Warm-up as You Walk In

Given

- Set `actions` (persistent/static)
- Set `states` (persistent/static)
- Function `T(s,a,s_prime)`

Write the pseudo code for:

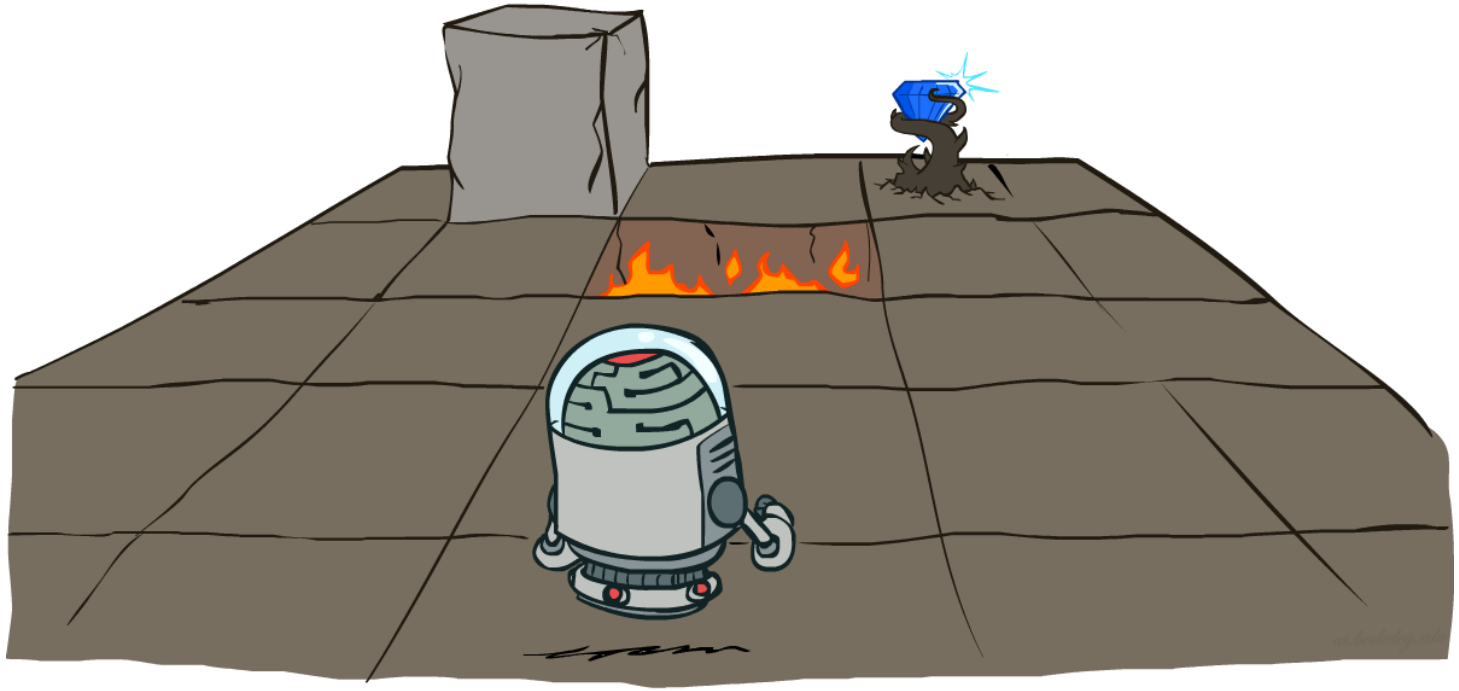
- `function V(s) return value`

that implements:

$$V(s) = \max_{a \in actions} \sum_{s' \in states} T(s, a, s')V(s')$$

AI: Representation and Problem Solving

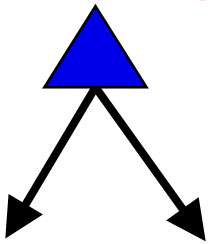
Markov Decision Processes



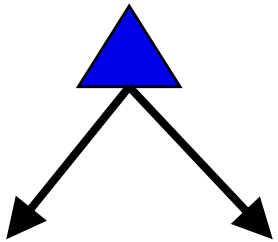
Instructor: Pat Virtue

Slide credits: CMU AI and http://ai.berkeley.edu

Minimax Notation



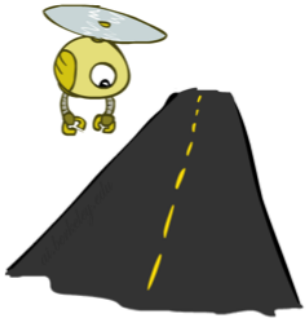
$$V(s) = \max_{a} V(s'),$$
$$\text{where } s' = result(s, a)$$

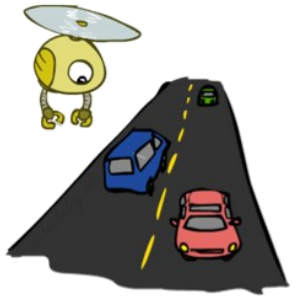


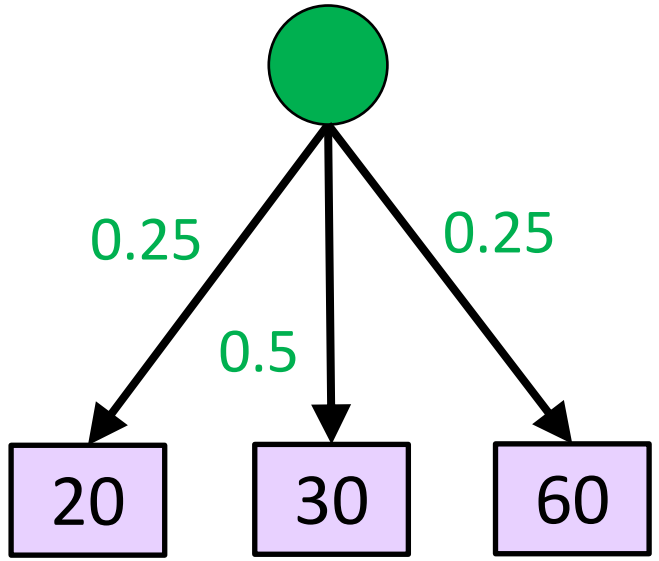
$$\hat{a} = \operatorname{argmax}_{a} V(s'),$$
$$\text{where } s' = result(s, a)$$

