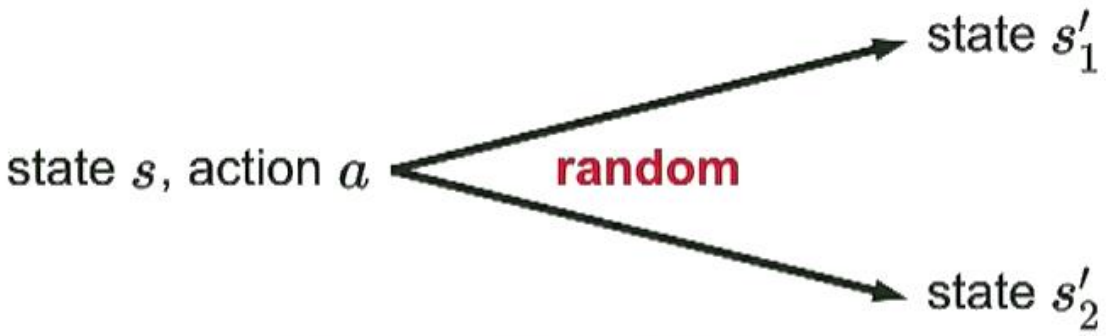
Markov Decision Processes

Deterministic Search

state s, action a —**deterministic**→ state $\mathrm{Succ}(s, a)$

Non-Deterministic Search

Uncertainty in the real world



Non-Deterministic Search

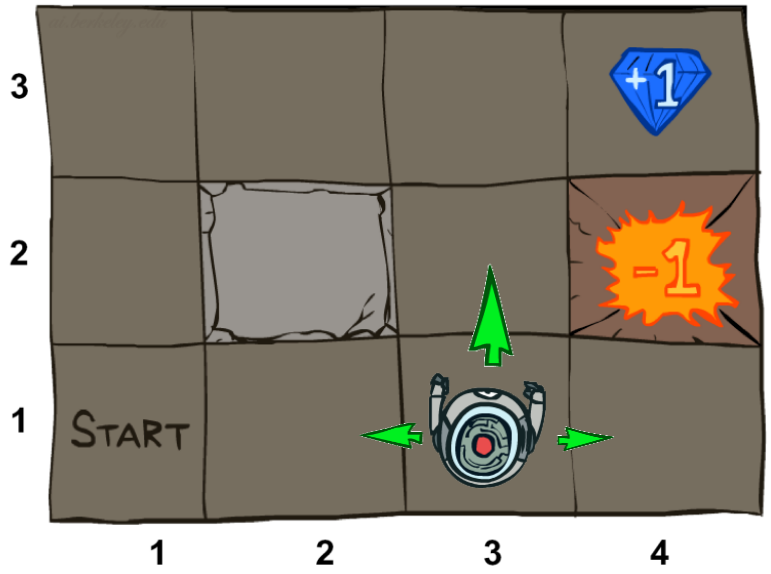
Robotics: decide where to move, but actuators can fail, hit unseen obstacles, etc.

Resource allocation: decide what to produce, don't know the customer demand for various products

Agriculture: decide what to plant, but don't know weather and thus crop yield

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.

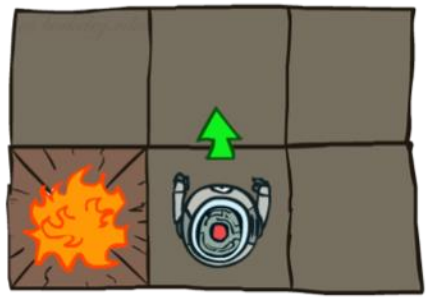


Example: Grid World

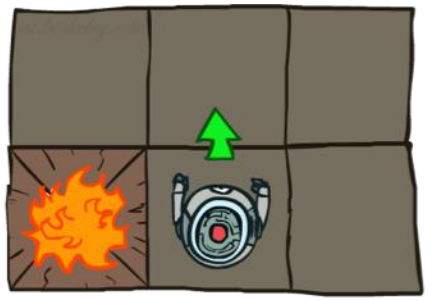
- The agent receives rewards each time step
  - ✓ Small “living” reward each step (can be negative)
  - ✓ Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards

Grid World Actions

Deterministic Grid World

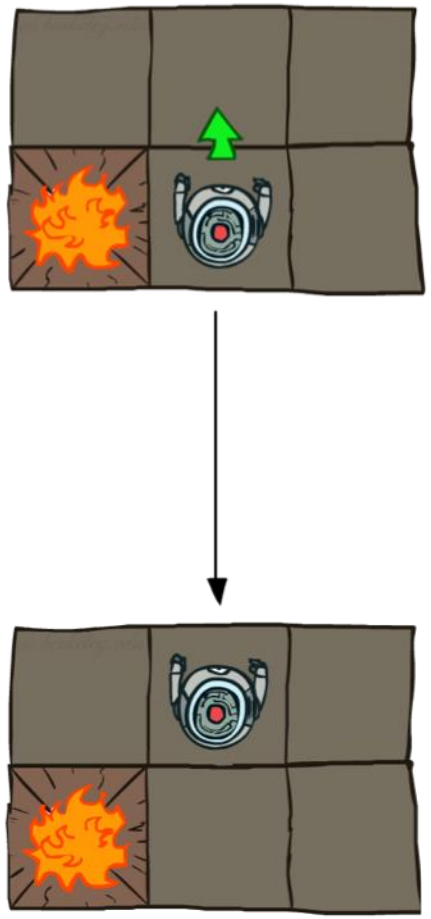


Stochastic Grid World

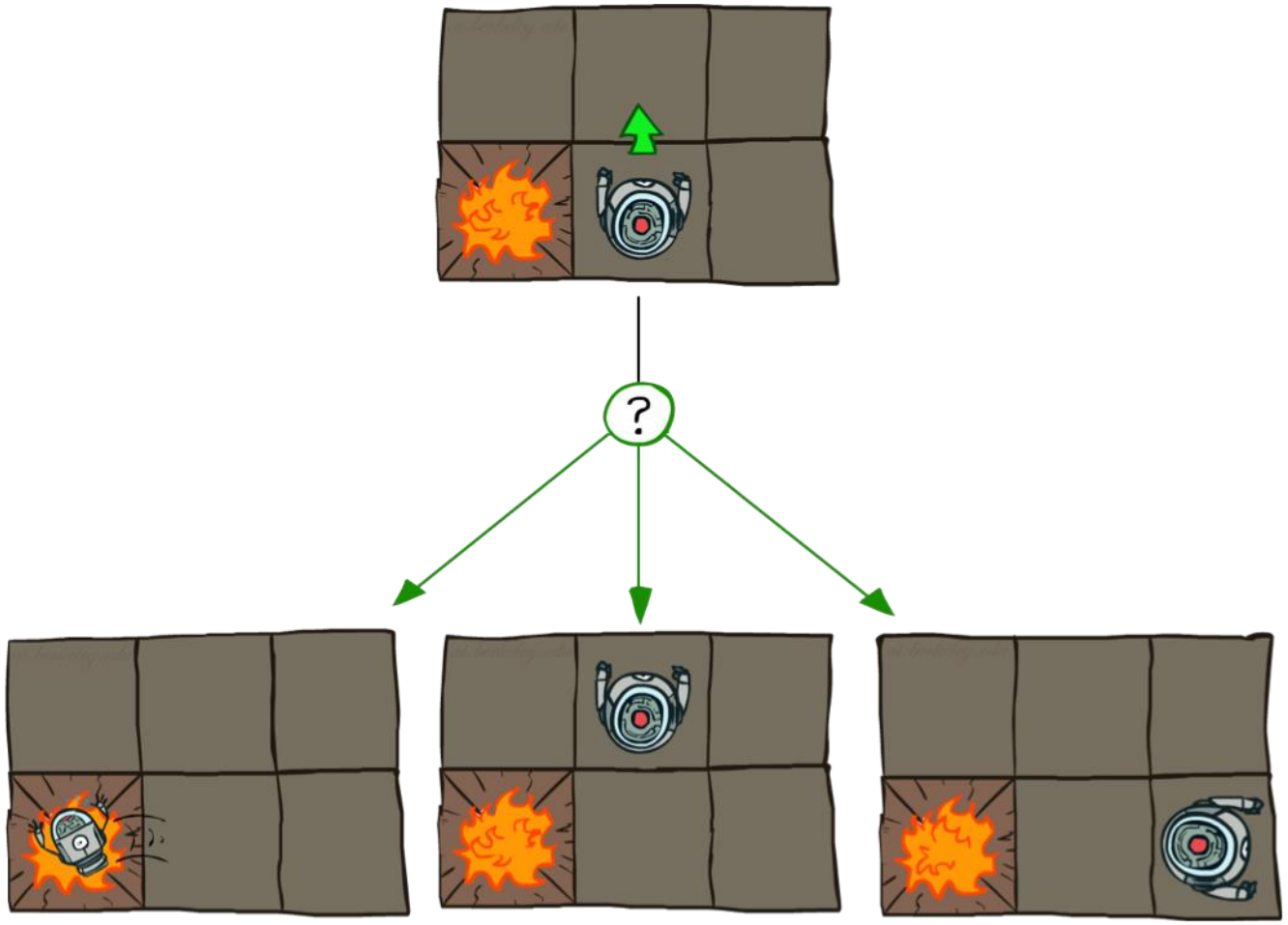


Grid World Actions

Deterministic Grid World



Stochastic Grid World



Markov Decision Processes

An MDP is defined by:

- A set of states s ϵ S
- A set of actions a ϵ 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

Markov Decision Processes

- MDPs are non-deterministic search problems
  - ✓ One way to solve them is with expectimax search
  - ✓ Others ways we will see later

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

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

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

Search vs. Markov Decision Processes

Definition: Markov decision process

States: the set of states
$s_{\text{start}} \in \text{States}$: starting state
$\text{Actions}(s)$: possible actions from state s
$T(s, a, s')$: probability of s' if take action a in state s
$\text{Reward}(s, a, s')$: reward for the transition (s, a, s')
$\text{IsEnd}(s)$: whether at end of game
$0 \leq \gamma \leq 1$: discount factor (default: 1)

Definition: search problem

States: the set of states
$s_{\text{start}} \in \text{States}$: starting state
$\text{Actions}(s)$: possible actions from state s
$\text{Succ}(s, a)$: where we end up if take action a in state s
$\text{Cost}(s, a)$: cost for taking action a in state s
$\text{IsEnd}(s)$: whether at end

- $\text{Succ}(s, a) \Rightarrow T(s, a, s')$
- $\text{Cost}(s, a) \Rightarrow \text{Reward}(s, a, s')$

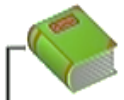
Policies: a solution to MDP

- 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



Definition: policy

A **policy** π is a mapping from each state $s \in \text{States}$ to an action $a \in \text{Actions}(s)$.

