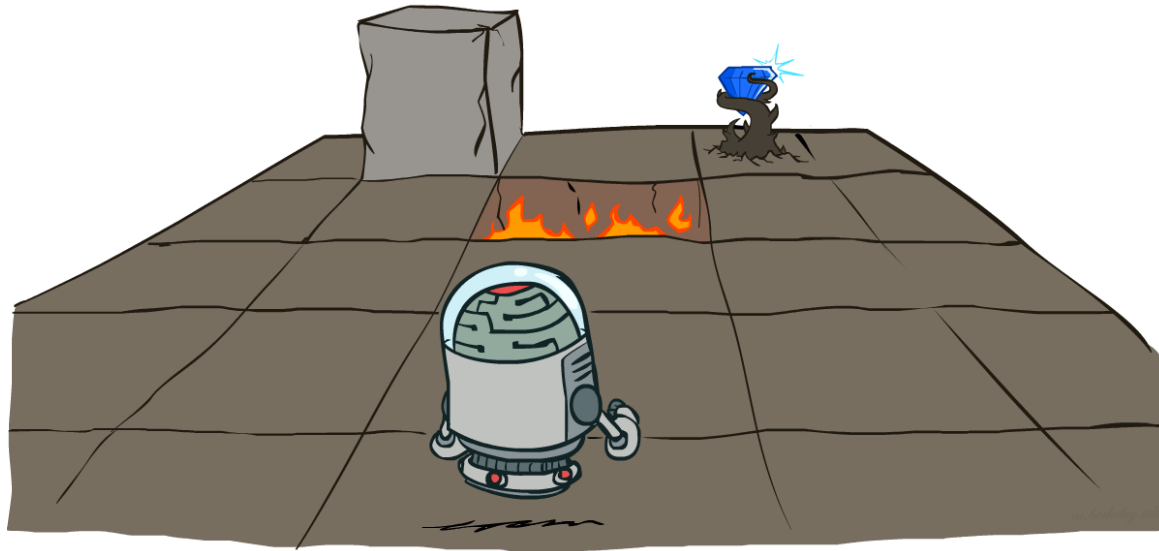


AI: Representation and Problem Solving

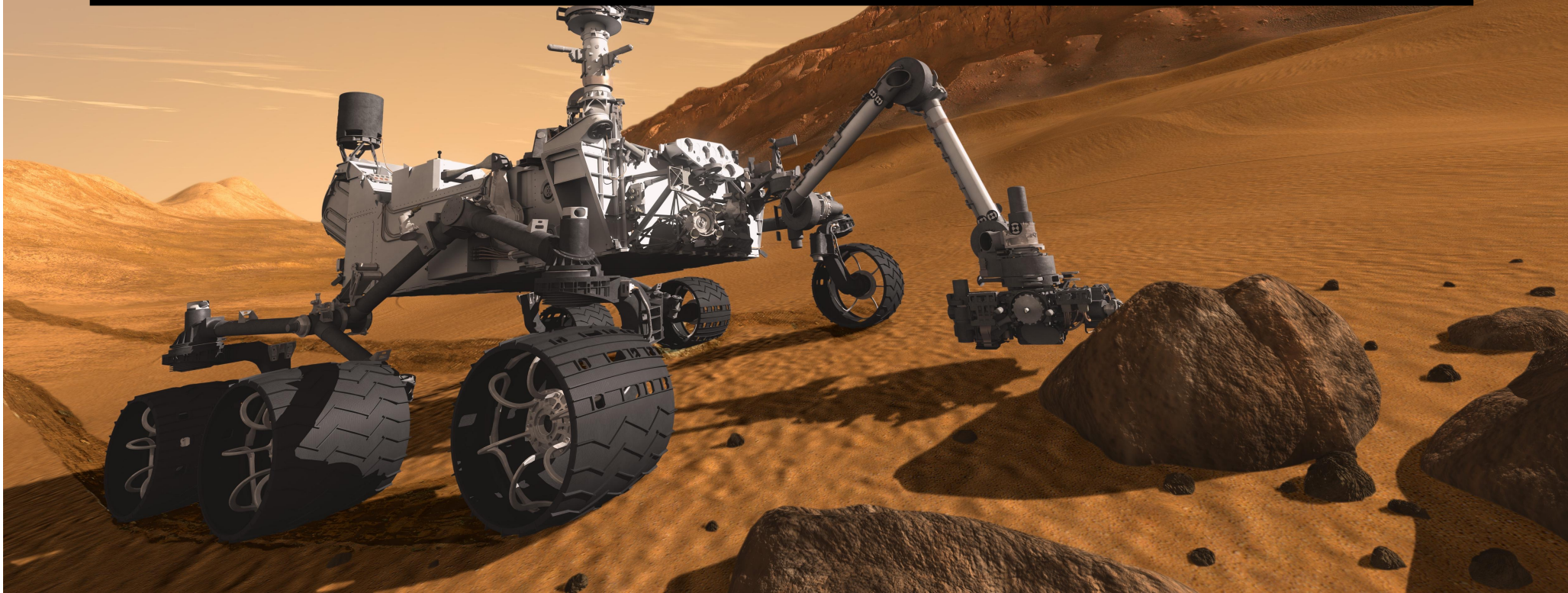
Markov Decision Processes



Instructors: Nihar Shah and Tuomas Sandholm

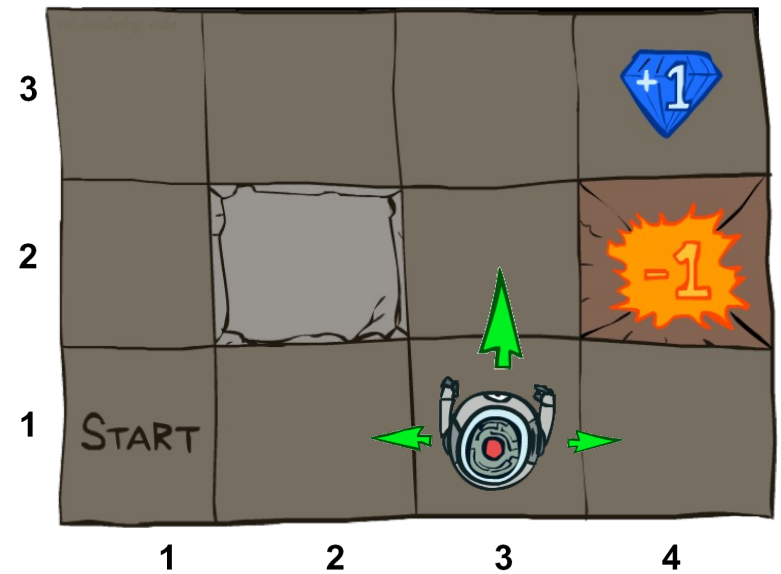
Slide credits: CMU AI and ai.berkeley.edu

The relationship between actions & consequences is not deterministic, but is stochastic (random)



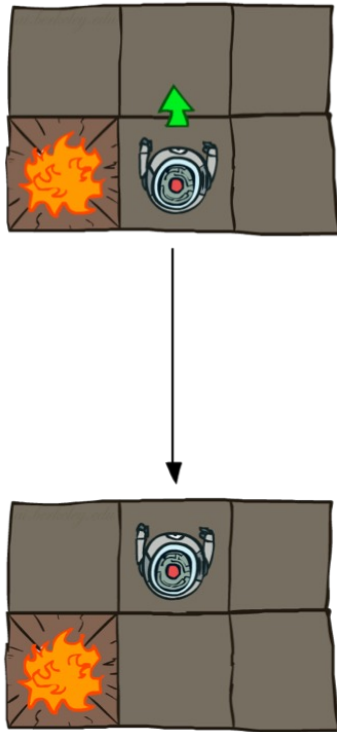
Example: Grid World

- A maze-like problem
 - The agent lives in a grid
 - Walls block the agent's path
- Noisy movement: 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
- The agent receives rewards each time step
 - Reward depends on agent's state and action
 - Can be positive, zero or negative
- Goal: maximize sum of rewards

