Artificial Intelligence Foundations and Applications

Partha Pratim Chakrabarti
Plaban Bhowmik
Sudeshna Sarkar
Centre of Excellence in Artificial Intelligence
IIT Kharagpur

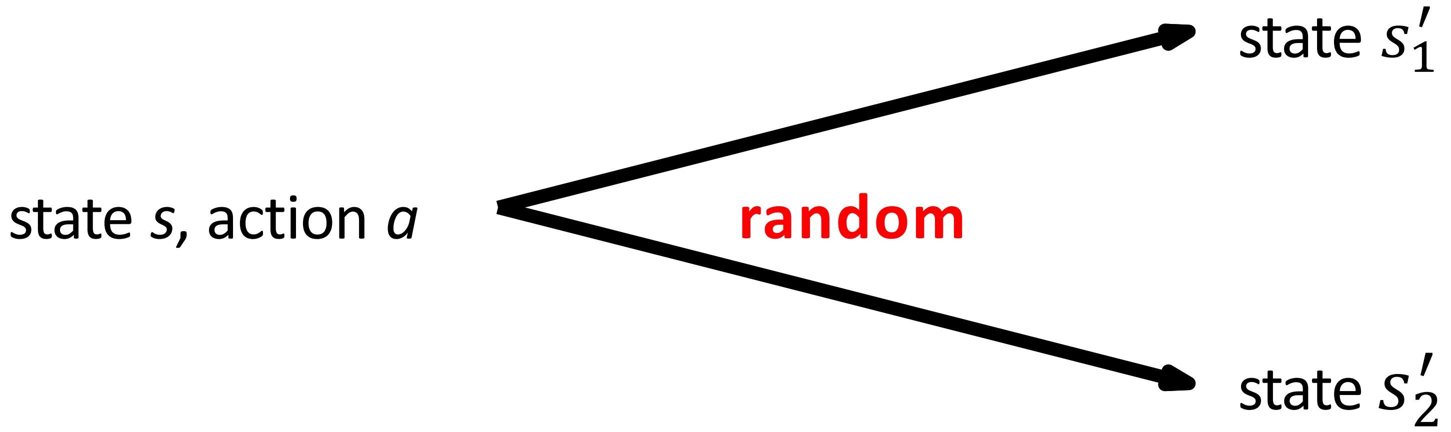
Planning

Stochastic Planning

MDP

Many slides adapted from
CS 188:University of California, Berkeley by Pieter Abbeel
CS221 : Stanford University by Percy Liang

Uncertainty in the real world



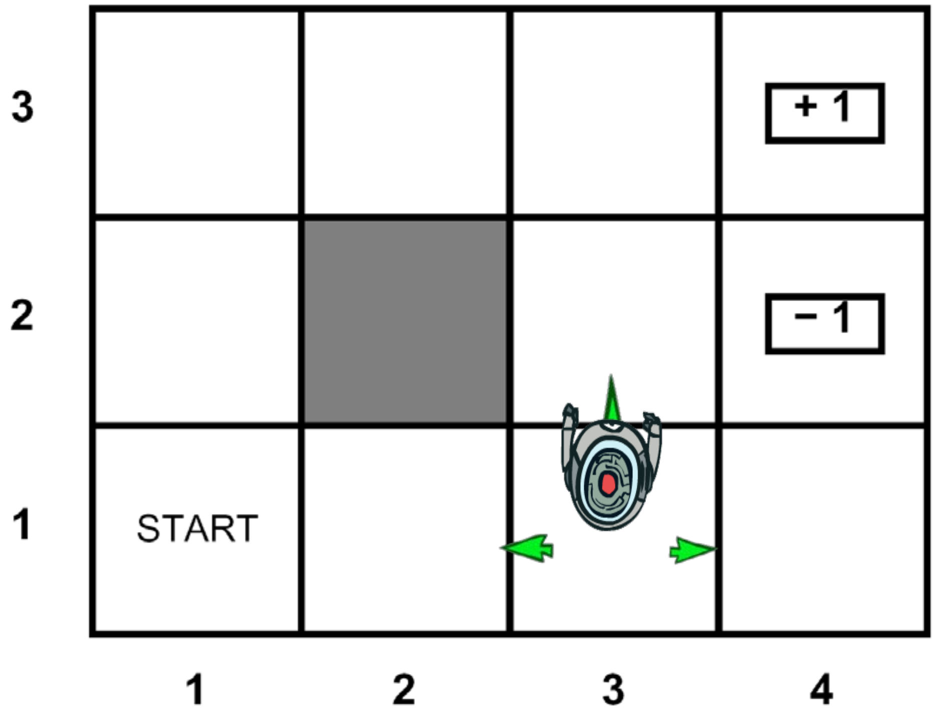
Randomness shows up in many places.

- caused by limitations of the sensors and actuators of the robot
- caused by market forces or nature

Example: Grid World

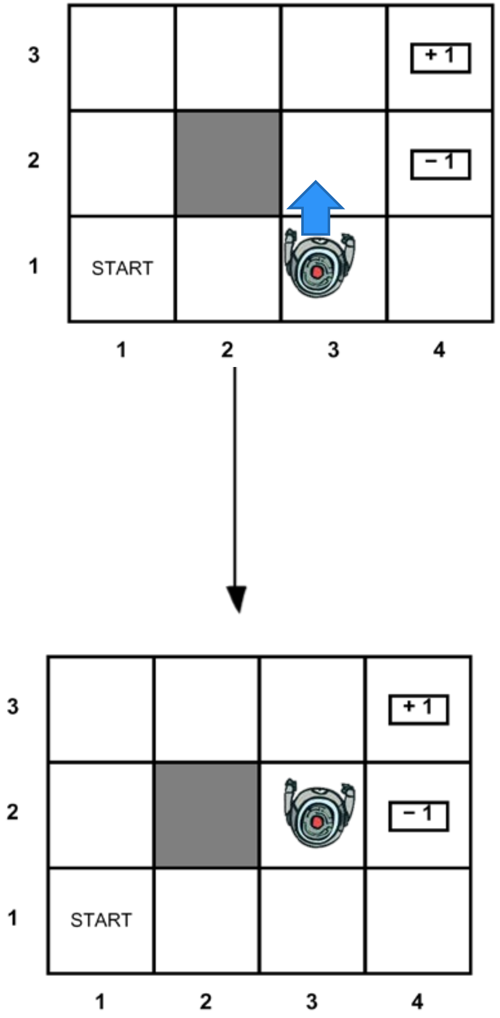
Noisy movement: actions do not always go as planned