Policies: quiz

CQ: If the environment is deterministic, an optimal solution to the grid world problem is the fixed action sequence:
down, down, right, right, and right.

(A) True

(B) False

(C) I don't know

	1	2	3	4
1	Start			
2		X		-1
3				+1

Policies: quiz

CQ: Consider the action sequence "down, down, right, right, and right". This action sequence could take the robot to more than one square with positive probability.

(A) True

(B) False

(C) I don't know

	1	2	3	4
1	Start			
2		X		-1
3				+1

Policies: quiz

CQ: Consider the action sequence "down, down, right, right, and right". This action sequence could take the robot to more than one square with positive probability.

(A) True

(B) False

(C) I don't know

	1	2	3	4
1	Start			
2		X		-1
3				+1

- The answer is true, because the action may or may not take the robot to +1, but will definitely take to more than more square. It can be any square.

Policies: quiz

The optimal policy of the grid world changes based on $R(s)$ for any non-goal state s. It shows a careful balancing of risk and reward.

	1	2	3	4
1	Start			
2		X		-1
3				+1

- Find the optimal policy when reward is zero.
- Find the optimal policy when reward is -0.1
- Find the optimal policy when reward is -10.

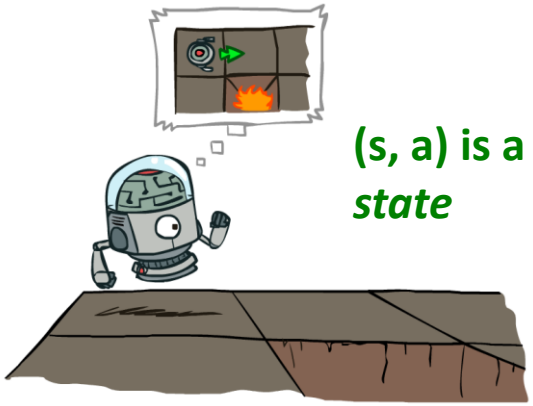
Policies: quiz

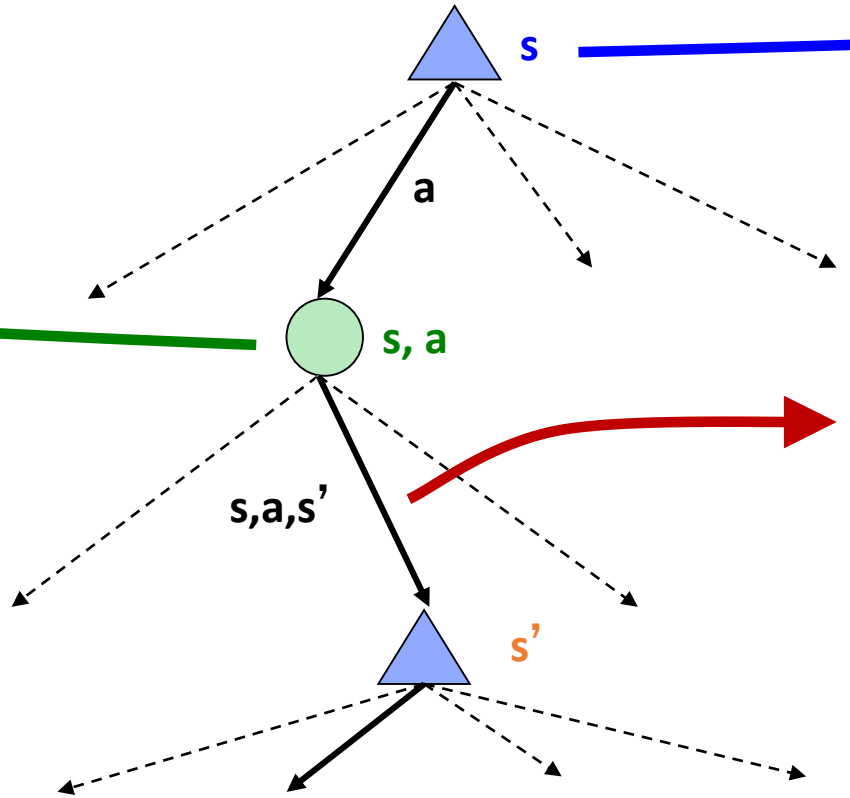
The optimal policy of the grid world changes based on $R(s)$ for any non-goal state s. It shows a careful balancing of risk and reward.

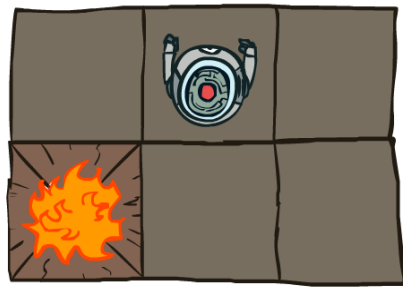
	1	2	3	4
1	Start			
2		X		-1
3				+1

- Find the optimal policy when reward is zero.
- Find the optimal policy when reward is -0.1
- Find the optimal policy when reward is -10.
- Find the optimal policy when reward is 10. The agent is happy for every move it take and never wanted to come out because the interstate reward is better then exit reward.

MDP Search Trees







s is a *state*

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

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

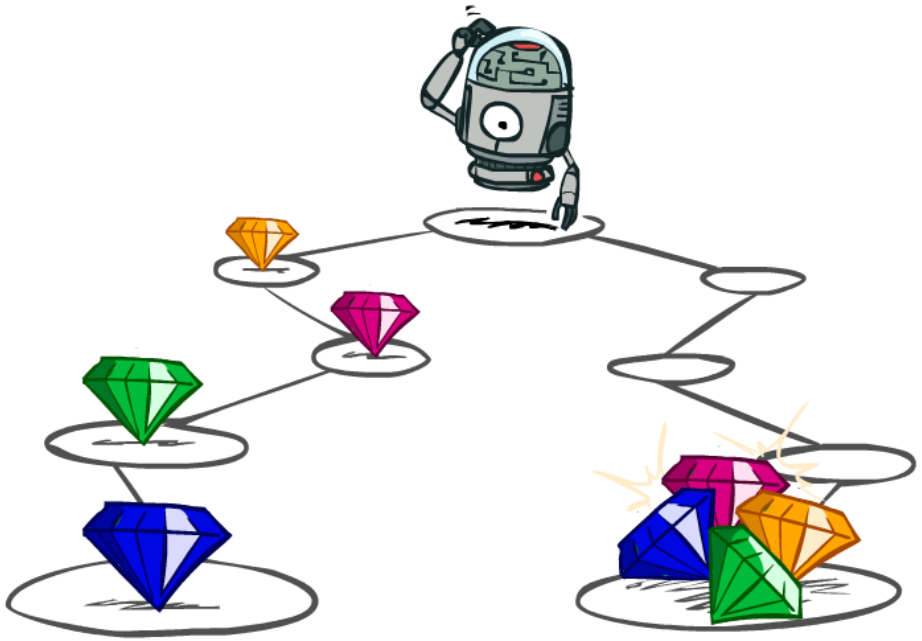
T(s,a,s') = P(s'|s,a)

R(s,a,s')

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] [1, 0, 0]

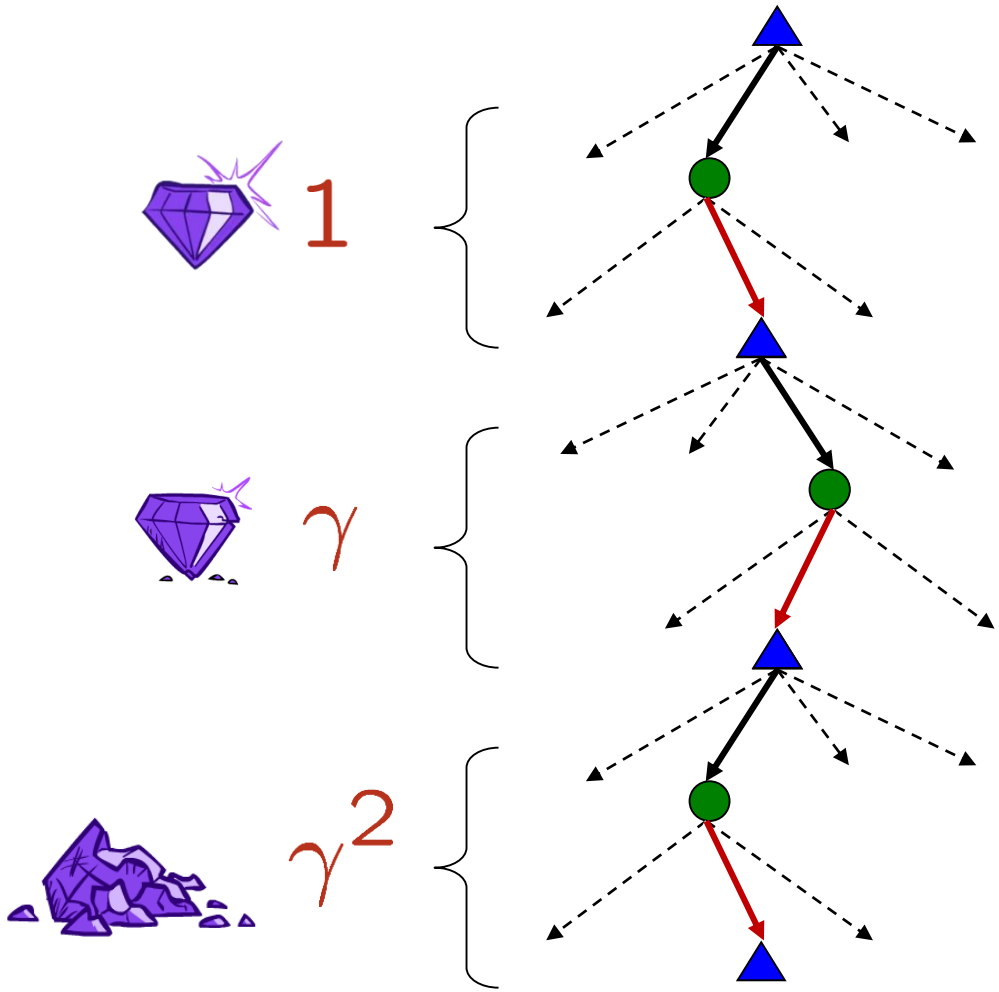


Discounting (gamma)

