CSE 473: Artificial Intelligence

Markov Decision Processes (MDPs)

Hanna Hajishirzi

Many slides over the course adapted from Luke Zettlemoyer,
Dan Klein, Pieter Abbeel, Stuart Russell or Andrew Moore

Outline (roughly next two weeks)

- Markov Decision Processes (MDPs)
  - MDP formalism
  - Value Iteration
  - Policy Iteration

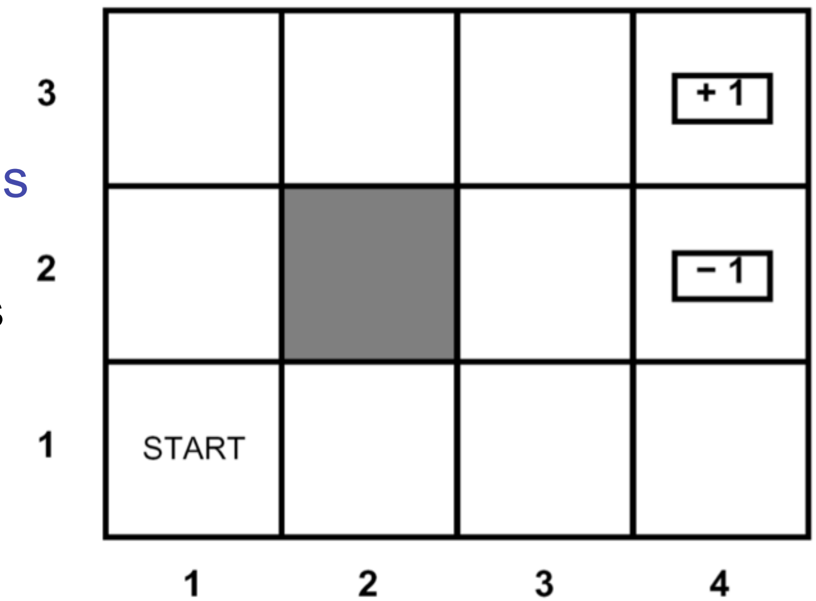
- Reinforcement Learning (RL)
  - Relationship to MDPs
  - Several learning algorithms

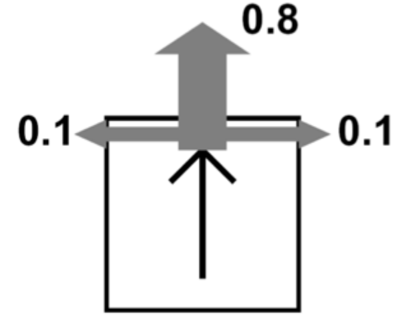
Non-deterministic Search

- Noisy execution of actions
  - Deterministic grid world vs. non-deterministic grid world

Example: Grid World

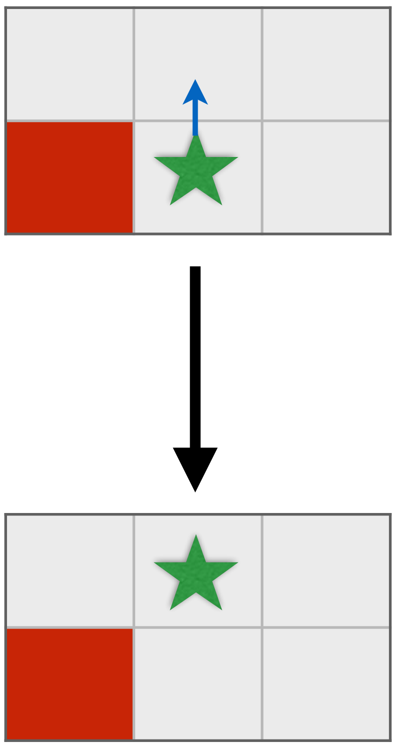
- A maze-like problem:
  - The agent lives in a grid
  - Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North
    (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Agent receives rewards each time step:
  - Small "living" reward each step
  - Big rewards come at the end
- Goal: maximize sum of rewards



