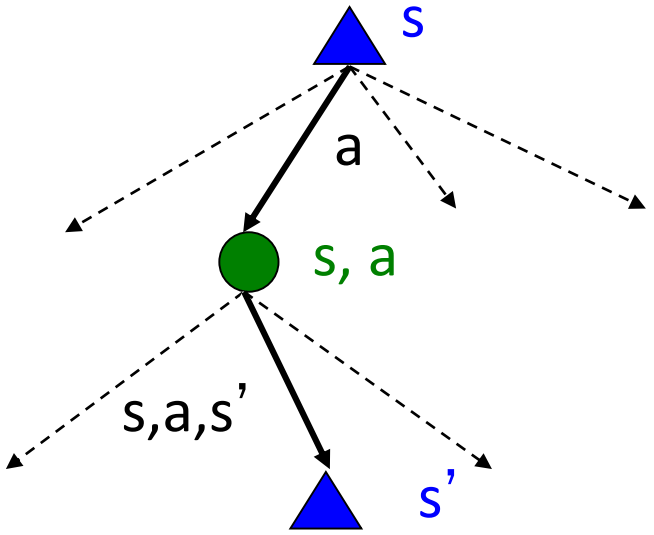
Quick recap: Bellman equations

- Recursive definition of values

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

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

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$



- These are called the Bellman equations
  - Can be solved using one-step lookahead relationship amongst optimal values

Quick recap: Value iteration algorithm

function Value-Iteration (MDP)

Initialize $V_0^*(s) = 0, \forall\, s$

for k=1…H do

for each state s in S do

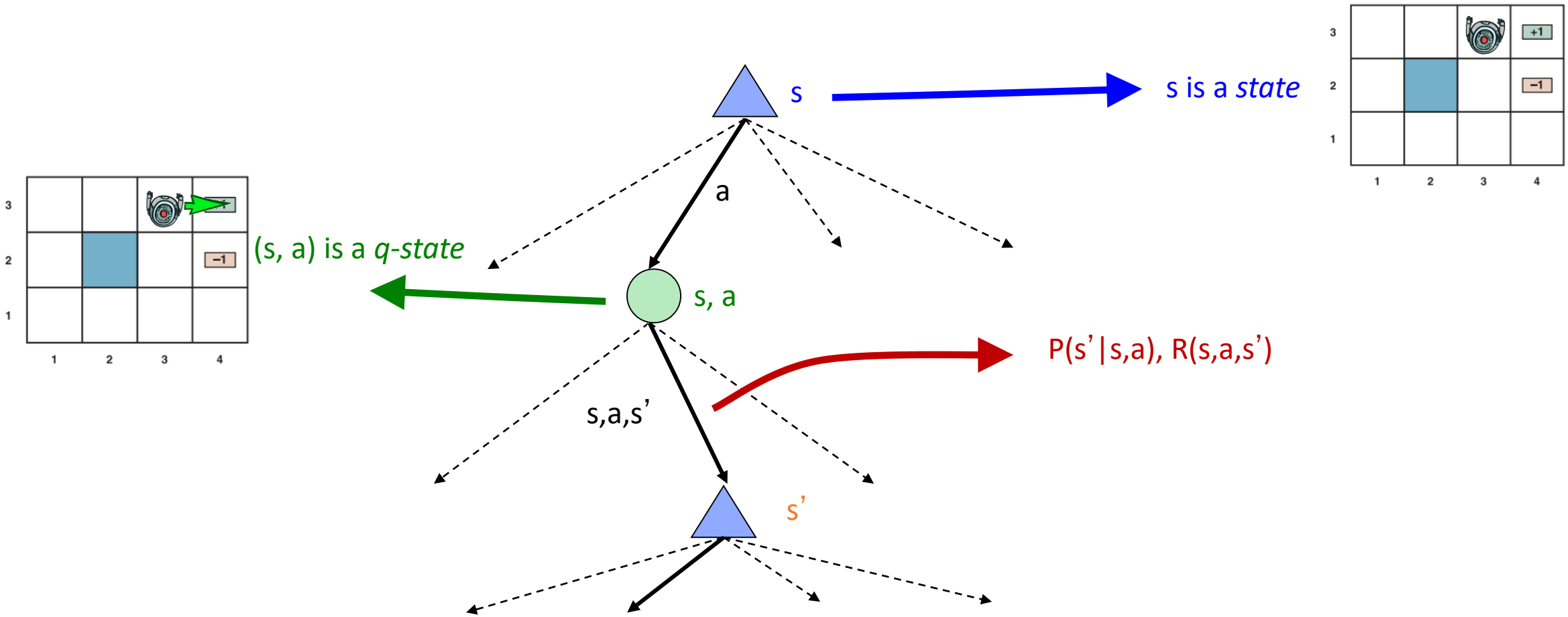
$$V_k^*(s) \leftarrow \max_a \sum P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