Expectations

Time:	20 min	+	30 min	+	60 min
	x		x		x
Probability:	0.25		0.50		0.25







Max node notation

$$V(s) = \max_a V(s'),$$
where $s' = result(s, a)$

Chance node notation

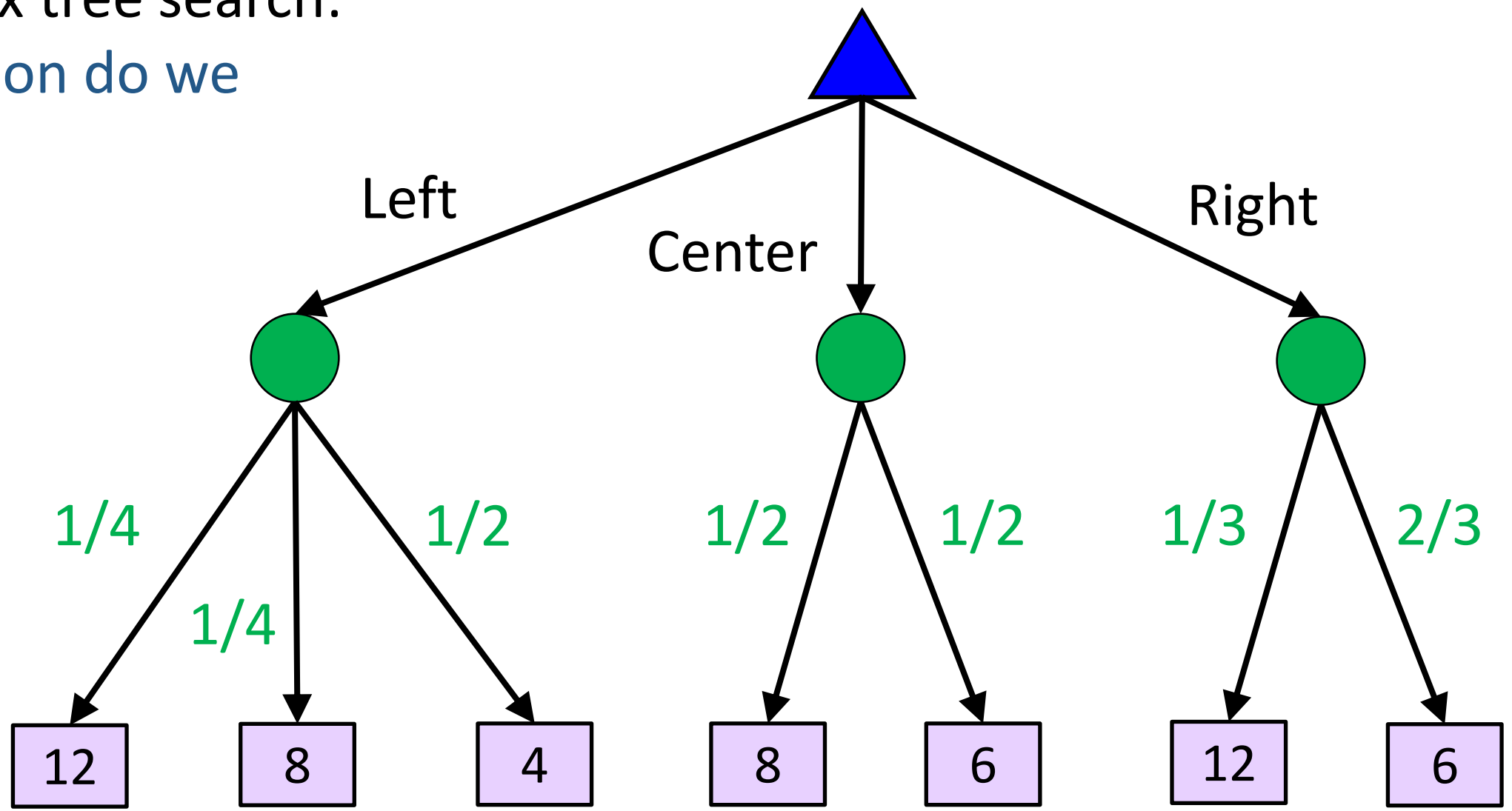
$$V(s) = \sum_{s'} P(s')\, V(s')$$

Previous Poll

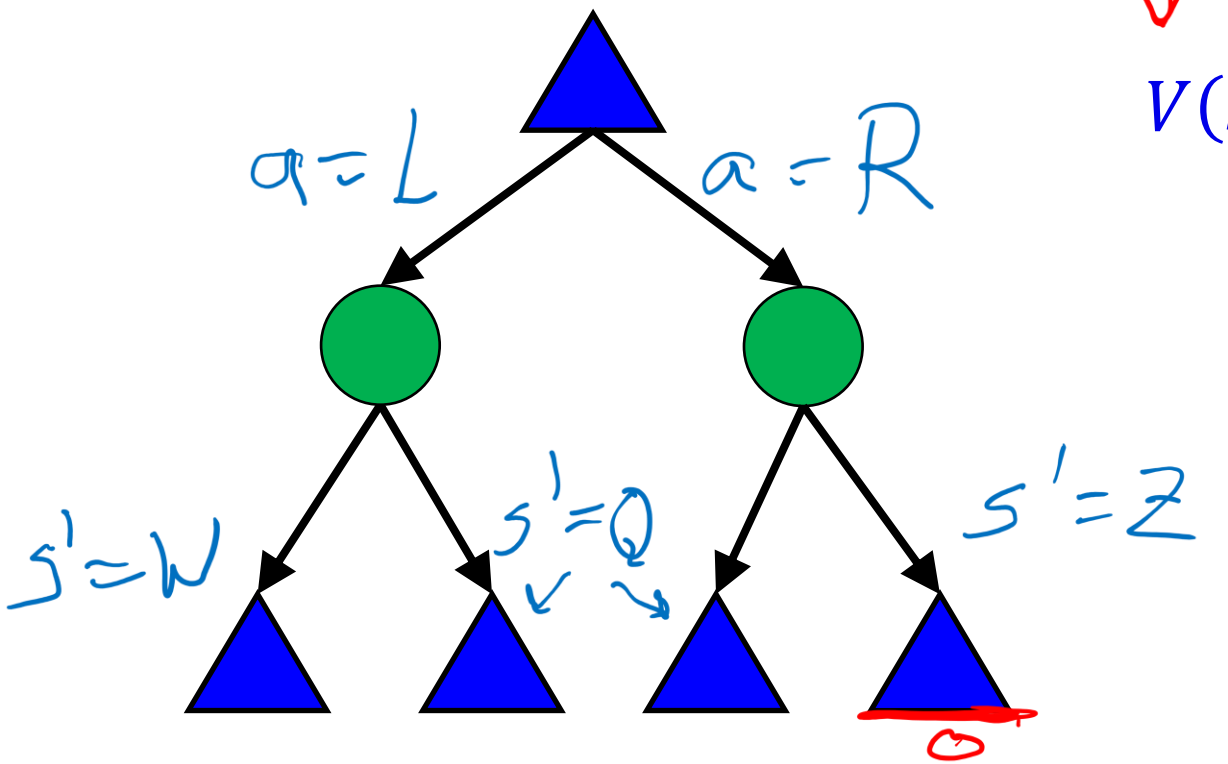
Expectimax tree search:

Which action do we choose?

A) Left
B) Center
C) Right
D) Eight



Expectimax Notation



$$V(s) = \max_{a} \sum_{s'} P(s'|s,a)\, V(s')$$

Warm-up as You Log In

Given

- Set `actions` (persistent/static)
- Set `states` (persistent/static)
- Function `T(s,a,s_prime)`

Write the pseudo code for:

- `function V(s) return value`

that implements:

$$V(s) = \max_{a \in actions} \sum_{s' \in states} T(s,a,s')V(s')$$

```
def V(s):
    best_val
    for a in actions
        for s' in states
            p = T(s, a, s')
            val += p V(s')
    if
    return best_val
```

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

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

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

MDP Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

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

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')], \quad \forall\, s$

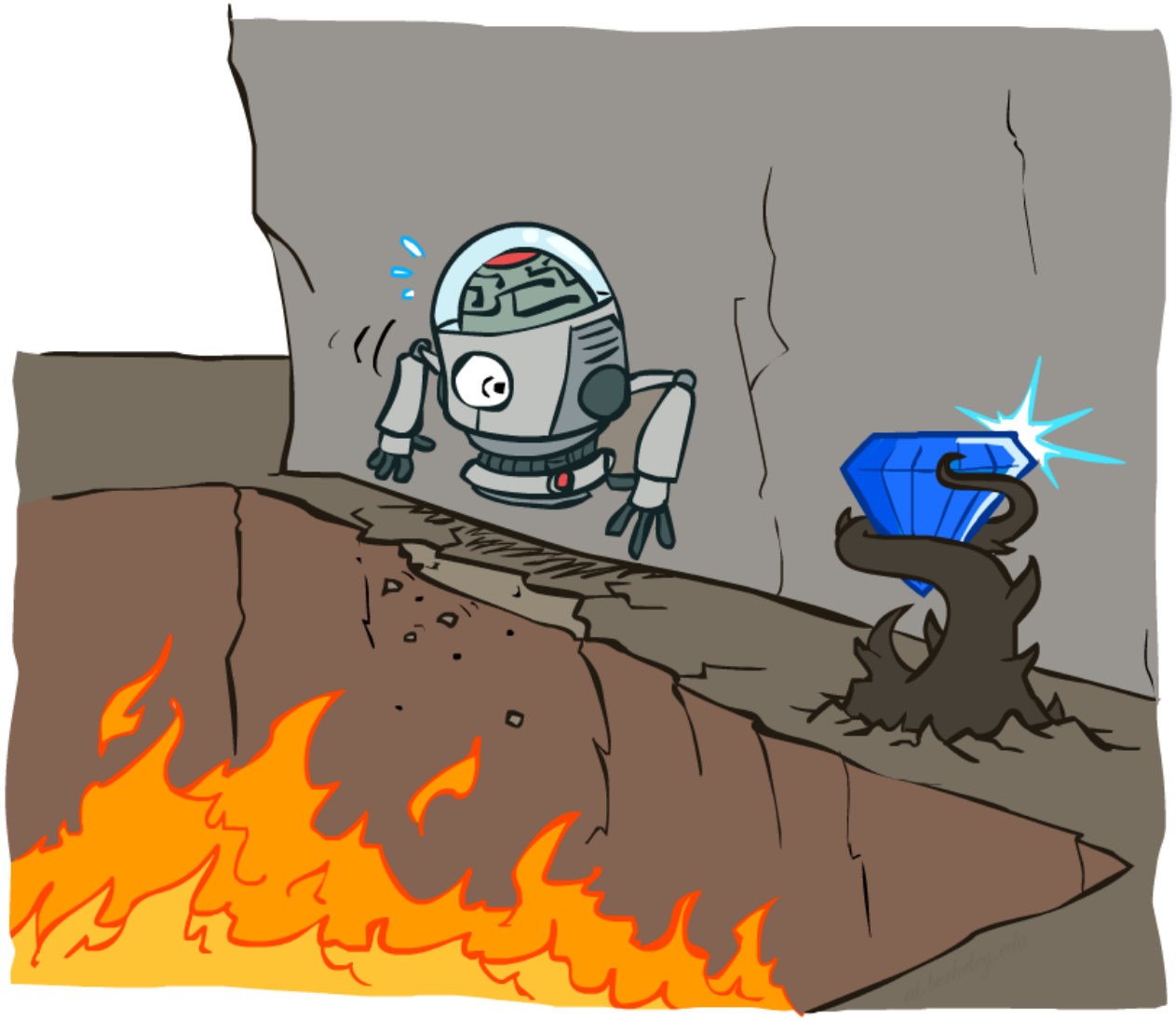
Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')], \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \quad \forall\, s$

