CS 471/571(Fall 2023): Introduction to Artificial Intelligence

Lecture 10: MDPs (Part 1)

Thanh H. Nguyen

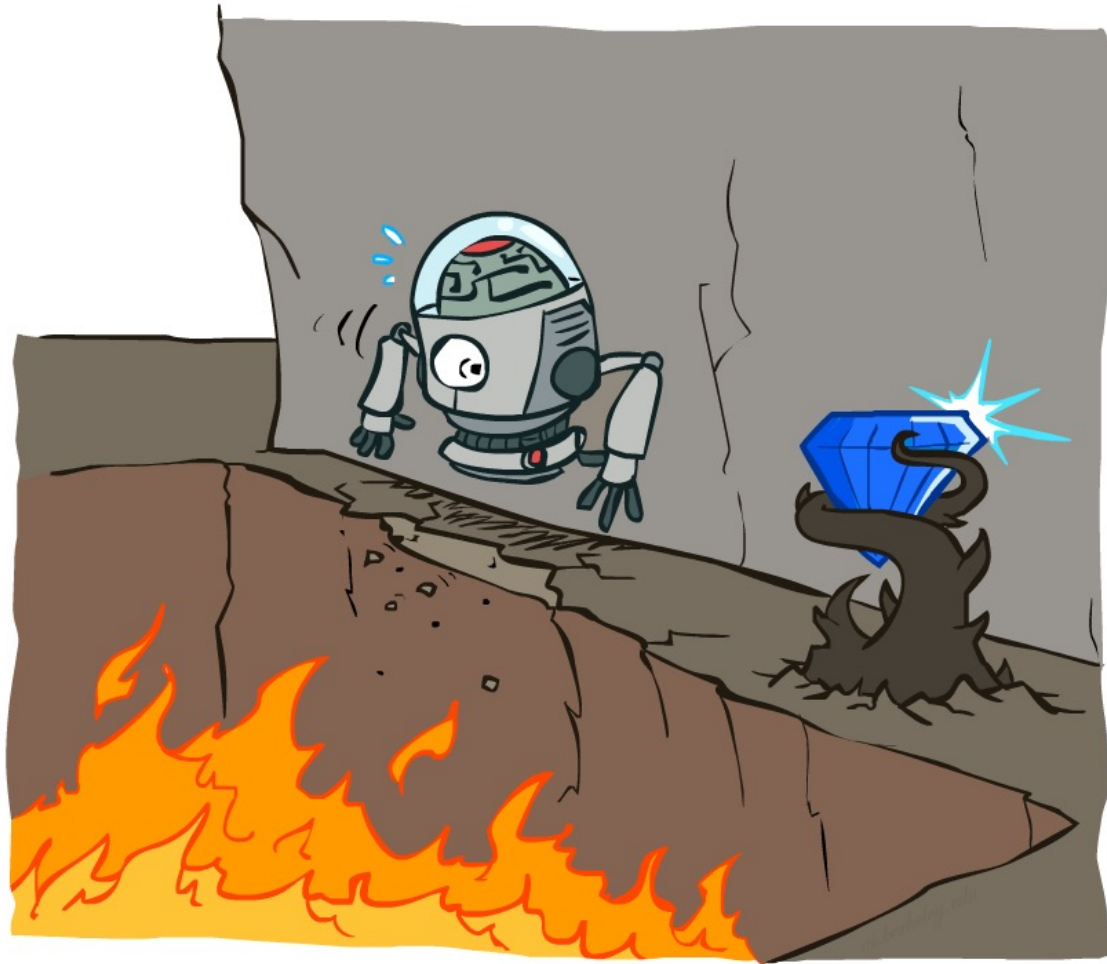Source: http://ai.berkeley.edu/home.html

Reminder

- Homework 2: CSPs and Games
 - Deadline: Oct 25th, 2023

- Project 2: Multi-agent Search
 - Deadline: November 03, 2023

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)
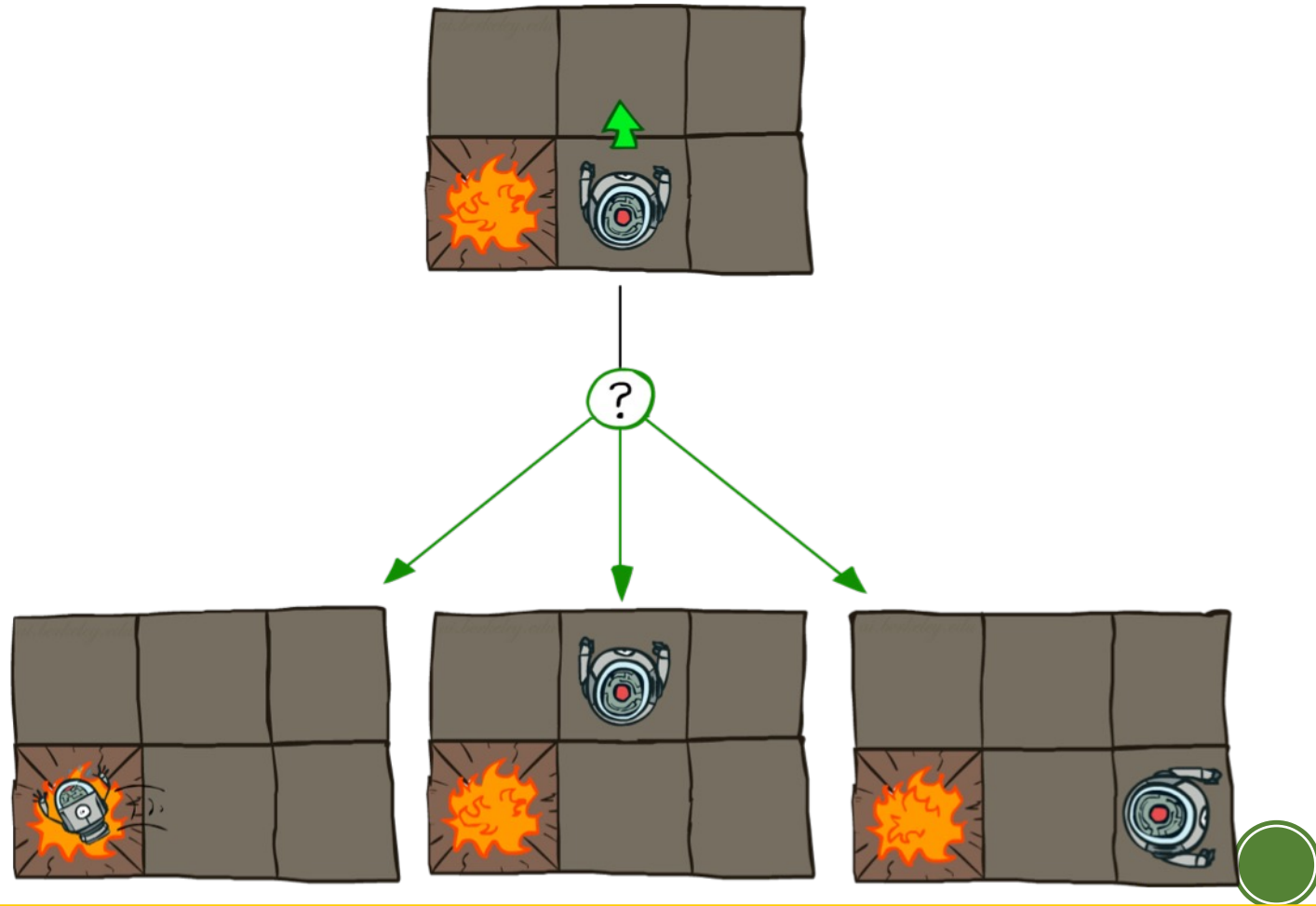