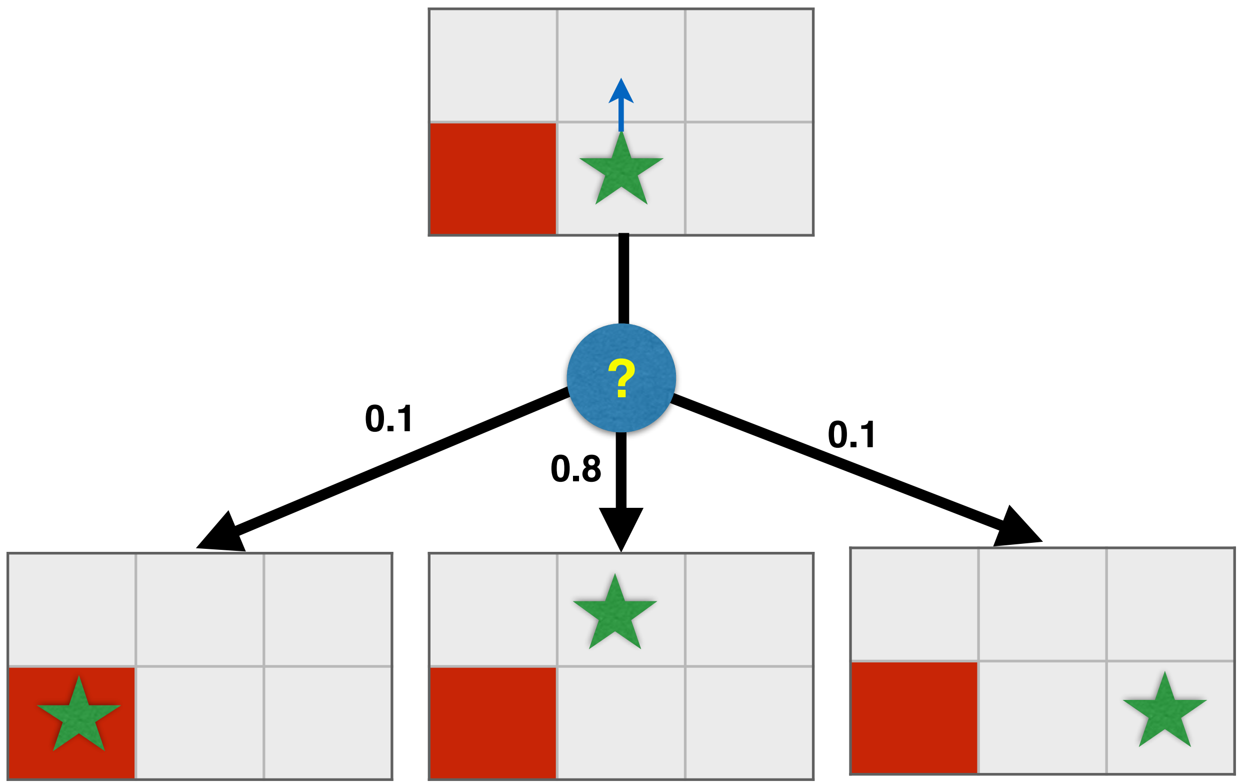


Grid World Actions

Deterministic



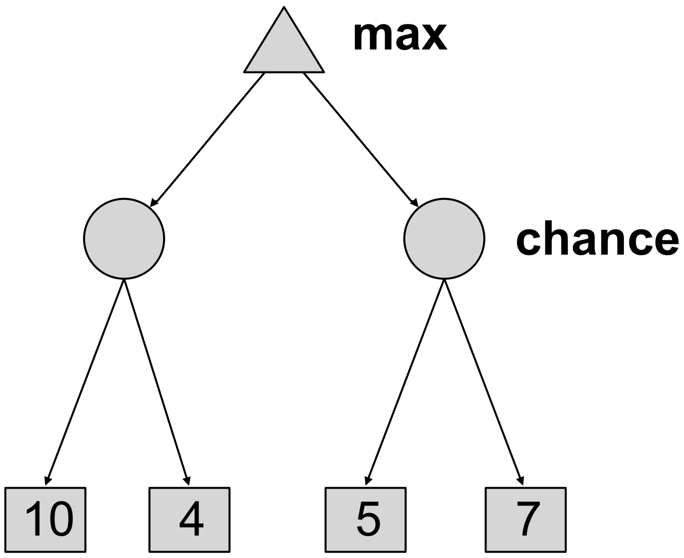
Stochastic



Review: Expectimax

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly

- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children

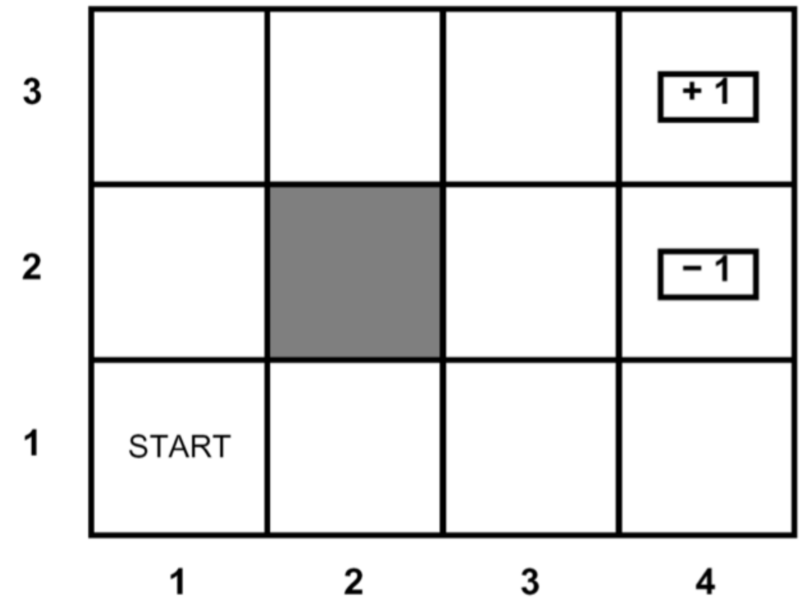


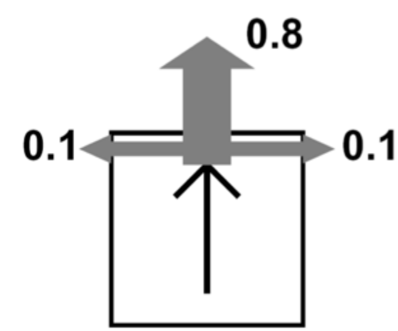
- Today, we'll learn how to formalize the underlying problem as a **Markov Decision Process**

Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ A
  - A transition function T(s,a,s')
    - Prob that a from 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 (or distribution)
  - Maybe a terminal state

- MDPs: non-deterministic search problems
  - Reinforcement learning: MDPs where we don't know the transition or reward functions





What is Markov about MDPs?

