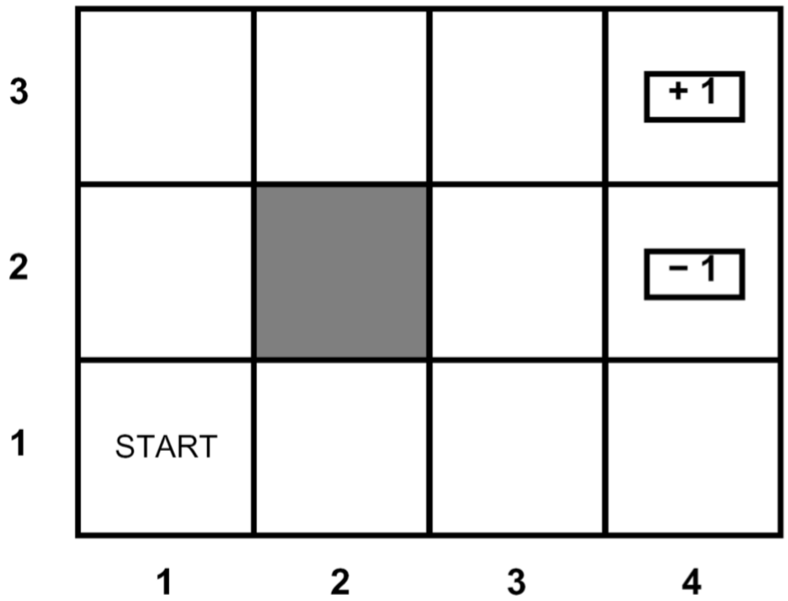
CSE 473

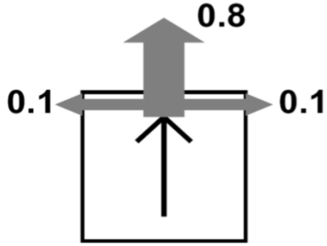
Lecture 16

Markov Decision Processes (MDPs) Part II

Recall: Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s,a,s')
    - Probability that action a in s leads to s'
      i.e., P(s' | s,a)
    - Also called "the model"
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state