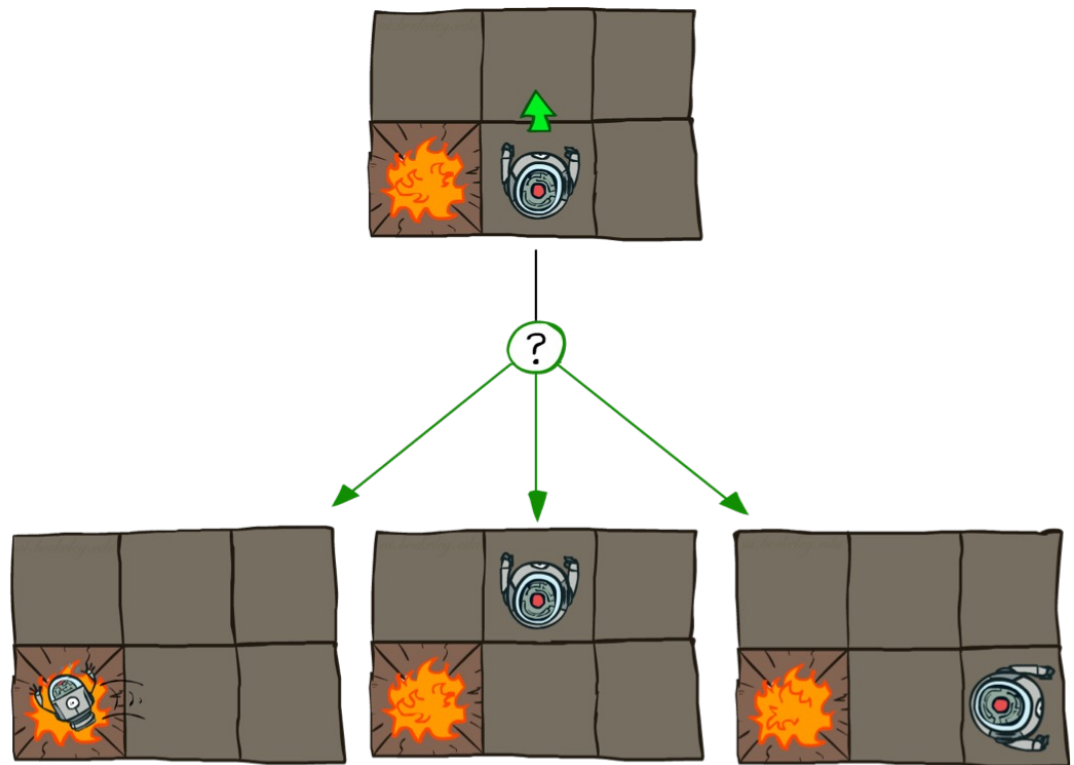


Grid World Actions

Deterministic Grid World



Stochastic Grid World

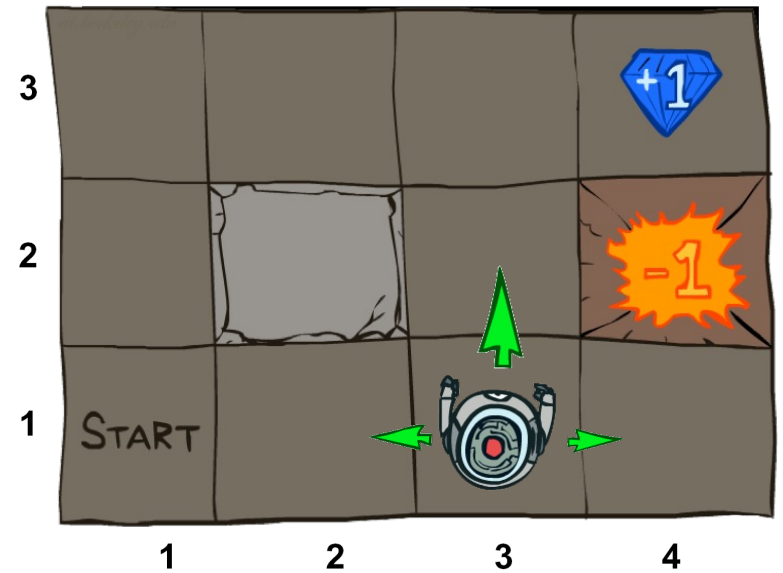


Markov Decision Processes

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 a from s leads to s' , i.e., $P(s' | s, a)$
- A reward function $R(s, a, s')$

All of this information is known beforehand



What is Markov about MDPs?

“Markov” generally means that given the present state, the future and the past are independent

For Markov decision processes, “Markov” means action outcomes depend only on the current state

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

=

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



Andrey Markov
(1856-1922)

Goal: Maximize sum of rewards

Finite time horizon setting:

- Consider MDP for T time steps
- Rewards summed over the time steps
- Want to choose actions that maximize *expected* reward

$$E[\sum_{t=1}^T \text{Reward at time } t]$$

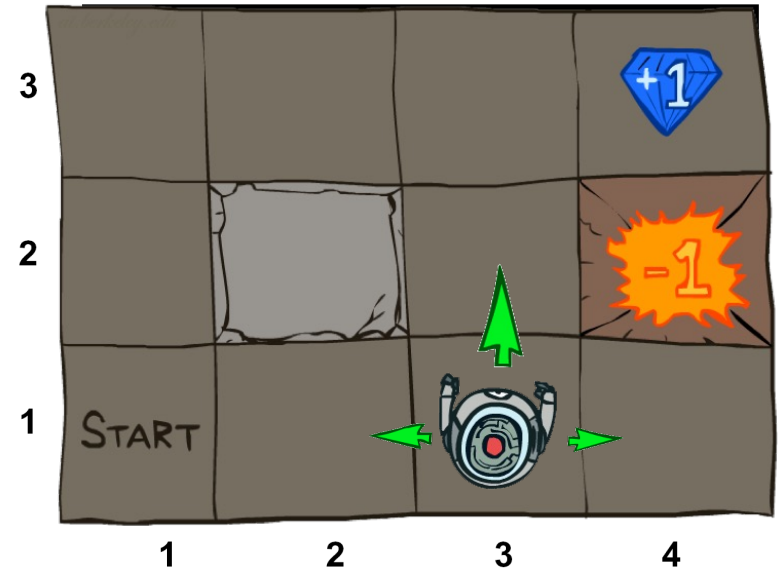
Markov Decision Processes

What actions will yield the highest expected reward?