- **Goal: maximize sum of rewards**
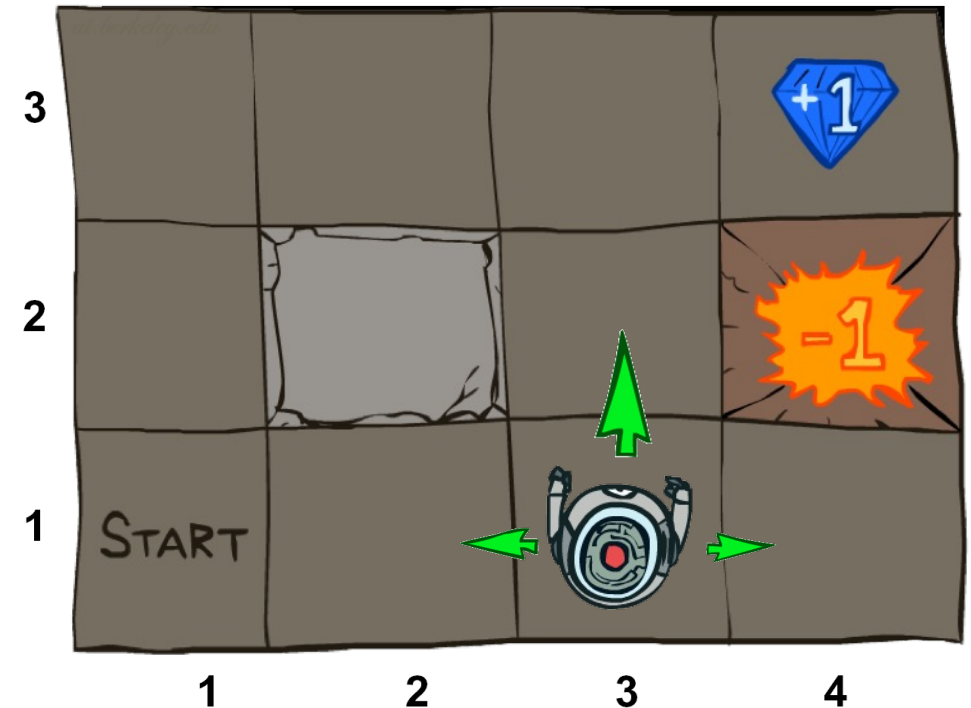
# Grid World Actions

Deterministic Grid World
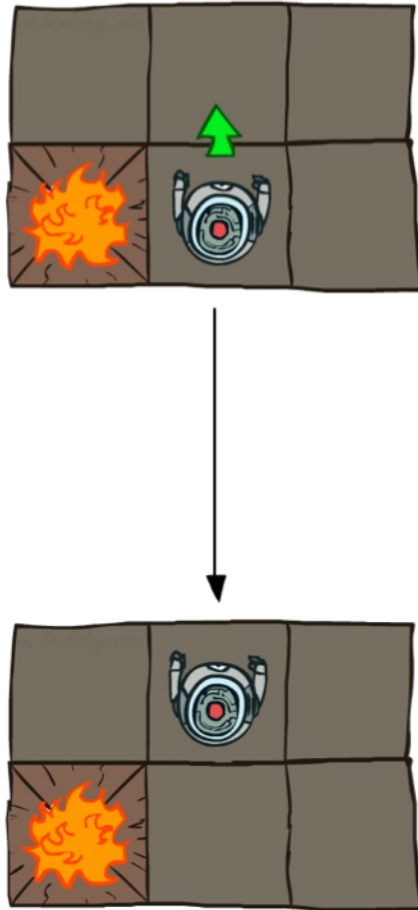
Stochastic Grid World

Markov Decision Processes

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

- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon
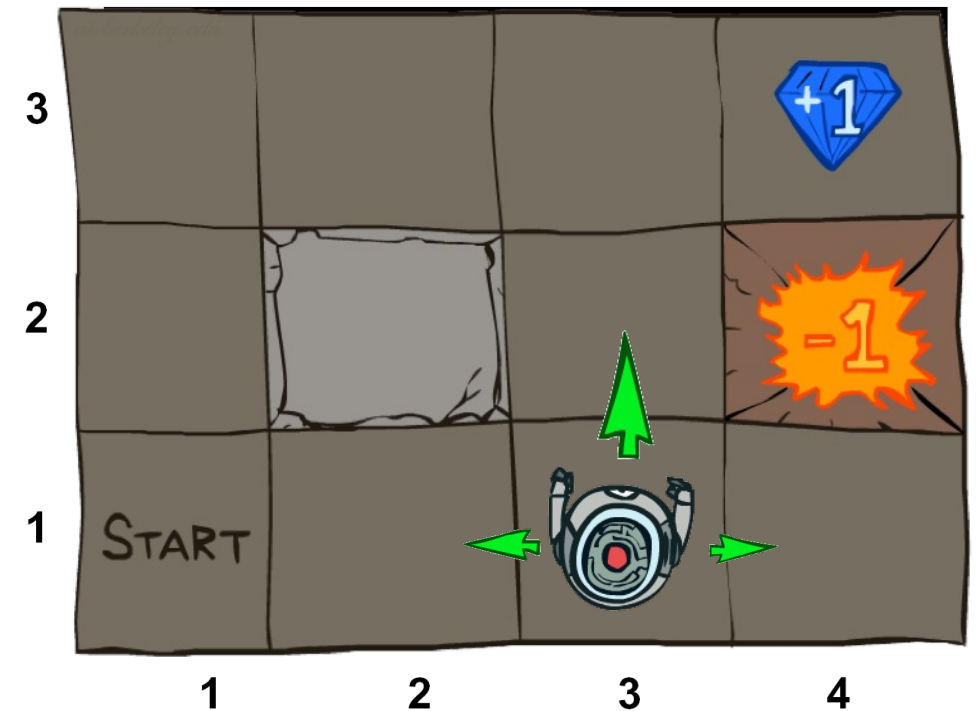
What is Markov about MDPs?