Policy evaluation: $V^{\pi}_{k+1}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')], \quad \forall\, s$

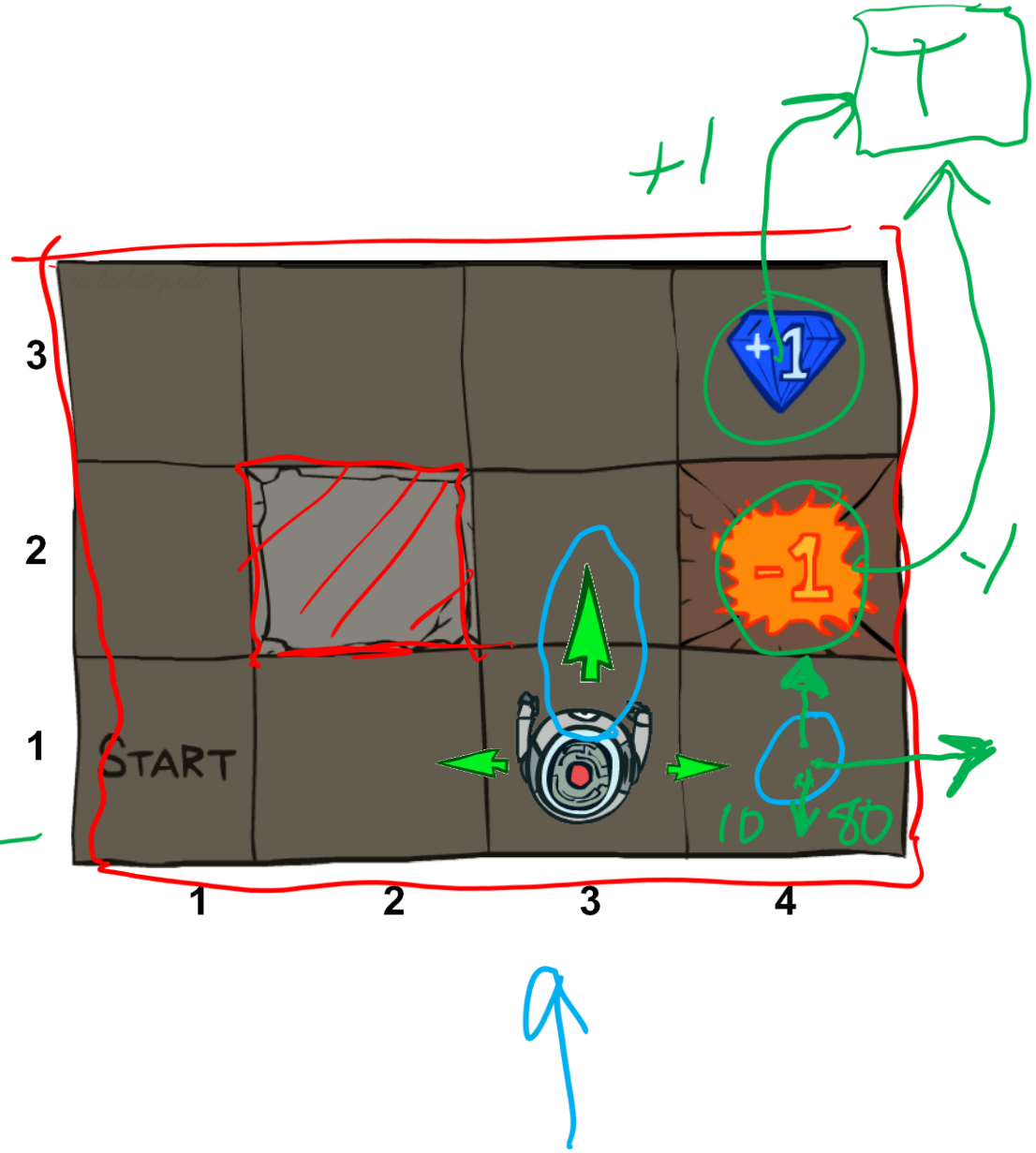
Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')], \quad \forall\, s$

Non-Deterministic Search

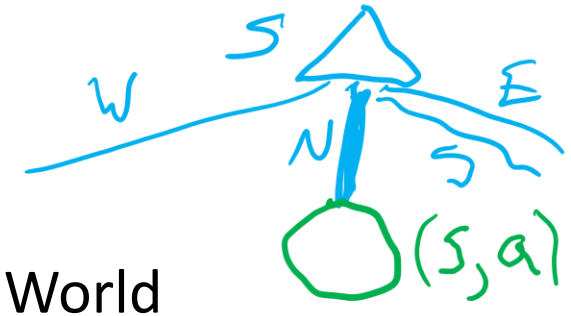


Example: Grid World

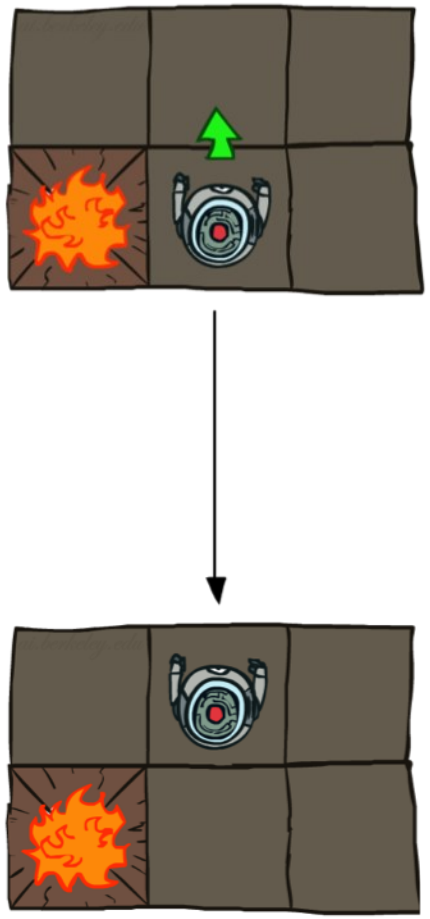
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is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