- 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



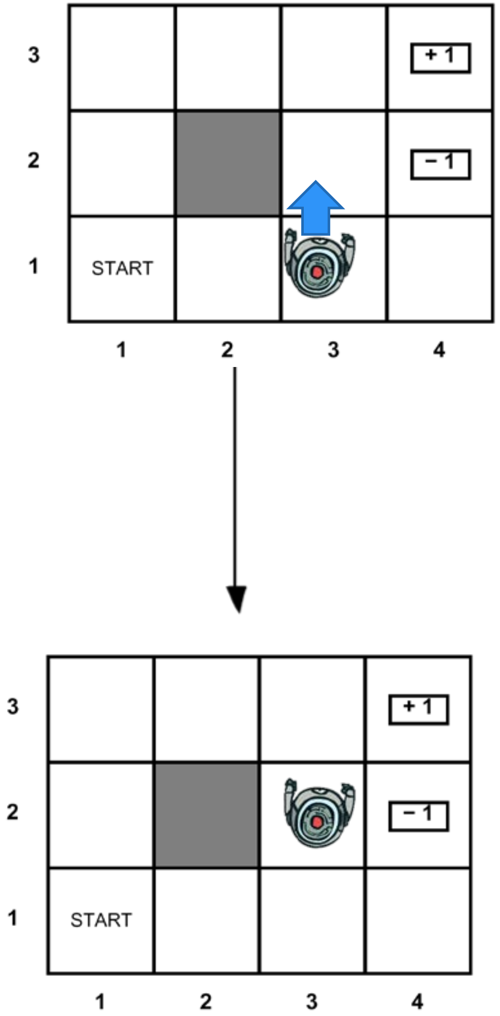
Grid World Actions

Deterministic Grid World

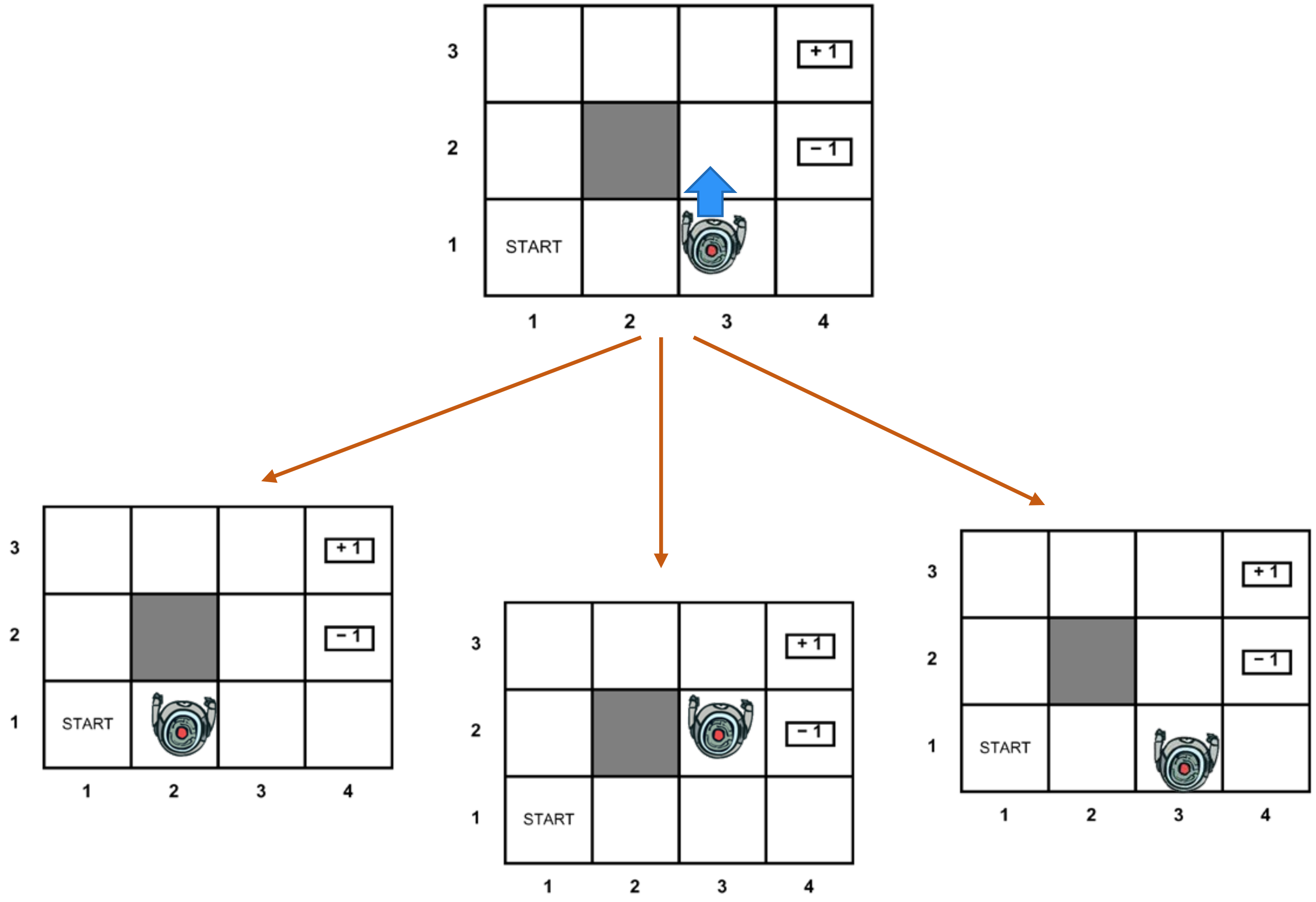


Grid World Actions

Deterministic Grid World



Stochastic Grid World

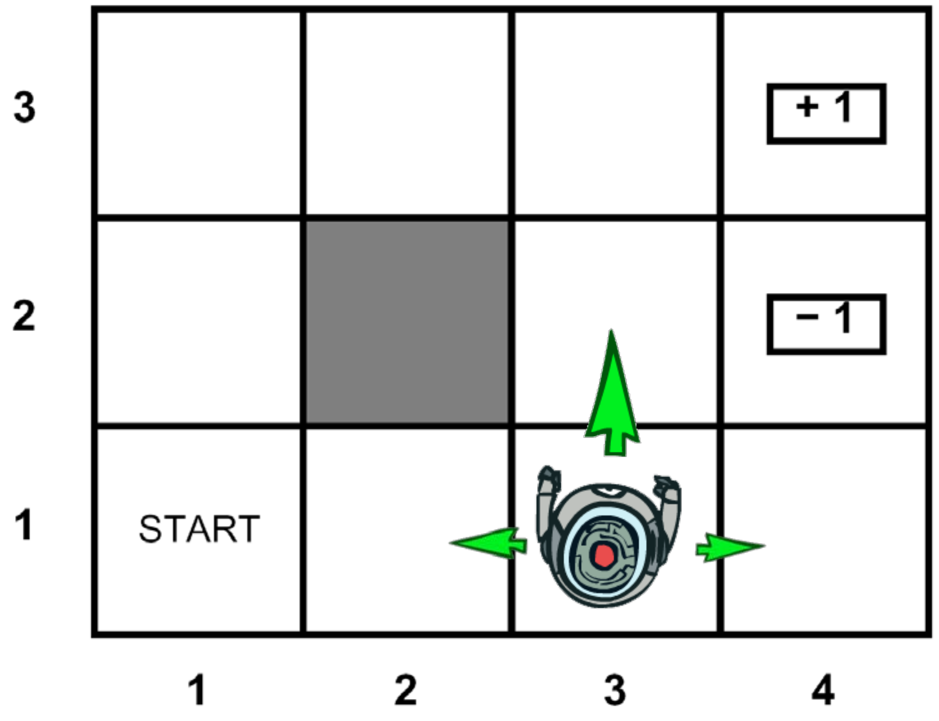


Markov Decision Processes

An MDP is defined by:

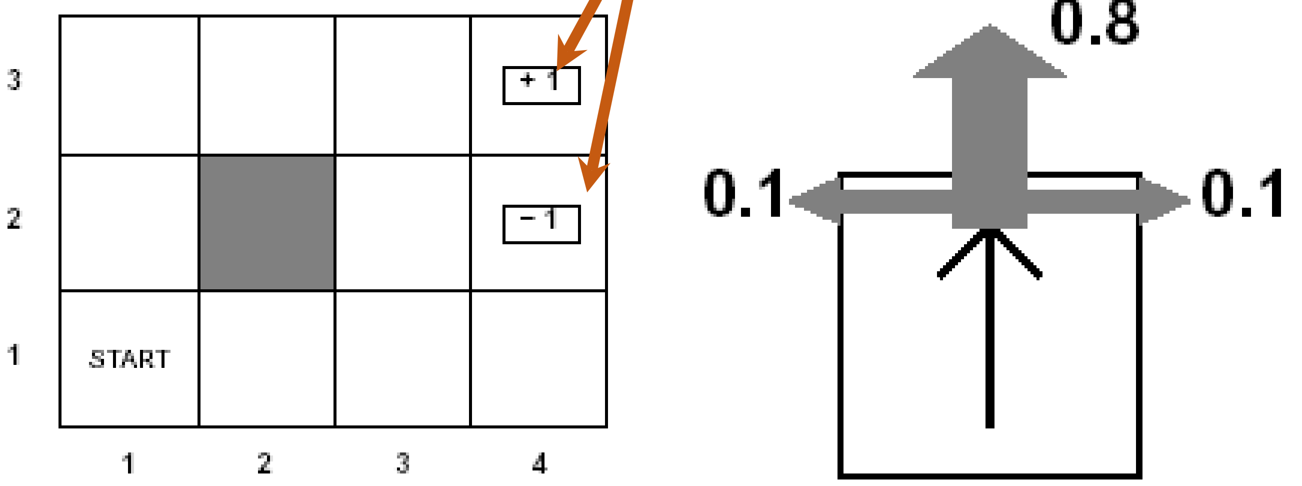
- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function $T(s, a, s')$ or $P(s'|s,a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - Sometimes just $R(s)$ or $R(s')$
- A start state
- Maybe a terminal state

Discount factor γ $0 \le \gamma \le 1$



[Demo – gridworld manual intro (L8D1)]

Example



$$R(s) = \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

What is Markov about MDPs?