This is called a value update or Bellman update/back-up

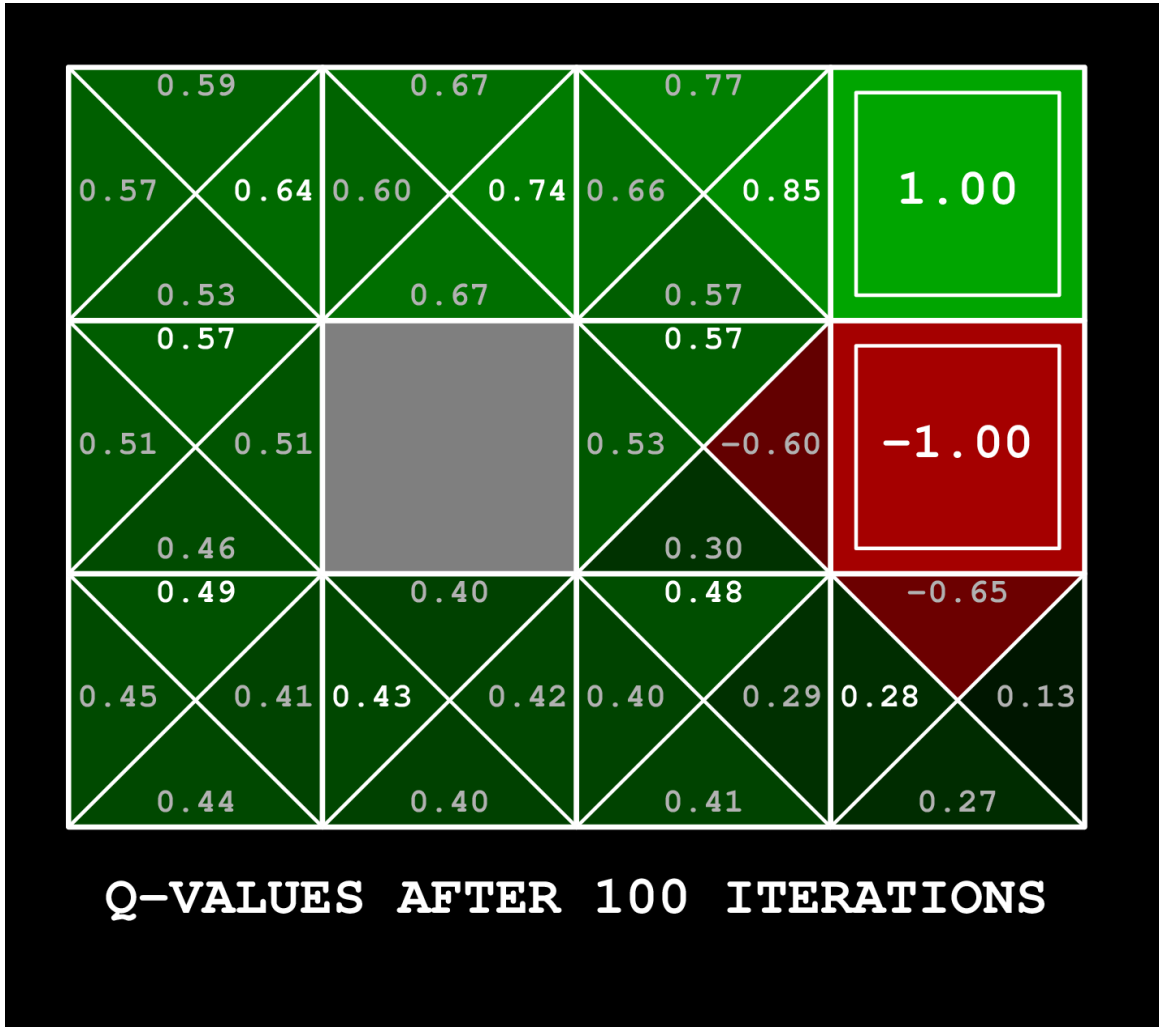
Implementation: Use two vectors of size |S| storing value functions, one at time k and one at k-1

MDP search tree



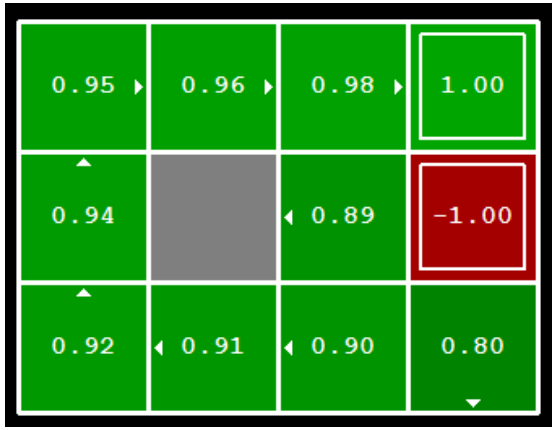
Solving MDPs: Policy Iteration

Gridworld: V and Q**



Extracting policy from V's*

- Consider having computed the optimal values V*(s)



- How should we act?
  - It's not obvious!
- We need to keep track of the optimal policy during value iteration

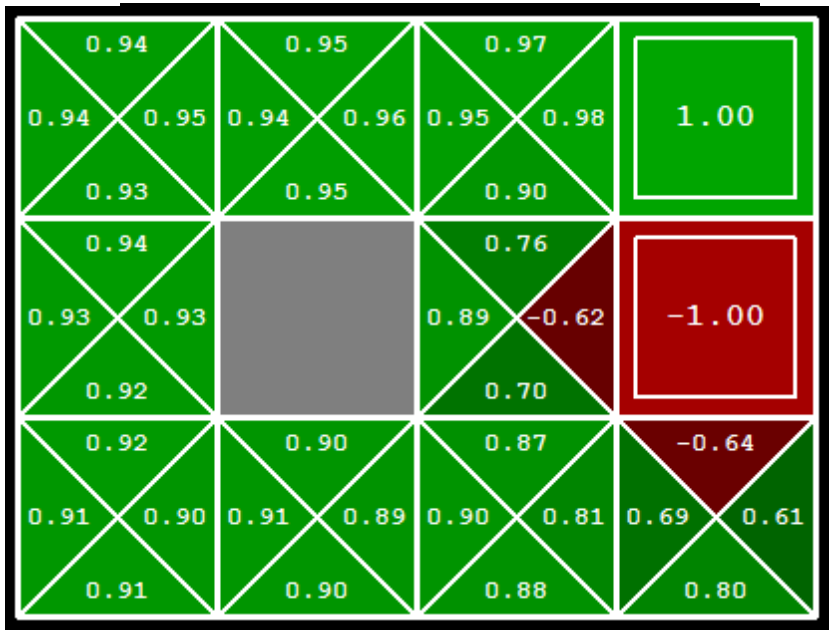
$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

- This is called policy extraction, since we retrieve the policy implied by the values