- Andrey Markov (1856-1922)
- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means:

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

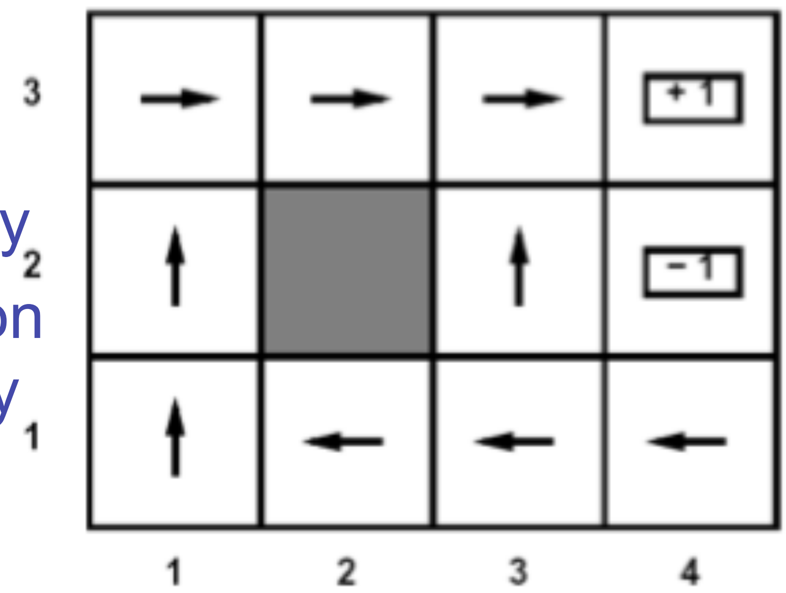
$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

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

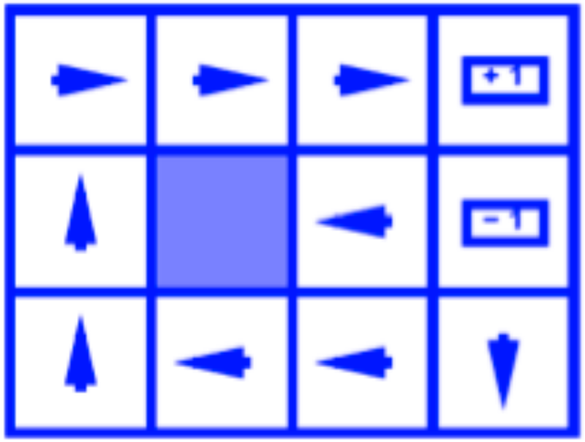
Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent
- Expectimax didn't compute the entire policy
  - It computed the action for a single state only