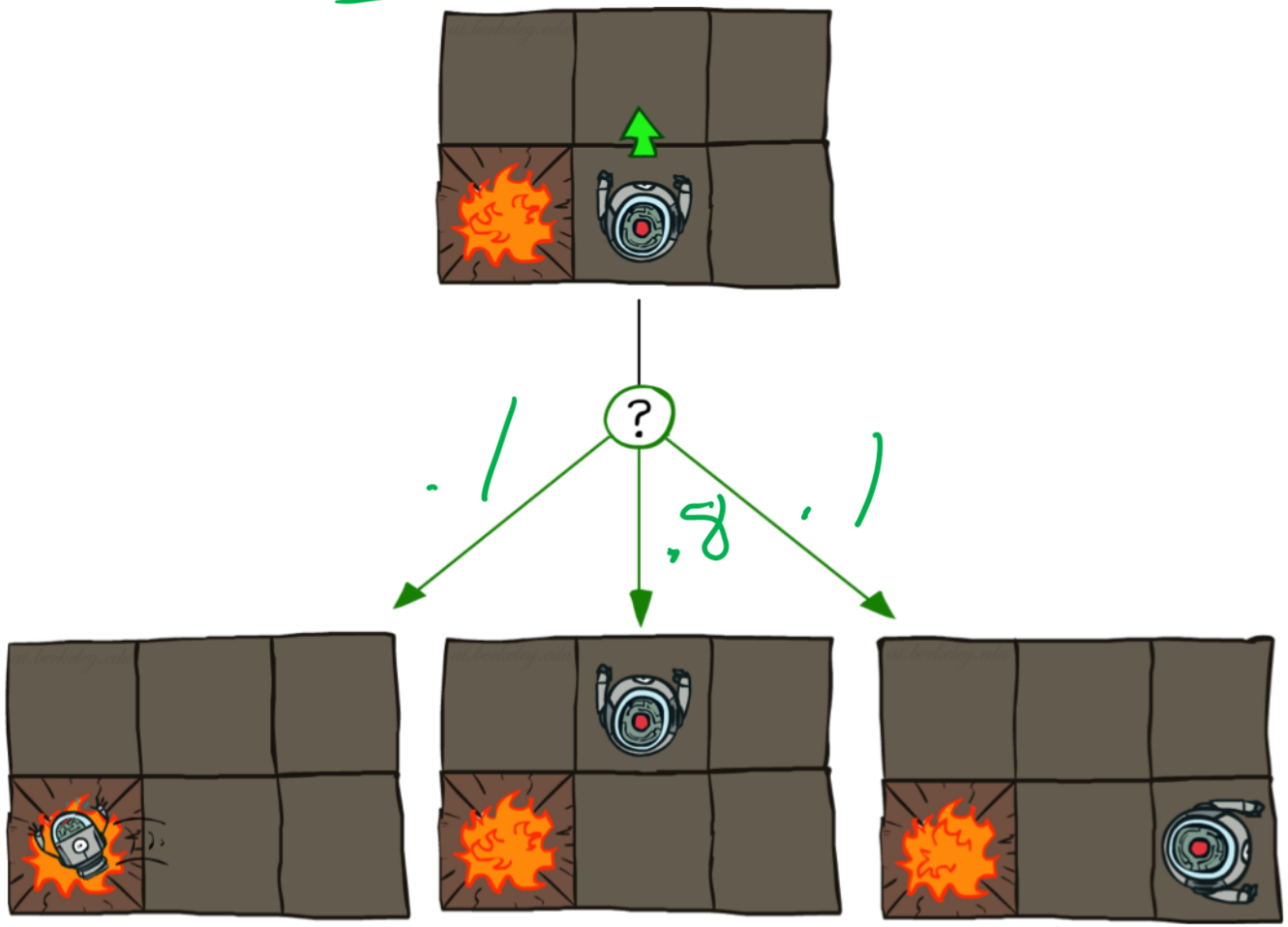
Grid World Actions



Deterministic Grid World



Stochastic Grid World



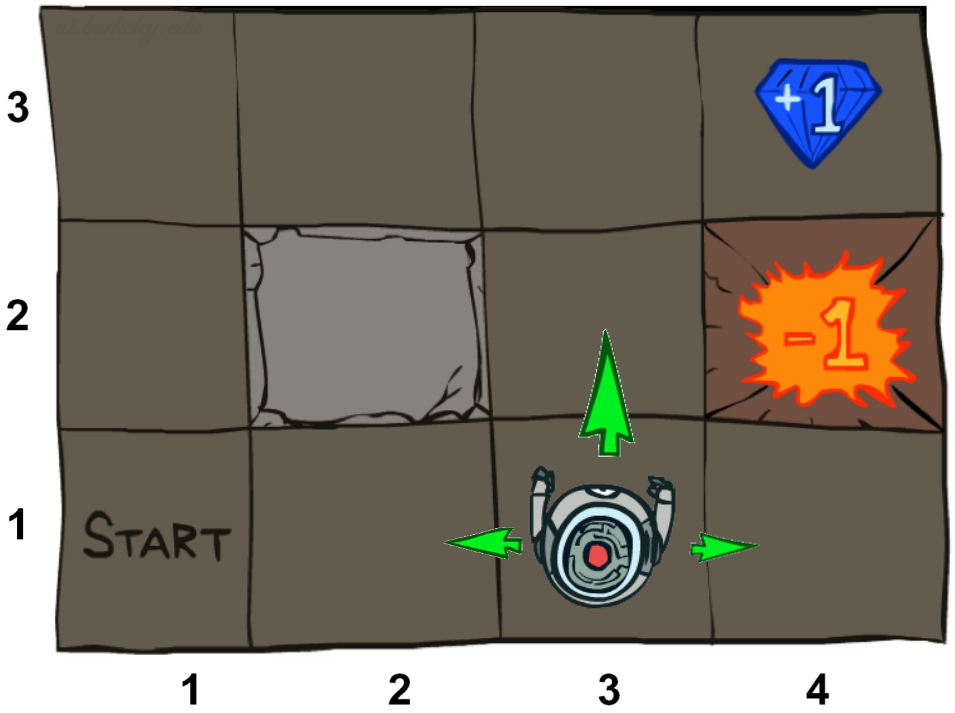
Markov Decision Processes

R(s,a) R(s,a,s')

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., $P(s' \mid s, a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - Sometimes just R(s) or R(s')
- Maybe a terminal state ← 0 possible action

episode



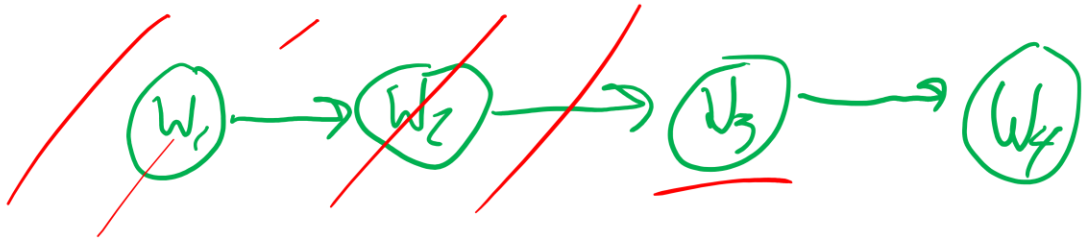
MDPs are non-deterministic search problems

- One way to solve them is with expectimax search
- We'll have a new tool soon

[Demo – gridworld manual intro (L8D1)]

Demo of Gridworld

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

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

$$=$$

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

Andrey Markov
(1856-1922)

This is just like search, where the successor function could only depend on the current state (not the history)

Policies

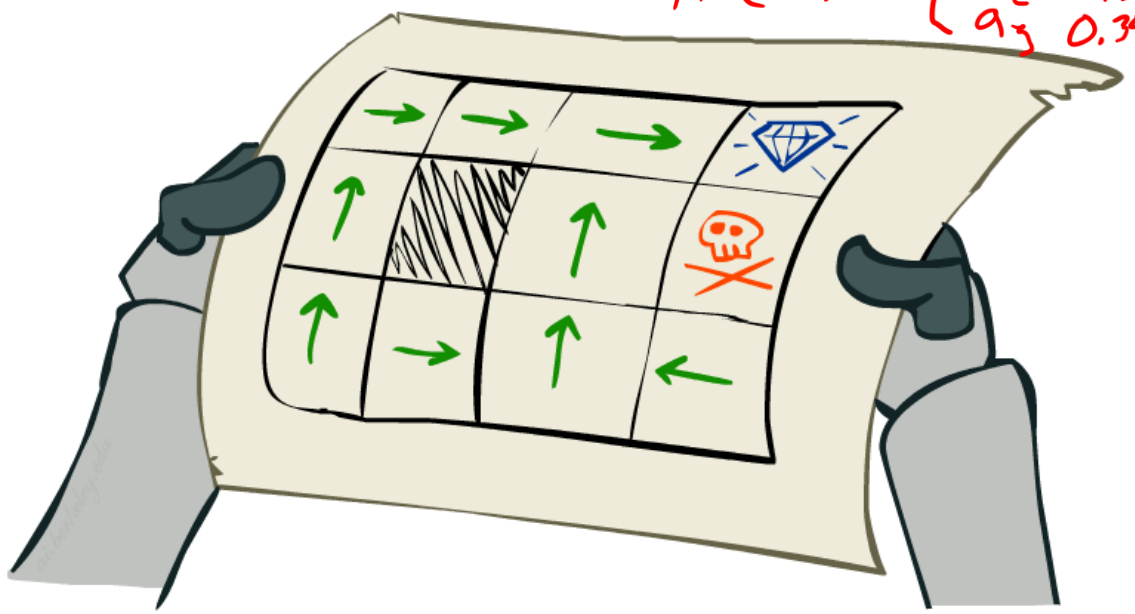
In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal