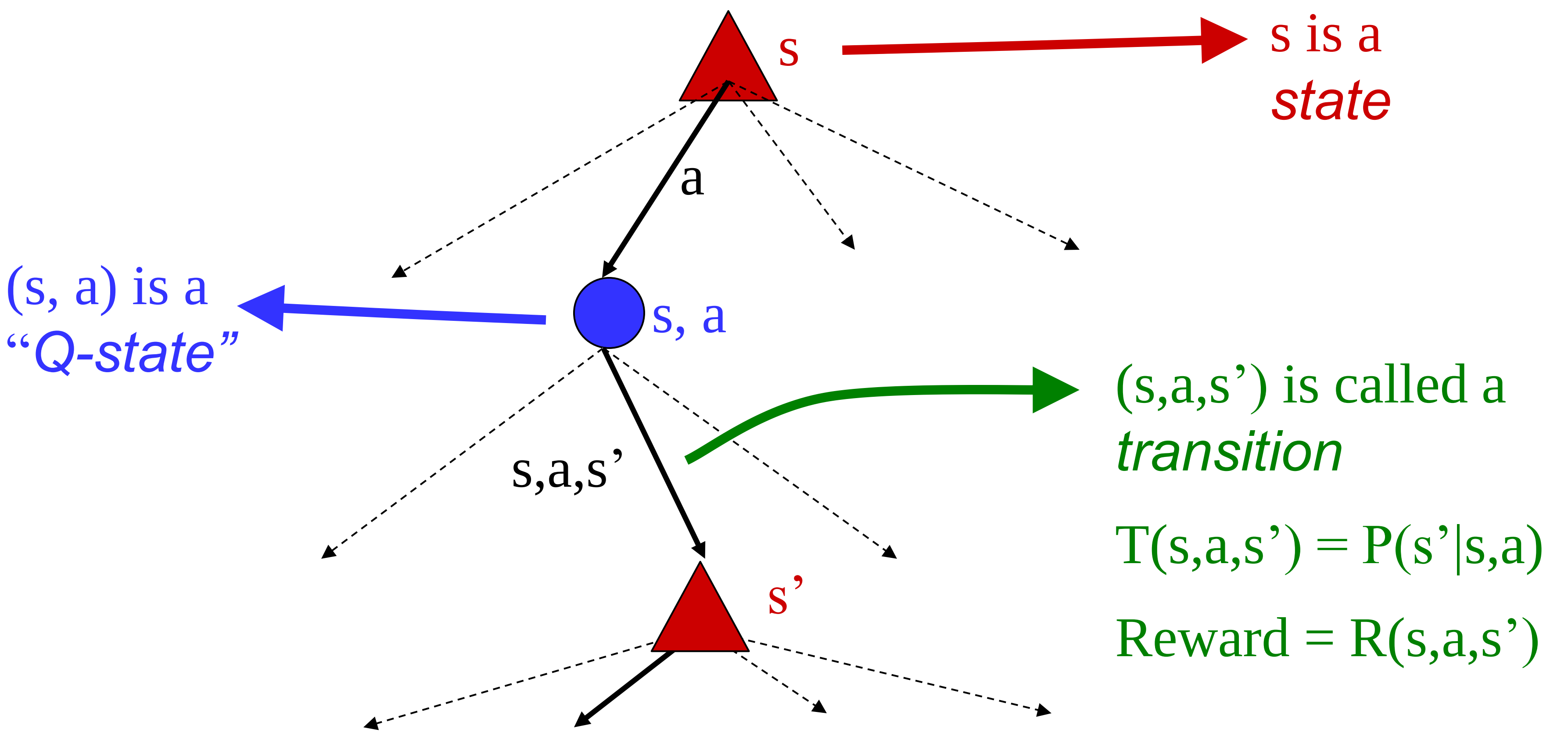


Action = North (N)

Similarly for actions E, S, W

MDP Search Trees

- Each MDP state gives an expectimax-like search tree



Utilities of Reward Sequences

- What is an "optimal" policy?
  - Each transition s,a,s' produces a reward (+ve, -ve, or 0)
  - Pick actions that maximize utility
  - Need to define utility of a *sequence* of rewards
- Idea 1:
  - Additive utility:

$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

- Problem: Infinite state sequences have infinite total reward

Defining Utilities

- Solution:
  - Discounting: Make infinite sum finite using γ $(0 < \gamma < 1)$

$$U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$$

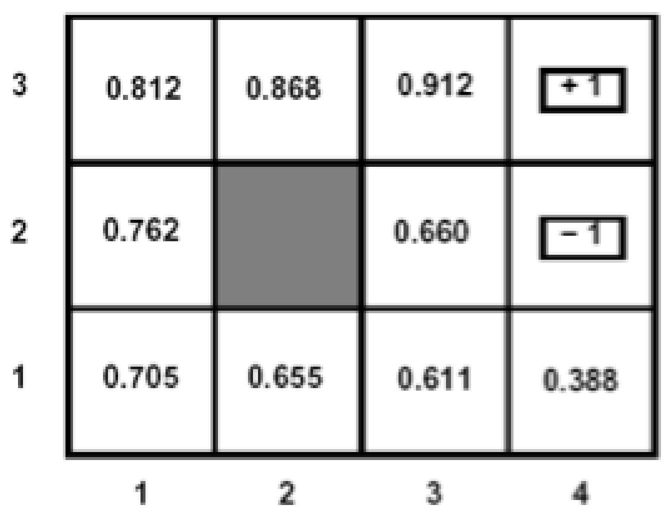
$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\text{max}}/(1 - \gamma)$$