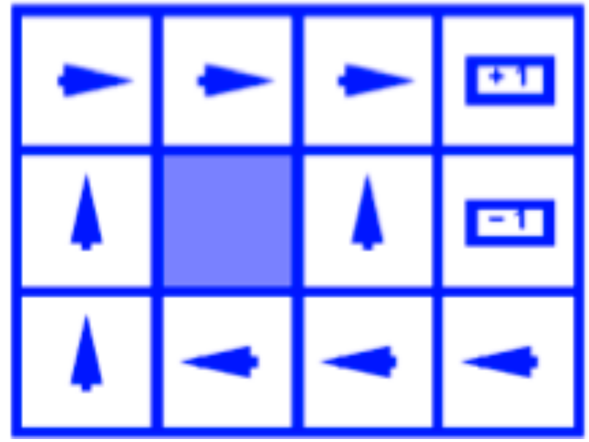


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

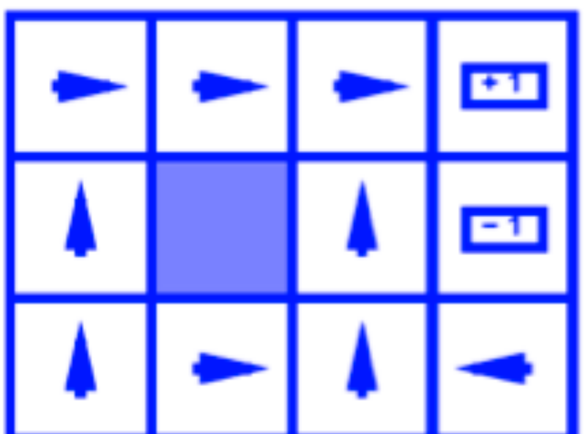
Example Optimal Policies



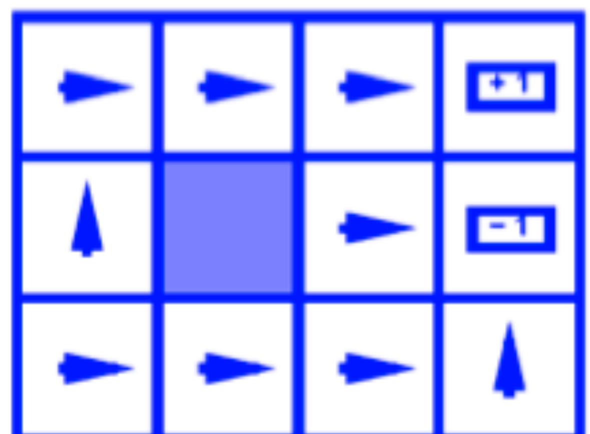
R(s) = -0.01



R(s) = -0.03



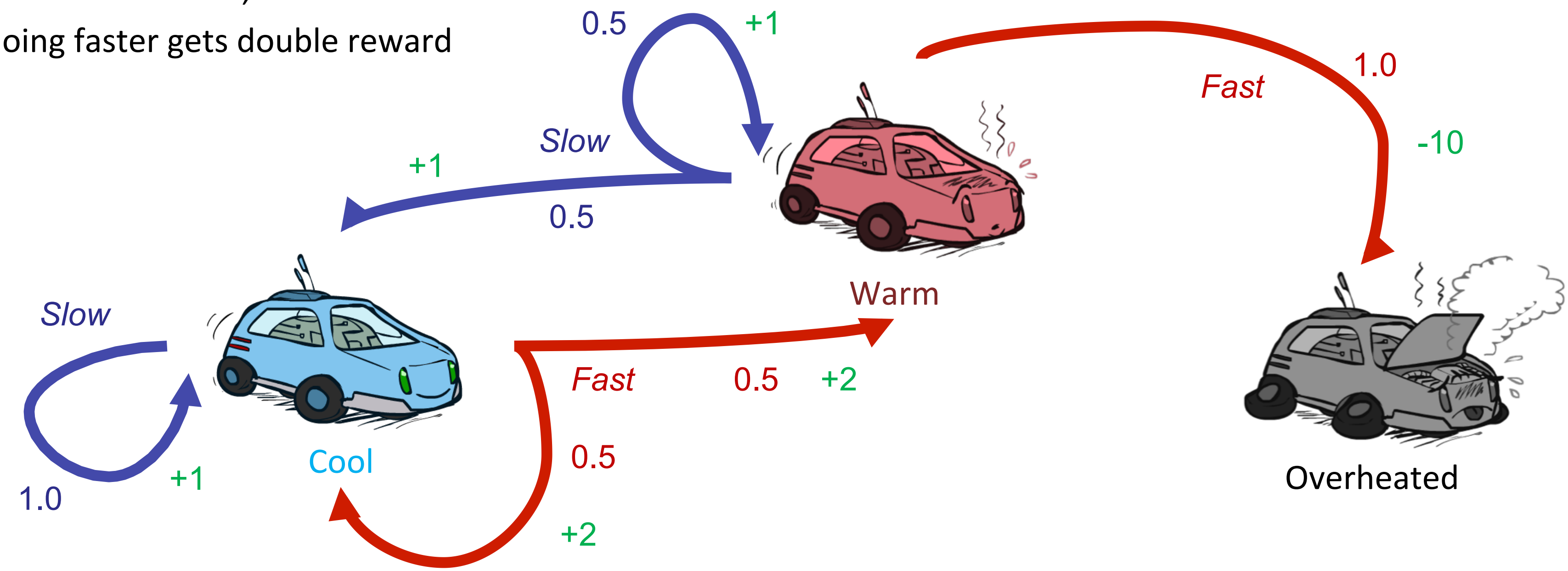
R(s) = -0.4



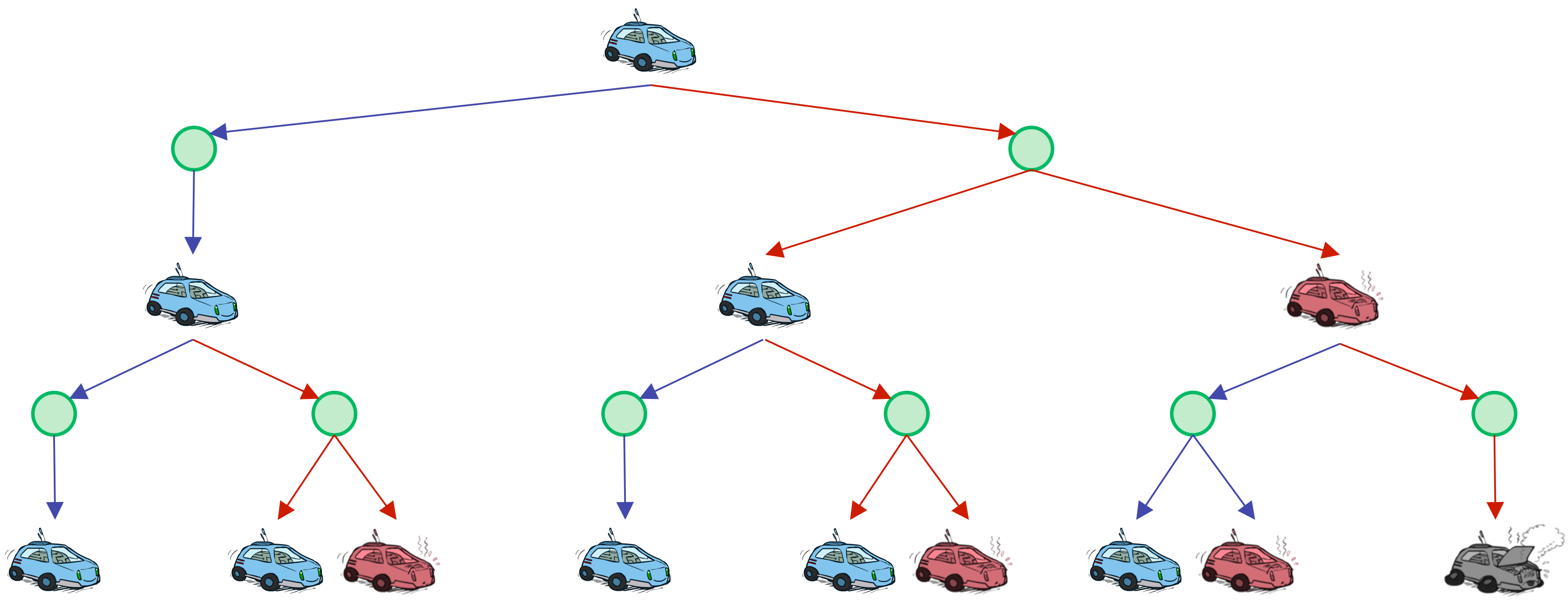
R(s) = -2.0

Another Example: Racing Car

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