For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$

- A policy π gives an action for each state
- An optimal policy is one that maximizes expected utility if followed
- An explicit policy defines a reflex agent

Expectimax didn't compute entire policies

- It computed the action for a single state only



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

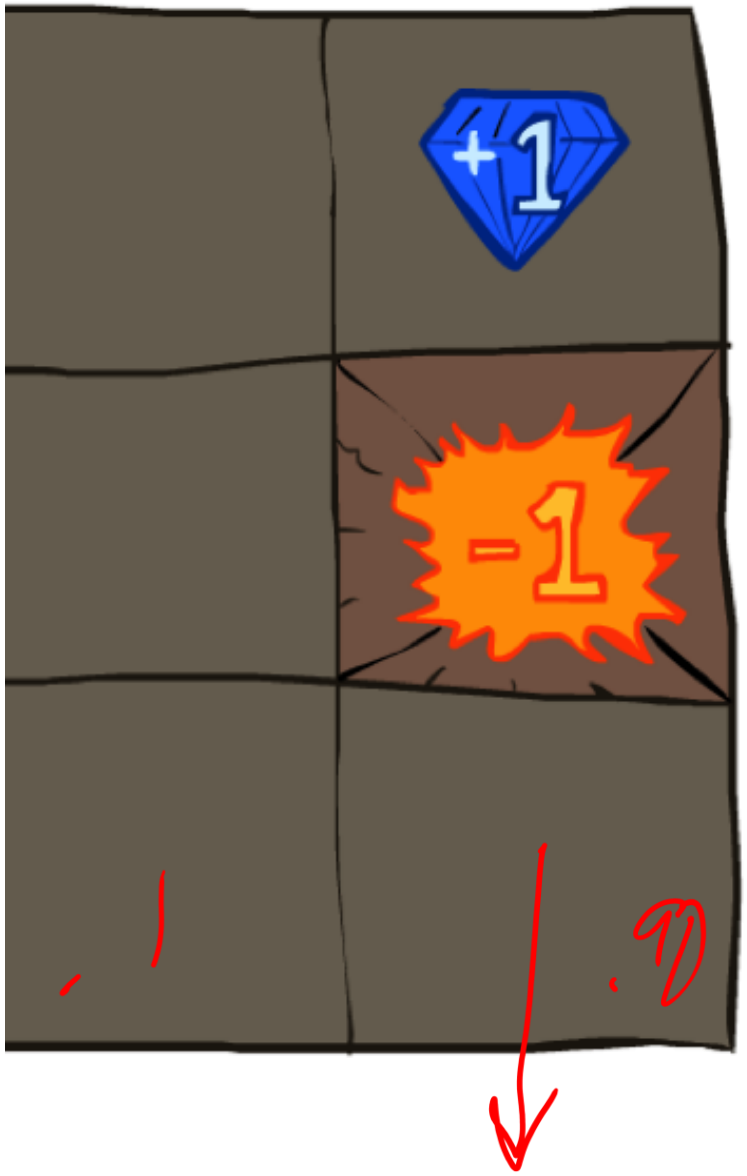
Poll 1

0.80 0.1 0.1

Which is the best move?

A. North
B. East
C. South
D. West

$R(s) = -10$



Poll 2

$R(s) \rightarrow a$

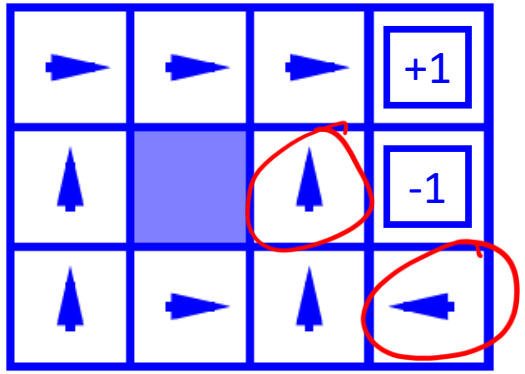
Which sequence of optimal policies matches the following sequence of living rewards:

{-0.01, -0.03, -0.04, -2.0}

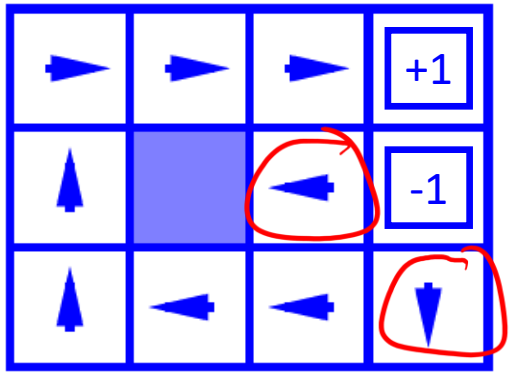
A B C D

I. {A, B, C, D}
II. {B, C, A, D}
III. {D, C, B, A}
IV. {D, A, C, B}

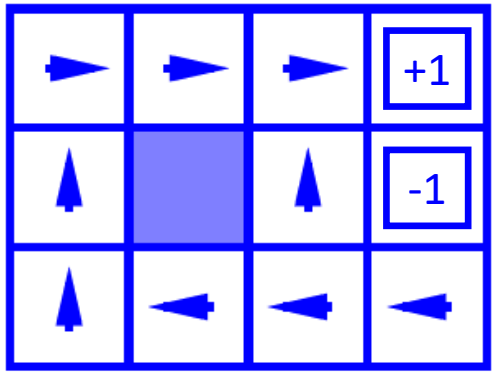
A)



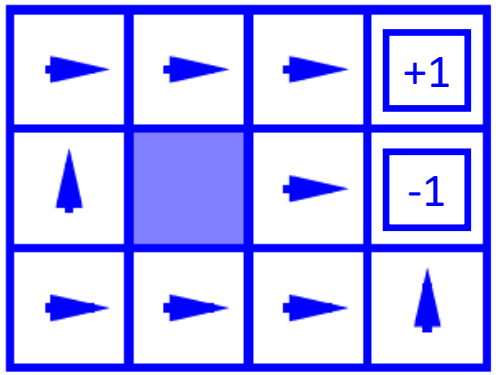
B)



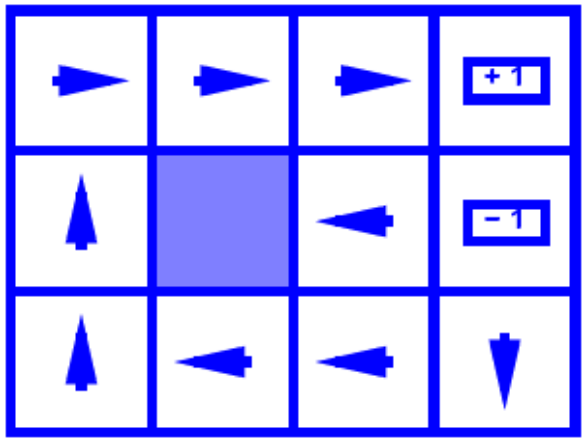
C)



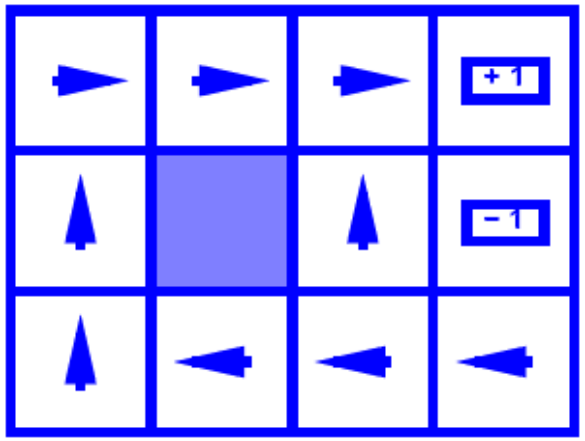
D)