- “Markov” : given the present state, the future and the past are independent
- 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)

Transportation example

Street with blocks numbered 1 to n.

Walking from s to $s + 1$ takes 1 minute.

Taking a magic tram from s to $2s$ takes 2 minutes. How to travel from 1 to n in the least time?

Tram fails with probability 0.5.

What are the states, actions, transition distribution, and rewards?

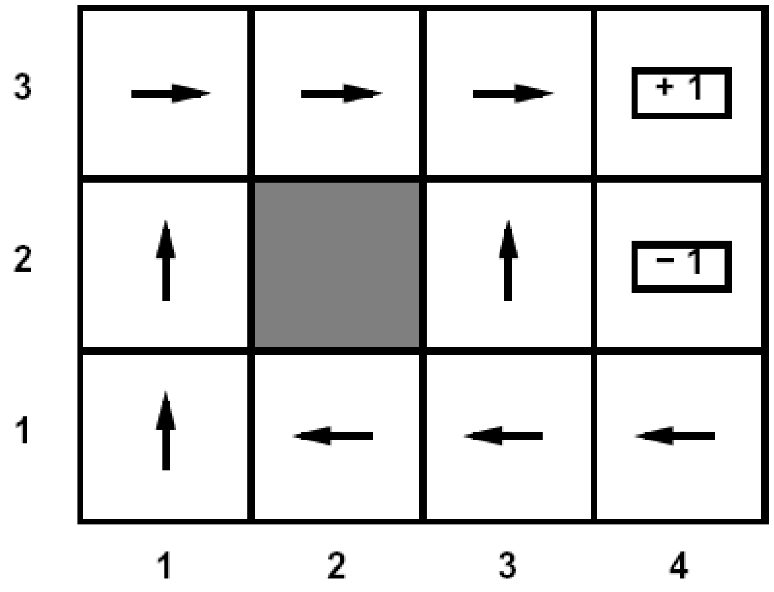
Policies

- 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

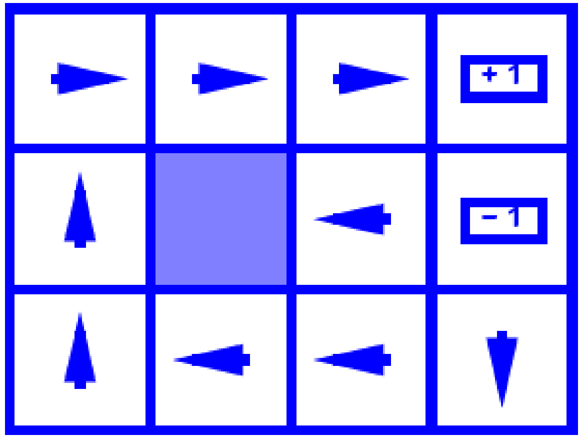
  policy $\pi^*: S \rightarrow A$

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

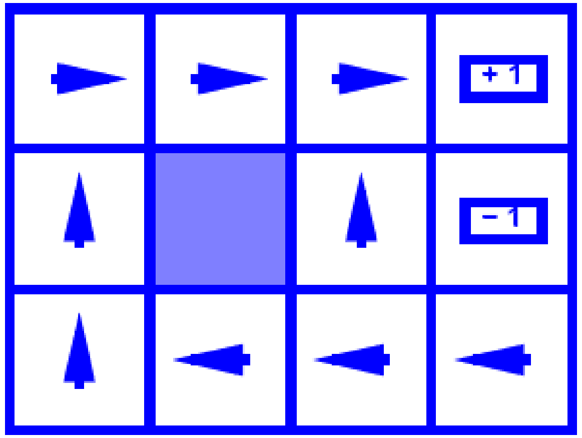


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

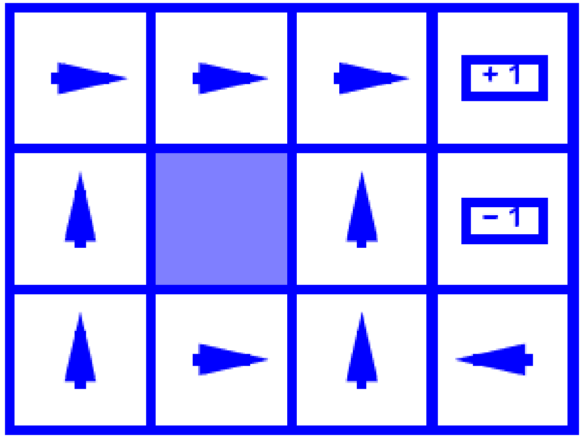
Optimal Policies



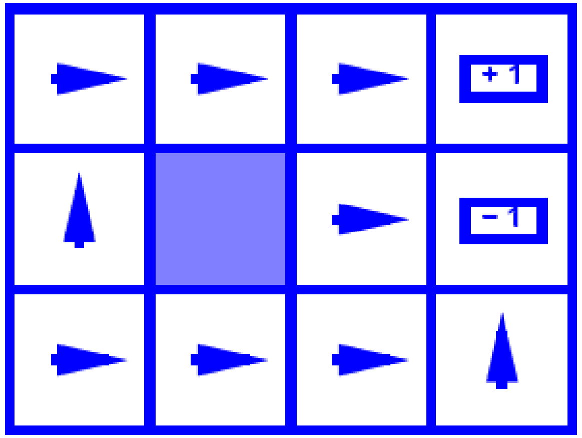
R(s) = -0.01



R(s) = -0.03



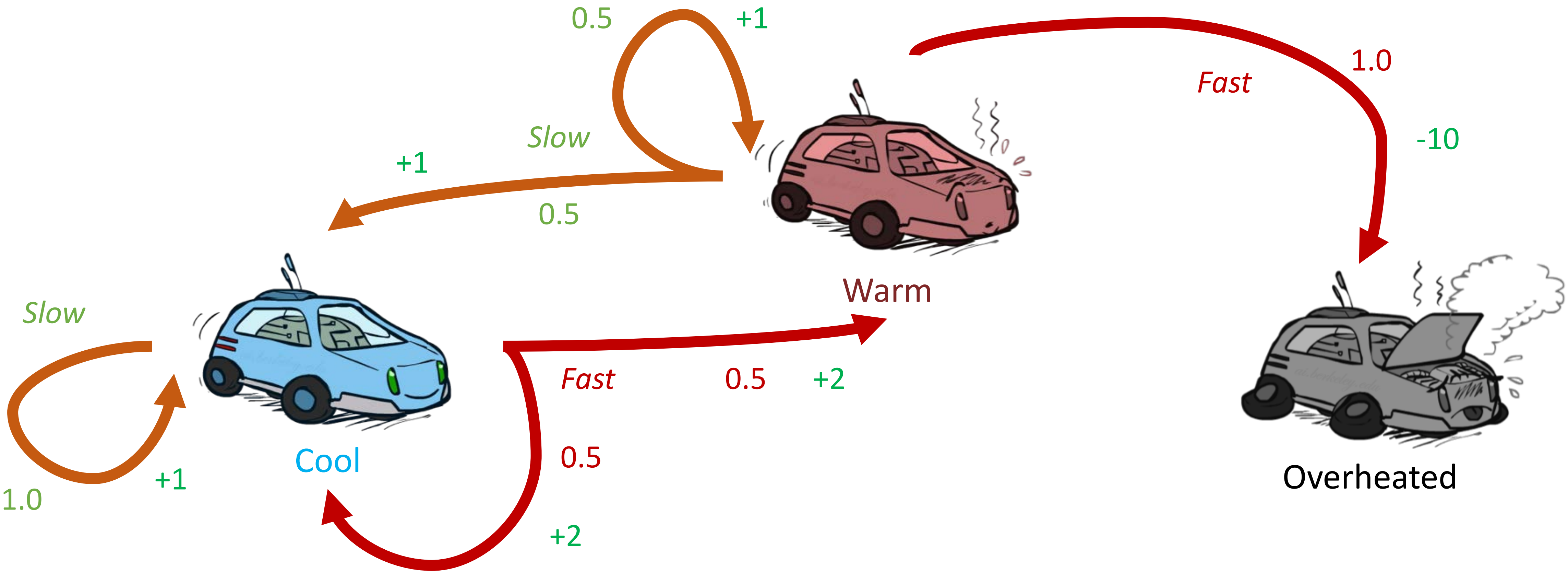
R(s) = -0.4



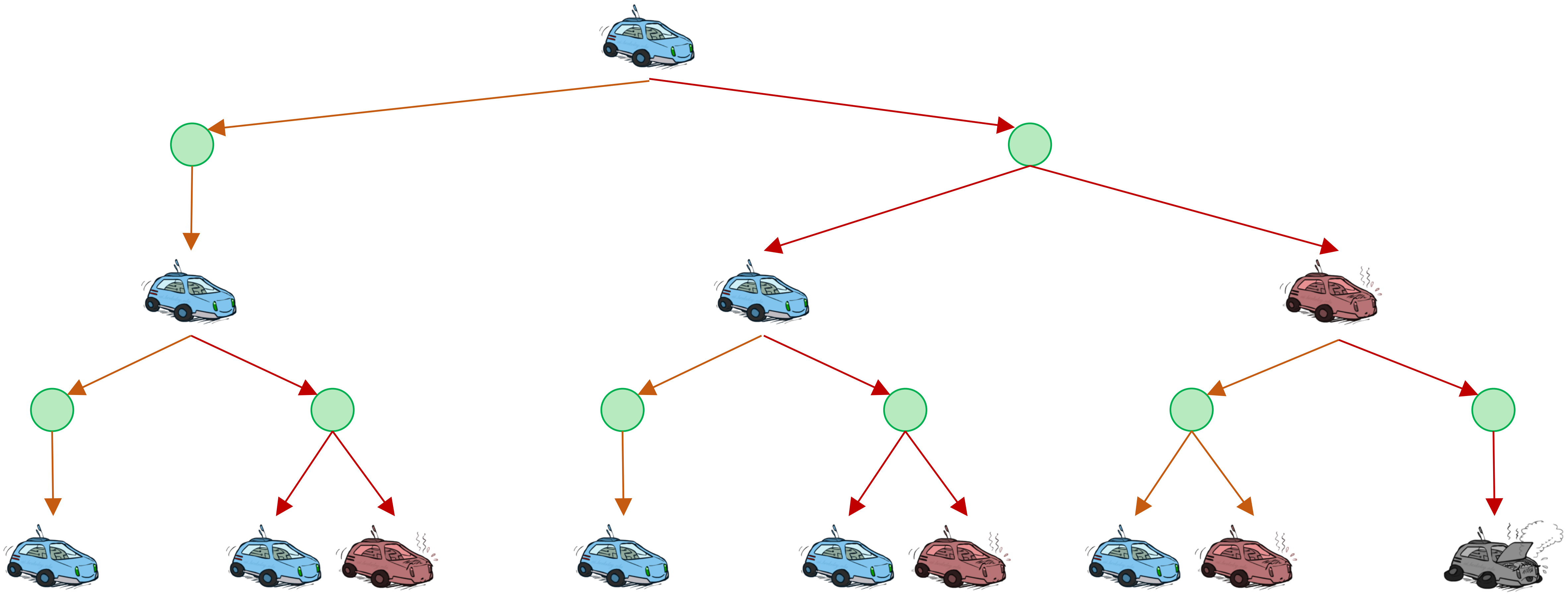
R(s) = -2.0

Example: Racing

- 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



Evaluating a policy

Definition: utility

- Following a policy yields a **random path**.