- "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)
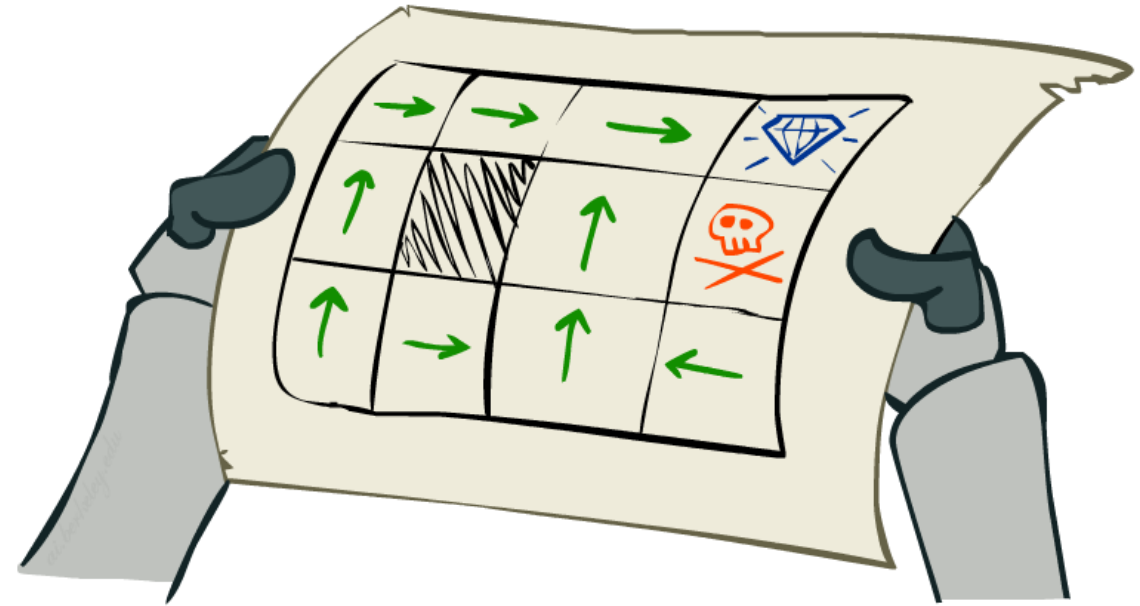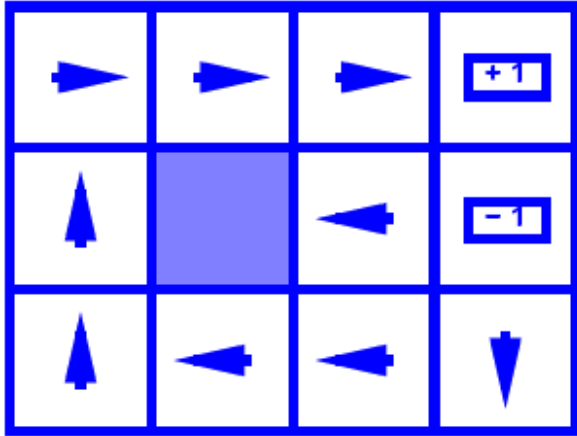
Policies

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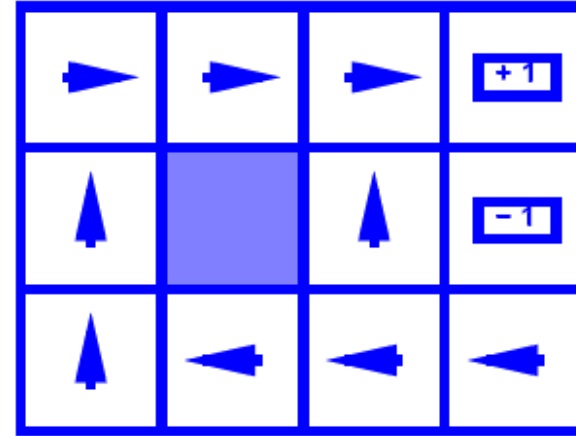 - An optimal policy is one that maximizes expected utility if followed