- For example:
  - ✓ 100 rupee now or 5 years later.
  - ✓ 100 rupee now or 0 later.
  - ✓ 100 rupee now or 120 rupee 2 years later
  - ✓ 100 rupee now or 500 rupee 5 years later

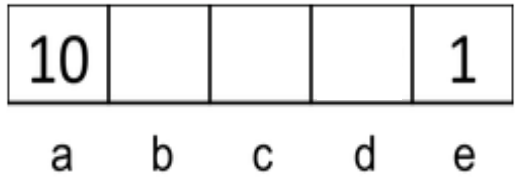
Discounting (gamma)

- 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])



Quiz: Discounting

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



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

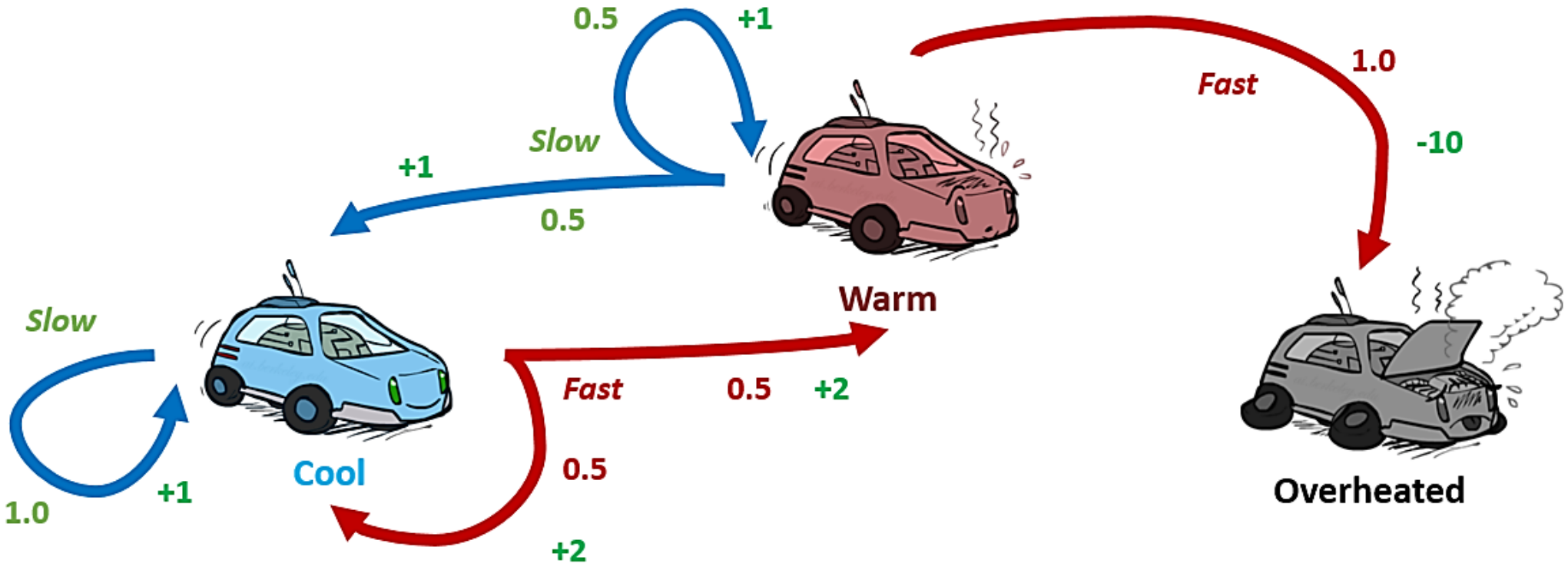
- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

Discounting example: Racing

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

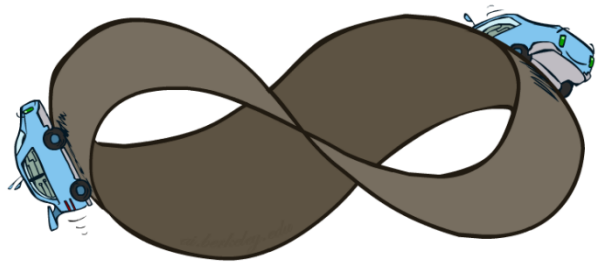
Discounting example: Racing

- Here the car keep going in a infinite loop, until it overheats.



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 $0 < \gamma < 1$
  - ✓ Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Some definitions

Definition: transition probabilities

The **transition probabilities** $T(s, a, s')$ specify the probability of ending up in state s' if taken action a in state s.

Definition: value (expected utility)

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

Definition: policy

A **policy** π is a mapping from each state $s \in \text{States}$ to an action $a \in \text{Actions}(s)$.

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 quantity).

Definition: optimal value

The **optimal value** $V_{\text{opt}}(s)$ is the maximum value attained by any policy.

Definition: value of a policy

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

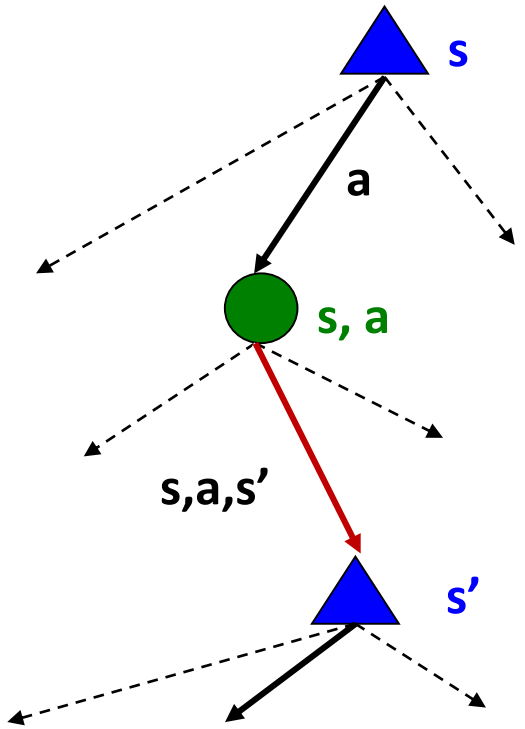
Solving MDPs

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



s is a *state*

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

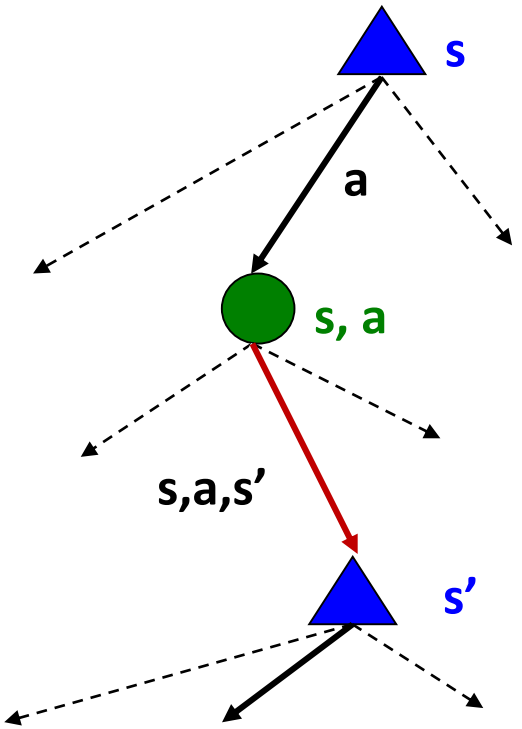
Values of States

- 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:

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

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

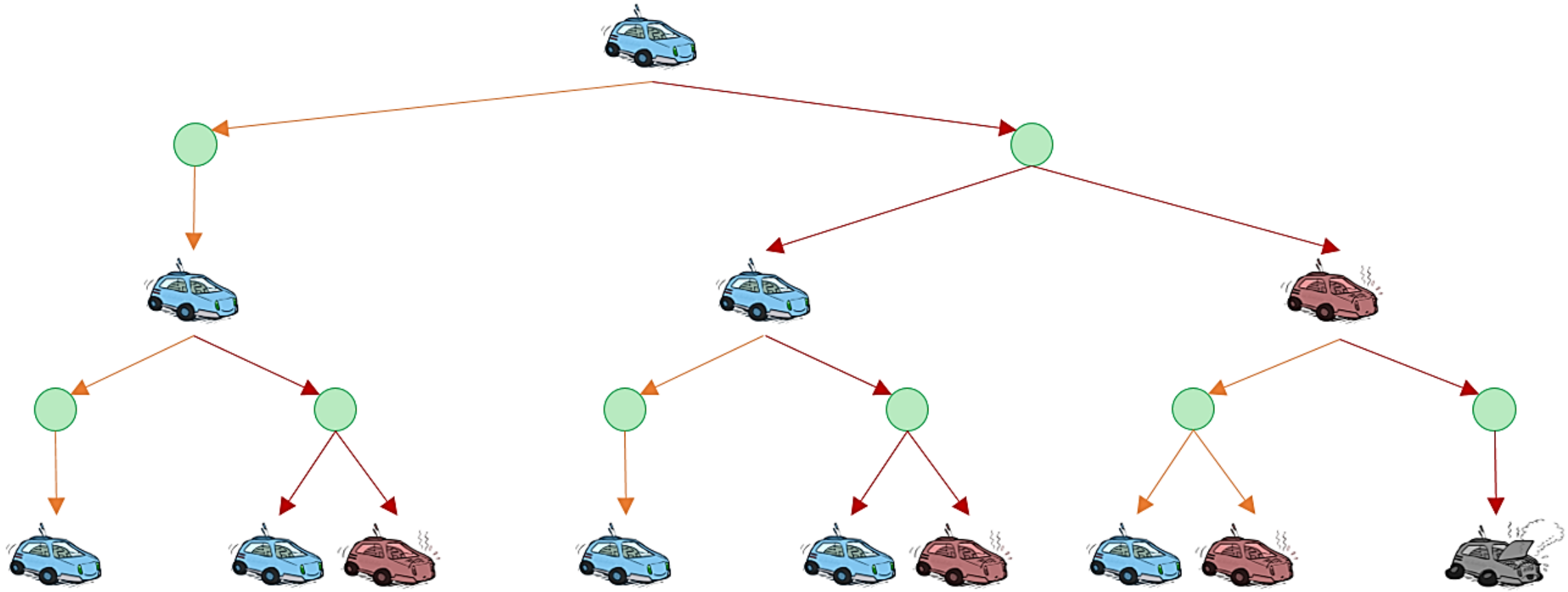
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