- The **utility** of a policy is the (discounted) sum of the rewards on the path (this is a random variable).

Path	Utility
[in; stay, 4, end]	4
[in; stay, 4, in; stay, 4, in; stay, 4, end]	12
in; stay, 4, in; stay, 4, end]	8
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	16
...	...

Definition: value (expected utility)

The **value** of a policy at a state is the **expected** utility.

Utilities

Two ways to define utilities

- Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$

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

Discounting

- **Definition: utility**

$s_0, a_1\, r_1\, s_1, a_2\, r_2\, s_2, \ldots$ (action, reward, new state).

The **utility** with discount γ is

$$u_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$$

Discount $\gamma = 1$

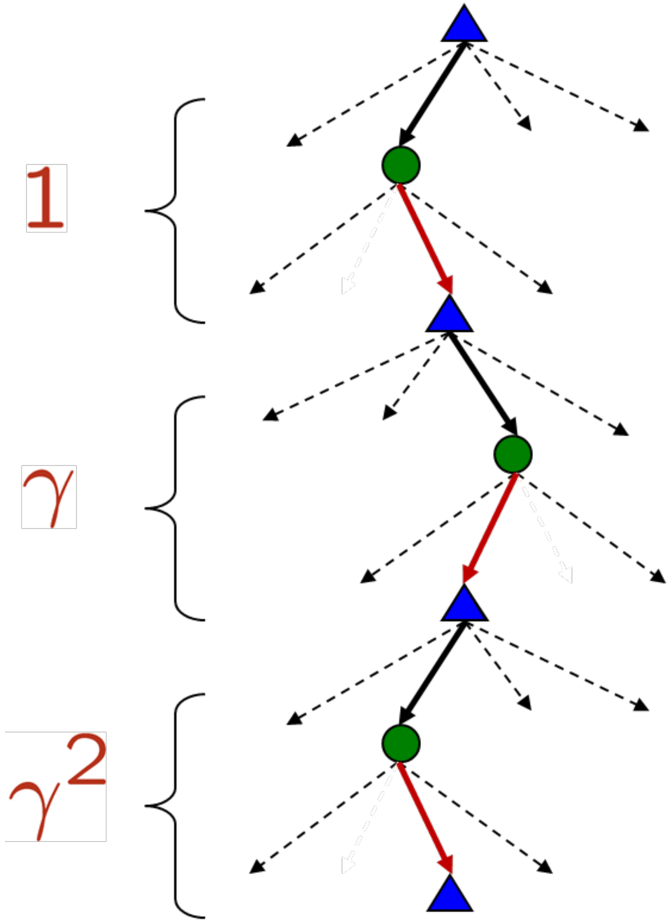
[stay, stay, stay, stay]: 4 + 4 + 4 + 4 = 16

Discount $\gamma = 0$

[stay, stay, stay, stay]: $4 + 0 \cdot (4 + \cdots) = 4$

Discount $\gamma = 0.5$

4+(1/2).4+((1/4).4+(1/8).4=7.5



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 $0 < \gamma < 1$

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

  - 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)