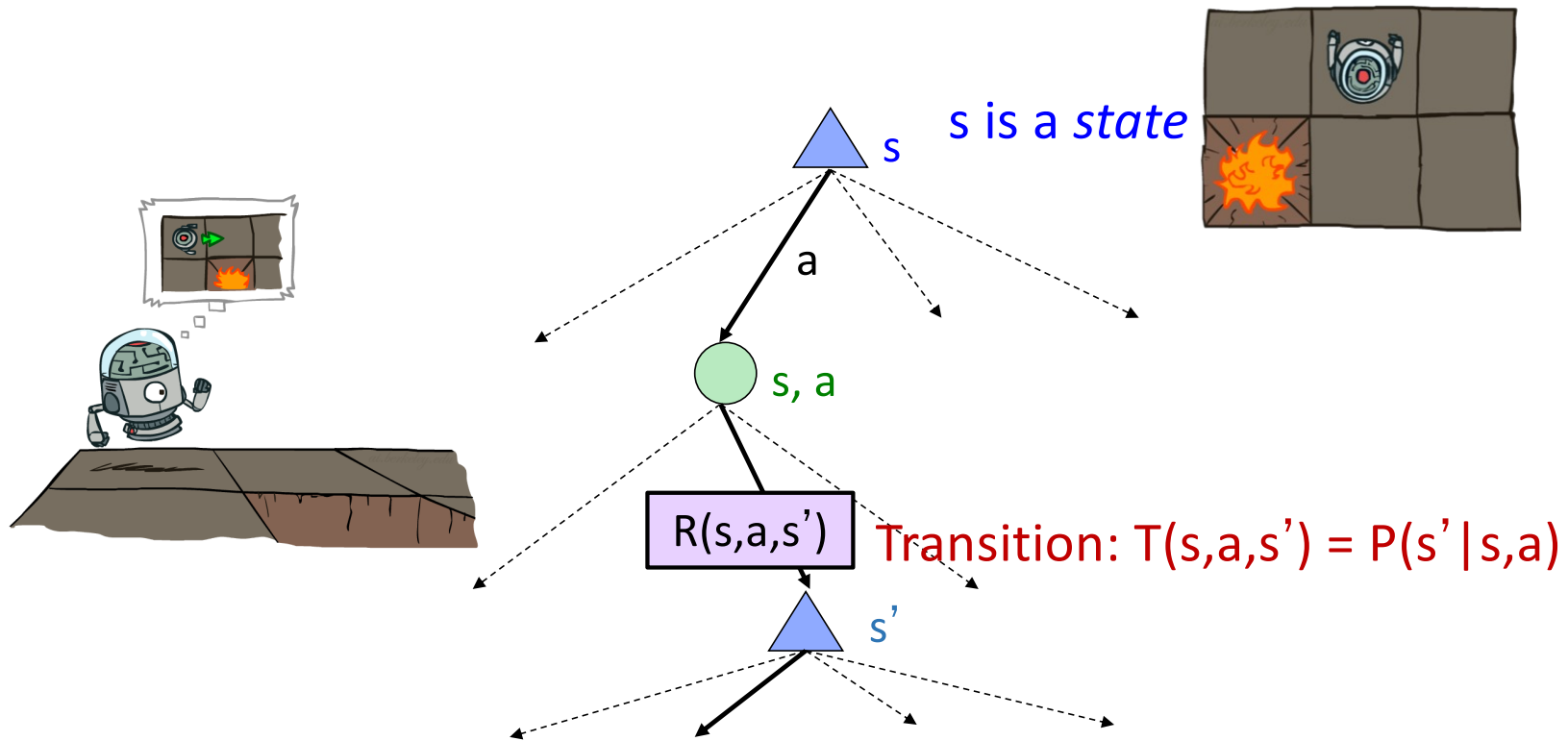
Suppose you choose the action “up”. Then suppose:

- There is an 80% chance you go up, and get a reward of +2
- There is a 10% chance you go left, and get a reward of -1
- There is a 10% chance you go right, and get a reward of -1
- The expected reward in this step under this action is $0.8*(+2) + 0.1*(-1) + 0.1*(-1) = 1.4$

Want to choose actions that maximize **total reward** across many time steps (not just in the current time step)



MDP Search Trees

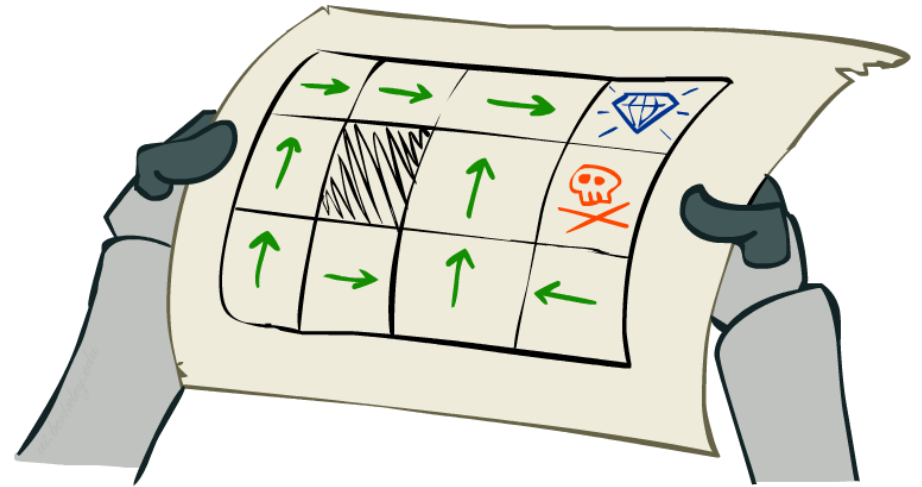


Policies

In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

For MDPs, we want an optimal **policy**

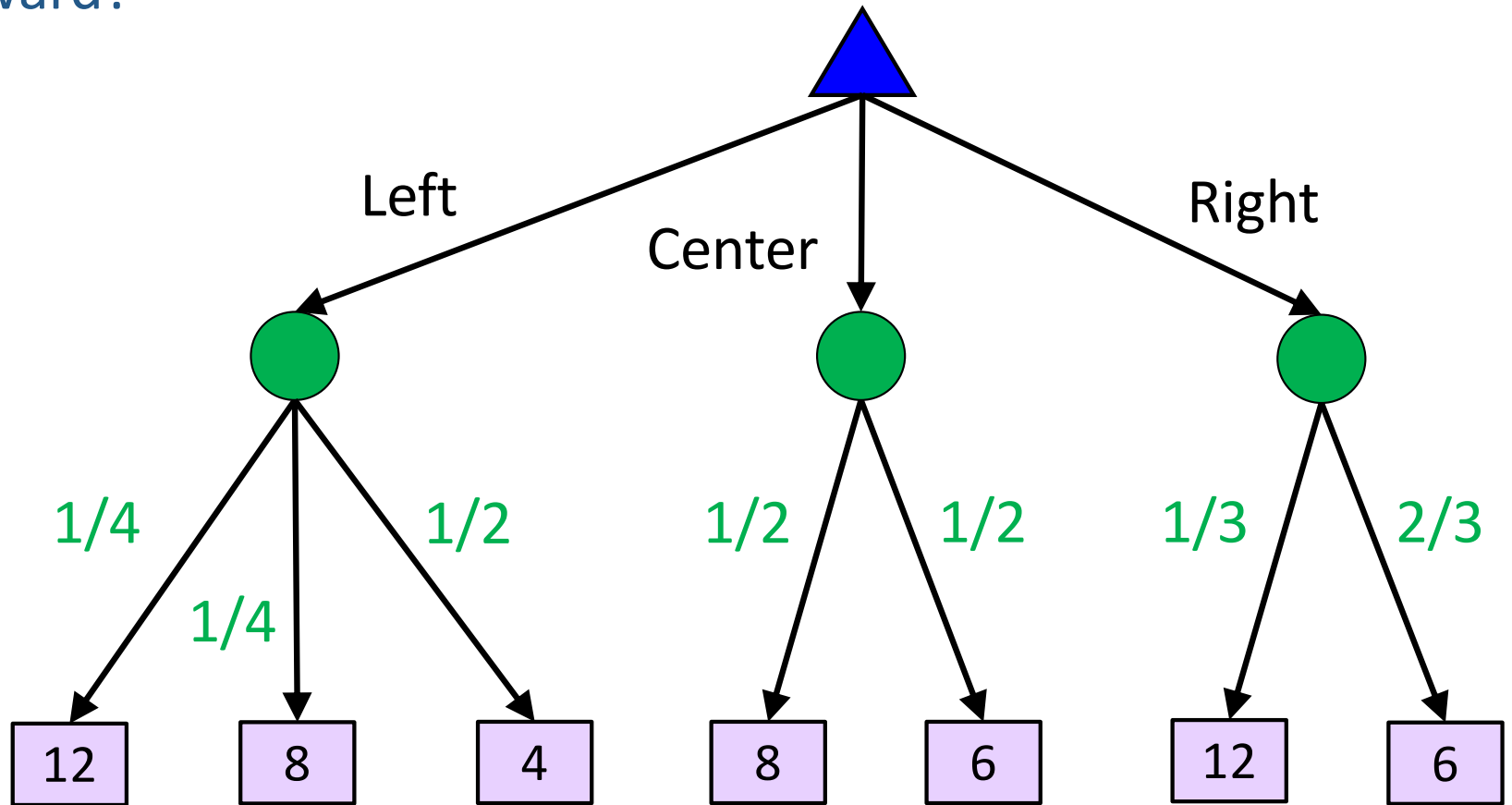
- A **policy** π gives an action for each state
- An **optimal** policy is one that maximizes expected utility if followed