Extracting policy from Q's*

- Consider having computed the optimal q-values Q*(s)
- How should we act?
  - It's trivial

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

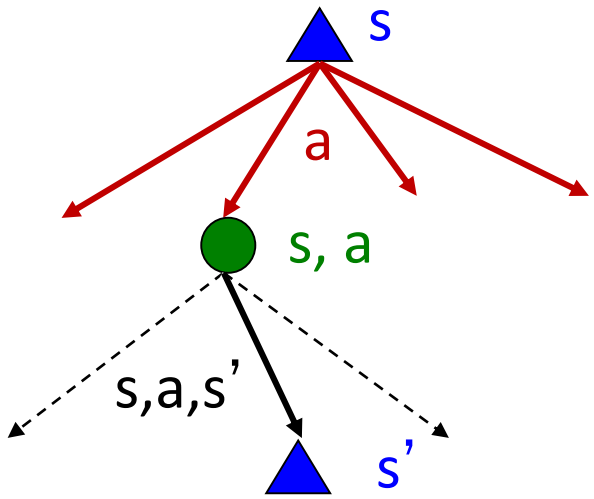


Issues with value iteration

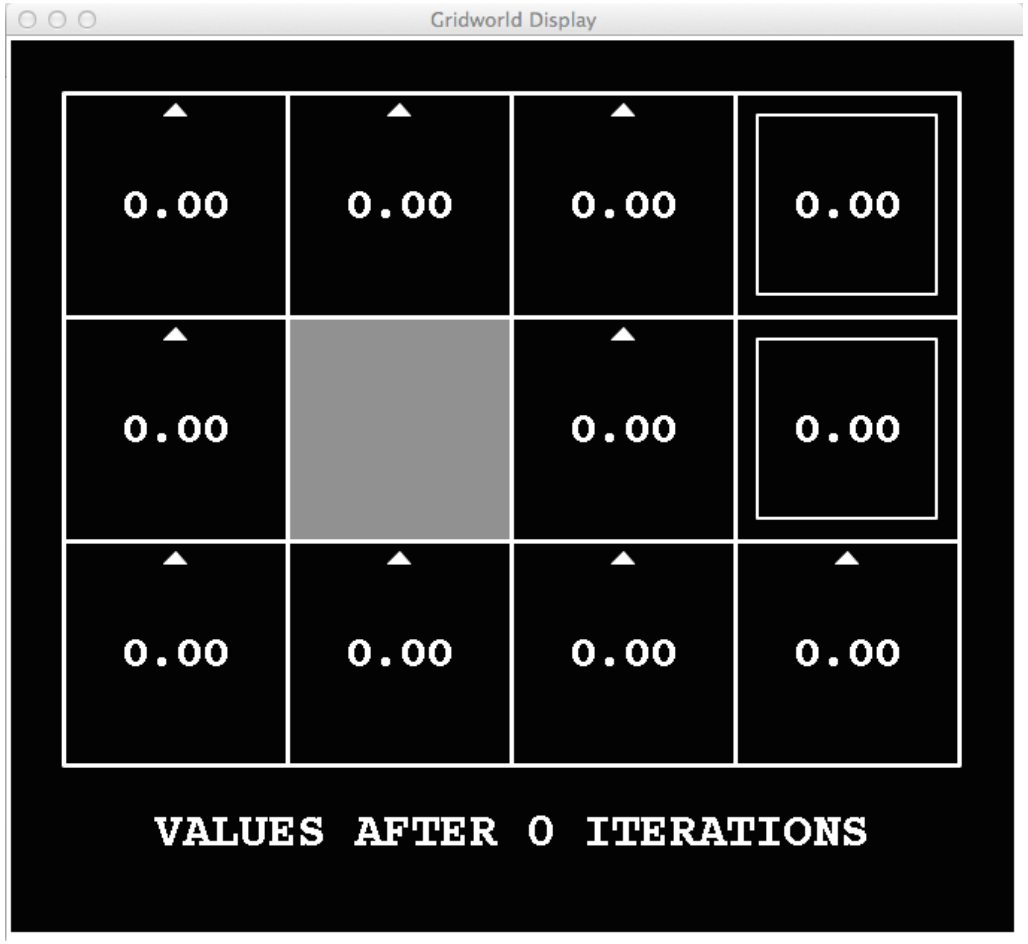
- Value iteration repeats the Bellman updates:

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

- Issues
  1. It's slow – $O(S^2A)$ per iteration
  2. The "max" at each state rarely changes
  3. The policy often converges long before the values