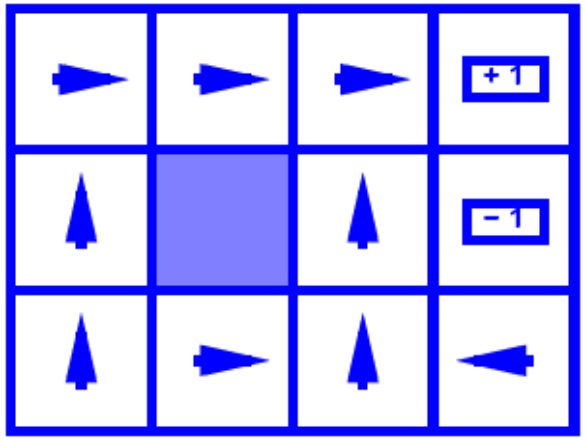
Optimal Policies



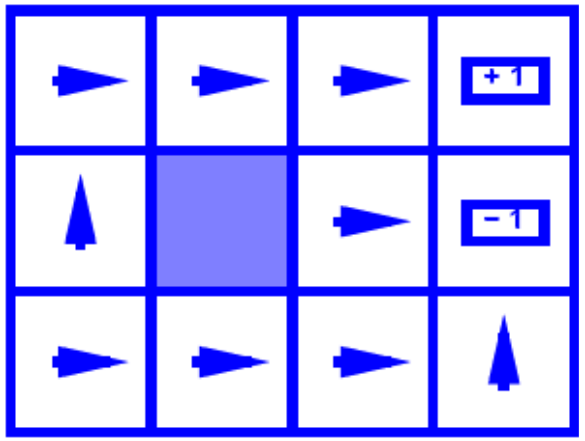
R(s) = -0.01



R(s) = -0.03

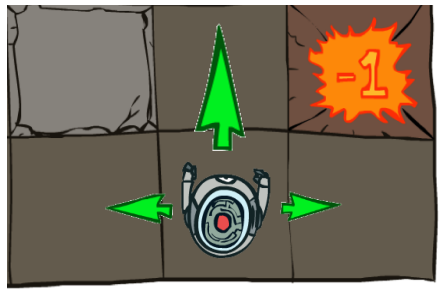


R(s) = -0.4



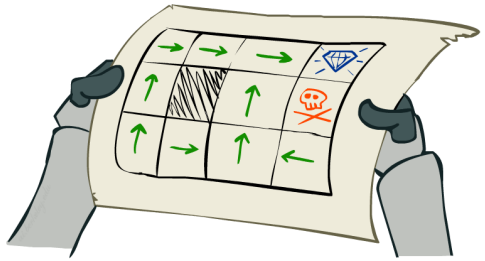
R(s) = -2.0

MDP Outline



MDP Setup

- Expectimax: State, actions, non-deterministic transition functions
  - Example: GridWorld



- Policies: Mapping states → actions
- Rewards
- Discounting, γ

Solving MDPs

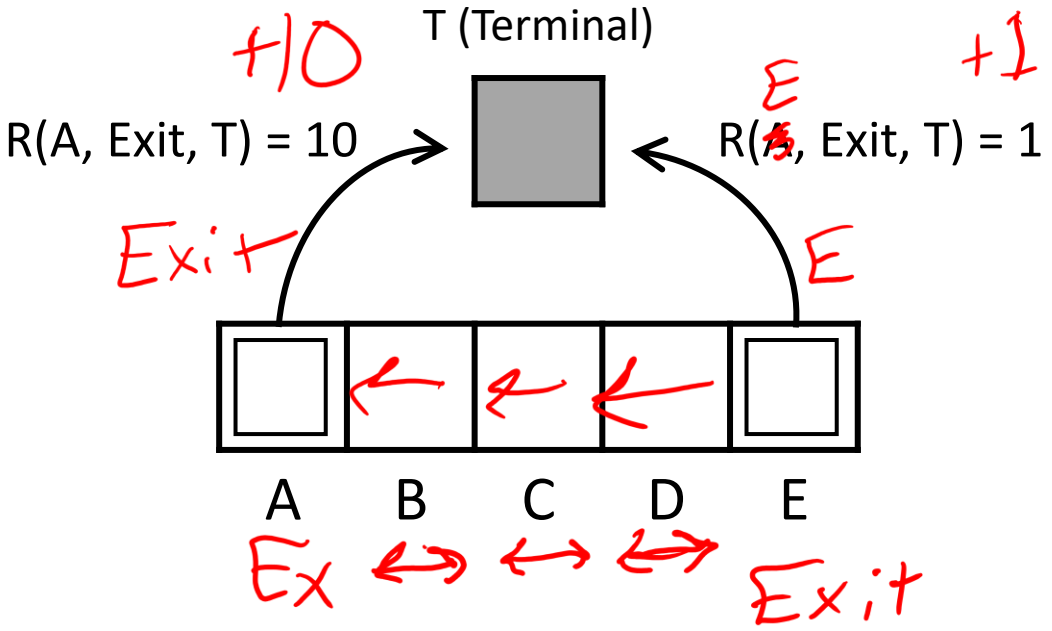
- Method 1) Value iteration
- Method 2) Policy Iteration