Solution method 1: Expectimax search

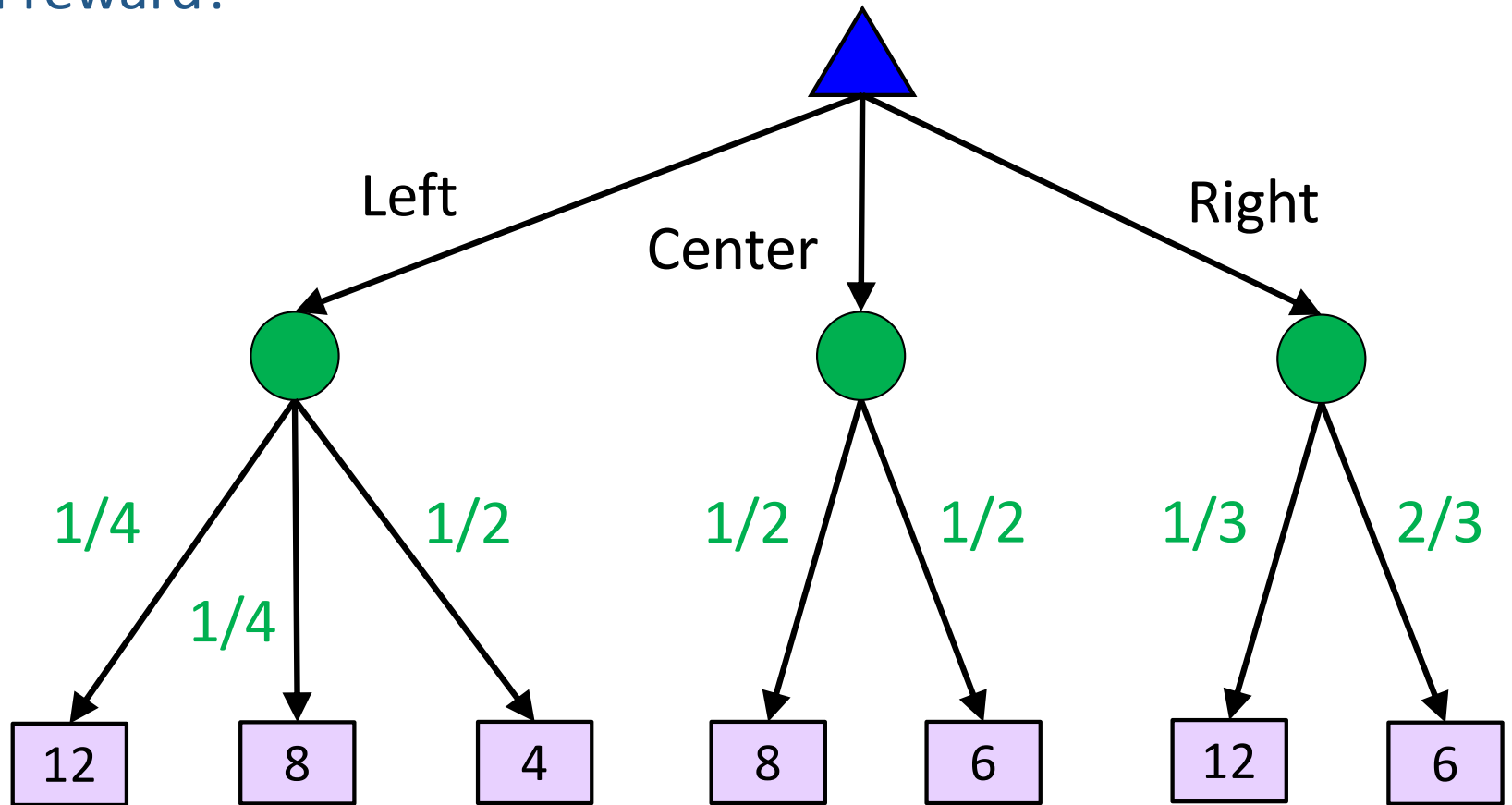
Suppose the MDP has only 1 state and will run only for 1 time step. If you were to take Left, what is the expected reward?

- A) 12
- B) 8
- C) 7
- D) 4



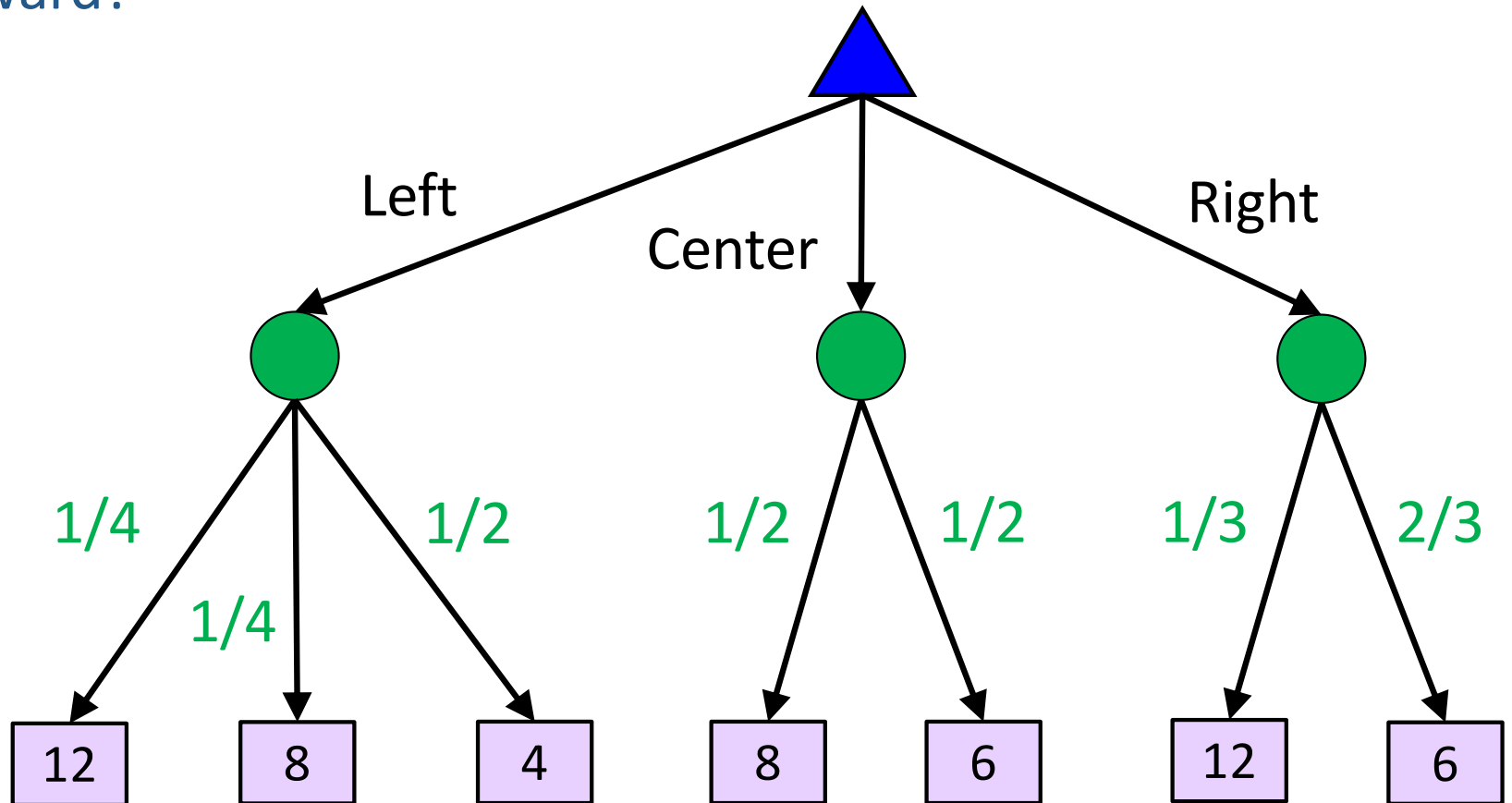
Suppose the MDP has only 1 state and will run only for 1 time step. If you were to take Center, what is the expected reward?