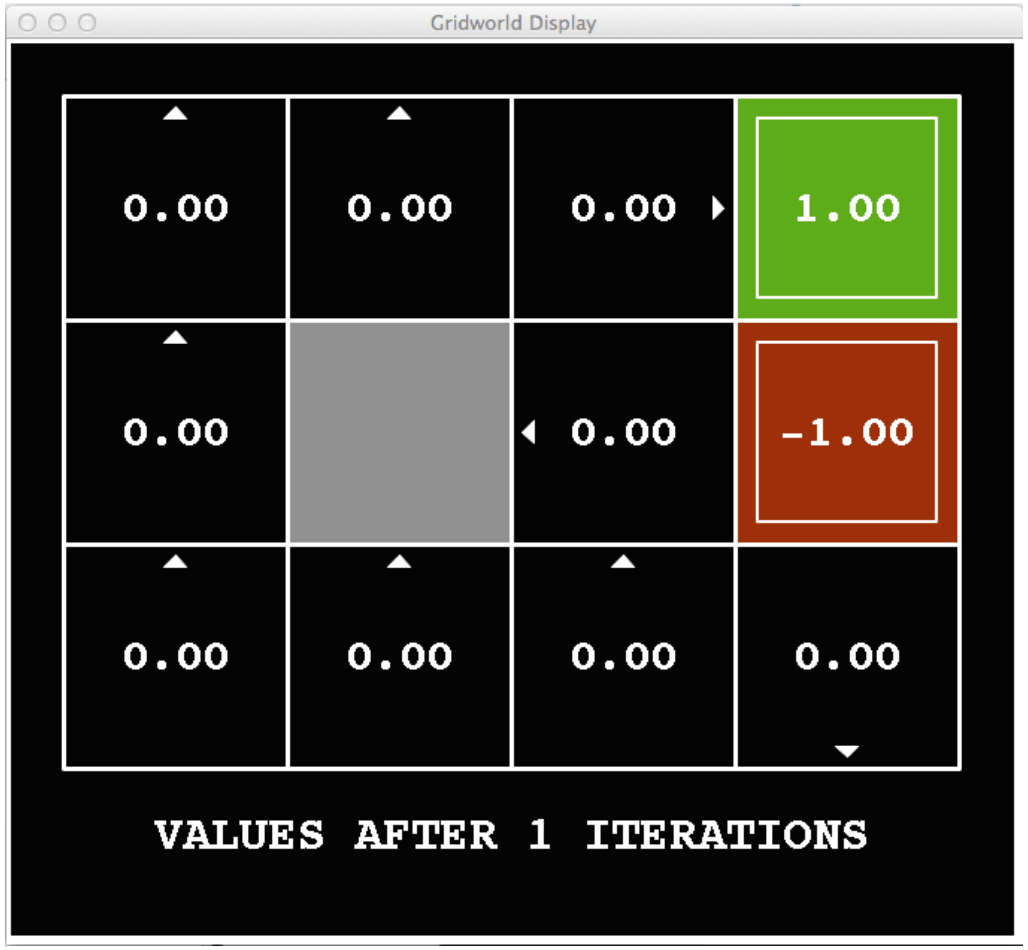


k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

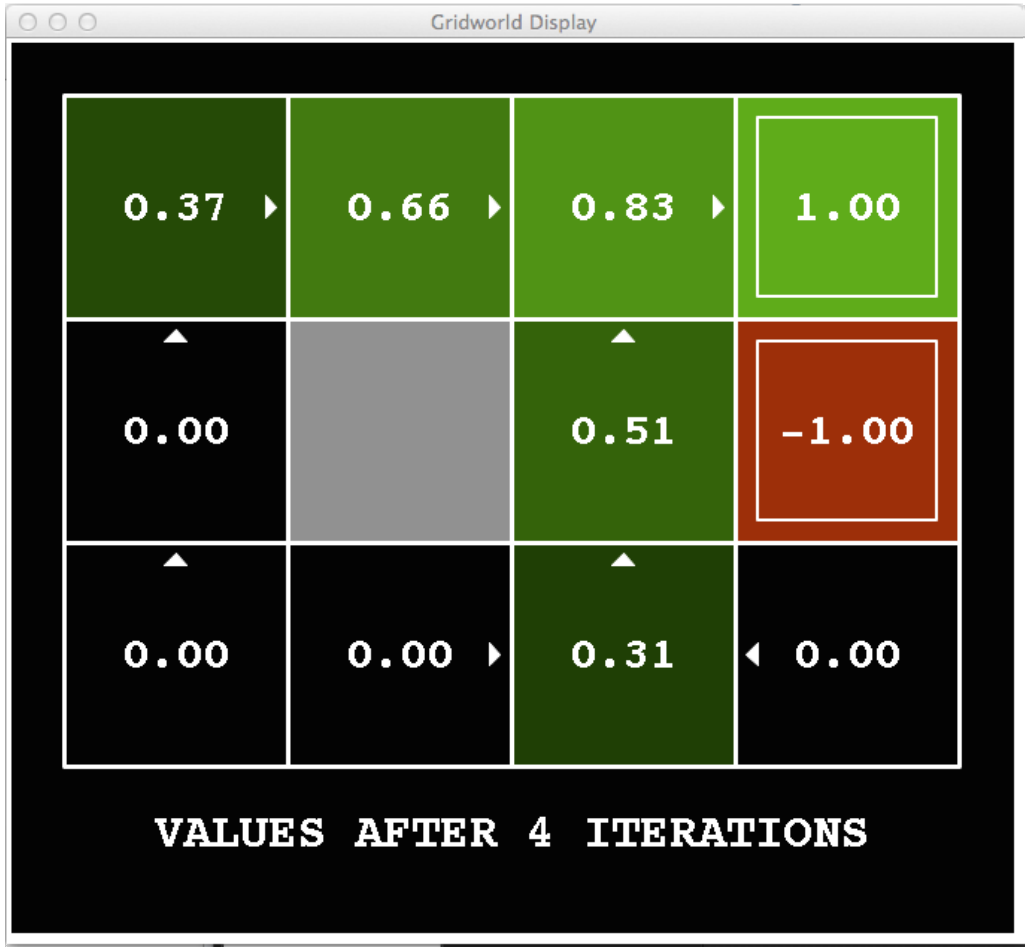
k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