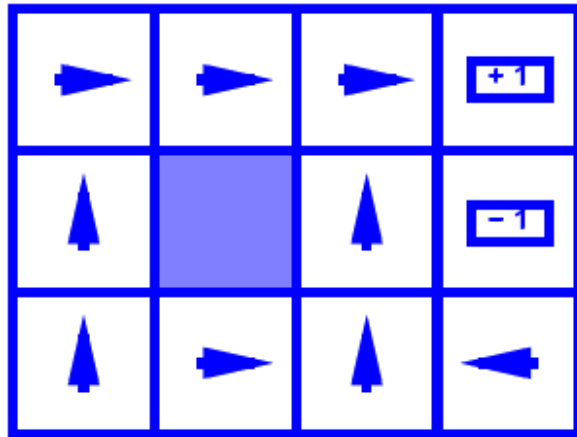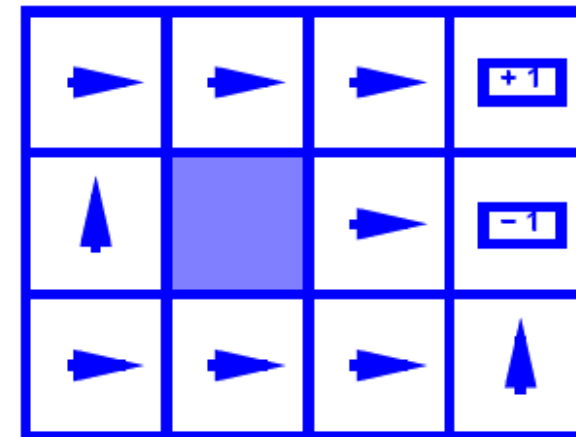Optimal policy when R(s, a, s') = -0.03 for all non-terminals s
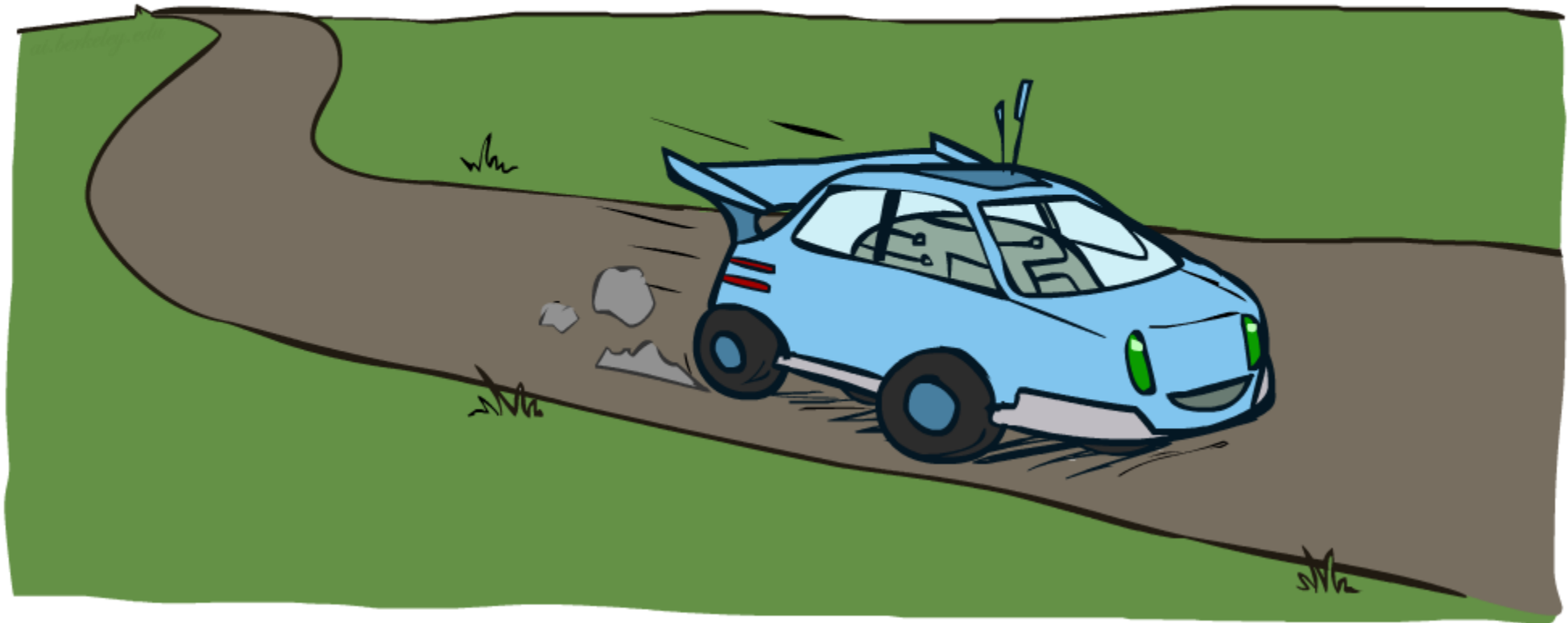
Optimal Policies
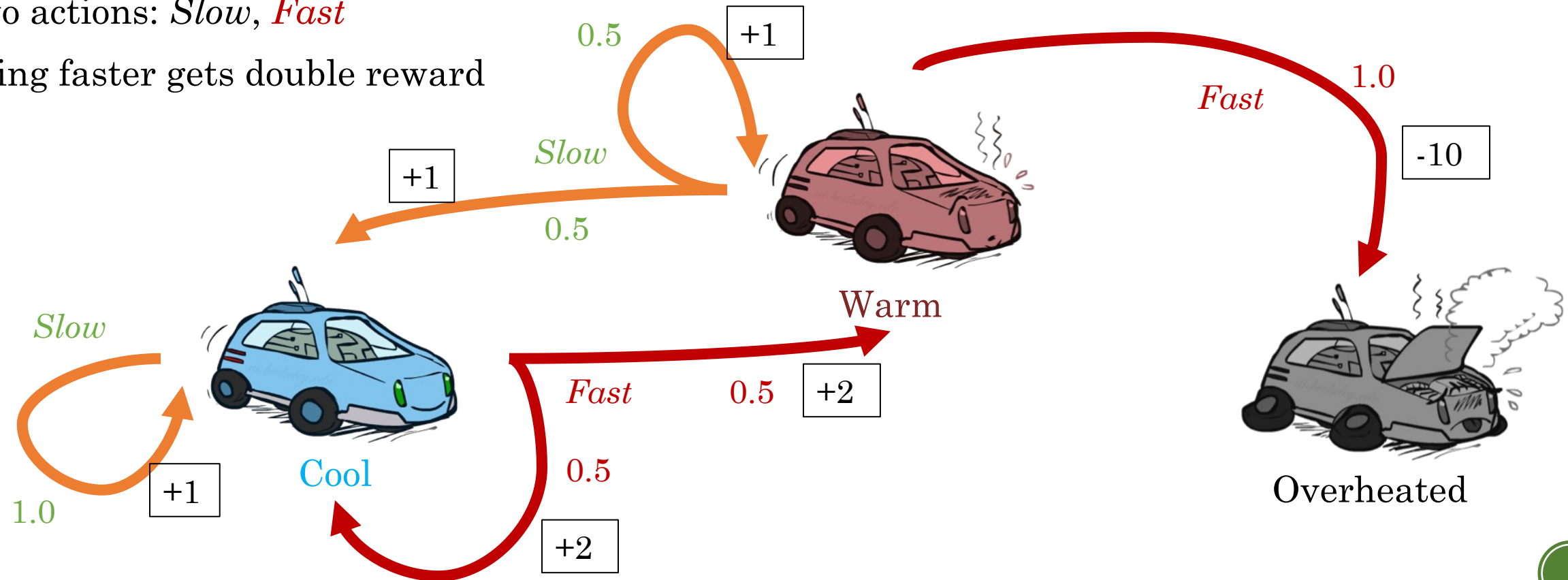

R(s) = -0.01

R(s) = -0.03

R(s) = -0.4

R(s) = -2.0

Example: Racing

Example: Racing

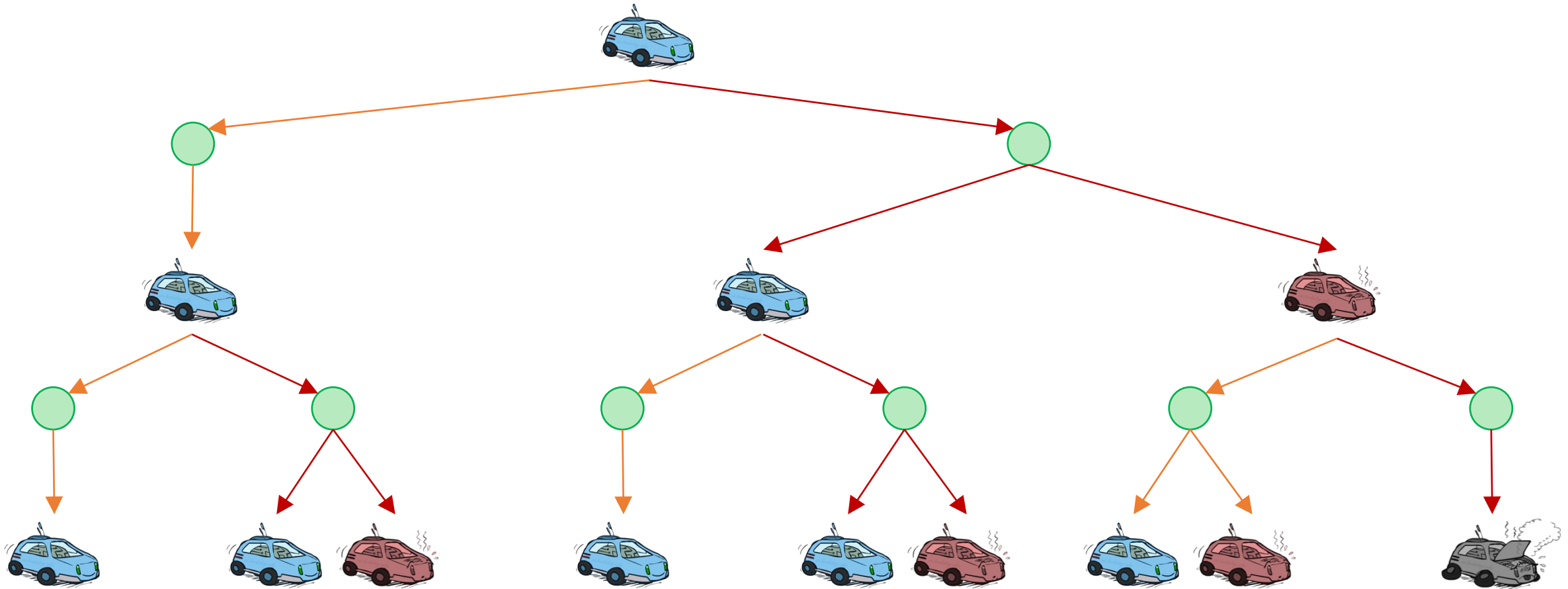- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