- A) 12
- B) 8
- C) 7
- D) 4



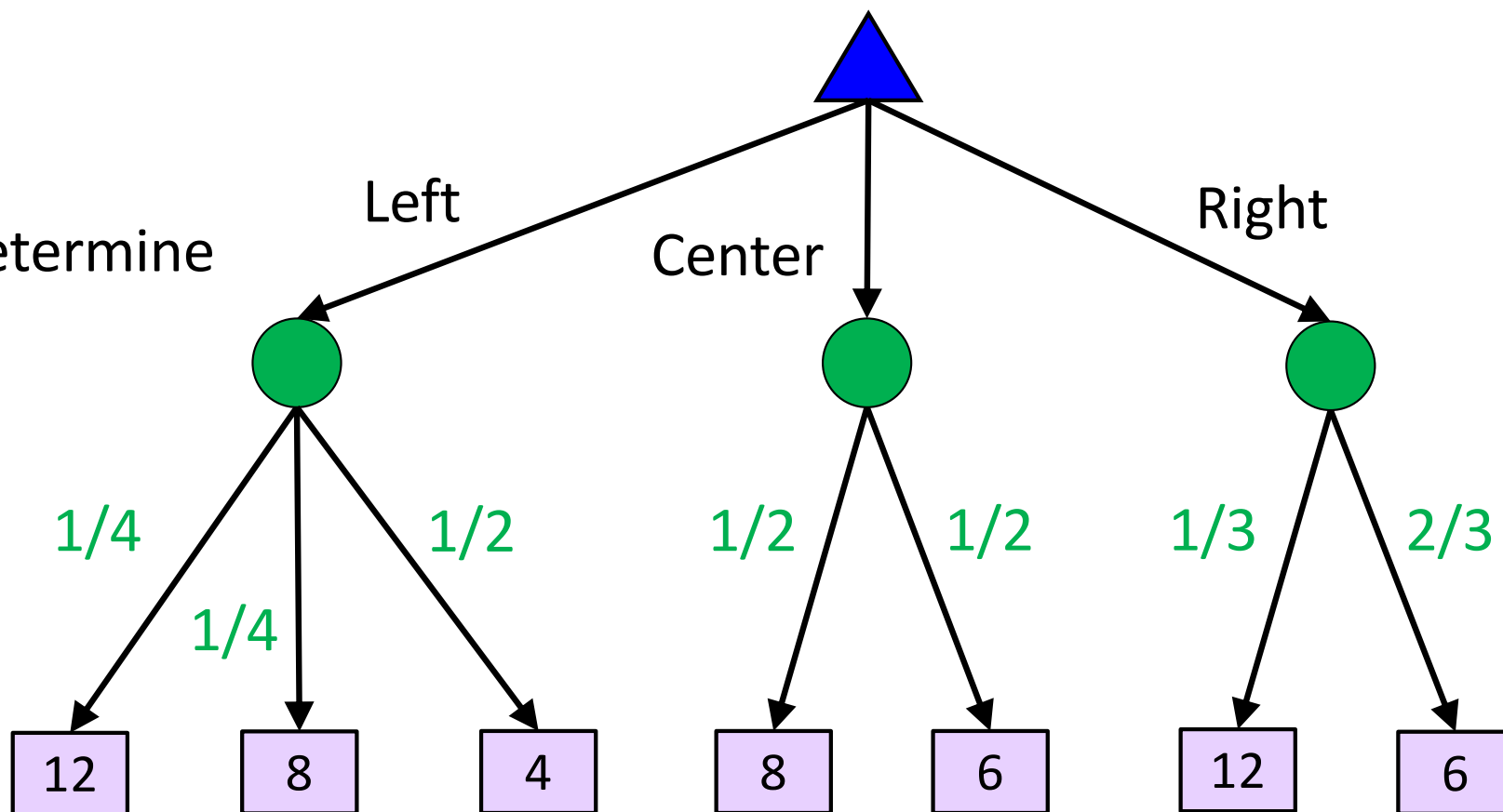
Suppose the MDP has only 1 state and will run only for 1 time step. If you were to take Right, what is the expected reward?

- A) 12
- B) 8
- C) 7
- D) 4



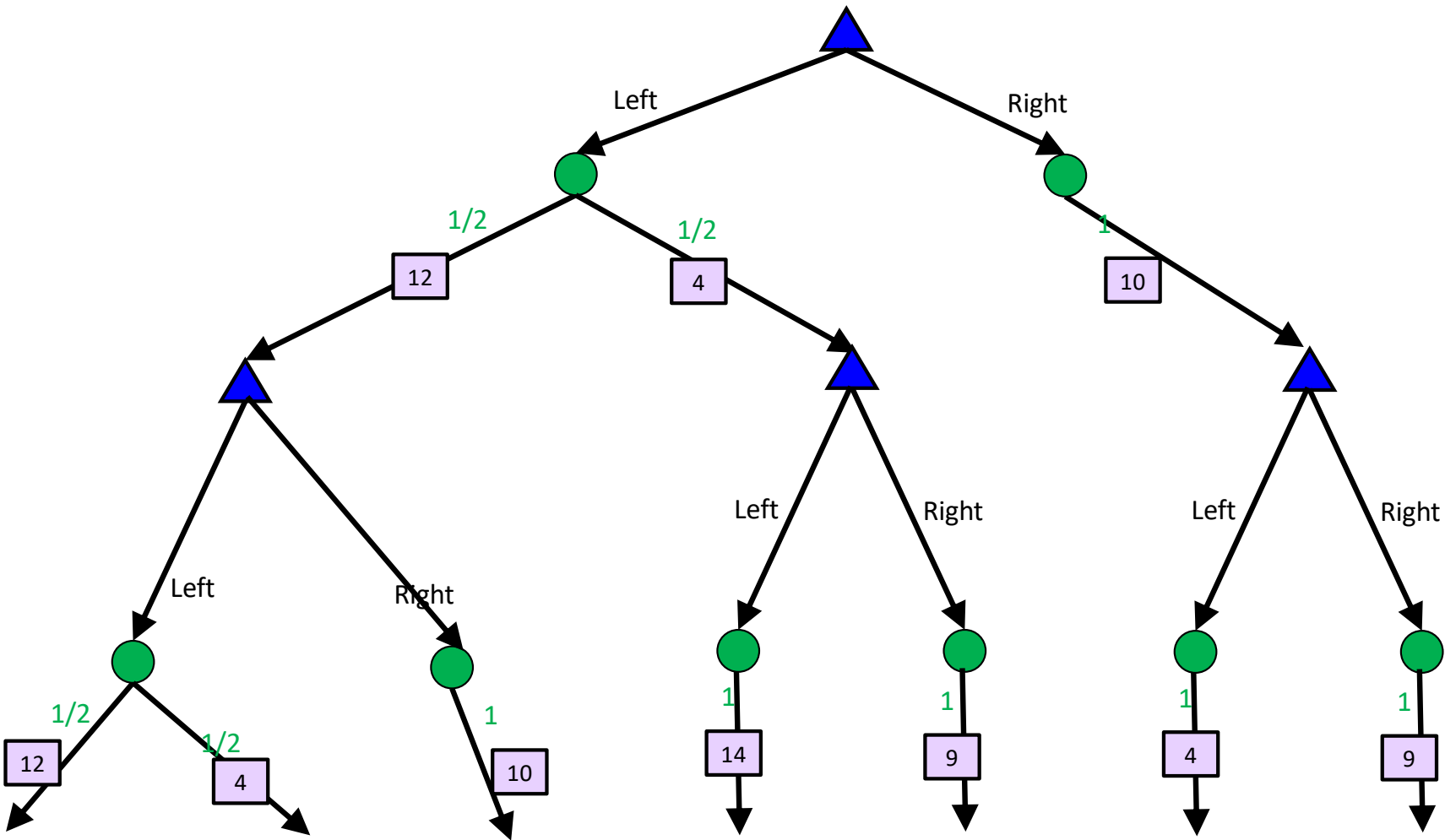
Suppose the MDP has only 1 state and will run only for 1 time step. Which action should we choose?