k=3

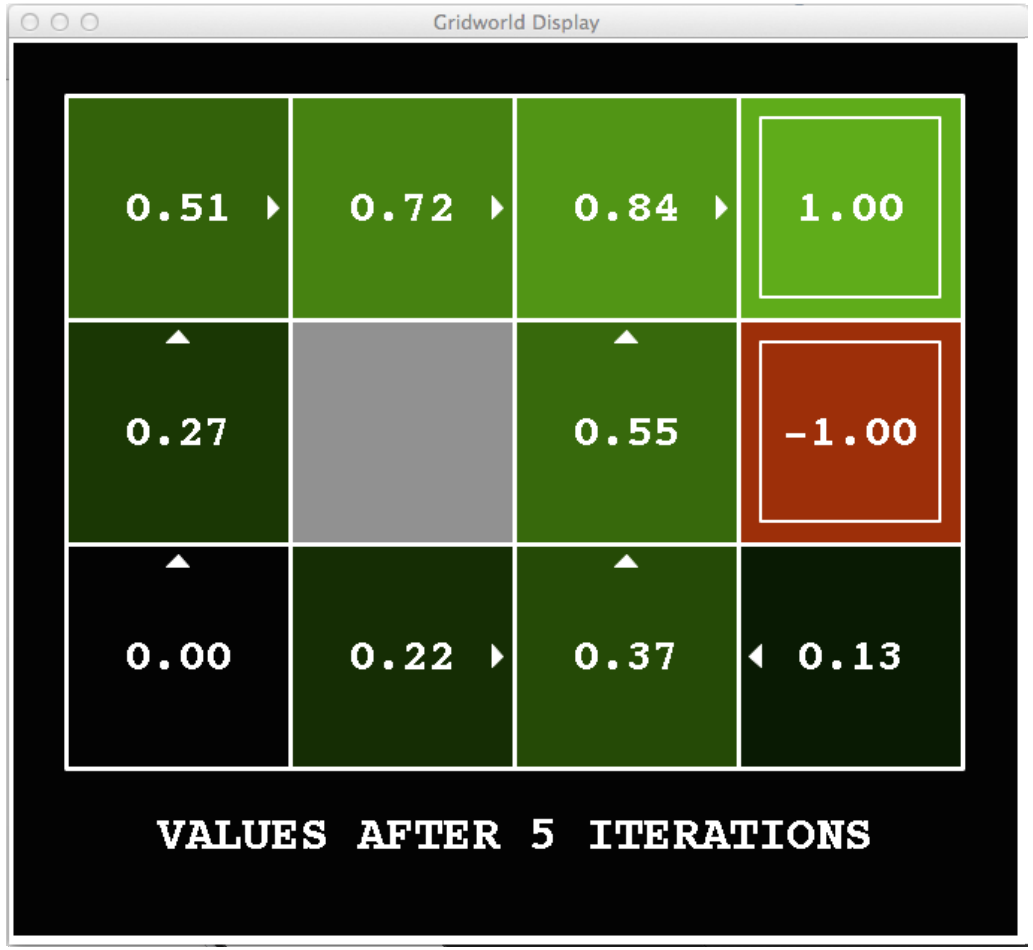
Noise = 0.2
Discount = 0.9
Living reward = 0

k=4



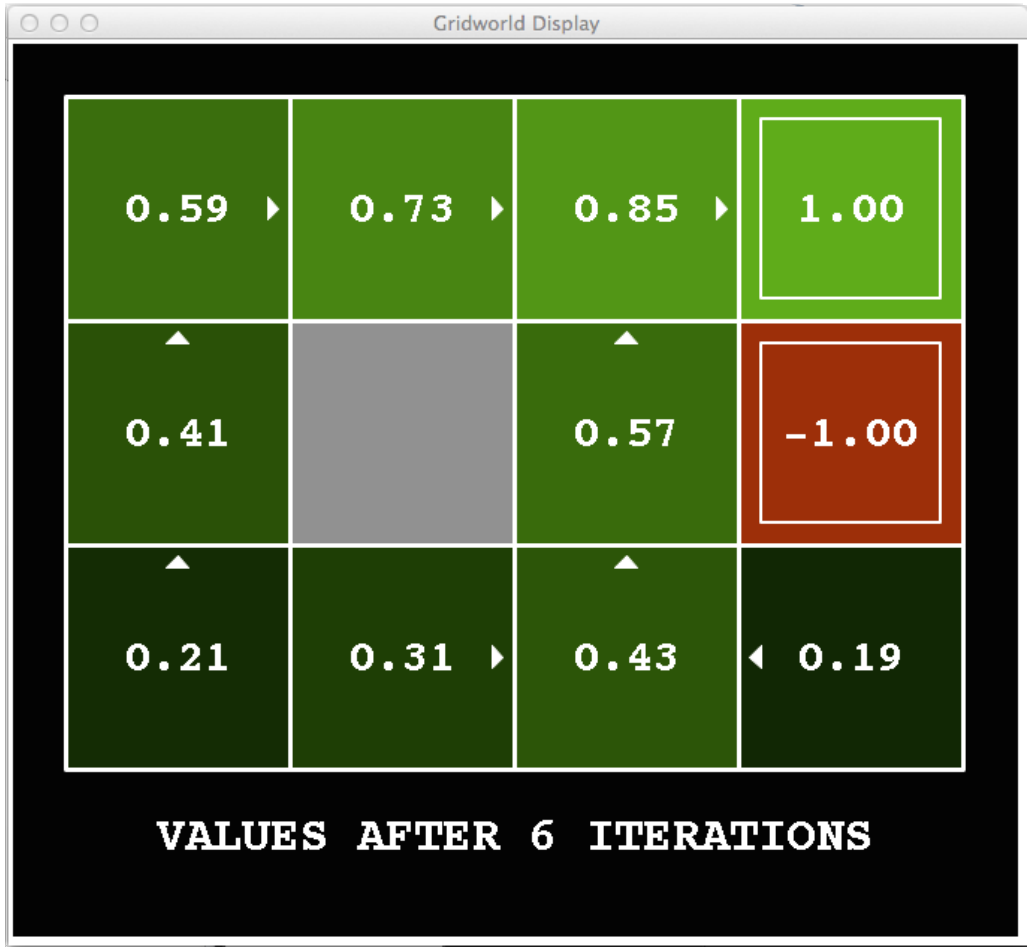
Noise = 0.2
Discount = 0.9
Living reward = 0