  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge

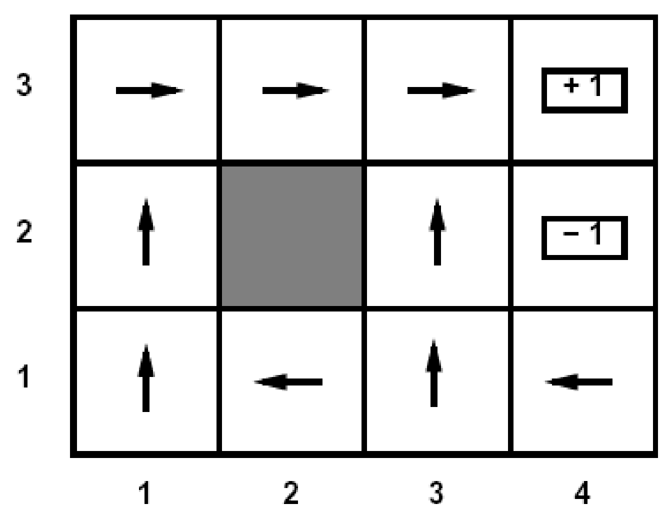
Defining the Optimal Policy

- Define the value of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally
- Define the value of a Q-state (s,a):
  $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally
- Define the optimal policy:
  $\pi^*(s)$ = optimal action from state s

Values



Optimal Policy



Bellman Equation

- Simple one-step look-ahead *recursive* relationship between optimal utility values
- Start with:

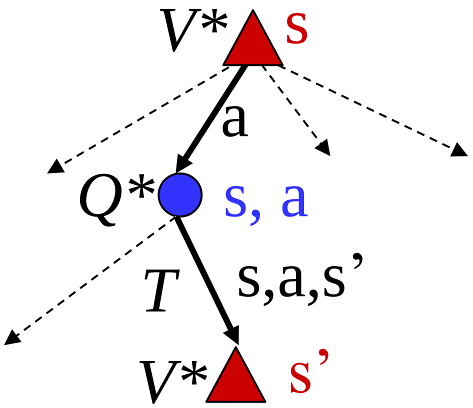
$$V^*(s) = \max_a Q^*(s,a)$$

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

- Combine to get Bellman Equation:

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

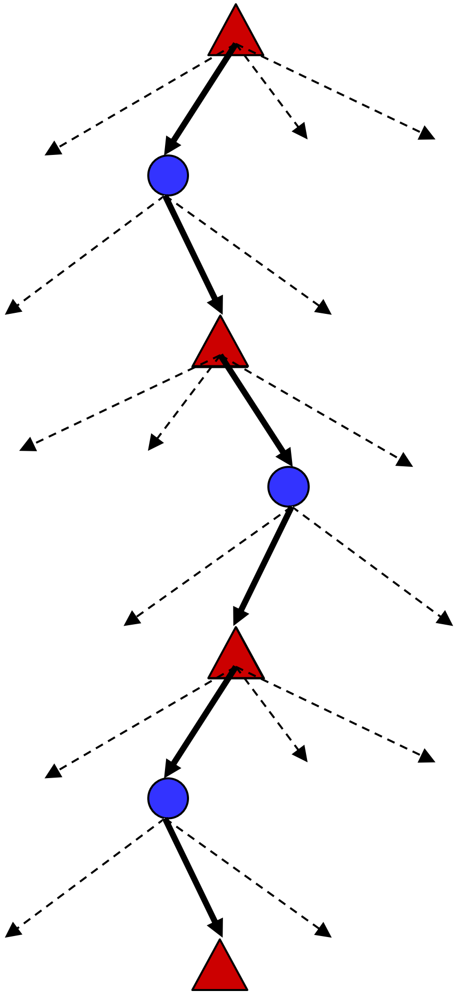
recursive



Richard Bellman
(1920-1984)

Why not use Expectimax?