- A) Left
- B) Center
- C) Right
- D) Cannot determine





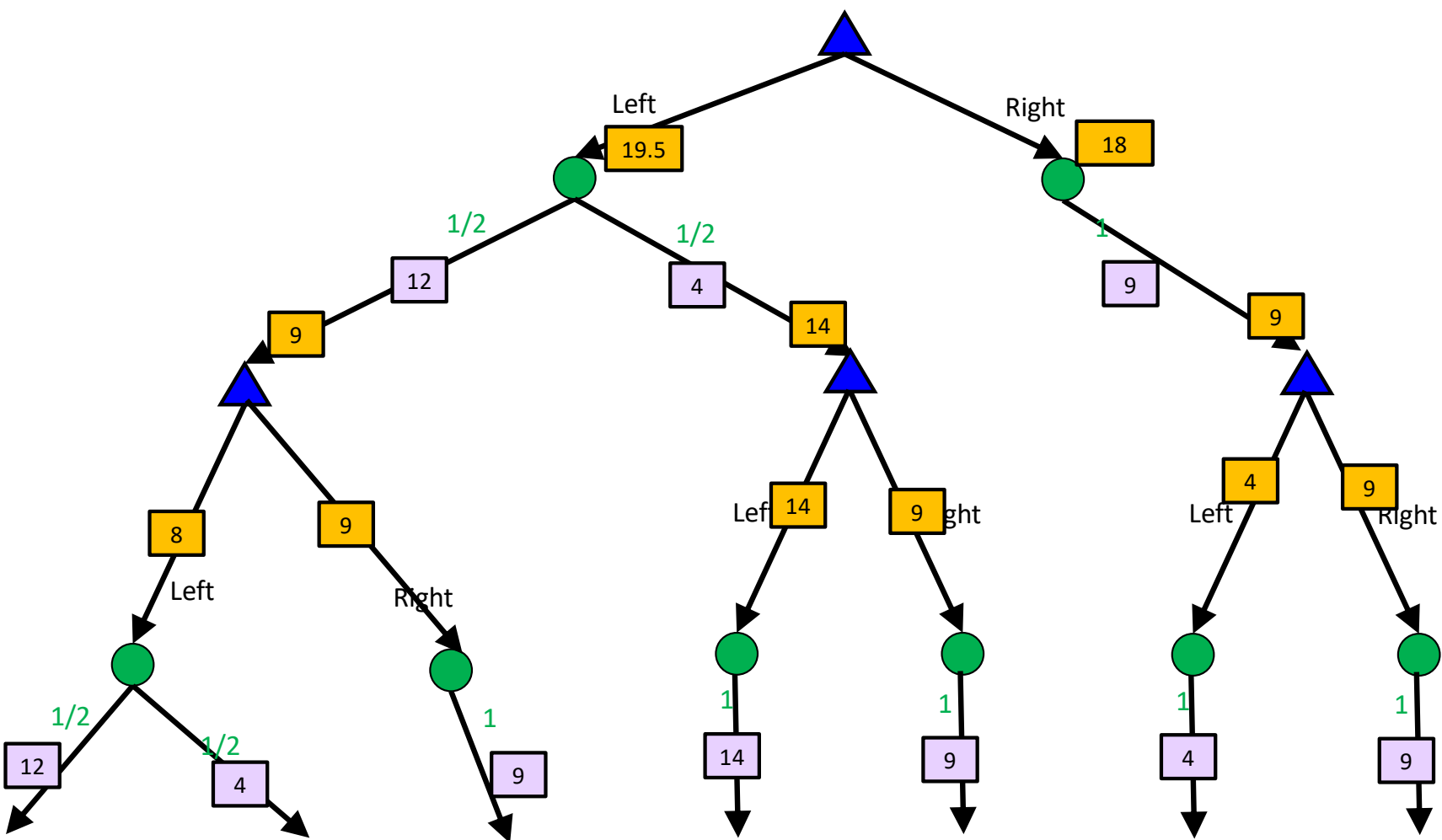
Solution method 1: Expectimax search

- Starting from the start state, expand the tree to depth k , for some chosen k
 - Previous example was for depth 1
 - Here is an example for depth 2...



Solution method 1: Expectimax search

- Starting from the start state, expand the tree to depth k , for some chosen k
- Move up the tree, recursively computing the reward obtained
 - Compute expectations at  nodes and max at  nodes



function EXPECTIMAX(node, depth):

if depth = 0:

return 0

if node is a MAX node  (denoted as s):

value = $-\infty$

for each possible action a:

value = max(value, EXPECTIMAX((s,a) , depth - 1))

return value

else if node is an EXPECTATION node  (denoted as (s,a)):

value = 0

for each child of node s', with probability given by the transition T(s,a,s'):

value += T(s,a,s') * (R(s,a,s') + EXPECTIMAX(s', depth))

return value

Goal: Maximize sum of rewards

Finite time horizon setting:

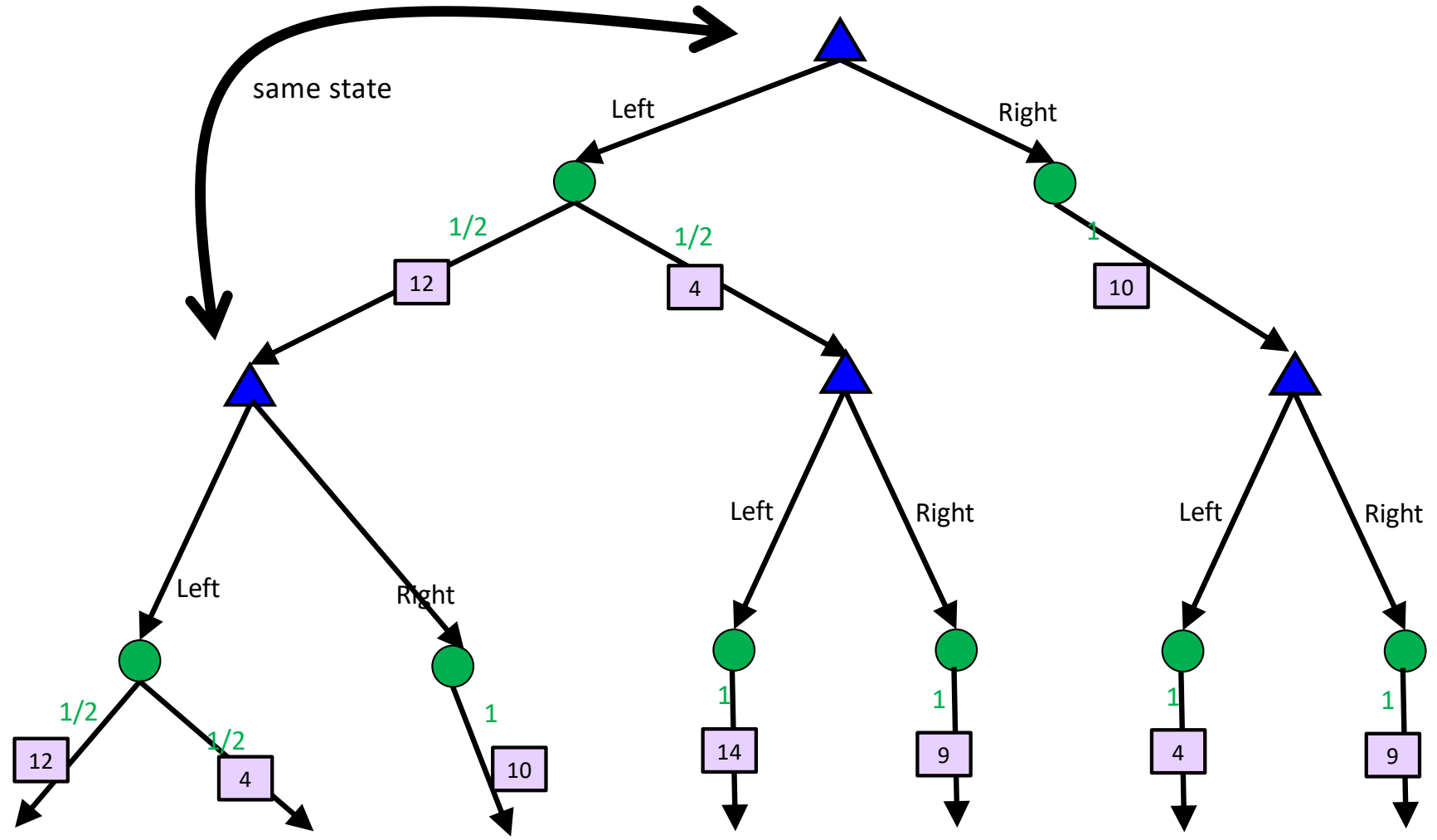
- Consider MDP for T time steps
- Rewards summed over the time steps
- Want to choose actions that maximize *expected* reward

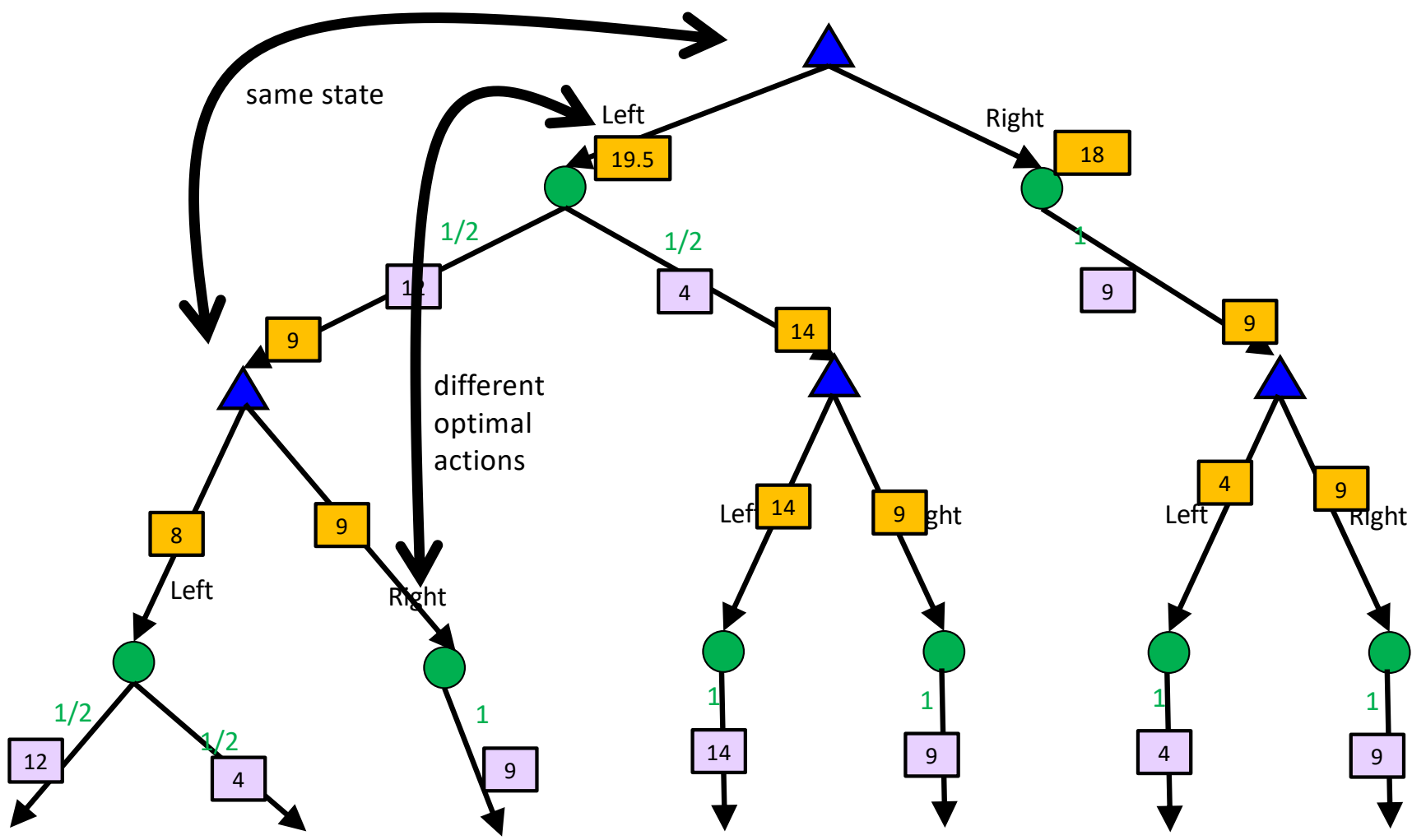
$$E[\sum_{t=1}^T \text{Reward at time } t]$$



Will the optimal choice of actions depend on time?

If you are in a state s at time 1 versus in the state s at time 10, can the optimal action be different?





Goal: Maximize sum of rewards

Finite time horizon setting:

- Consider MDP for T time steps
- Rewards summed over the time steps
- Want to choose actions that maximize *expected* reward

$$E[\sum_{t=1}^T \text{Reward at time } t]$$



Will the optimal choice of actions depend on time?

If you are in a state s at time 1 versus in the state s at time 10, can the optimal action be different?

YES

Goal: Maximize sum of rewards

Finite time horizon setting:

- Consider MDP for T time steps
- Rewards summed over the time steps
- Want to choose actions that maximize *expected* reward

$$E[\sum_{t=1}^T \text{Reward at time } t]$$

Challenges

- Optimal action depends on time step
- Need to compute it separately for each time step...
 - Computational cost + headache
- May not know horizon T in advance

Goal: Maximize sum of rewards

Infinite time horizon setting:

- Want to choose actions that maximize *expected* reward

$$E[\sum_{t=1}^{\infty} \text{Reward at time } t]$$

Goal: Maximize sum of rewards

Infinite time horizon setting:

- Want to choose actions that maximize *expected* reward

$$E[\sum_{t=1}^{\infty} \text{Reward at time } t]$$

Challenge

This can be infinite for many choices of actions...
Infinity vs. infinity?