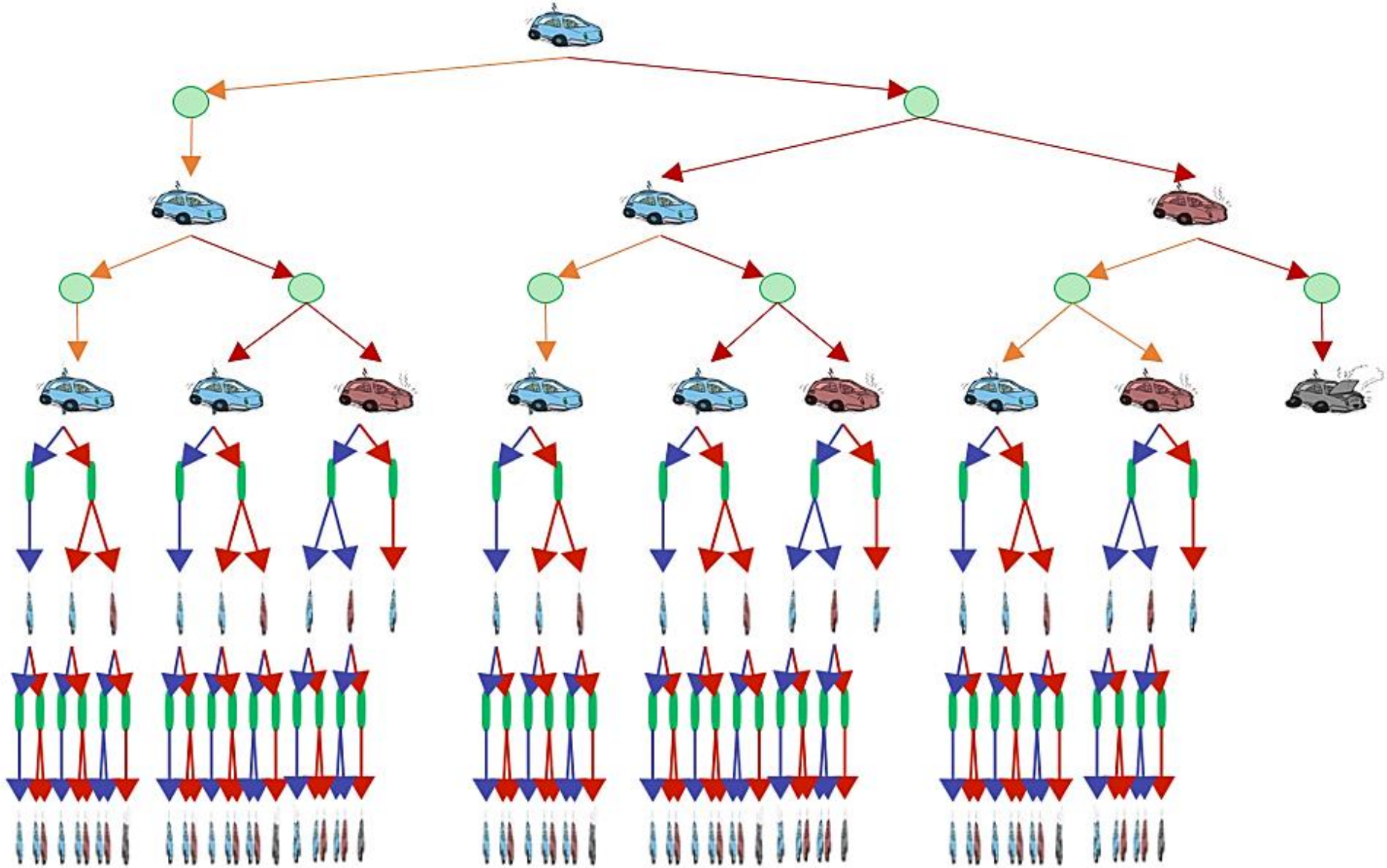
s is a *state*

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

Values of States: racing example

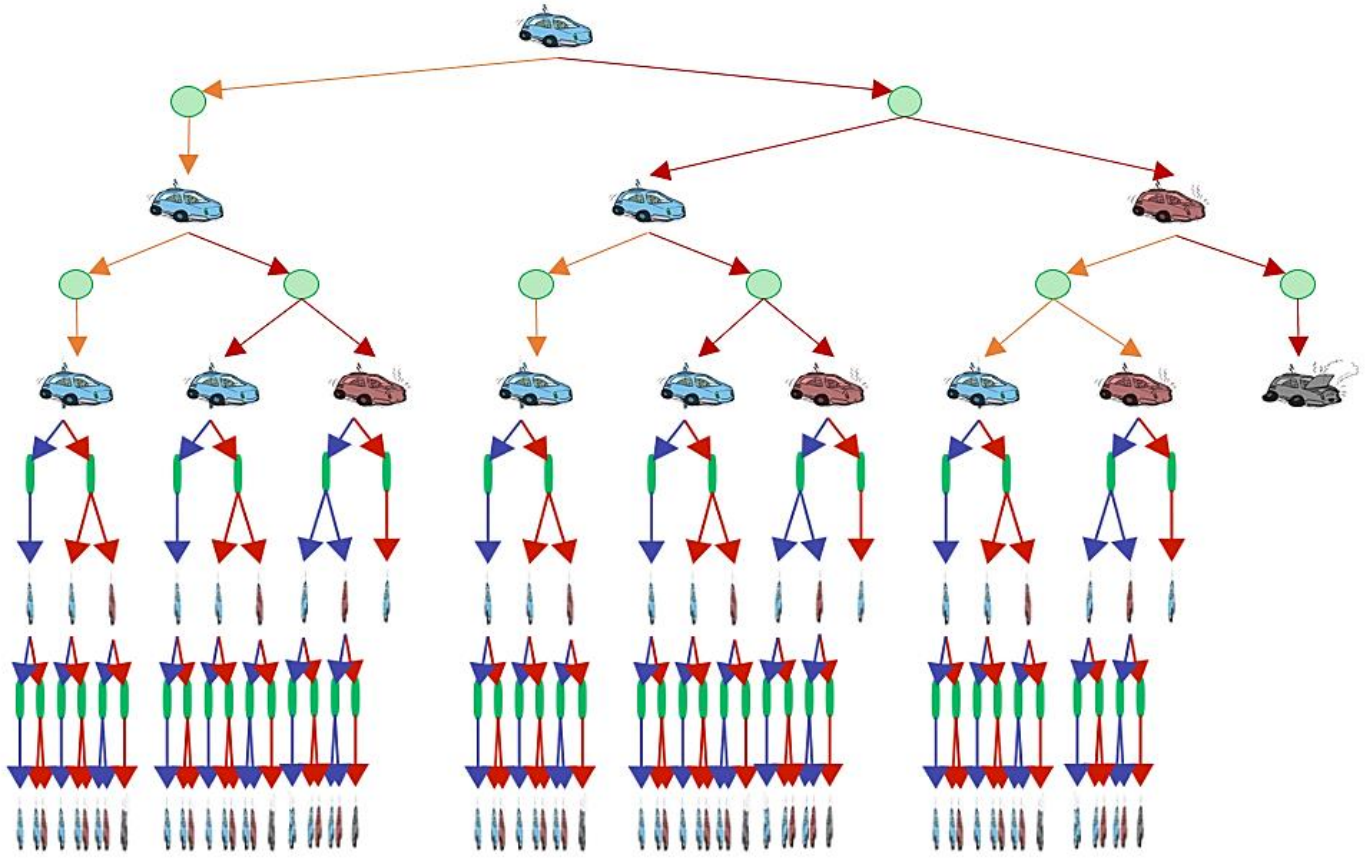


Values of States: racing example



Values of States: racing example

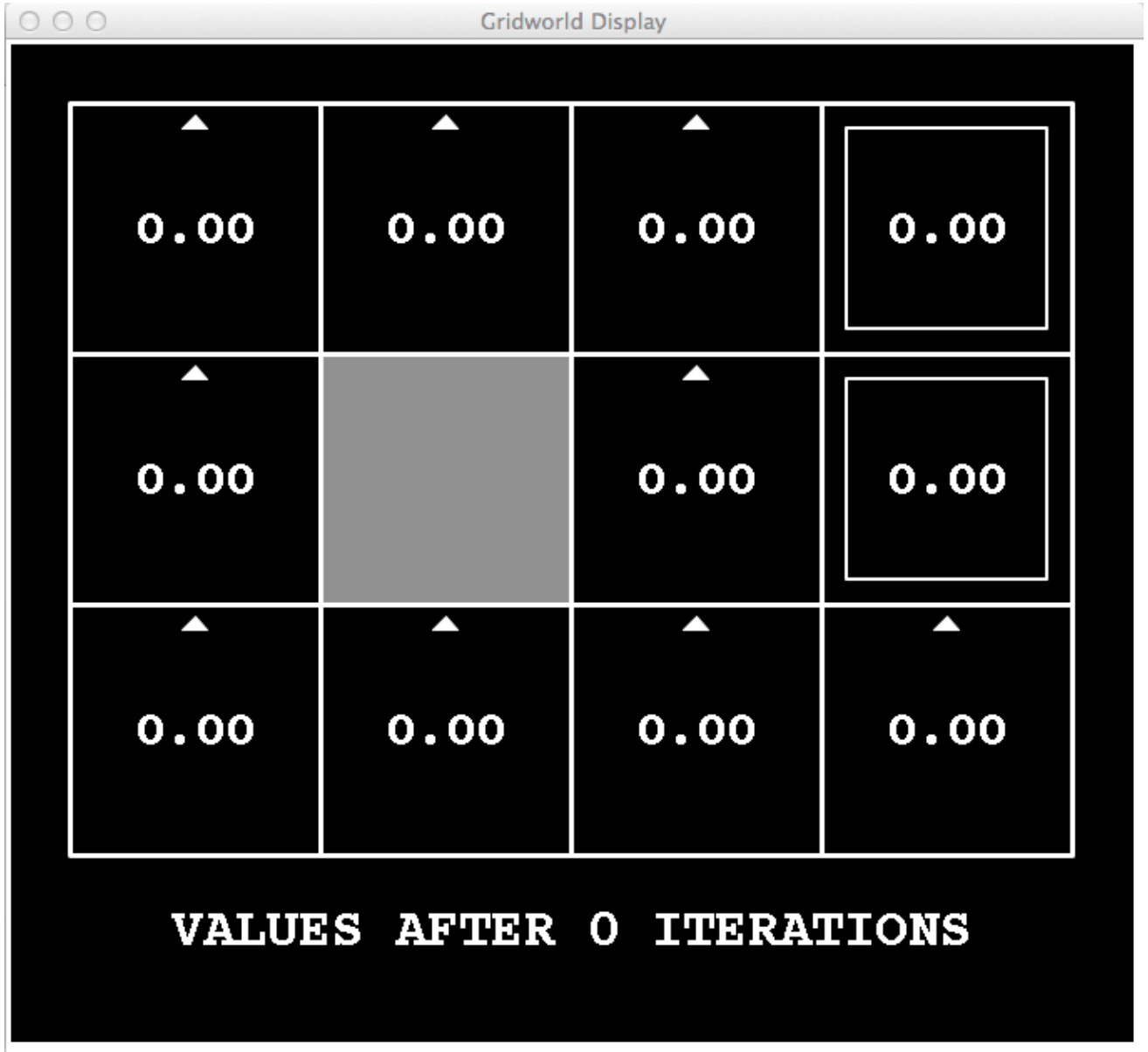
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter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