- Problems:
  - The tree is usually infinite
  - Same states appear over and over
  - Need to search once for each state
- Idea: Value iteration
  - Compute optimal values for all states all at once iteratively
  - Bottom-up dynamic programming
  - Simple table look-up for any state
- Calculates estimates $V_k^*(s)$ in iteration k
  - The optimal value considering only *next k* time steps (next *k* rewards)
  - As $k \rightarrow \infty$, V_k approaches the optimal value



Value Iteration (VI)

- Idea:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth i+1:

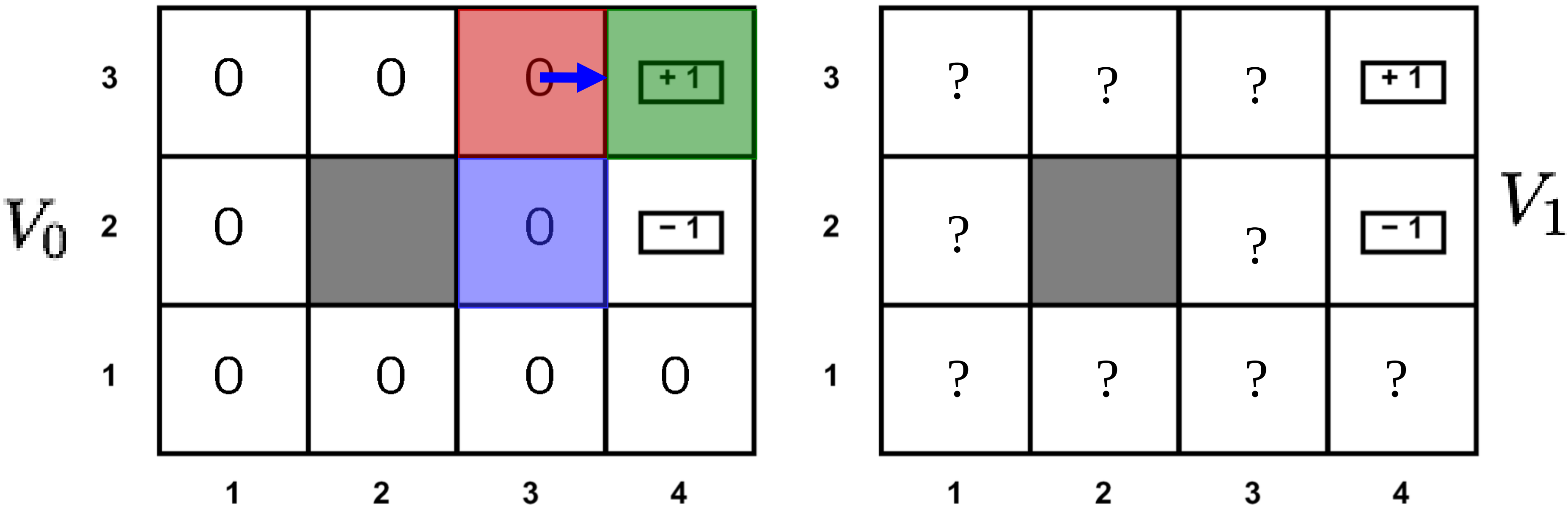
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

  - This is called a value update or Bellman update
  - Repeat until convergence

- Theorem: VI will converge to optimal values
  - Basic idea: approximations get refined towards optimal values