k=5



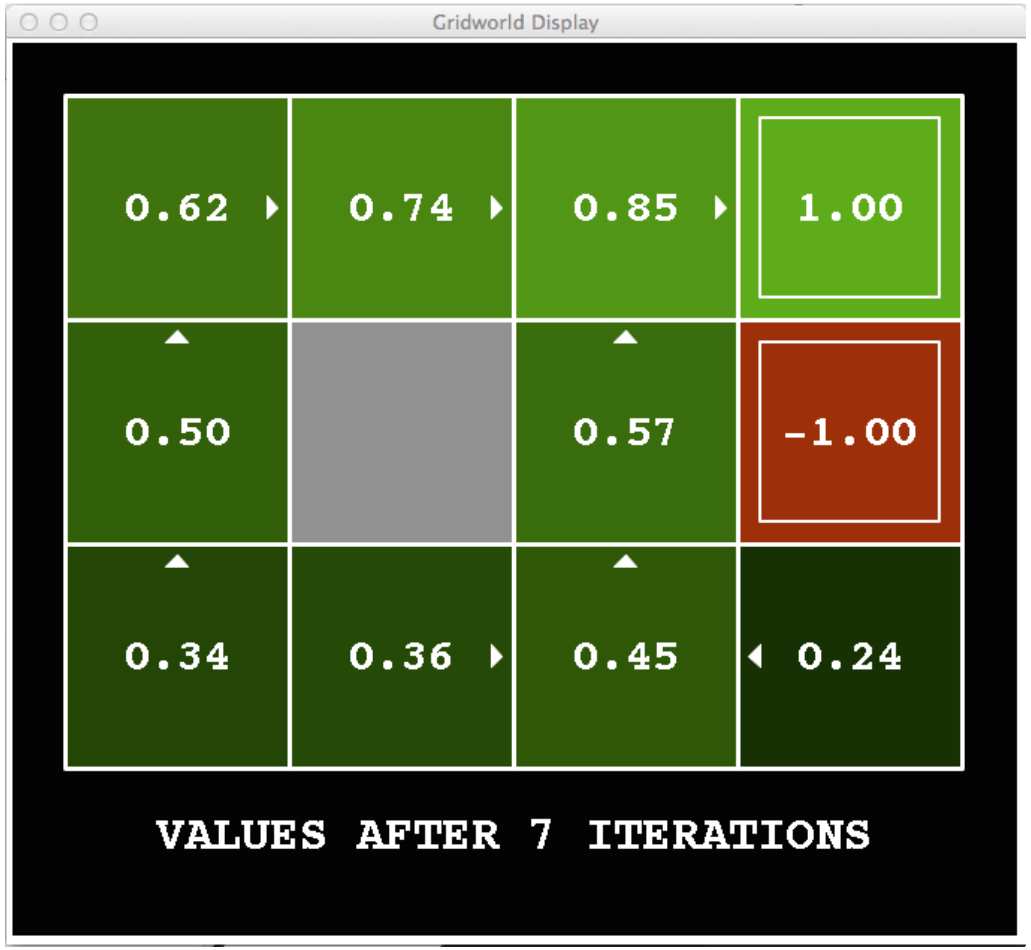
Noise = 0.2
Discount = 0.9
Living reward = 0