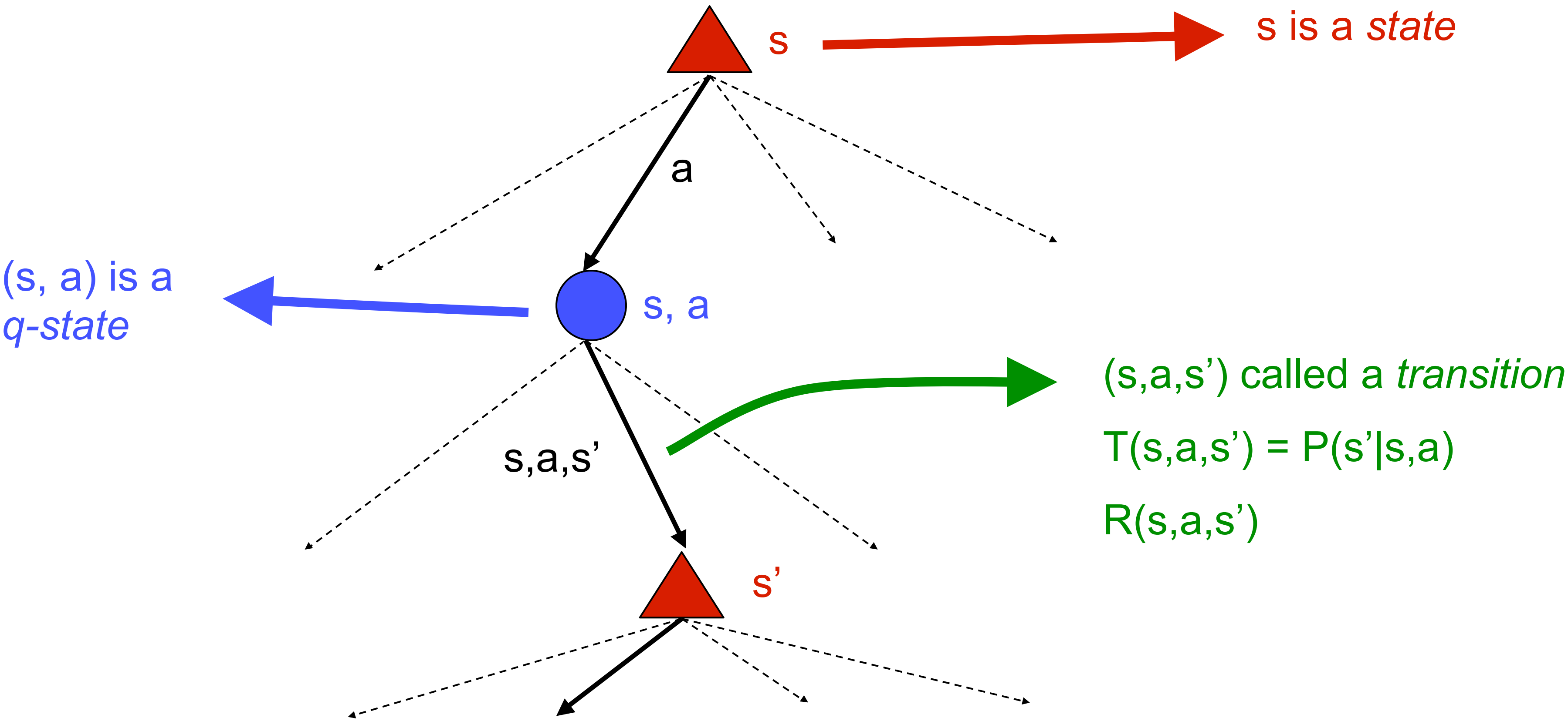


Racing Car Search Tree



MDP Search Trees

- Each MDP state gives an expectimax-like search tree

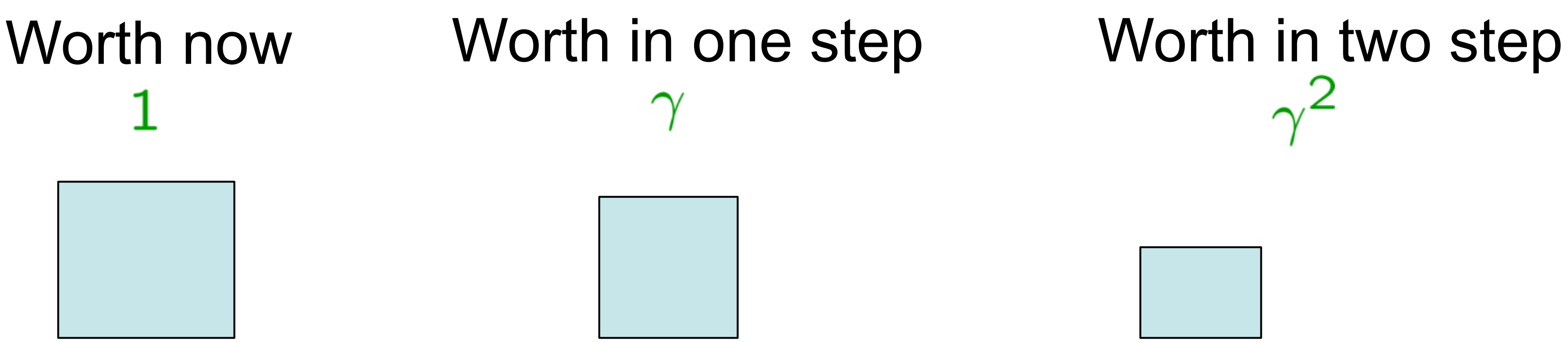


Utilities of Sequences

- What preference should an agent have over reward sequences?
- More or less:
  - [1, 2, 2]   or   [2, 3, 4]
- Now or later:
  - [0, 0, 1]   or   [1, 0, 0]

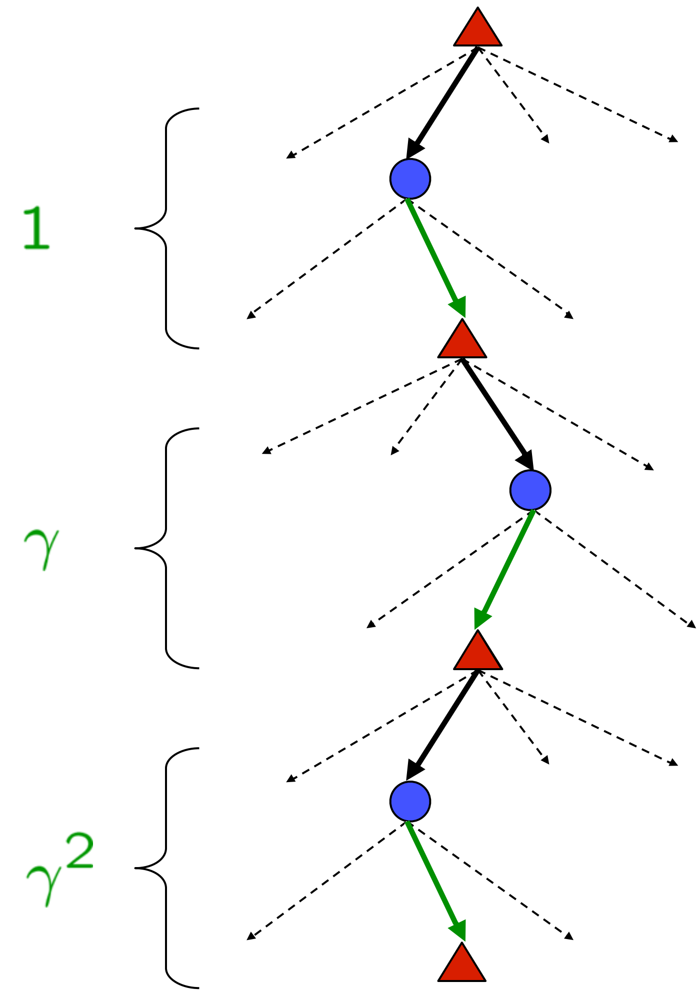
Discounting