Value Iteration

Simple Deterministic Example

- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

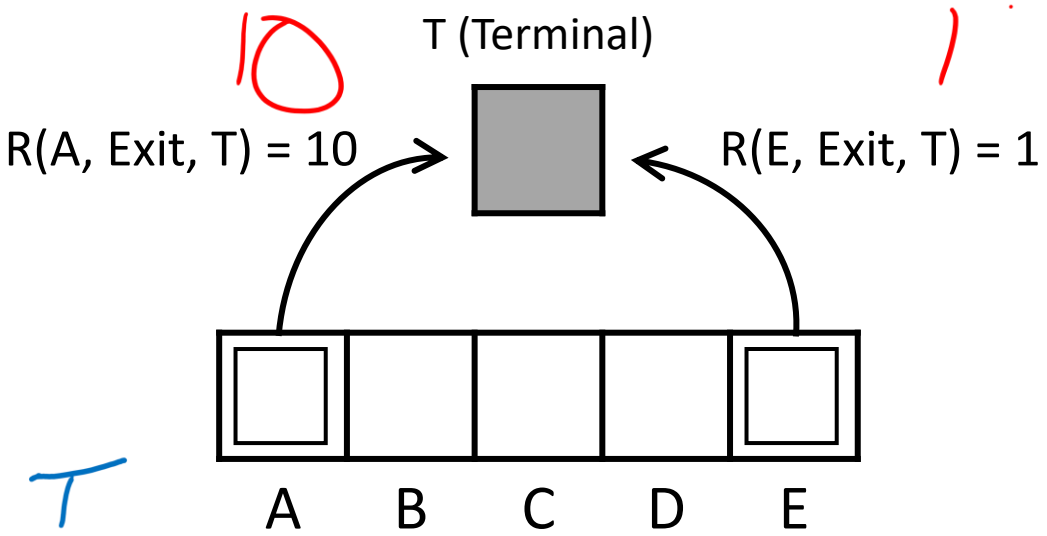
$$V(s) = \max_a [R(s, a, s') + V(s')]$$

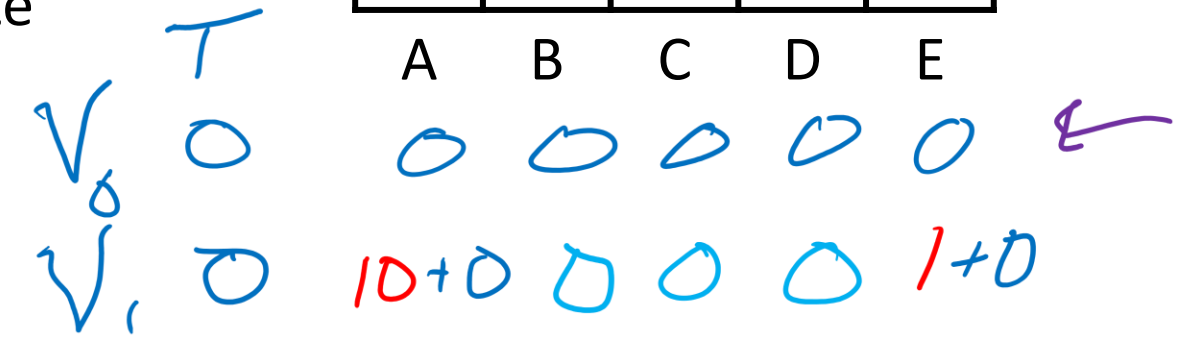


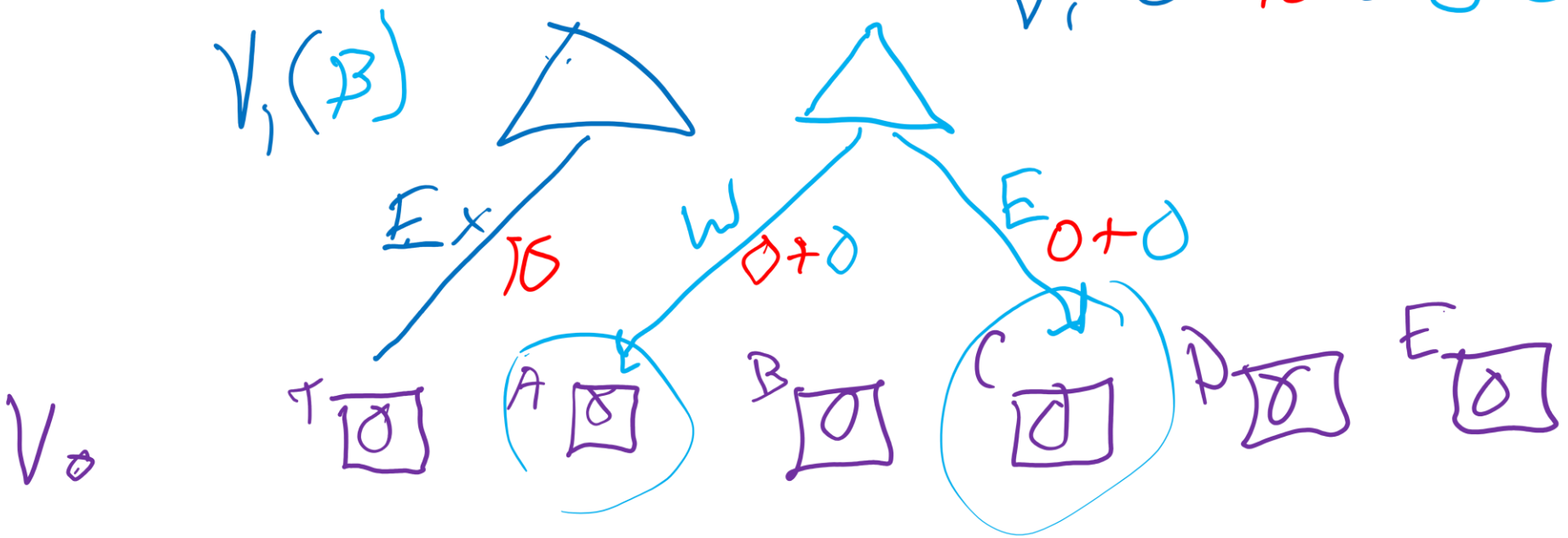
Simple Deterministic Example

- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

$$V_{k+1}(s) = \max_a [R(s,a,s') + V_k(s')]$$



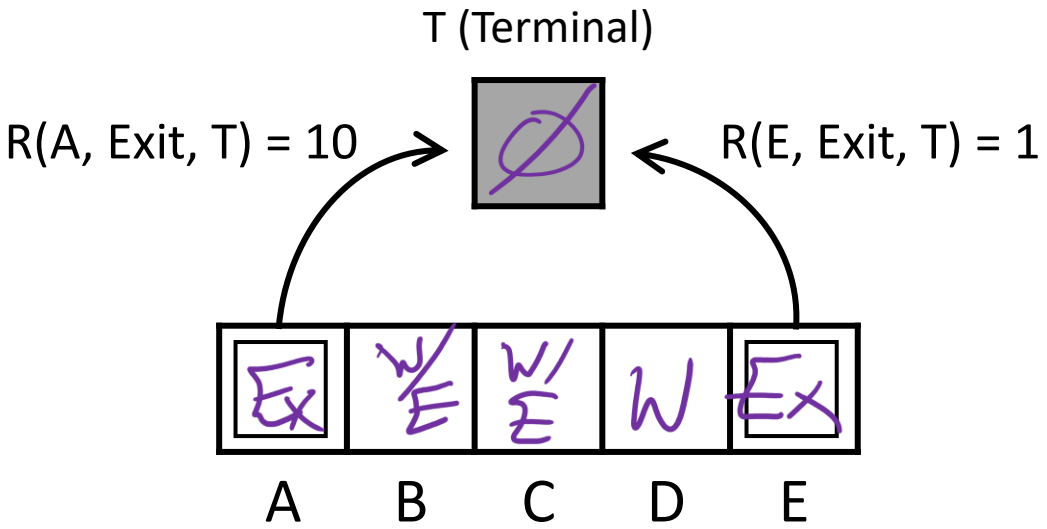




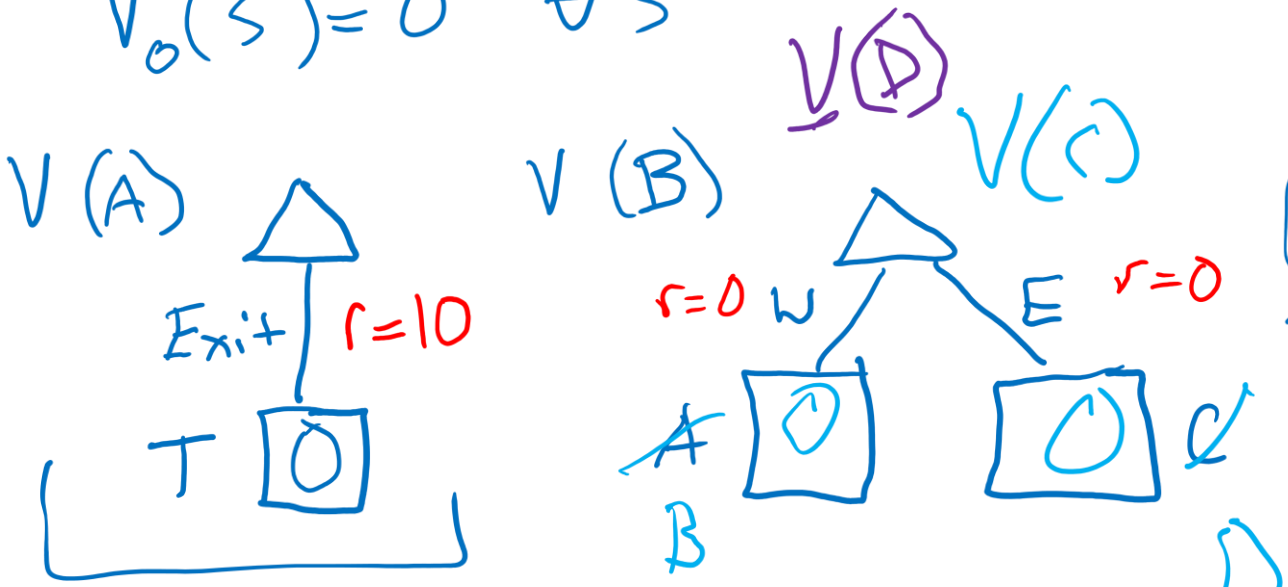
Simple Deterministic Example

- Actions: B, C, D: East, West
- Actions: A, E: Exit
- Transitions: deterministic
- Rewards only for transitioning to terminal state