k=6



Noise = 0.2
Discount = 0.9
Living reward = 0

k=7

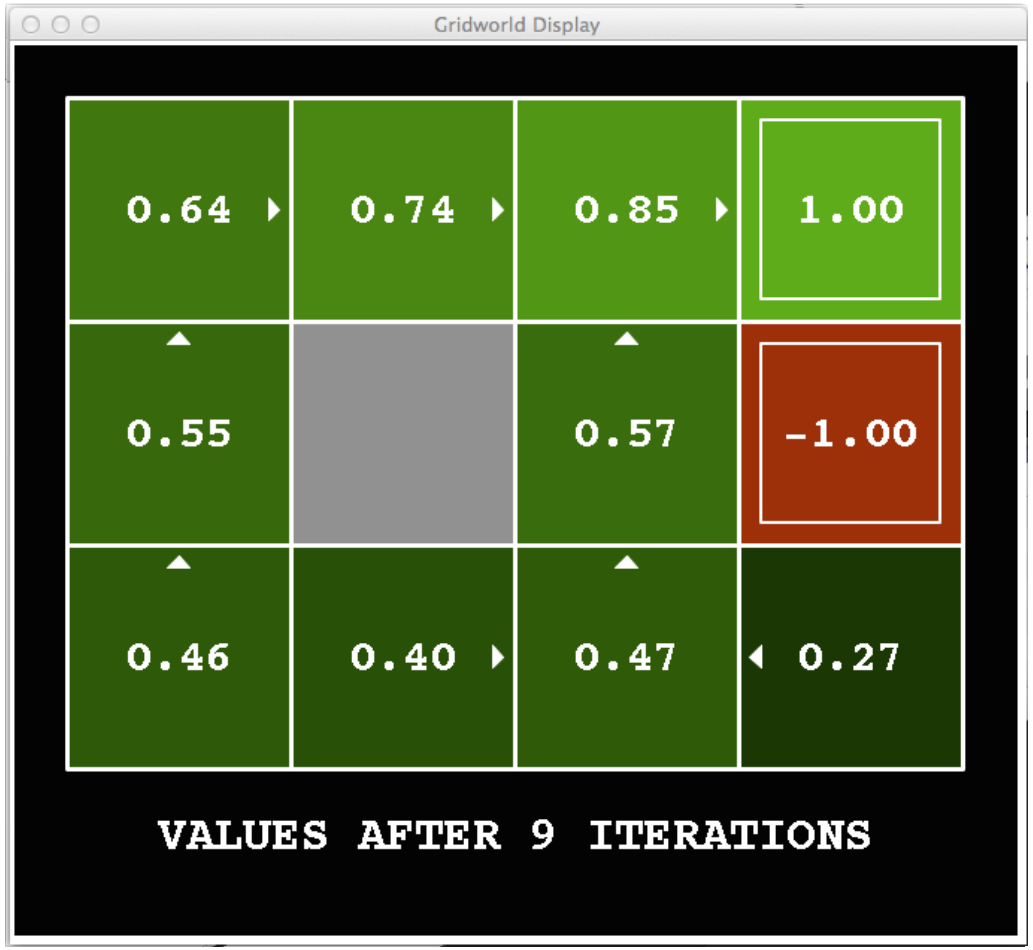


Noise = 0.2
Discount = 0.9
Living reward = 0

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

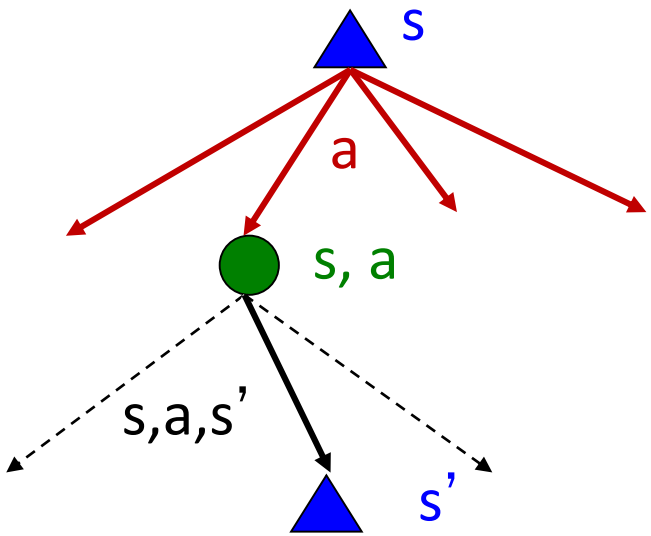
Noise = 0.2
Discount = 0.9
Living reward = 0

k=100

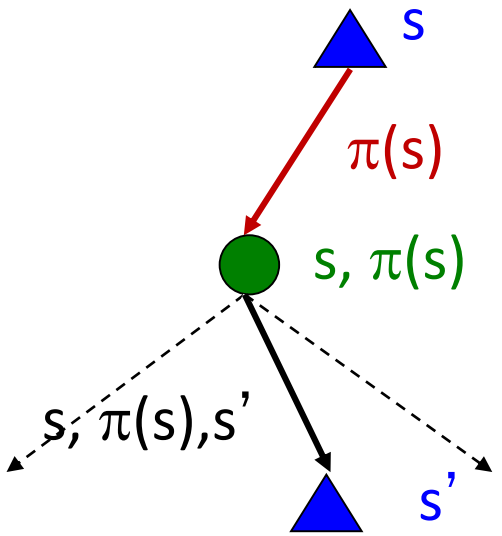
Noise = 0.2
Discount = 0.9
Living reward = 0

Policy evaluation

Do the optimal action



Do what π says to do



- In value iteration, we max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then only one action per state
  - $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π
  - The value depends now on which policy we fixed

Policy evaluation

- Recall value iteration

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

- Policy evaluation

$$V_k^\pi(s) \leftarrow \sum_{s'} P(s'|s,\pi(s))(R(s,\pi(s),s') + \gamma V_{k-1}^\pi(s'))$$

  - At convergence

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

  - Could also compute $Q^\pi(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V^\pi(s'))$

Example: policy evaluation

Always go east

Always go north

Example: policy evaluation

Always go east

Always go north

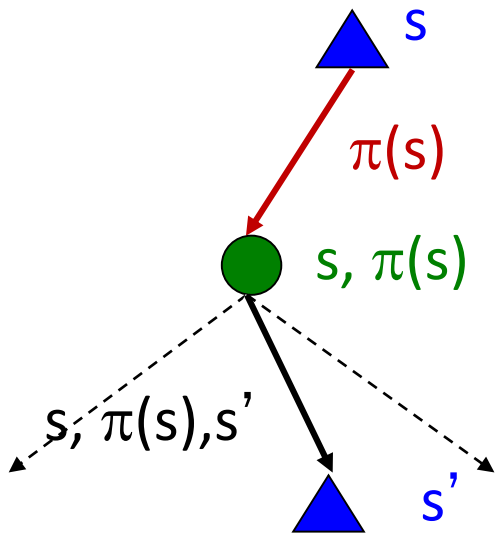
Policy evaluation revisited

- Idea 1: modify Bellman updates

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

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

  - Efficiency: $O(S^2)$ per iteration, compare to $O(S^2A)$ of value iteration



- Idea 2: It is just a linear system; solve it, e.g., with Numpy

  variables: $V^\pi(s)$

  constants: P, R

$$\forall s \quad V^\pi(s) = \sum_{s'} P(s'|s,\pi(s))\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

Policy iteration

One iteration of policy iteration

1. Policy evaluation for fixed current policy π:
   - Iterate until values converge:

$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} P(s'|s, \pi(s)) \left[R(s, \pi(s), s') + \gamma V^{\pi}_{k}(s')\right]$$

2. Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_{a} \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

- Repeat steps until policy converges
- At convergence: optimal policy!
- Can converge (much) faster than value iteration under some conditions

Policy iteration guarantees

Theorem. Policy iteration is guaranteed to converge and at convergence, the current policy and its value function are the optimal policy and the optimal value function!

- At convergence: optimal policy!
- Can converge (much) faster than value iteration under some conditions

Comparisons

- Both value iteration and policy iteration are dynamic programs for solving MDPs
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We could track the policy, or take the max over actions to implicitly re-compute it
- In policy iteration:
  - Evaluate a 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)

Summary

- Given an MDP find the optimal policy π*

$$\pi^* = \text{argmax}_\pi \, E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi\right]$$

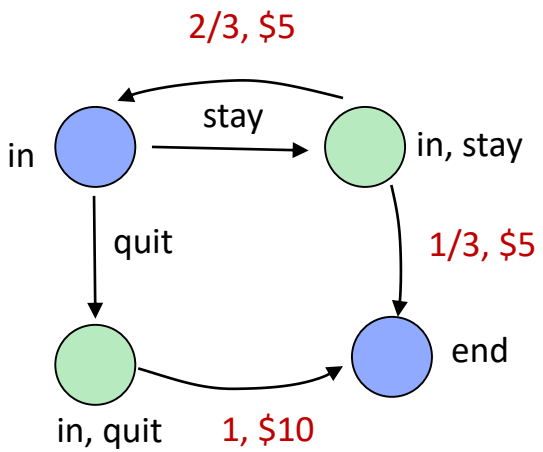
- Exact methods
  - Value Iteration
  - Policy Iteration

Limitations

- Iteration over / storage for all states and actions: requires small, discrete state-action space
- Update equations require access to dynamics model and reward function

Quiz: What is V^{stay}?