Values of States: time limited values

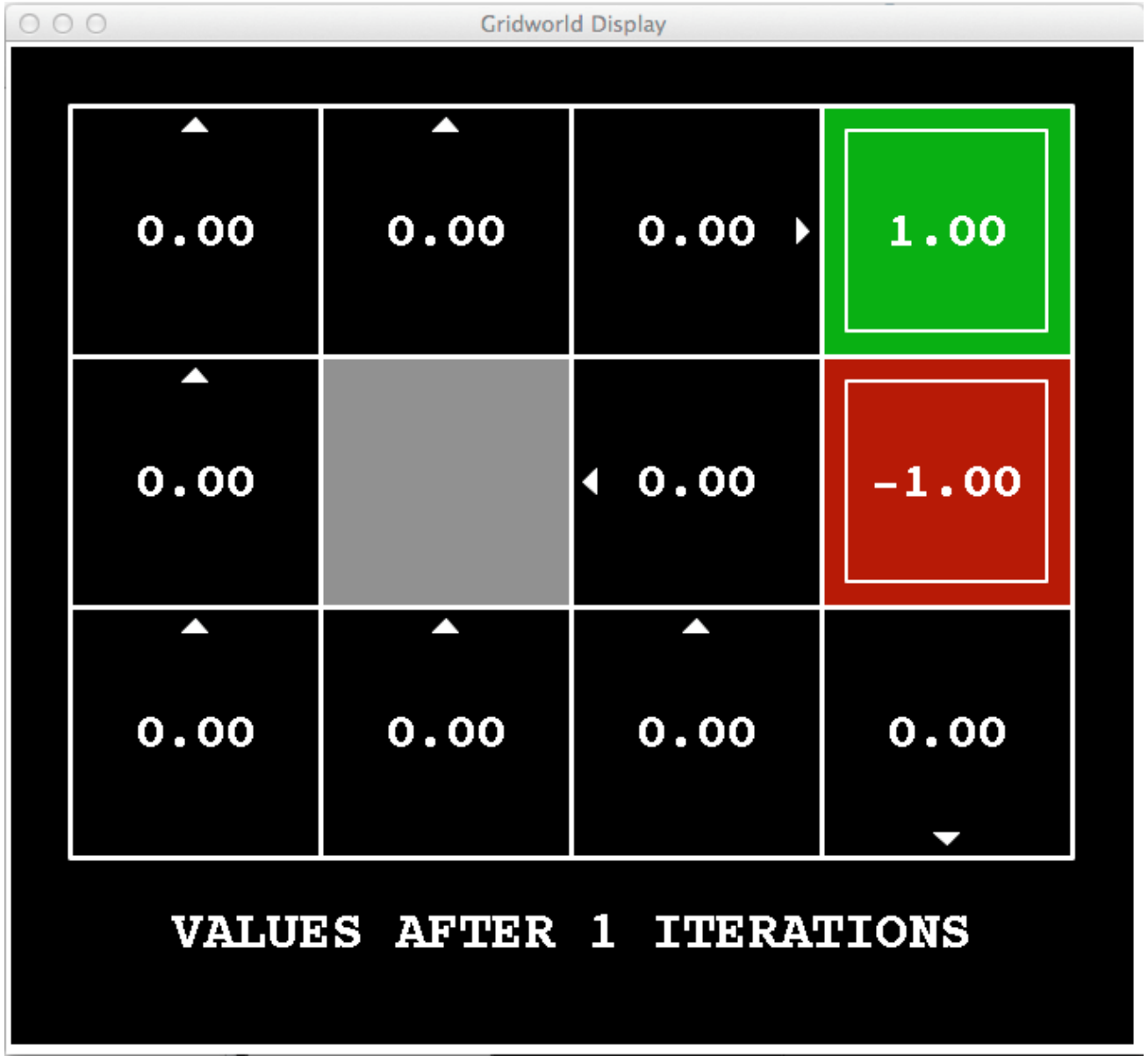
- Time-limited values as key idea.
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
- We use **value-iteration** where we start from depth or bottom.
- For this we need, time-limited values as key idea.