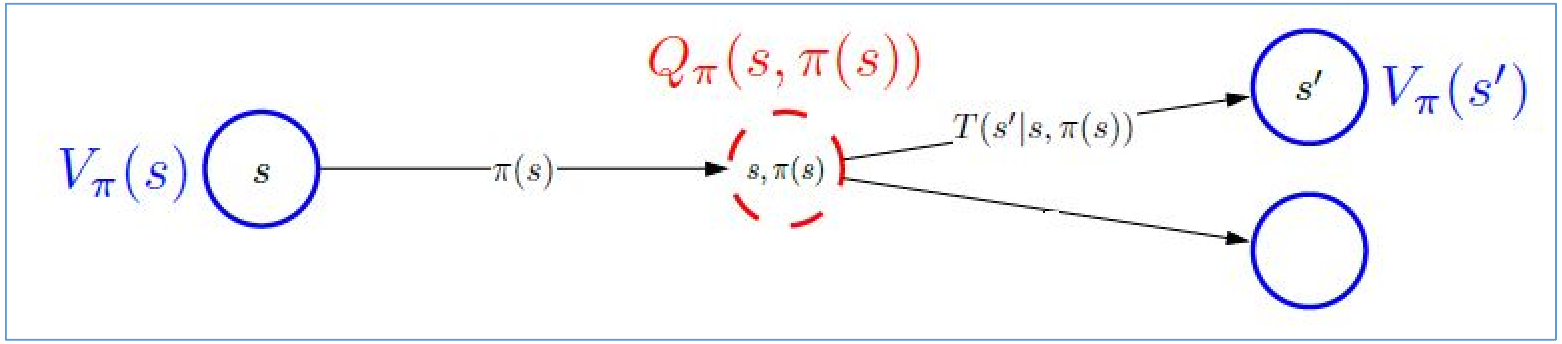
Policy evaluation

Definition: value of a policy

- Let $V_\pi(s)$ be the expected utility received by following policy π from state s.

Definition: Q-value of a policy

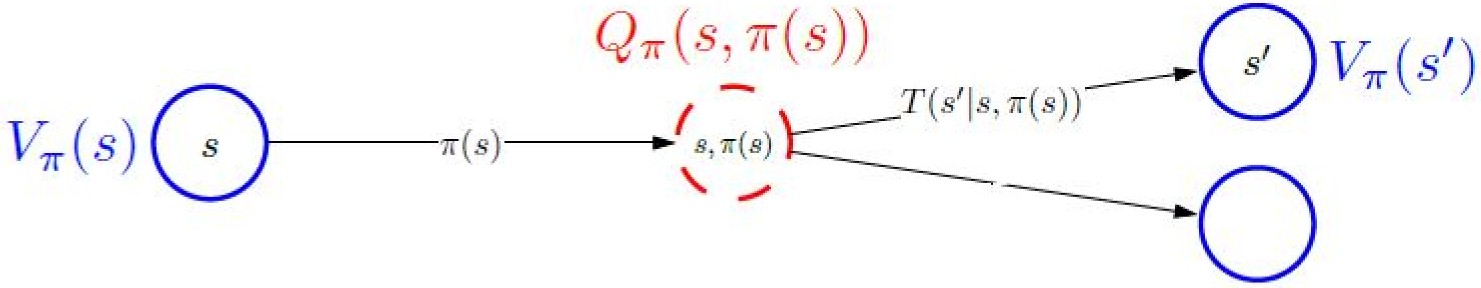
- Let $Q_\pi(s, a)$ be the expected utility of taking action a from state s, and then following policy π.



Policy Evaluation : Recurrences

- Value function for a policy $\pi: S \rightarrow A$

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} T(s' | s, \pi(s)\,)\, V^{\pi}(s')$$



$$V^{\pi}(s) = \begin{cases} 0 \text{ if IsEnd}(s) \\ Q^{\pi}(s, \pi(s)) \end{cases}$$

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

Optimal value and policy

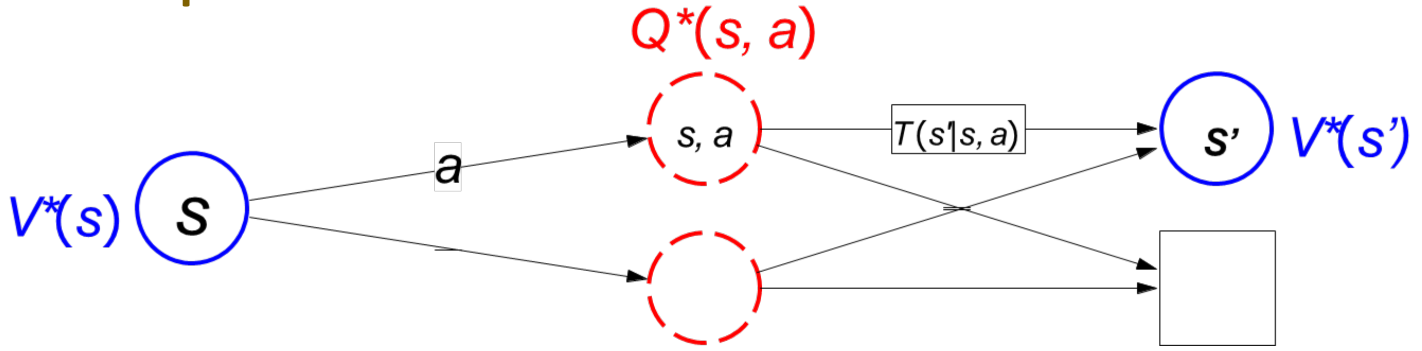
Goal: find policy that maximizes expected utility

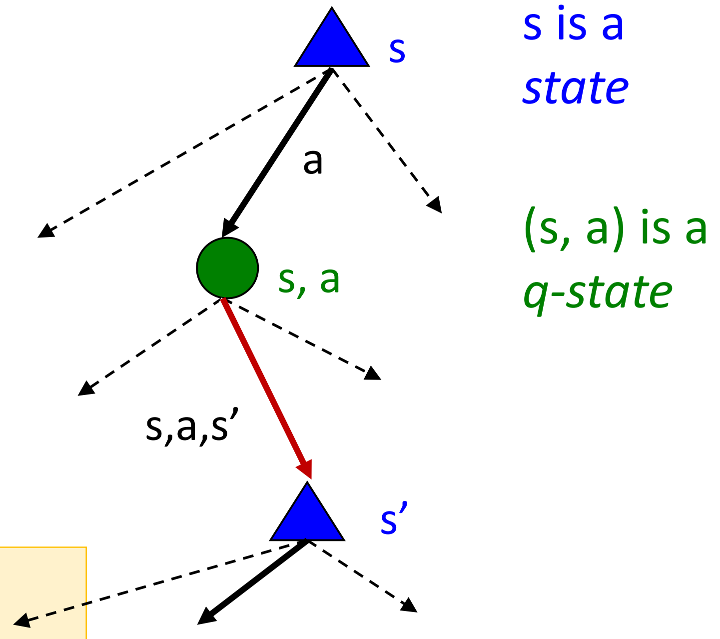
Optimal Value: The optimal value $V^*(s)$ is the maximum value attained by any policy, starting from state s.

Optimum value function

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Optimal value





The value (utility) of a state s:

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

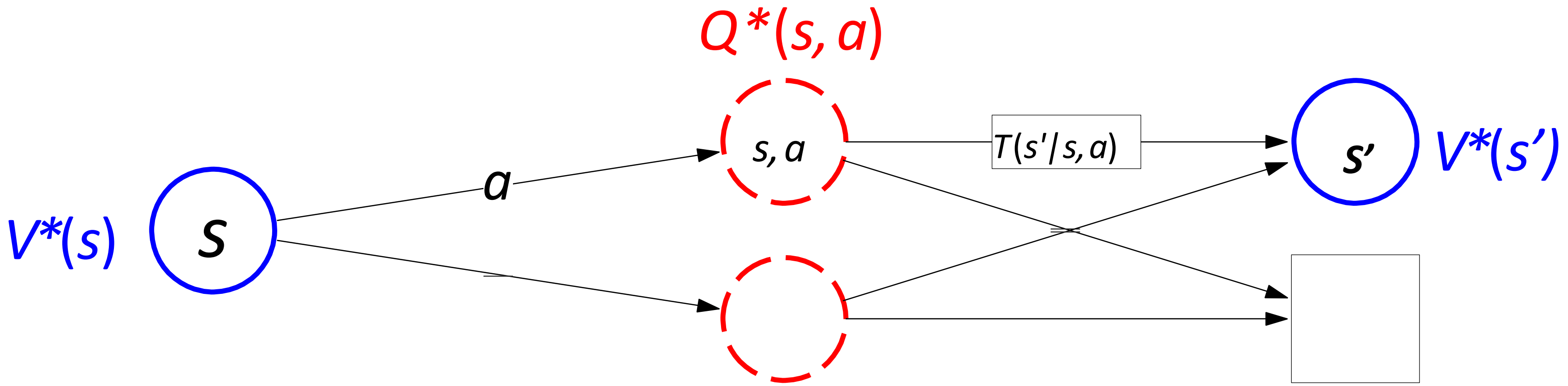
The value (utility) of a q-state (s,a):