- At each round
  - You have the option to stay or quit
  - If quit, the game is ended and you get $10
  - If stay, you get $5 and then roll the dice
    - If the result is 1 or 2, the game is ended
    - Otherwise, the game continues to the next round



	end	in
V_0^{stay}	0	0
V_1^{stay}	0	1/3*[5+V_0^{stay}(end)] + 2/3*[(5+V_0^{stay}(in)]= 5
V_2^{stay}	0	1/3*[5+V_1^{stay}(end)] + 2/3*[(5+V_1^{stay}(in)]= 25/3

Reinforcement Learning

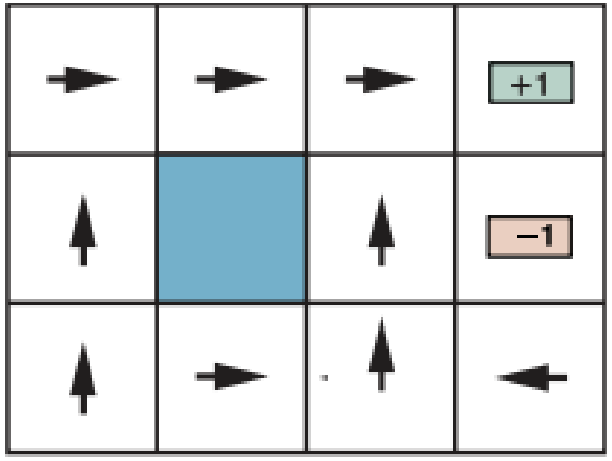
Defining RL problems

- Still assume a MDP:
  - A set of states S
  - A set of actions A per state
  - A transition model $P(s' | s, a)$
  - A reward function $R(s, a, s')$
- Still looking for a policy $\pi(s)$
- New twist: don't know *P* and *R*
  - We don't know which states are good or what the actions do
  - Must actually try out actions and states to learn

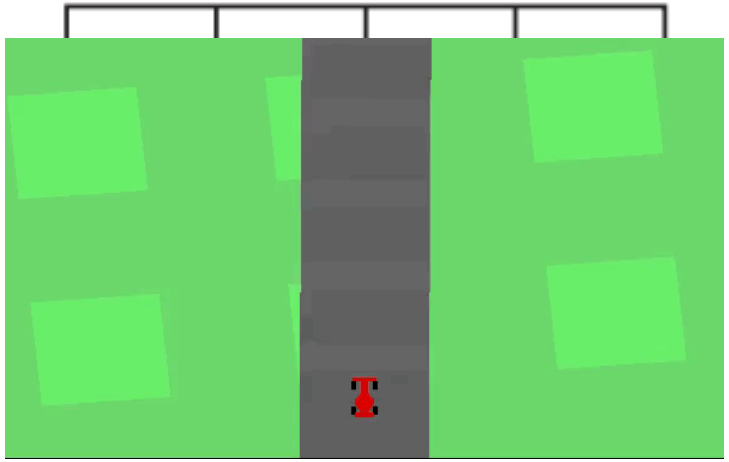
Defining RL problems

- Still assume a MDP:
  - A set of states S
  - A set of actions A per state
  - A transition model $P(s' | s, a)$
  - A reward function $R(s, a, s')$
- Still looking for a policy $\pi(s)$
- New twist: don't know *P* and *R*
  - We don't know which states are good or what the actions do
  - Must actually try out actions and states to learn

Offline vs Online



MDP (offline solution)



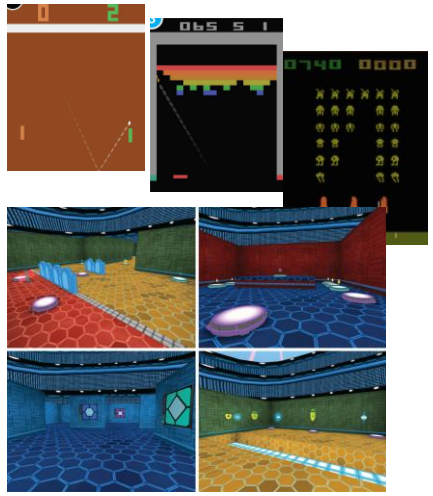
RL (online learning)

Some RL success stories

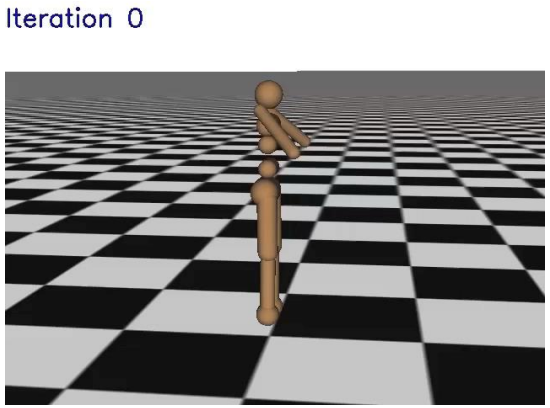
[Kohn and Stone, 2004]

[Ng et al., 2004]

[Tedrake et al., 2005]



[Mnih et al. 2013, 2015]



[Schulman et al, 2016]

[silver, Huang et al, 2016]

Learning to walk [Kohn and Stone, 2004]

Initial

Training

After learning [1K trials]

Learning to walk [Tedrake et al., 2005]

Our bot