Example: Bellman Updates

Example: γ=0.9, noise=0.2, living penalty=0



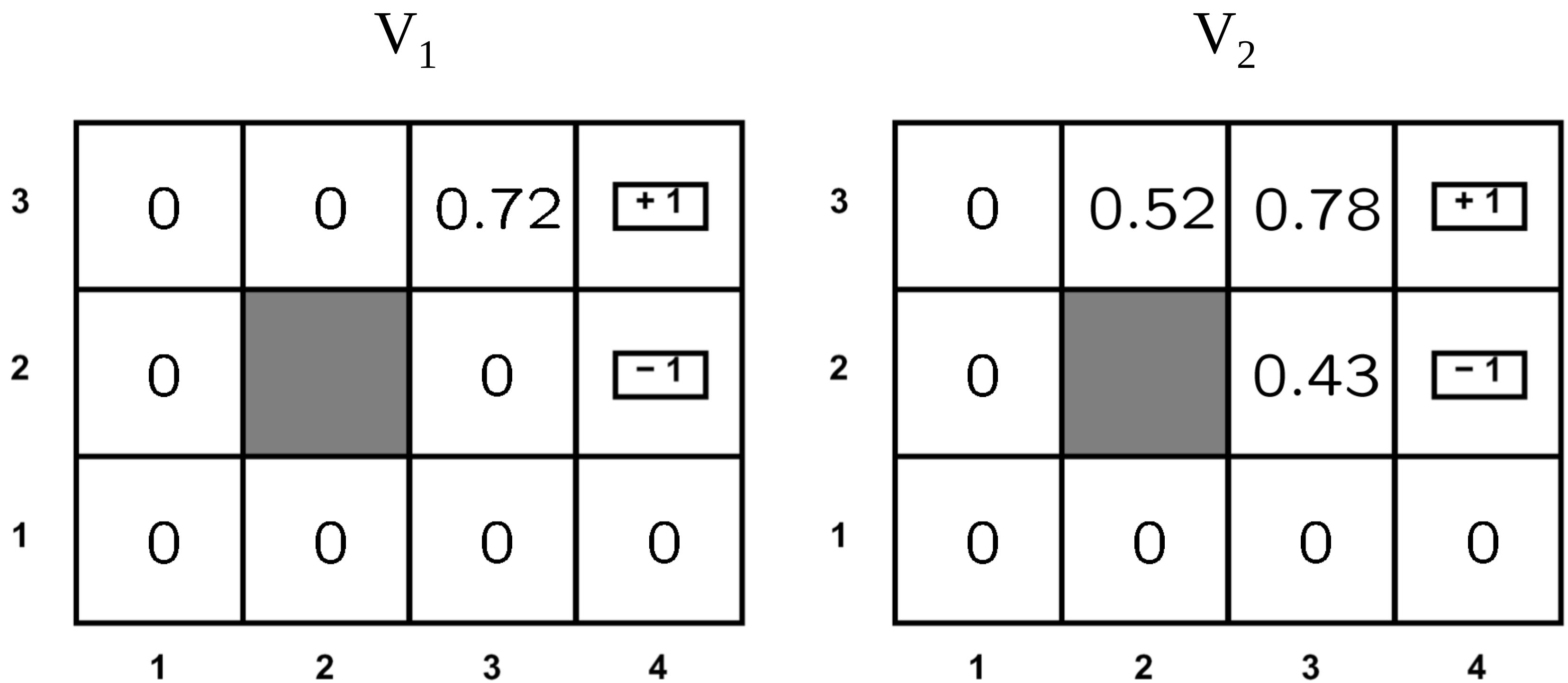
$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right] = \max_a Q_{i+1}(s,a)$$

$$Q_1(\langle 3,3 \rangle, \text{right}) = \sum_{s'} T(\langle 3,3 \rangle, \text{right}, s') \left[R(\langle 3,3 \rangle, \text{right}, s') + \gamma V_i(s') \right]$$

$$= 0.8 * [0.0 + 0.9 * 1.0] + 0.1 * [0.0 + 0.9 * 0.0] + 0.1 * [0.0 + 0.9 * 0.0]$$