- It is reasonable to maximize the sum of rewards
- It also makes sense to prefer rewards now to rewards later
- One solution: value of rewards decay exponentially



Discounting

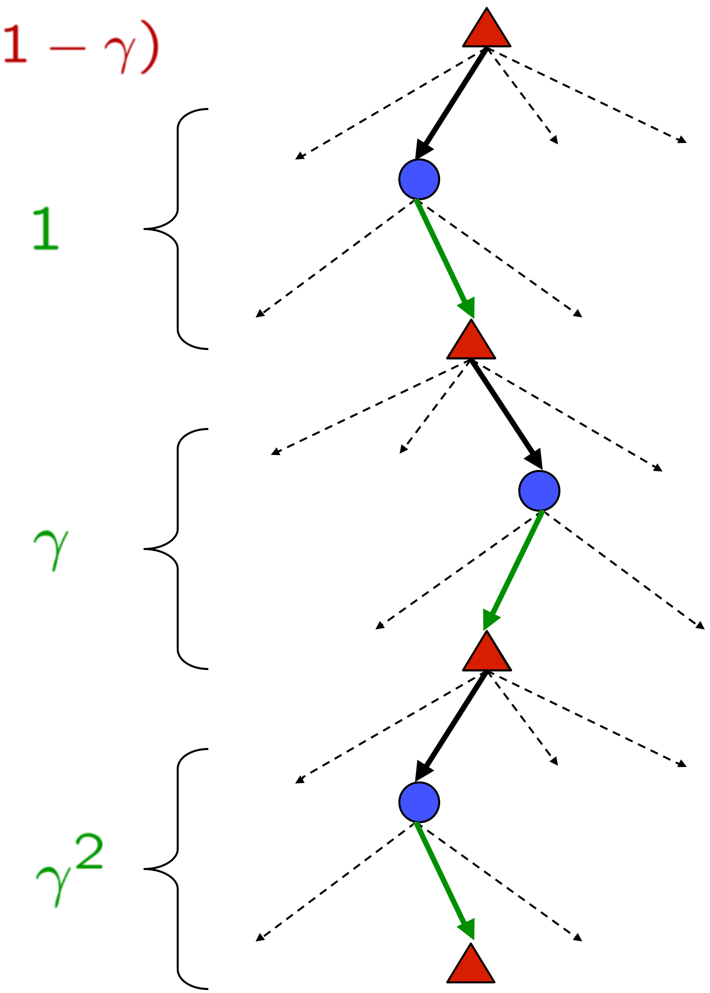
- How to discount?
  - Each time we descend, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge
- Example: discount of 0.5
  - $U([1, 2, 3]) = 1*1 + .5*2 + .25*3$
  - $U([1,2,3]) < U([3,2,1])$