Racing Search Tree

MDP Search Trees

▪ Each MDP state projects an expectimax-like search tree

s — s is a *state*

a

(s, a) is a *q-state*

s, a

(s,a,s') called a *transition*

$T(s,a,s') = P(s' \mid s,a)$

$R(s,a,s')$

s,a,s'

s'
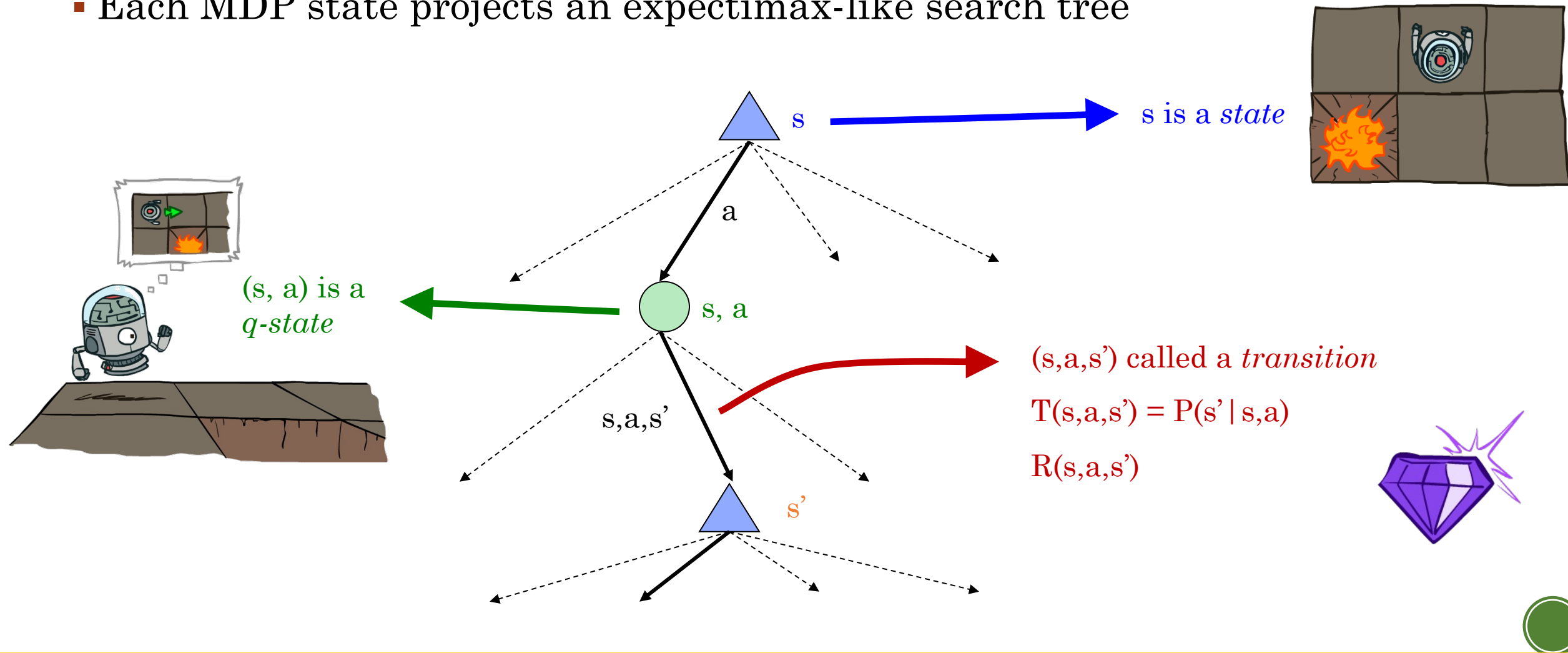
Utilities of Sequences
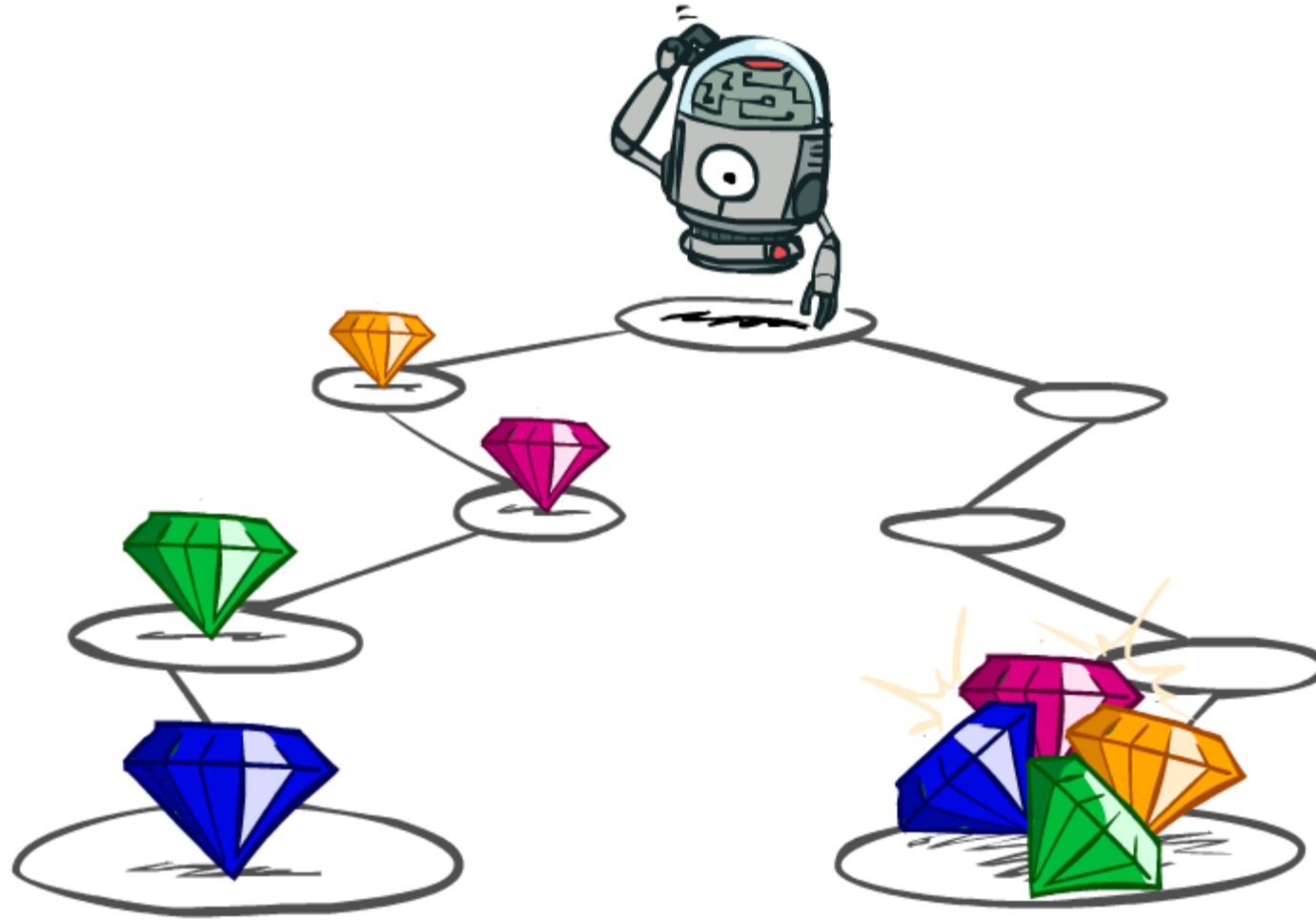
Utilities of Sequences