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

The optimal policy:

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

Optimal value



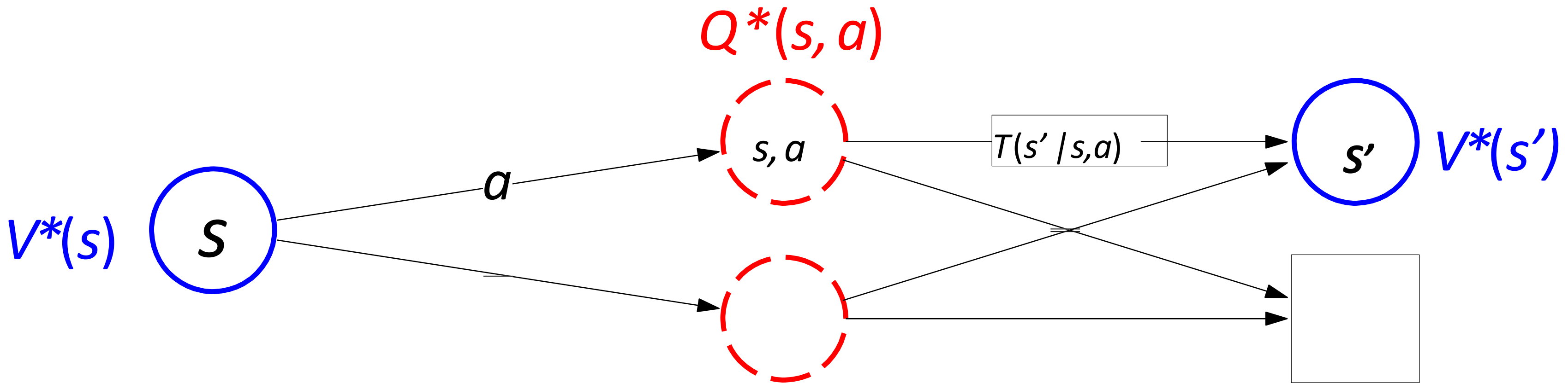
- Optimal value if take action *a* in state *s*:

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

Optimal value from state *s*:

$$V^*(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \max_a Q^*(s,a) & \text{otherwise} \end{cases}$$

How to get the optimal policy?



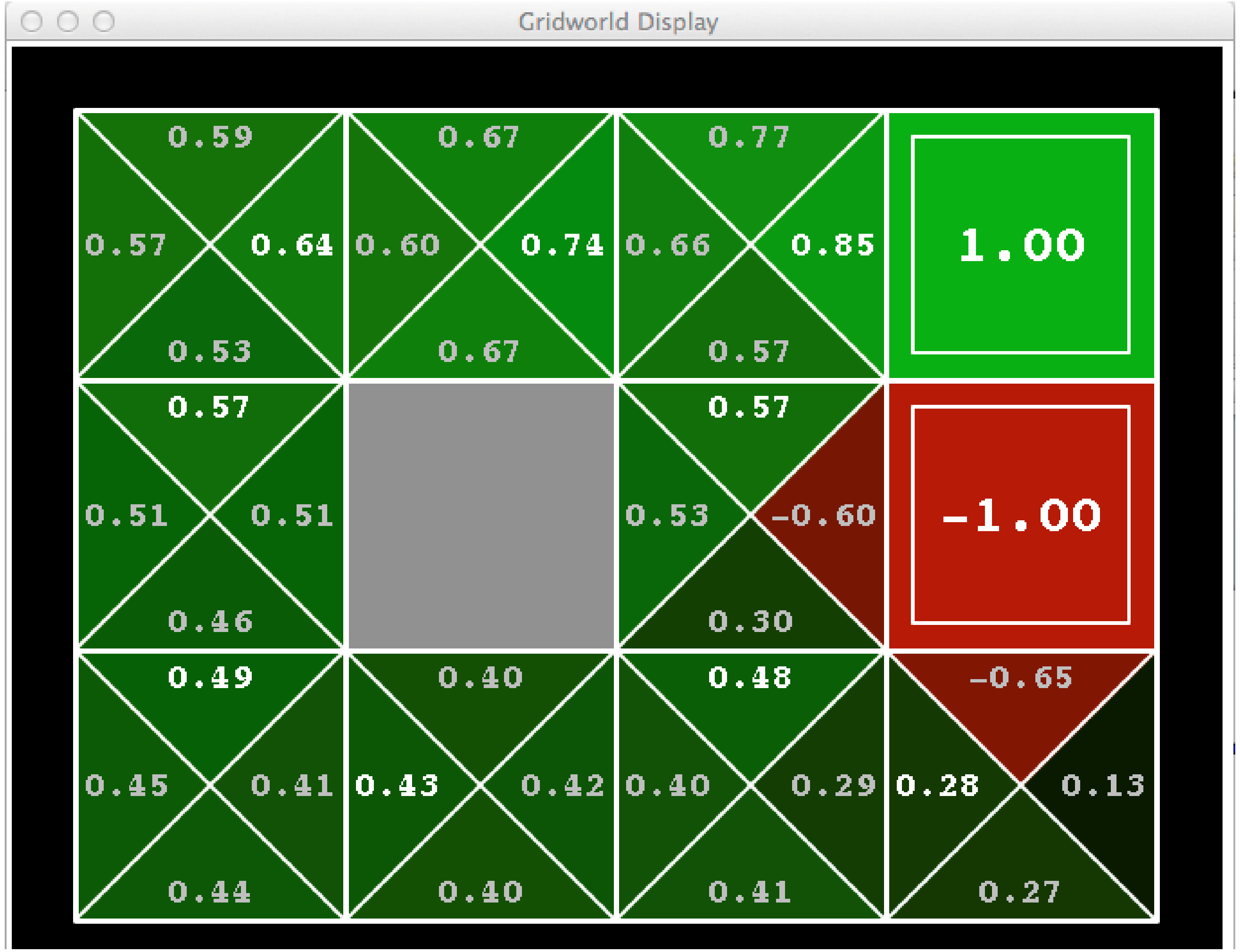
- Given Q*, read off the optimal policy:

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

Gridworld V* Values

Noise = 0.2
Discount = 0.9
Living reward = 0

Gridworld Q* Values

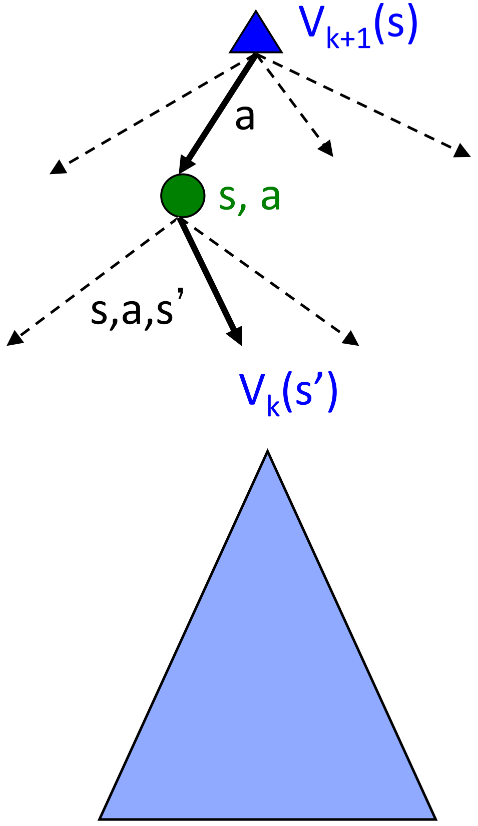


Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

1. For each state s, initialize $V(s)$:= 0.
2. **for** until convergence **do**
3. For every state, update

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



- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Example: Value Iteration



V_2

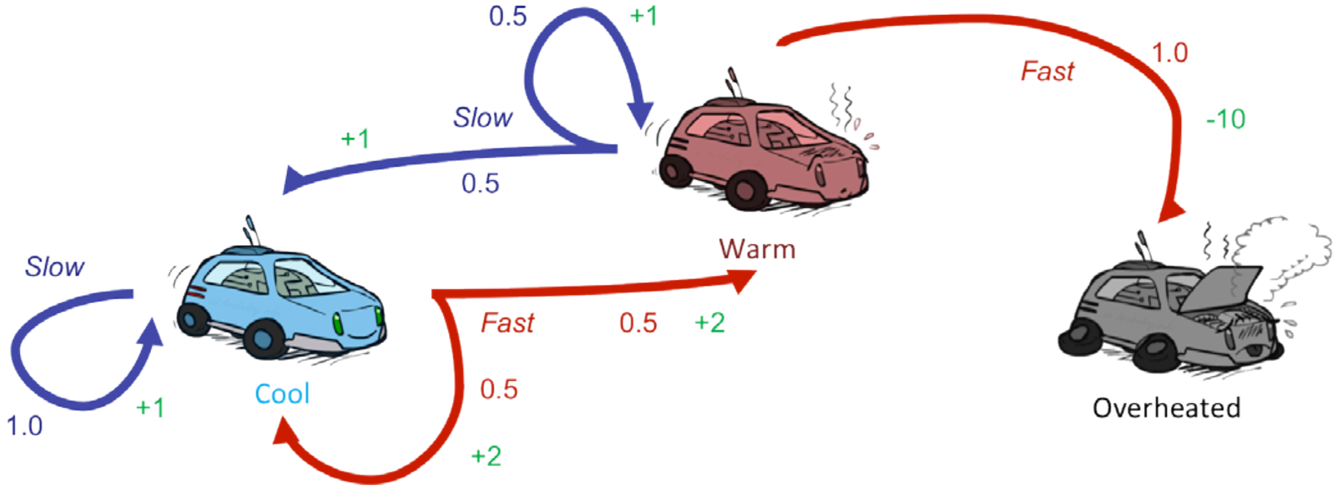
V_1

S: 1

F: .5*2+.5*2=2

V_0

0	0	0

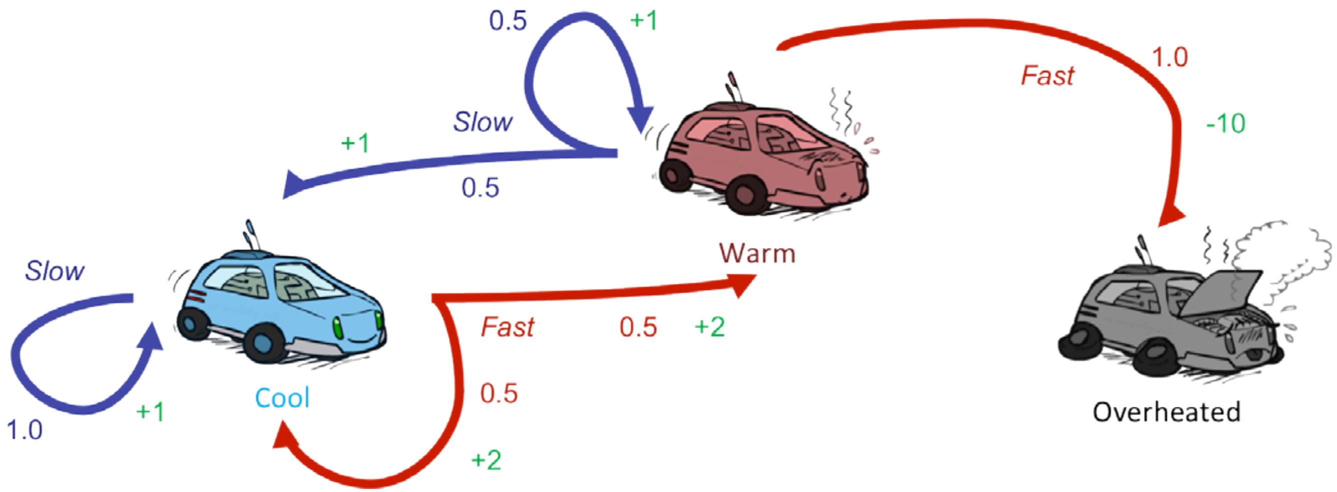


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	S: .5*1+.5*1=1 F: -10	
V_0	0	0	0

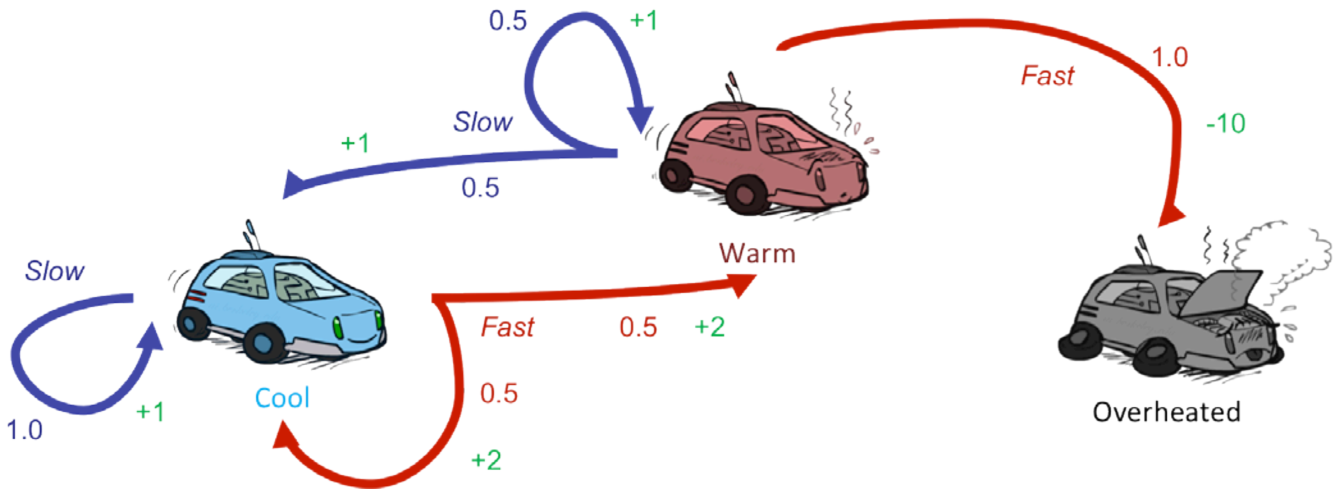


Assume no discount!