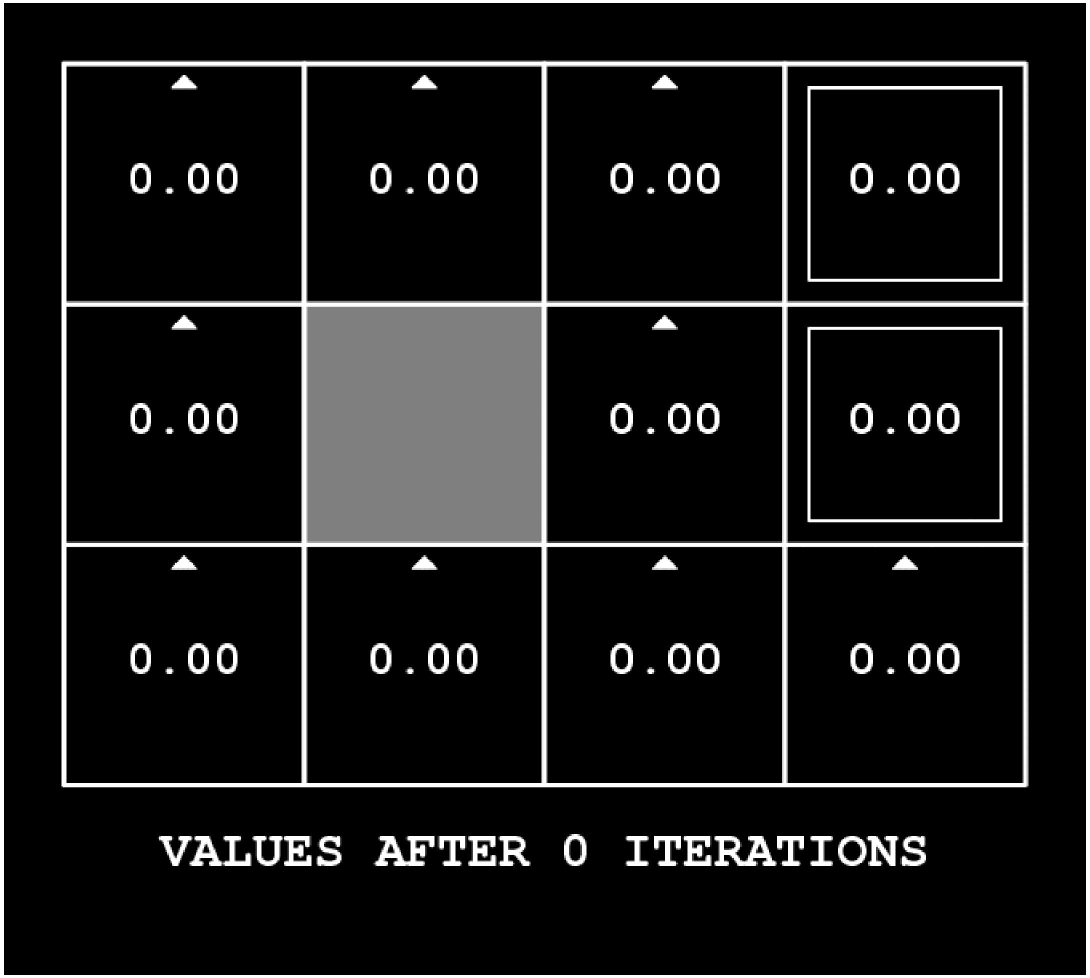
= 0.72

Example: Value Iteration



- Information propagates outward from terminal states and eventually all states have correct value estimates

Example: Value Iteration (Movie)

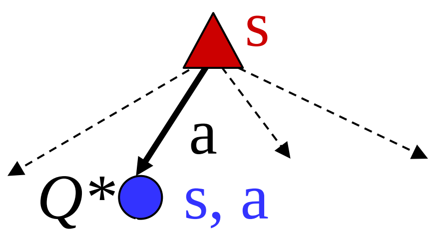


Optimal Policy: Computing Actions

- Which action to chose in state s:

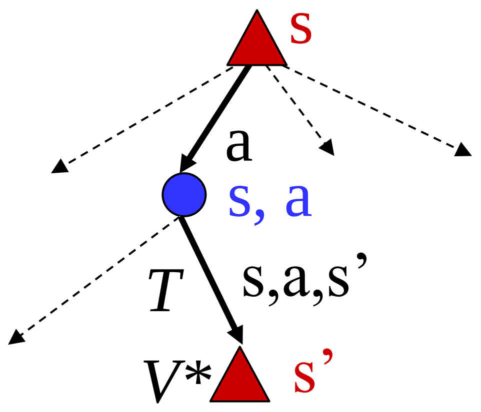
  - Given optimal *Q**?

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



  - Given optimal values *V**?

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



Value Iteration Complexity

- Problem size:
  - |A| actions and |S| states
- Each Iteration

  For all s:
  $$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V_i(s')\right]$$

  - Time: $O(|A| \cdot |S|^2)$
  - Space: $O(|S|)$
- Num of iterations to converge
  - Can prove that it can be exponential in the discount factor γ

Is there a faster alternative to value iteration?

Yeah, crazy little thing called policy iteration!

Next Time

- Policy Iteration and Reinforcement Learning
- To Do
  - Read chapters 17 and 21