Values of States: time limited values



Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States: time limited values



Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States: time limited values

Noise = 0.2
Discount = 0.9
Living reward = 0

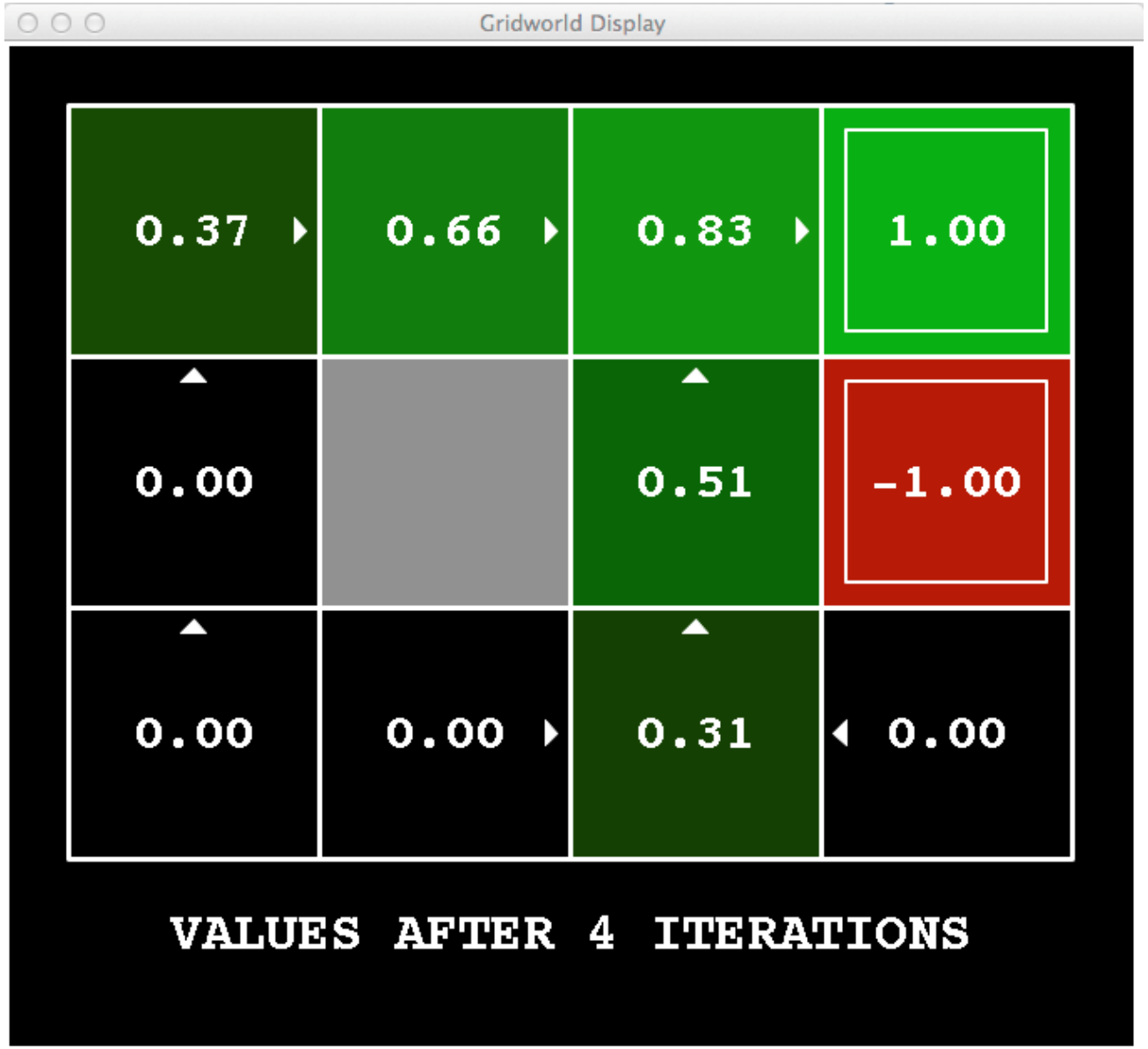
Values of States: time limited values

Find for 4 iteration

Noise = 0.2
Discount = 0.9
Living reward = 0

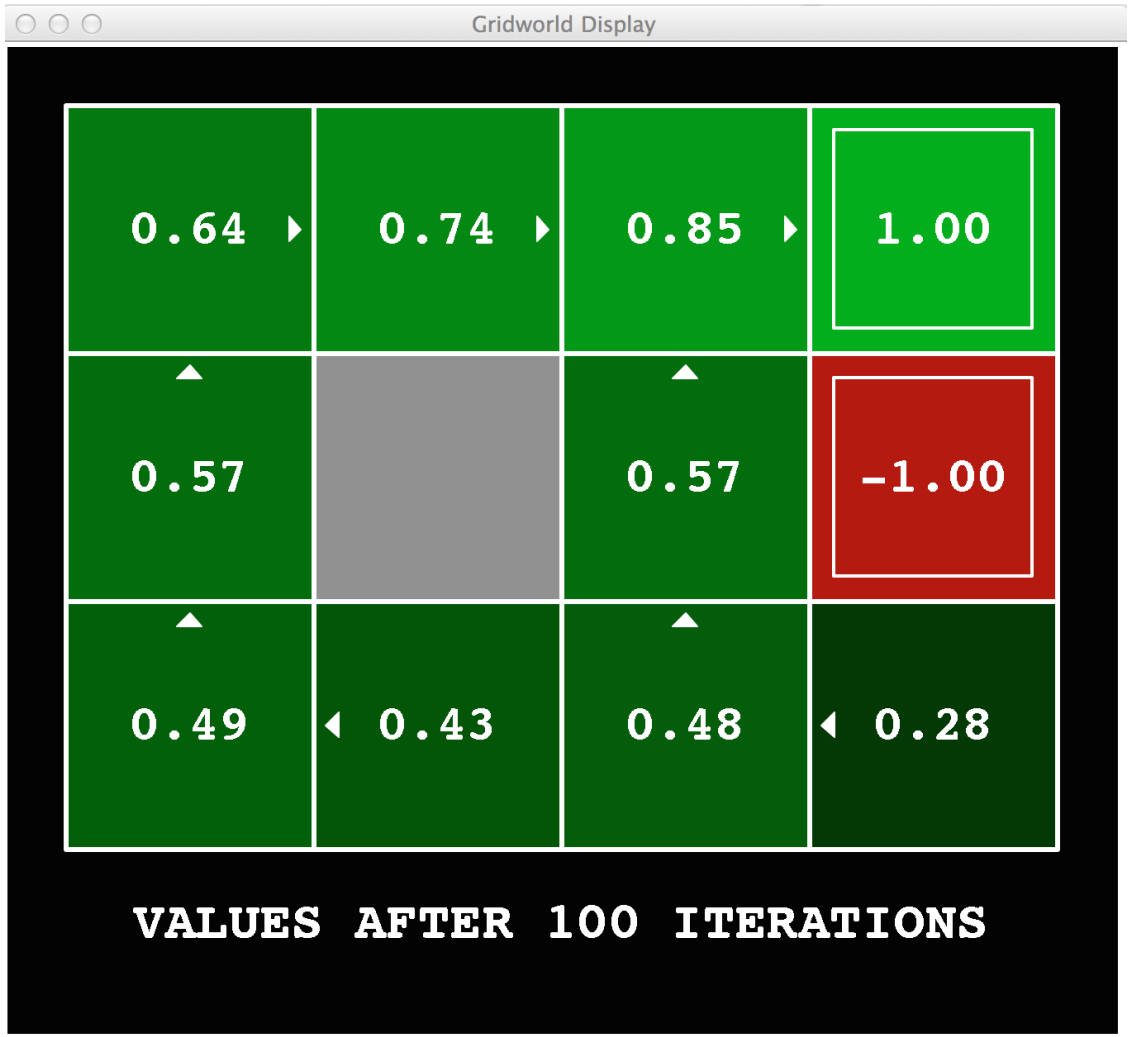
Values of States: time limited values

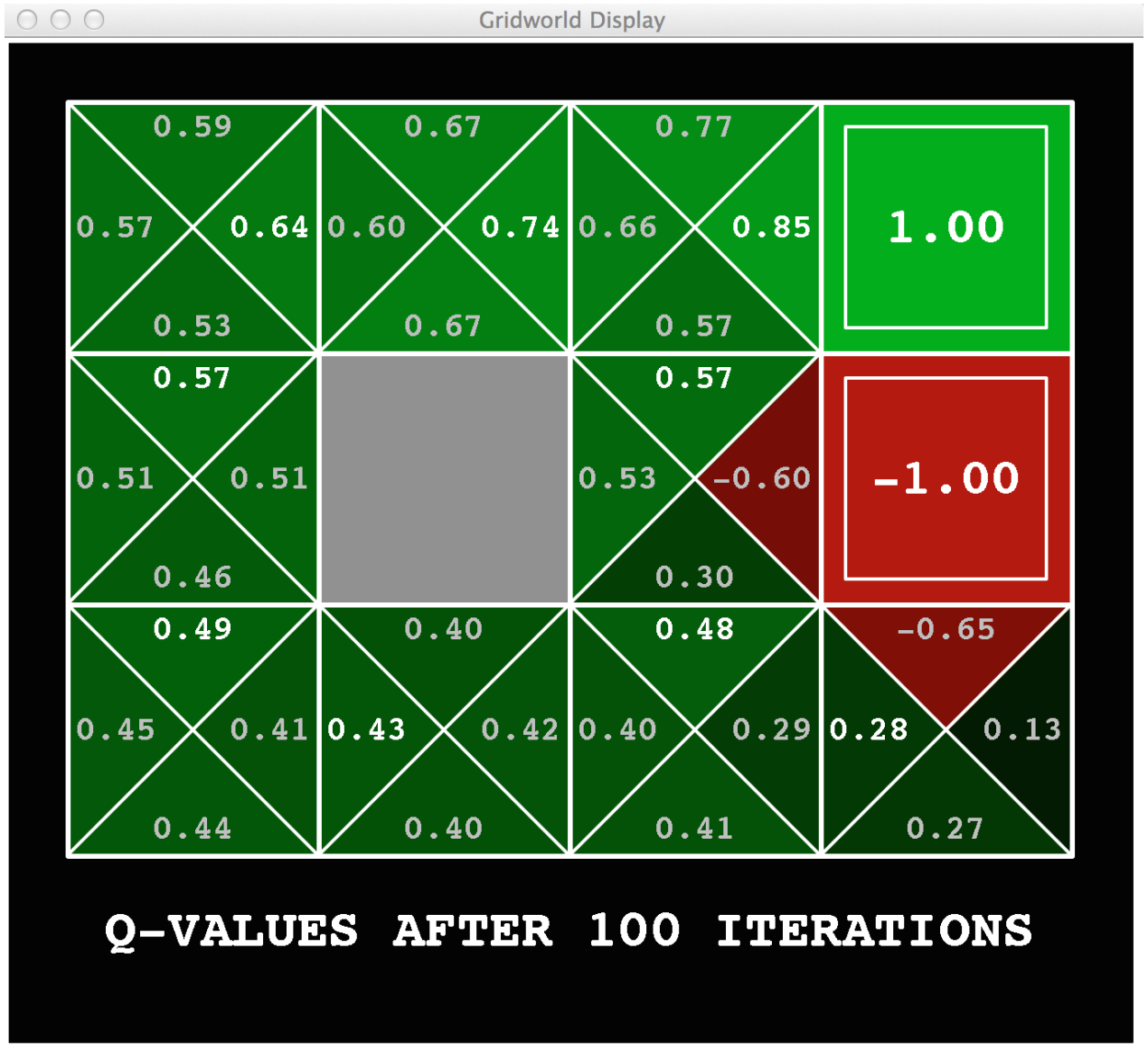
And so on …..