Discounting

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

- Typically discount rewards by $\gamma < 1$ each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge



Quiz: Discounting

- Given:

10				1
a	b	c	d	e

  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For $\gamma = 1$, what is the optimal policy?

10				1

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

10				1

- Quiz 3: For which ° are West and East equally good when in state d?

Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider stationary preferences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r_0', r_1', r_2', \ldots]$$
$$\Leftrightarrow$$
$$[r_0, r_1, r_2, \ldots] \succ [r_0', r_1', r_2', \ldots]$$

- Only two ways to define stationary utilities
  - Additive utility:

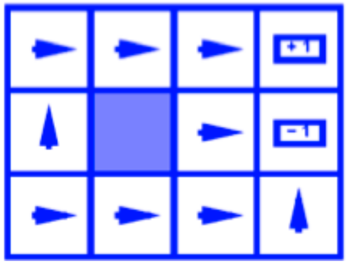
$$U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$$

  - Discounted utility:

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

Infinite Utilities?!

- Problem: what if the game lasts forever?
  - Infinite state sequences have infinite rewards



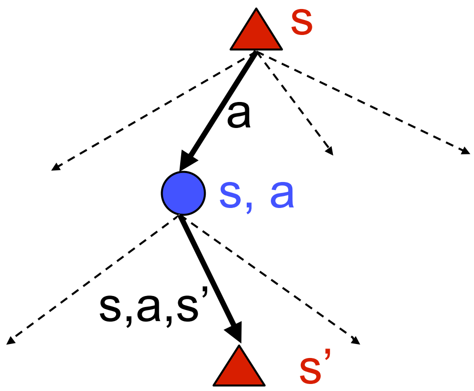
- Solutions:
  - Finite horizon:
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)
  - Discounting: for $0 < \gamma < 1$

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

    - Smaller γ means smaller "horizon" – shorter term focus

Recap: Defining MDPs

- Markov decision processes:
  - States S
  - Start state s_0
  - 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 (or return) = sum of discounted rewards

Solving MDPs

- We want to find the optimal policy π^*:
  - Find best action for each state such that it maximizes Utility (or return) = sum of discounted rewards

Optimal Utilities

- Define the value of a state s:

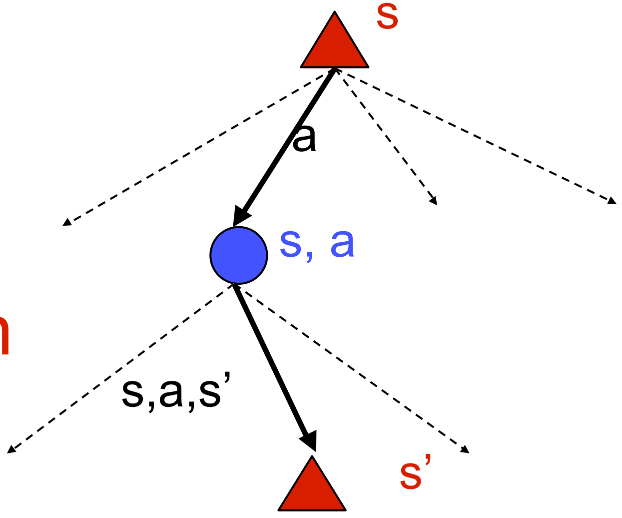
  $V^*(s)$ = expected utility starting in s and acting optimally

- Define the value of a q-state (s,a):

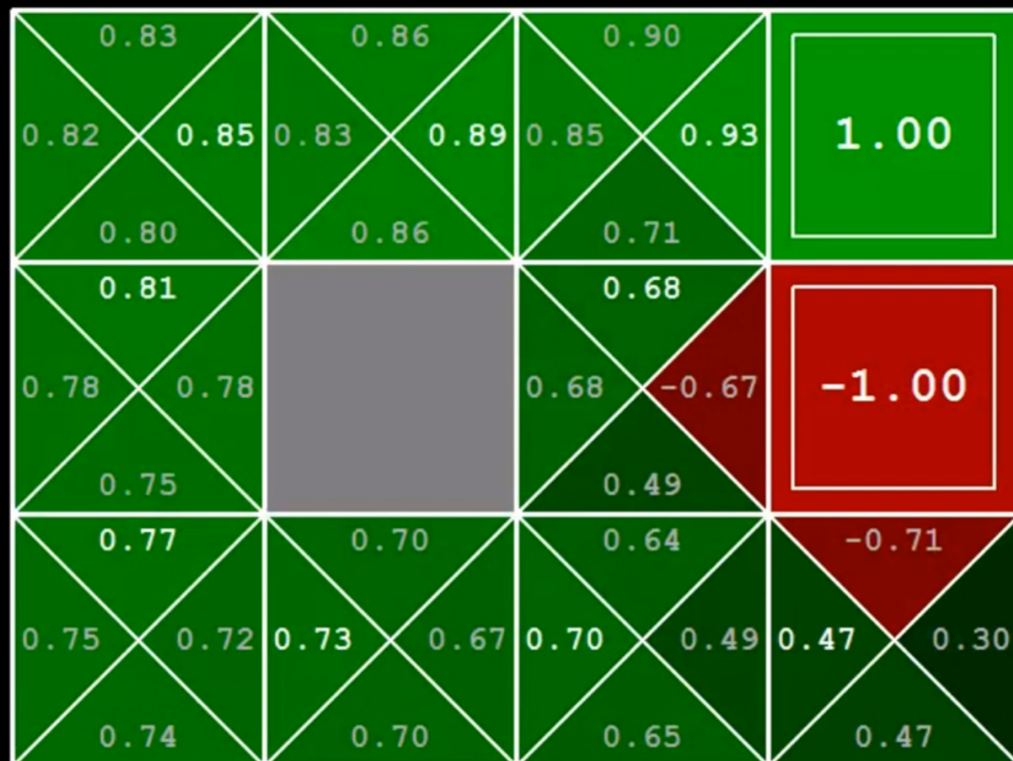
  $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally

- Define the optimal policy:

  $\pi^*(s)$ = optimal action from state s



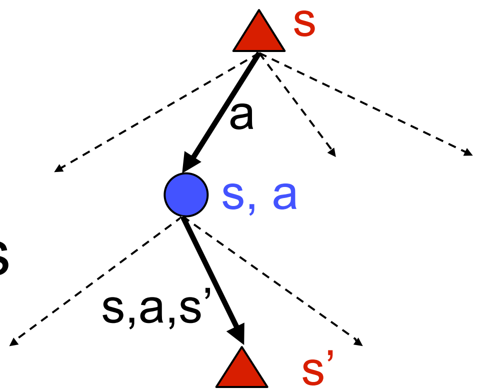

0.85
0.89
0.93
1.00
0.81
0.68
-1.00
0.77
0.73
0.70
0.47
VALUES AFTER 100 ITERATIONS



Q-VALUES AFTER 100 ITERATIONS

The Bellman Equations

- Definition of "optimal utility" leads to a simple one-step lookahead relationship amongst optimal utility values:
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax does



- Formally:

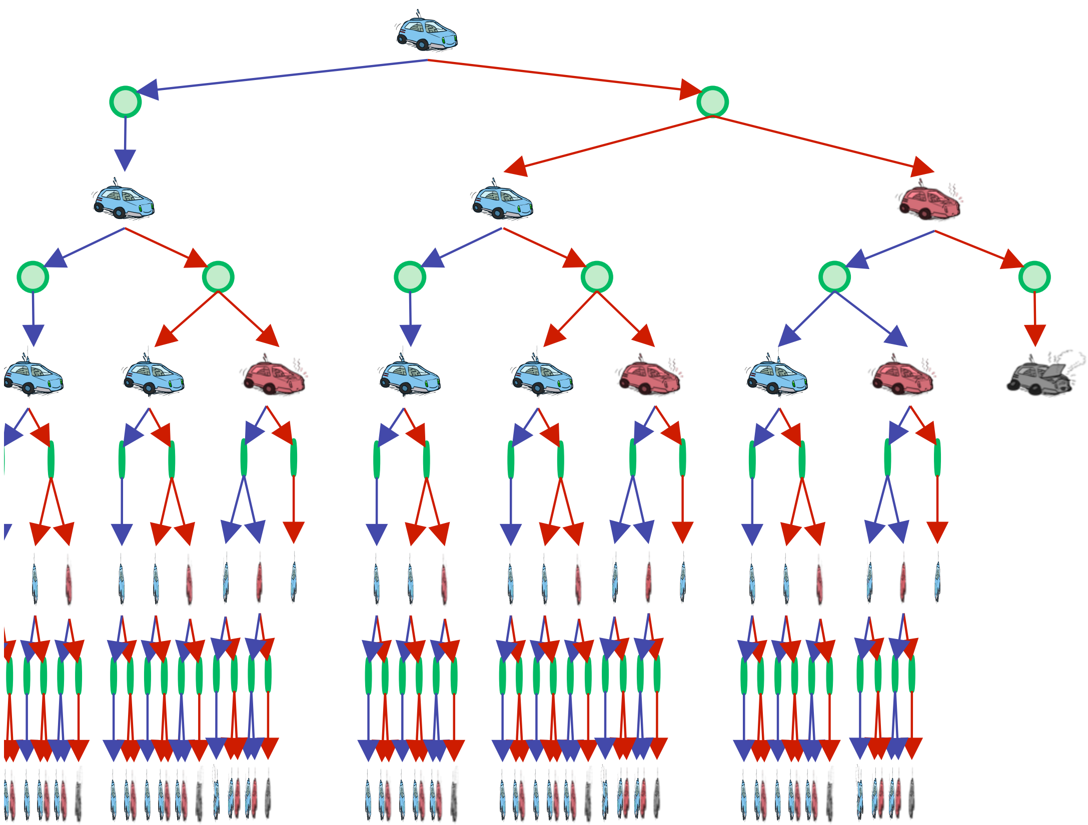
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

Racing Car Search Tree

- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if $\gamma < 1$



Time Limited Values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s