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

Example: Value Iteration

V_2			
V_1	2	1	0
V_0	0	0	0

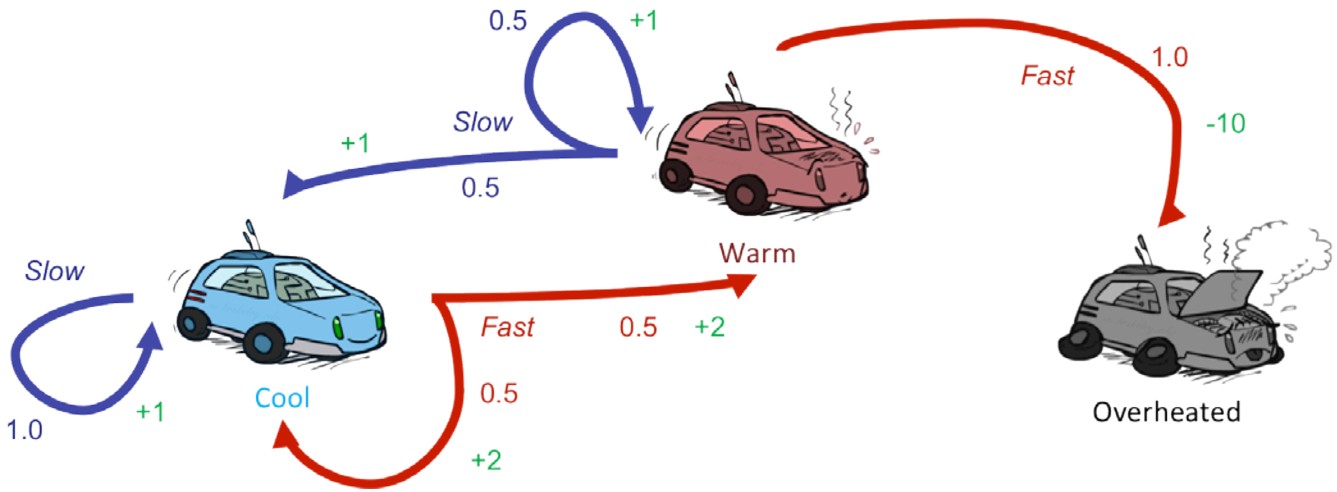


Assume no discount!

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

Example: Value Iteration

V_2	S: 1+2=3 F: .5*(2+2)+.5*(2+1)=3.5		
V_1	2	1	0
V_0	0	0	0

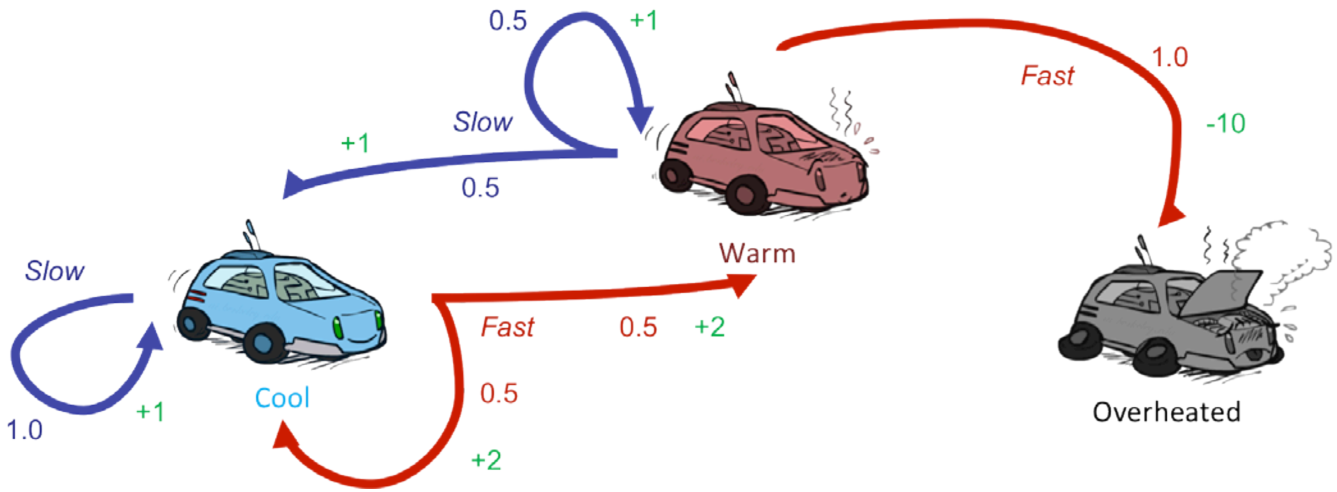


Assume no discount!

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

Example: Value Iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

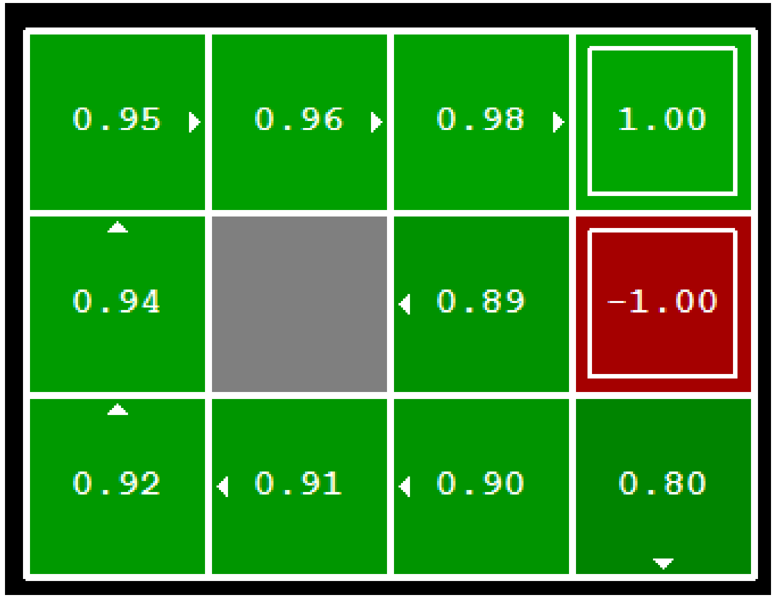
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)

- How should we act?

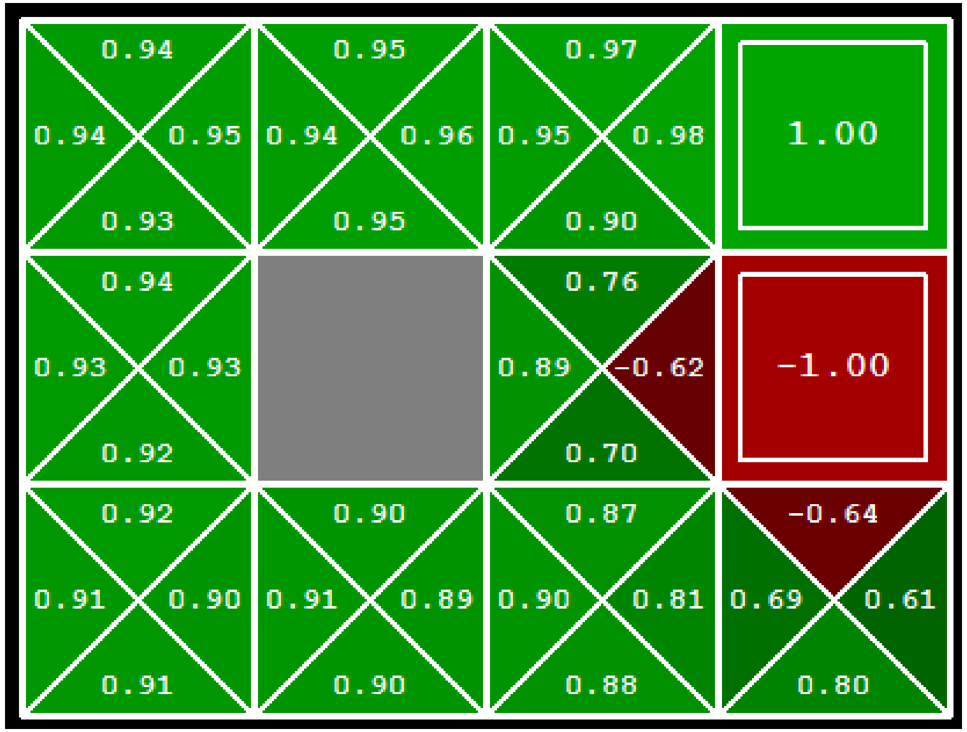
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$



- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

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

- Important lesson: actions are easier to select from q-values than values!

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy Evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy Improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values

Repeat steps until policy converges

Policy Iteration

Policy Evaluation: For fixed current policy π, find values with policy evaluation:

- Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma\, V_k^{\pi_i}(s')\right]$$

Policy Improvement: For fixed values, get a better policy using policy extraction

- One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don’t track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we’re done)
- Both are dynamic programs for solving MDPs