Noise = 0.2
Discount = 0.9
Living reward = 0

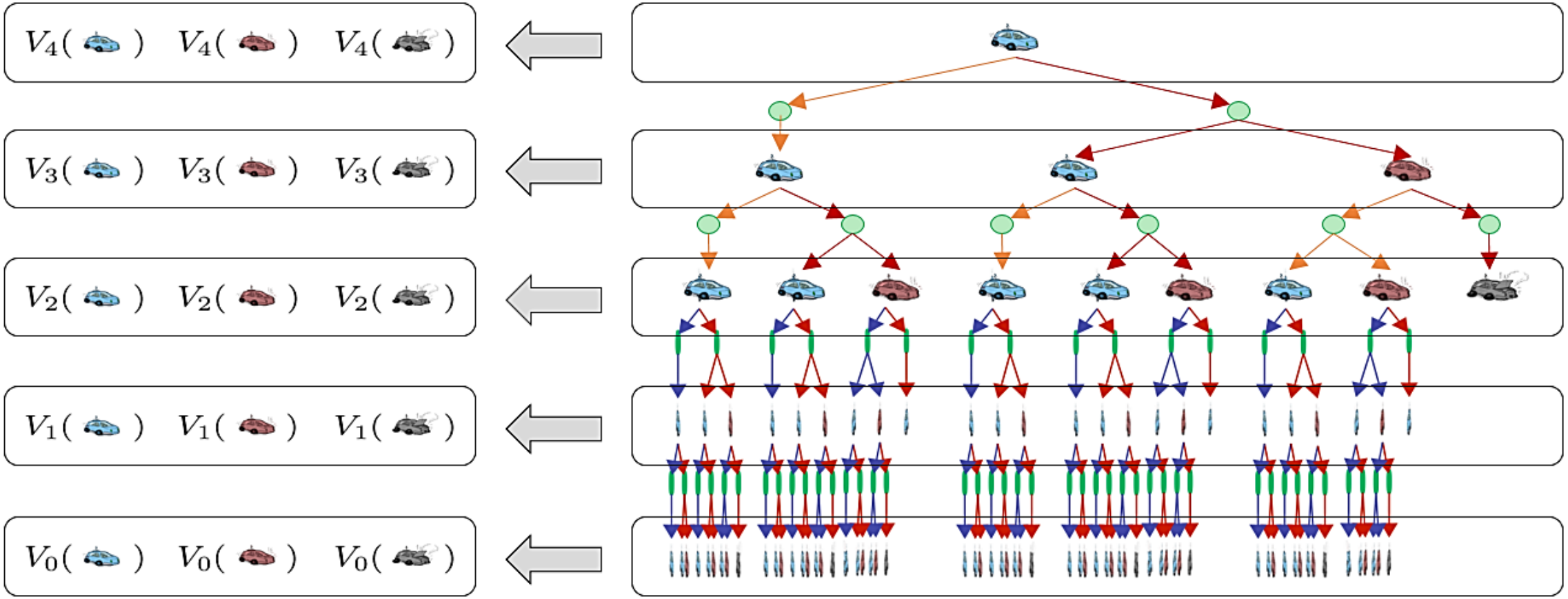
Values of States: time limited values





Time-Limited Values

- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps

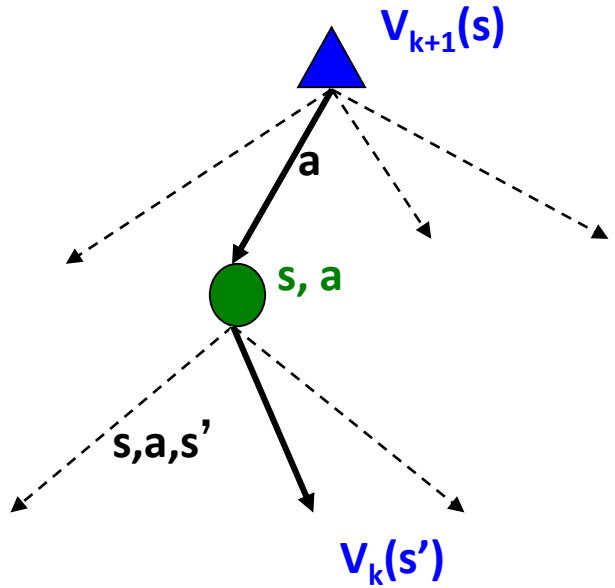


Values of States: value iteration

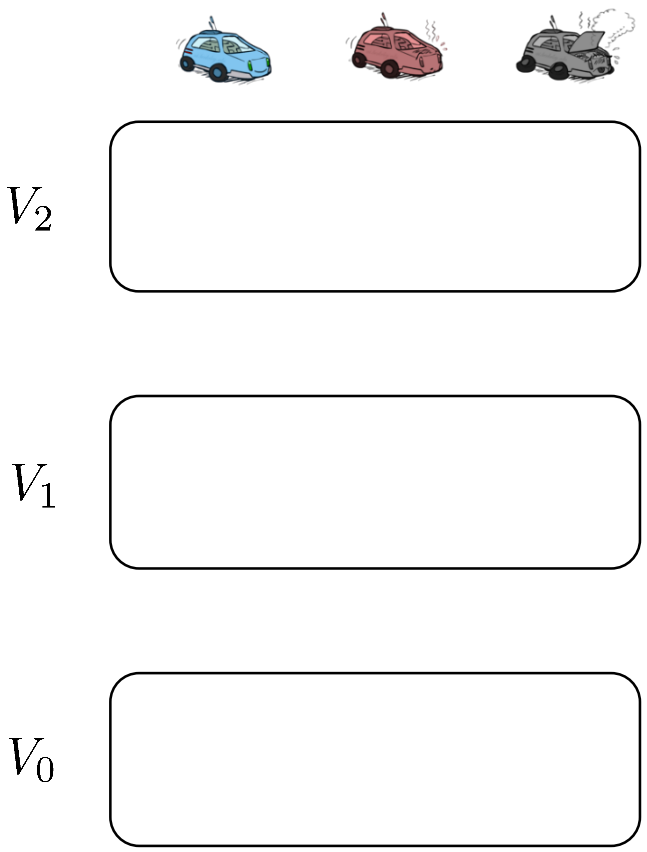
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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

- Repeat until convergence
- 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



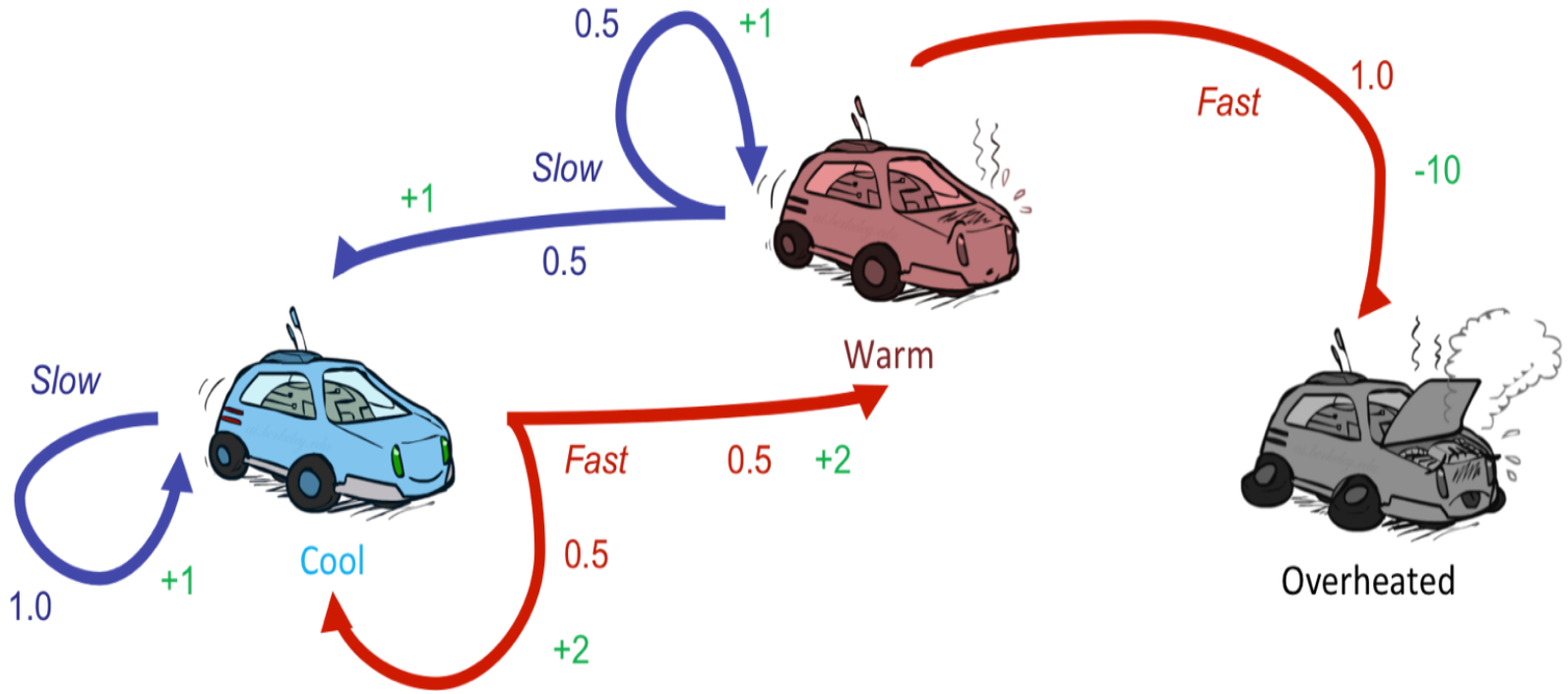
Value iteration: example



V_2

V_1

V_0

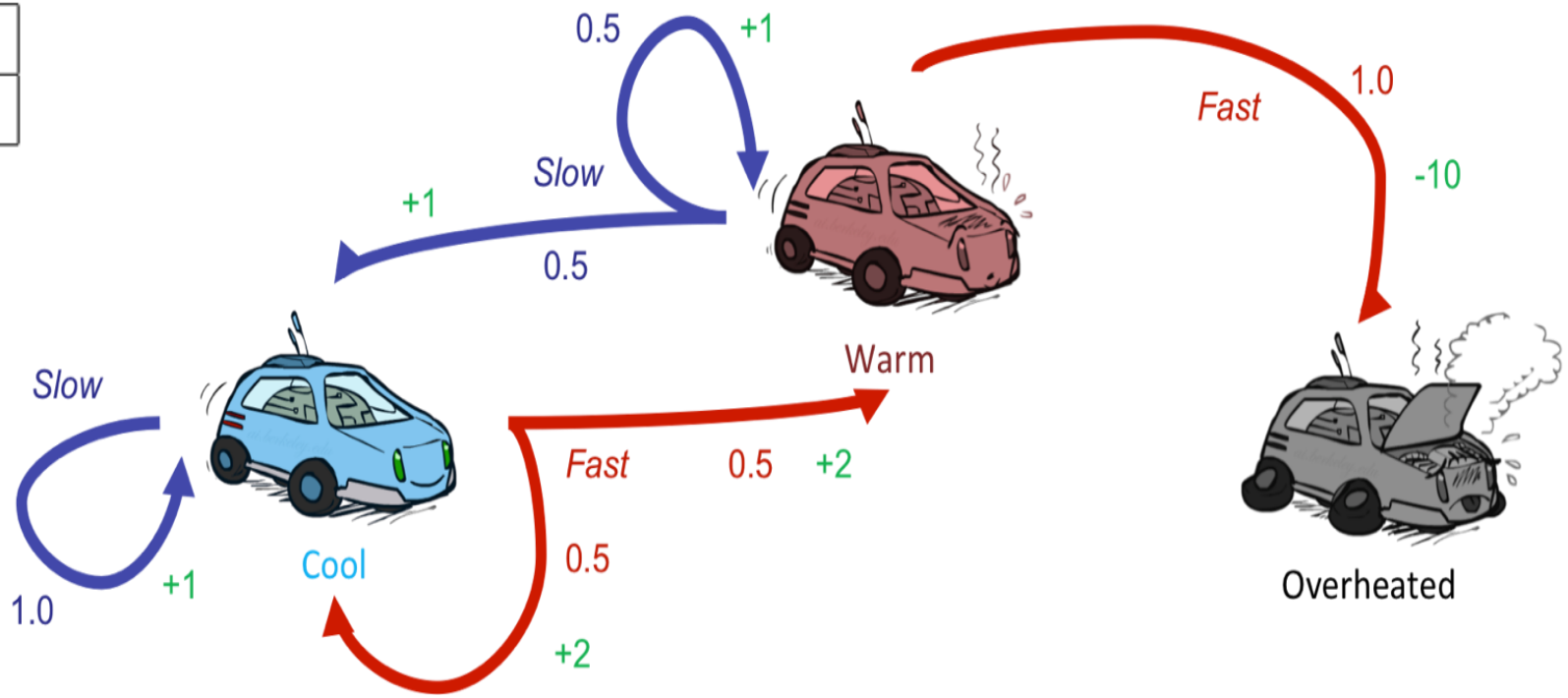


$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Assume no discount!

Value iteration: example

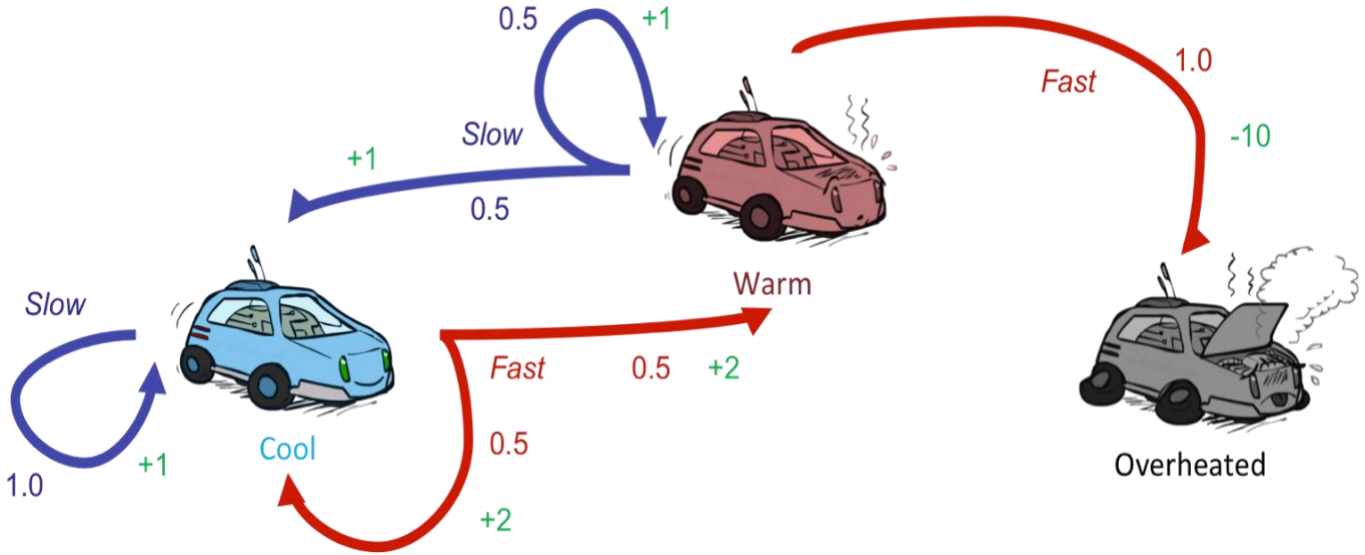
	cool	warm	overheated
V_0	0	0	0



Assume no discount!