- 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]
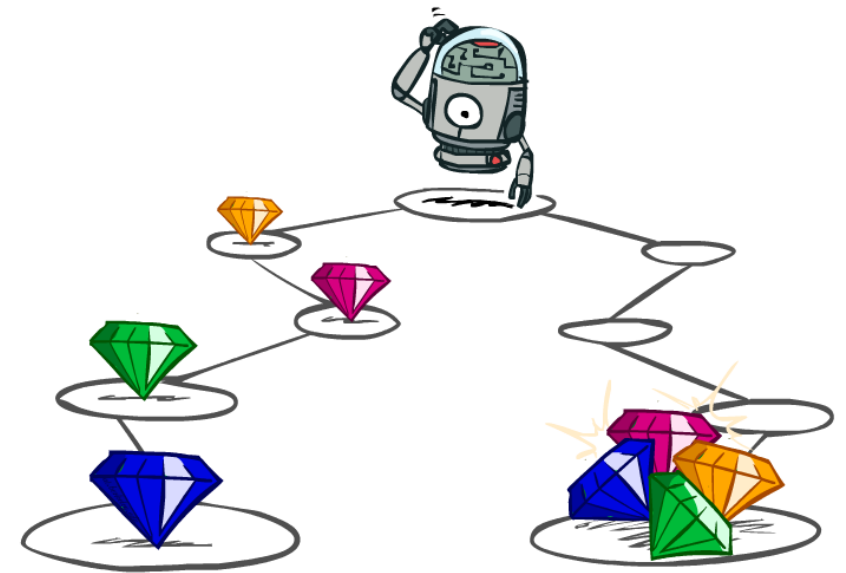
Discounting

- 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

$$1$$

$$\gamma$$

$$\gamma^2$$

Worth Now   Worth Next Step   Worth In Two Steps

Discounting

- **How to discount?**
  - Each time we descend a level, we multiply in the discount once

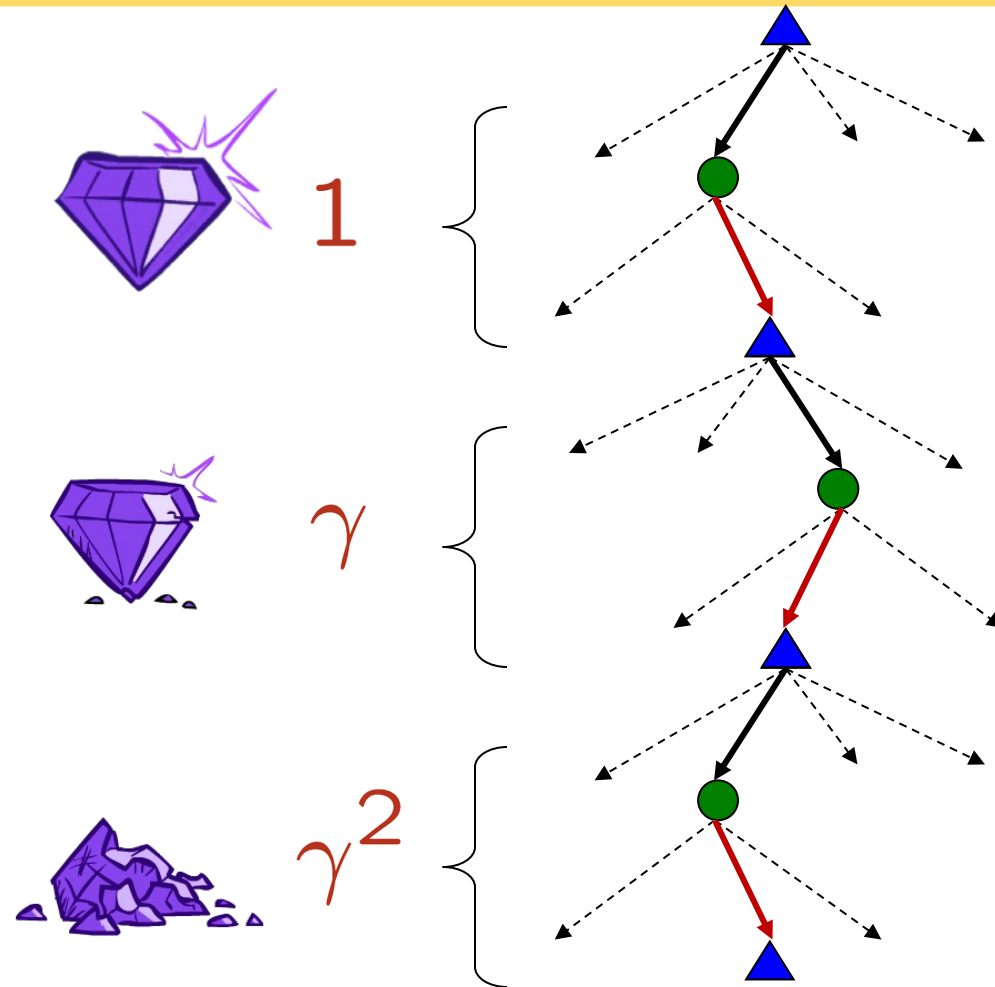- **Why discount?**
  - Sooner rewards probably do have higher utility than later rewards
  - Also helps our algorithms converge

- **Example: discount of 0.5**
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])