Discounting

It's reasonable to maximize the sum of rewards

It's also reasonable to prefer rewards now to rewards later

One solution: values of rewards decay exponentially



1

Worth Now



γ

Worth Next Step



γ^2

Worth In Two Steps

Goal: Maximize sum of rewards

Infinite time horizon setting with discounted rewards:

Want to choose actions that maximize *expected* reward

$$E[\sum_{t=1}^{\infty} \gamma^t \text{ Reward at time } t]$$

for some γ in $(0,1)$

If you are in some state s at some time t_1
versus
if you are in that state s at some time t_2

Does the optimal action need to differ?

Goal: Maximize sum of rewards

$$E \left[\sum_{t=1}^{\infty} \gamma^t \text{Reward at time } t \right]$$

Does the optimal action need to differ with time?

1. Markov property means that the MDP evolution does not change with time
2. What about optimal actions? Let's look at future reward starting at t_1 or t_2

Finite horizon: Future reward = $E \left[\sum_{t=t_1}^T \text{Reward at time } t \right]$ vs. $E \left[\sum_{t=t_2}^T \text{Reward at time } t \right]$

Infinite horizon with discounted reward:

$E \left[\sum_{t=t_1}^{\infty} \gamma^t \text{Reward at time } t \right] = E \left[\sum_{t=1}^{\infty} \gamma^t \text{Reward at time } t \right]$ (via change of variables)

vs. $E \left[\sum_{t=t_2}^{\infty} \gamma^t \text{Reward at time } t \right] = E \left[\sum_{t=1}^{\infty} \gamma^t \text{Reward at time } t \right]$

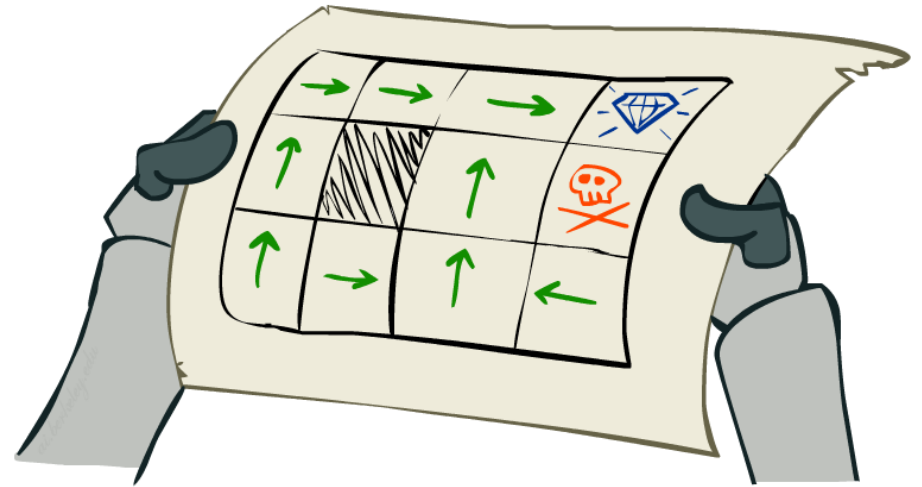
Time invariant!!

Policies

In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

For MDPs, we want an optimal policy

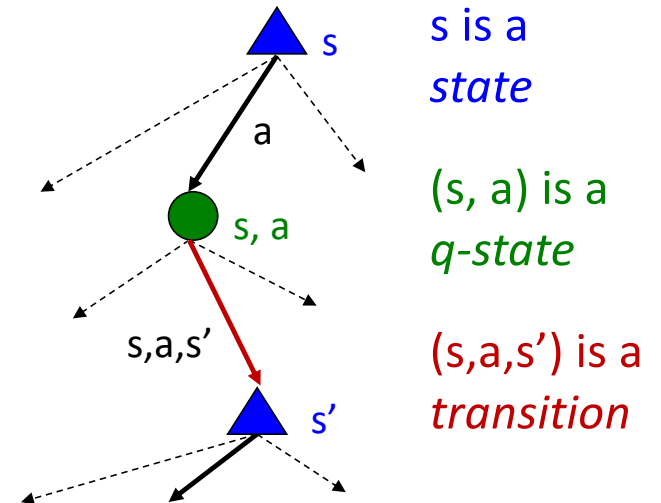
- A policy π gives an action for each state
- An optimal policy is one that maximizes expected utility if followed
- **Action depends only on the state you are in, and not on when**
- Policy $\pi: S \rightarrow A$



Solution method 2: Value iteration

Optimal Quantities

- The value (utility) of a state s :
 $V^*(s)$ = expected total reward starting in s and acting optimally
- The value (utility) of a q-state (s,a) :
 $Q^*(s,a)$ = expected total reward starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
 $\pi^*(s)$ = optimal action from state s



Optimal Quantities

Bellman equations

- The value (utility) of a state s :

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

- The value (utility) of a q-state (s,a) :

$Q^*(s,a)$ = expected total reward starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

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

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

for every state s

Optimal Quantities

- The value (utility) of a state s :
 $V^*(s)$ = expected total reward starting in s and acting optimally
- The value (utility) of a q-state (s,a) :
 $Q^*(s,a)$ = expected total reward starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
 $\pi^*(s)$ = optimal action from state s

Question: If you have Q^* , how to get V^* ?

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

Optimal Quantities

- The value (utility) of a state s :
 $V^*(s)$ = expected total reward starting in s and acting optimally
- The value (utility) of a q-state (s,a) :
 $Q^*(s,a)$ = expected total reward starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
 $\pi^*(s)$ = optimal action from state s

Question: If you have Q^* , how to get π^* ?

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a)$$

Optimal Quantities

- The value (utility) of a state s :
 $V^*(s)$ = expected total reward starting in s and acting optimally
- The value (utility) of a q-state (s,a) :
 $Q^*(s,a)$ = expected total reward starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
 $\pi^*(s)$ = optimal action from state s

Question: If you have V^* , how to get Q^* ?

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

Optimal Quantities

- The value (utility) of a state s :

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

- The value (utility) of a q-state (s,a) :

$Q^*(s,a)$ = expected total reward starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

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

Question: If you have V^* , how to get π^* ?

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

“Policy extraction”

Solution method 2: Value iteration

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

If we can estimate V^* , we can find the optimal policy

Key idea:

- Initialize V^* (either a guess or set it to 0)
- Keep updating the guess using the equation above

Solution method 2: Value iteration

Start with initial guess of the value function V , e.g., $V(s) = 0$ for all s

Given vector of $V(s)$ values, do one iteration of expectimax for each state:

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

Repeat

Solution method 2: Value iteration

Example on the board

Solution method 2: Value iteration

Start with initial guess of the value function V , e.g., $V(s) = 0$ for all s

Given vector of $V(s)$ values, do one iteration of expectimax for each state:

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

Repeat

Complexity of each iteration:

$$O(S^2A)$$

Theorem: will converge to unique optimal values

Convergence

How do we know the V_k vectors are going to converge?

Proof sketch: Suppose rewards are bounded in $[R_{\text{MIN}}, R_{\text{MAX}}]$

- For any state V_k and V_{k+1} can be viewed as depth $k+1$ expectimax results in nearly identical search trees
- The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
- That last layer is at best all R_{MAX}
- It is at worst R_{MIN}
- But everything is discounted by γ^k that far out
- So V_k and V_{k+1} are at most $\gamma^k \max |R|$ different
- So as k increases, the values converge