$$V_{k+1}(s) = \max_a [R(s,a,s') + V_k(s')]$$

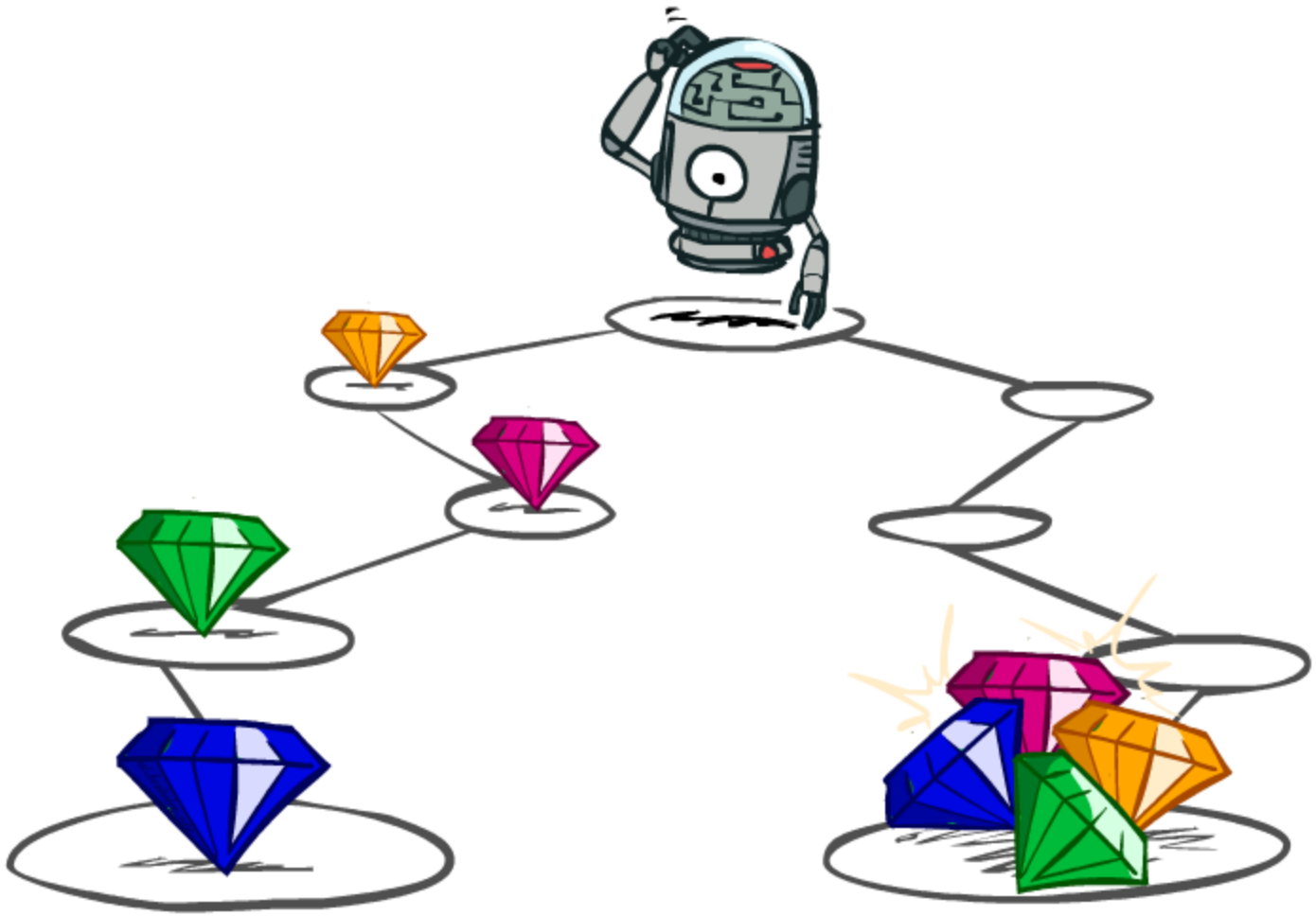


$V_0(s) = 0 \quad \forall s$



	T	A	B	C	D	E
V_0	0	0	0	0	0	0
V_1	0	10	0	0	0	1
V_2	0	10	10	0	1	1
V_3	0	10	10	10	1	1
V_4	0	10	10	10	10	1

Utilities of Sequences

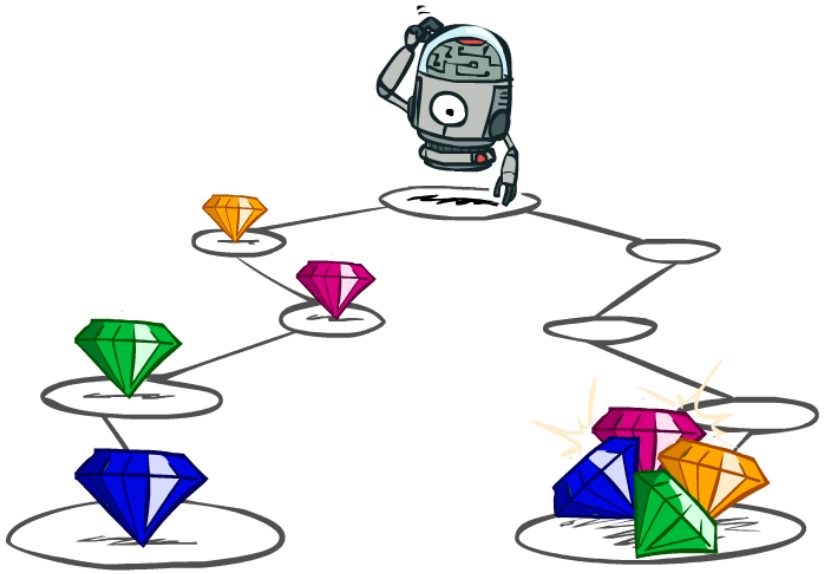


Utilities of Sequences

What preferences 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

$\gamma = 0.9$

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

100

γ

Worth Next Step

90

γ^2 $.9^2 = .81$

Worth In Two Steps

81

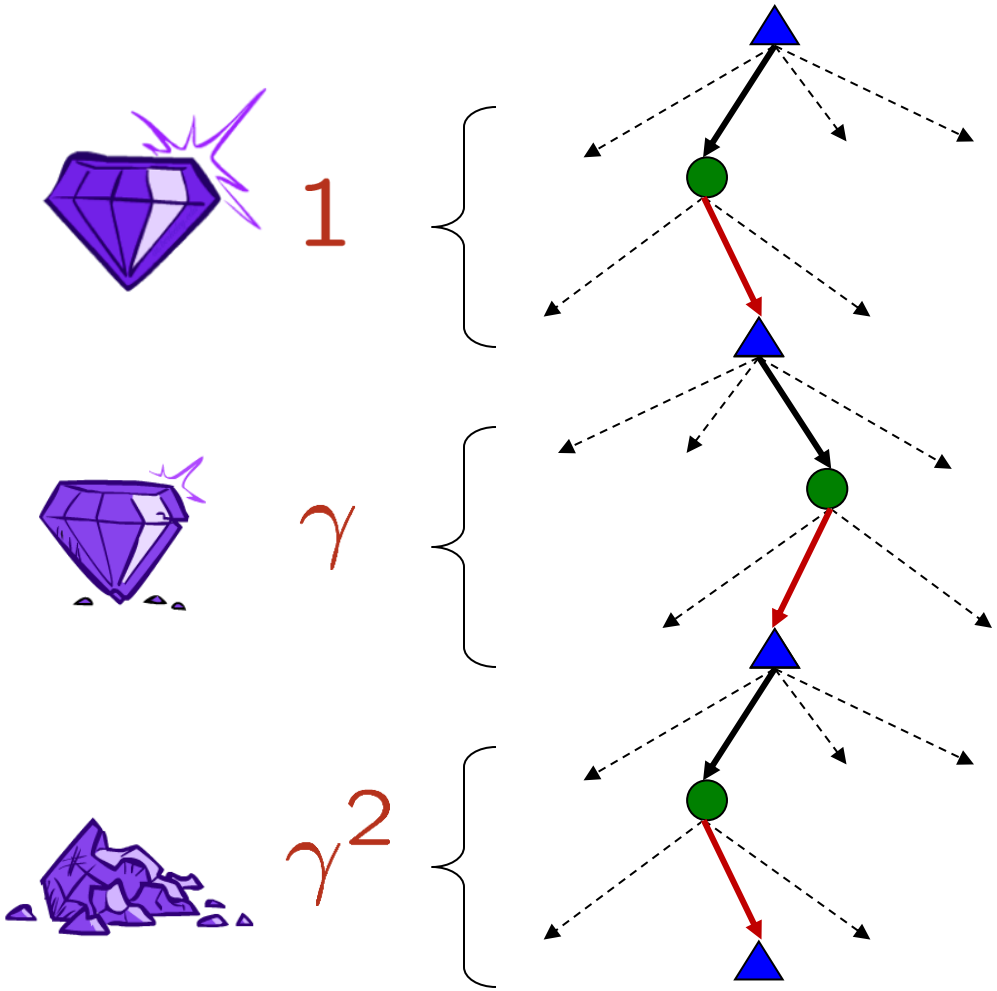
Discounting

How to discount?

- Each time we descend a level, we multiply in the discount once

Why discount?

- Sooner rewards probably do have higher utility than later rewards
- Also helps our algorithms converge
- Important: use $0 < \gamma < 1$



Poll

What is the value of this ordered sequence of rewards [2,4,8] with $\gamma = 0.5$?

A. 3
B. 6
C. 7
D. 14

Bonus: What is the value of [8,4,2] with $\gamma = 0.5$?