1

γ

γ^2

Stationary Preferences

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$

$$\updownarrow$$

$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

- Then: there are only two ways to define utilities
  - Additive utility: $\quad U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $\quad U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$
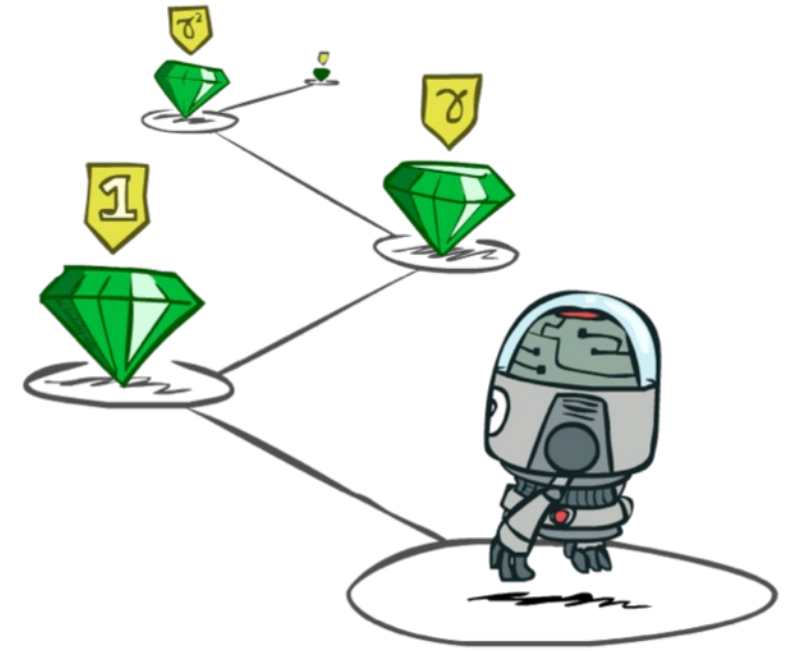
Quiz: Discounting

- 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 γ are West and East equally good when in state d?
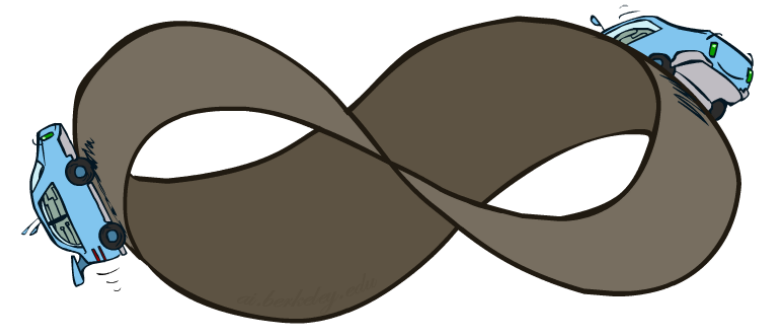
Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?

- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)

 - Discounting: use $0 < \gamma < 1$

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

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

  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)
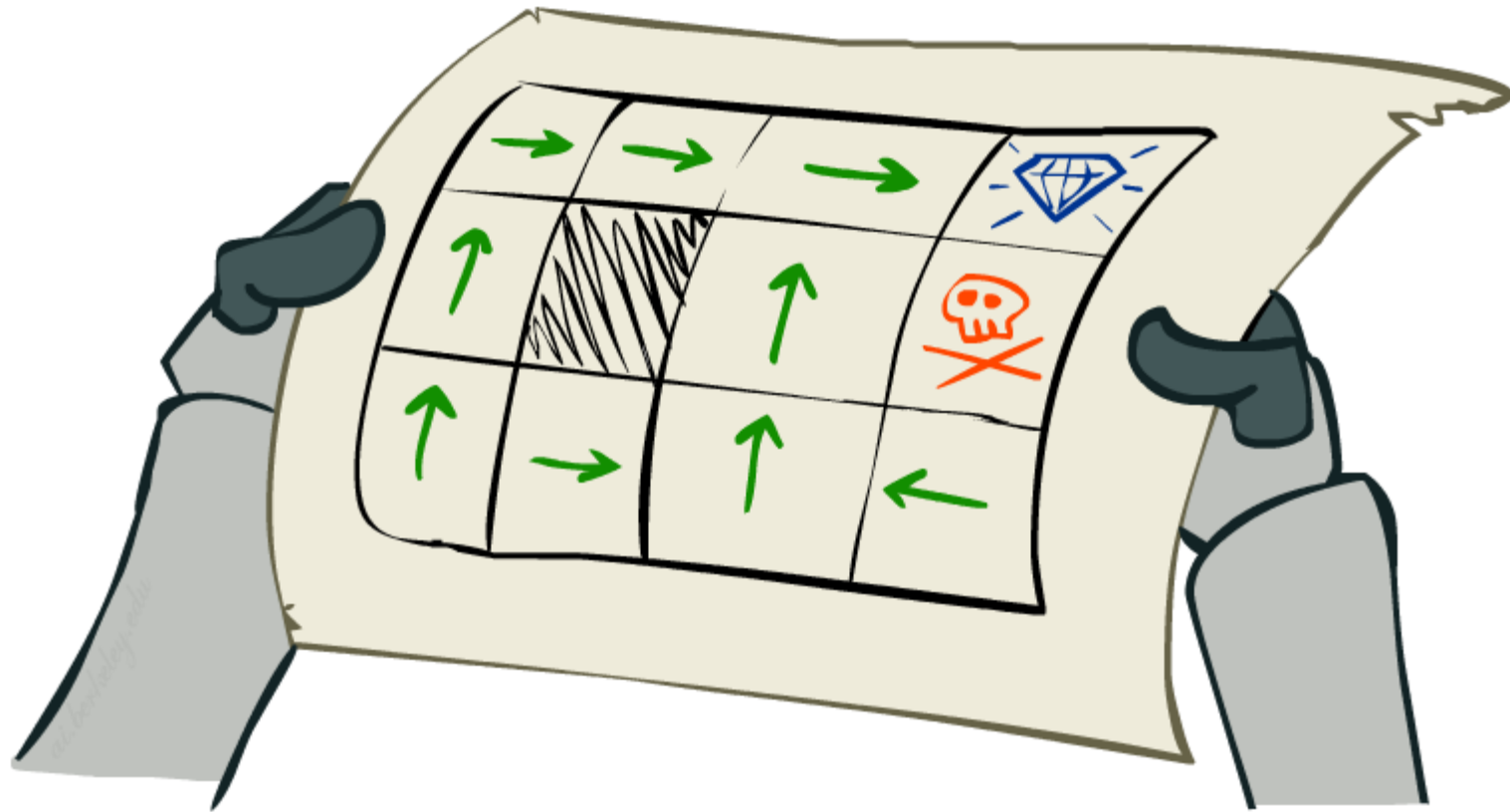
Recap: Defining MDPs

- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of 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 = sum of (discounted) rewards
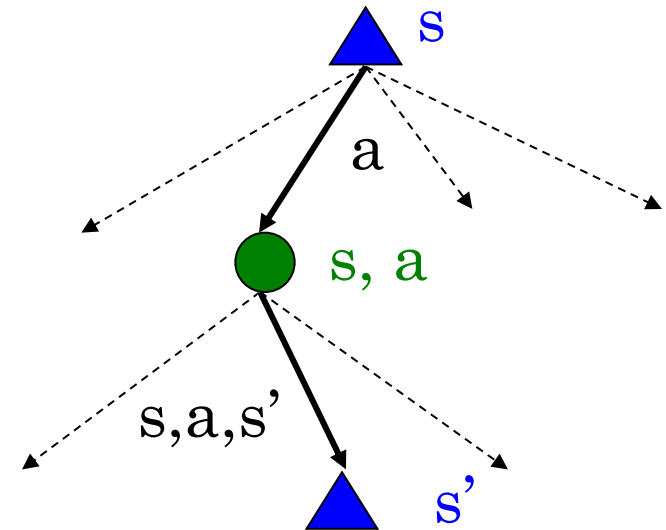
Solving MDPs

Optimal Quantities

- ■ The value (utility) of a state s:

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

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

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

- ■ The optimal policy:

  π*(s) = optimal action from state s

s is a *state*

(s, a) is a *q-state*
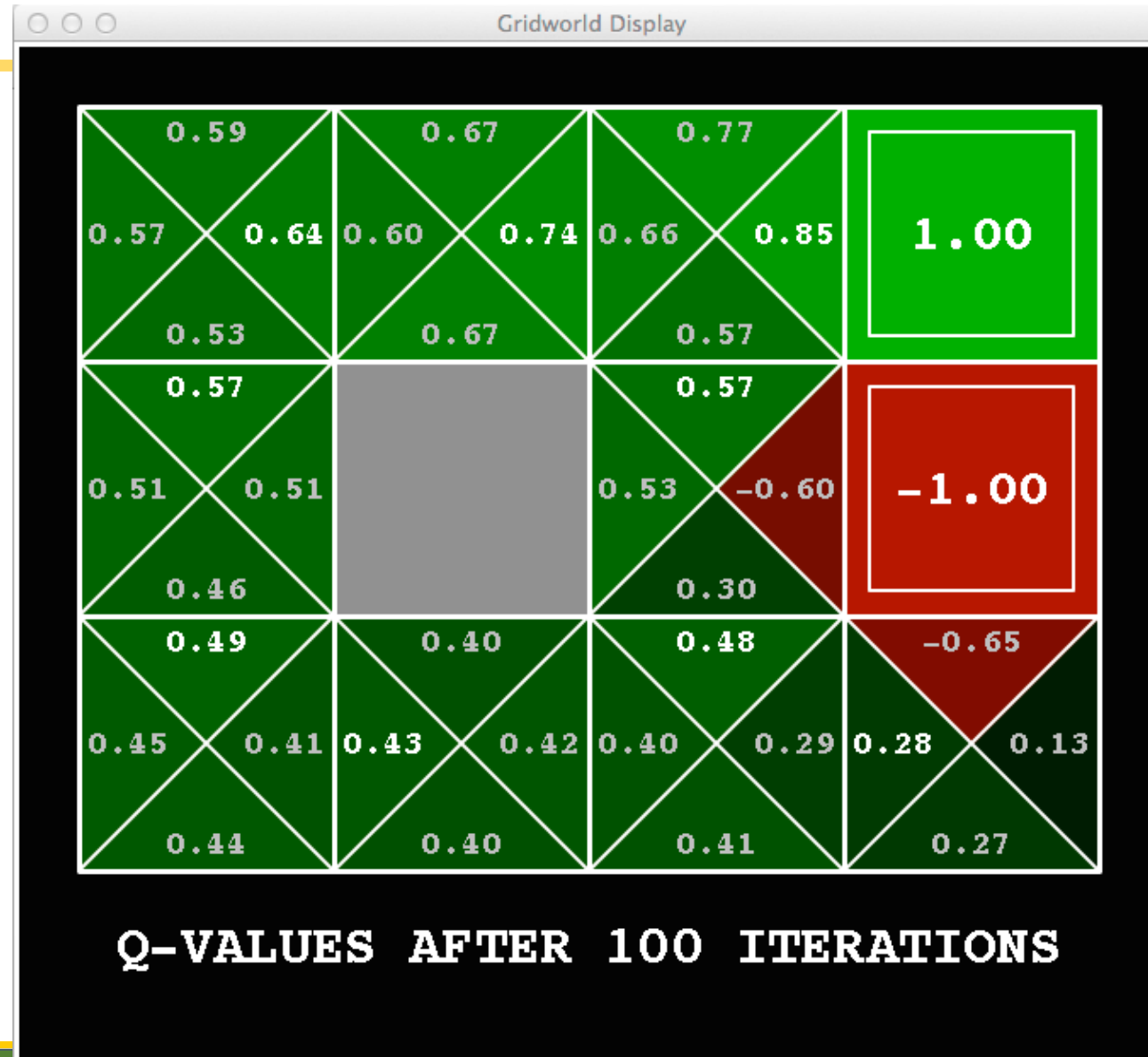
(s,a,s') is a *transition*
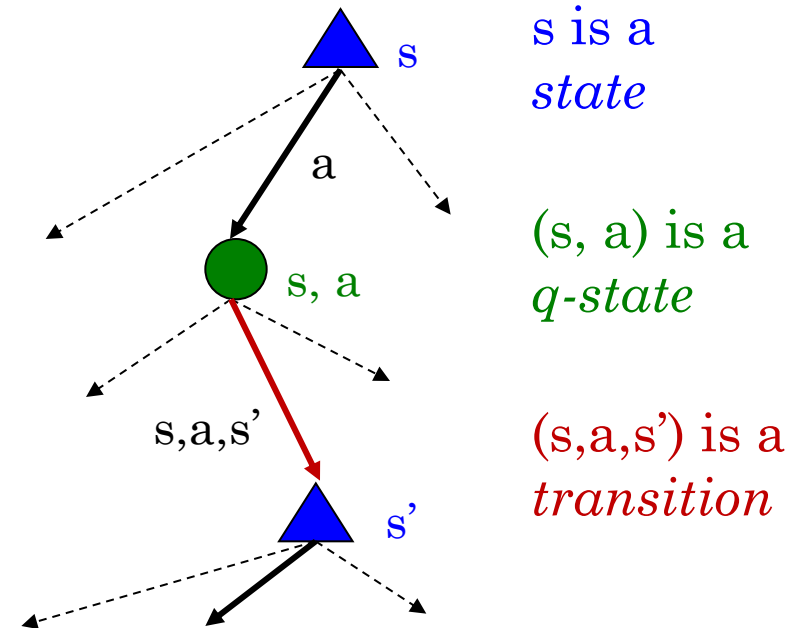
Snapshot of Demo – Gridworld V Values

Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo – Gridworld Q Values

Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!

- Recursive definition of value:

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

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

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