Value iteration: example

	cool	warm	overheated
V_0	0	0	0

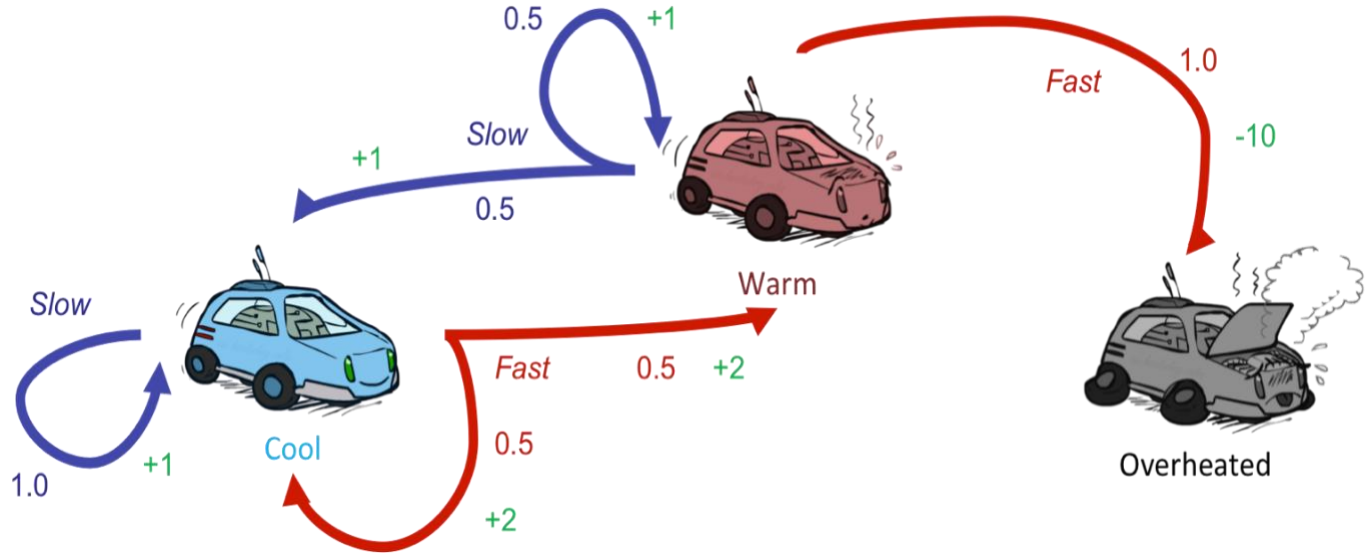


$$
\begin{aligned}
V_1(cool) &= \max\{1\cdot[1+\ 1\ \cdot 0],\ 0.5\cdot[2+\ 1\ \cdot 0]+0.5\cdot[2+\ 1\ \cdot 0]\} \\
&= \max\{1,2\} \\
&= \boxed{2}
\end{aligned}
$$

Assume no discount!

Value iteration: example

	cool	warm	overheated
V_0	0	0	0



$$V_1(cool) = \max\{1\cdot[1+\mathbf{1}\cdot 0],\ 0.5\cdot[2+\mathbf{1}\cdot 0]+0.5\cdot[2+\mathbf{1}\cdot 0]\}$$
$$= \max\{1,2\}$$
$$= \boxed{2}$$

$$V_1(warm) = \max\{0.5\cdot[1+\mathbf{1}\cdot 0]+0.5\cdot[1+\mathbf{1}\cdot 0],\ 1\cdot[-10+\mathbf{1}\cdot 0]\}$$
$$= \max\{1,-10\}$$
$$= \boxed{1}$$

$$V_1(overheated) = \max\{\}$$
$$= \boxed{0}$$

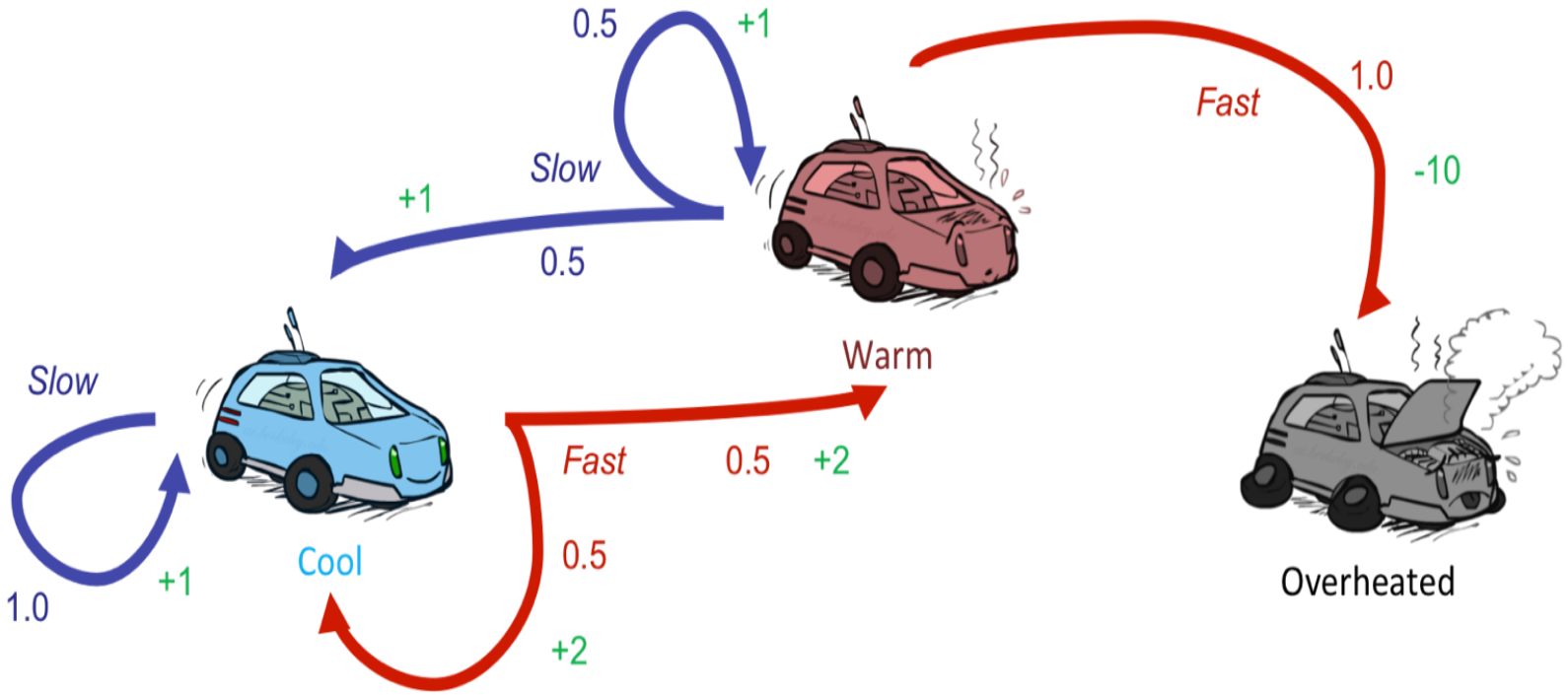
Assume no discount!

Value iteration: example

V_2

V_1 2 1 0

V_0 0 0 0

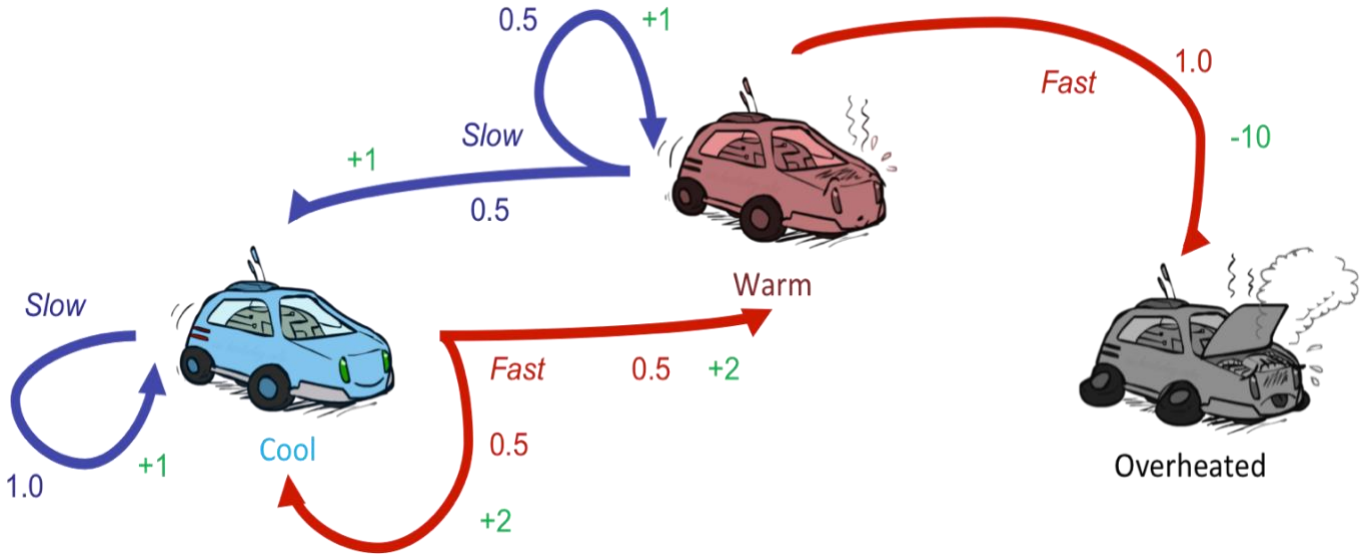


$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Assume no discount!

Value iteration: example

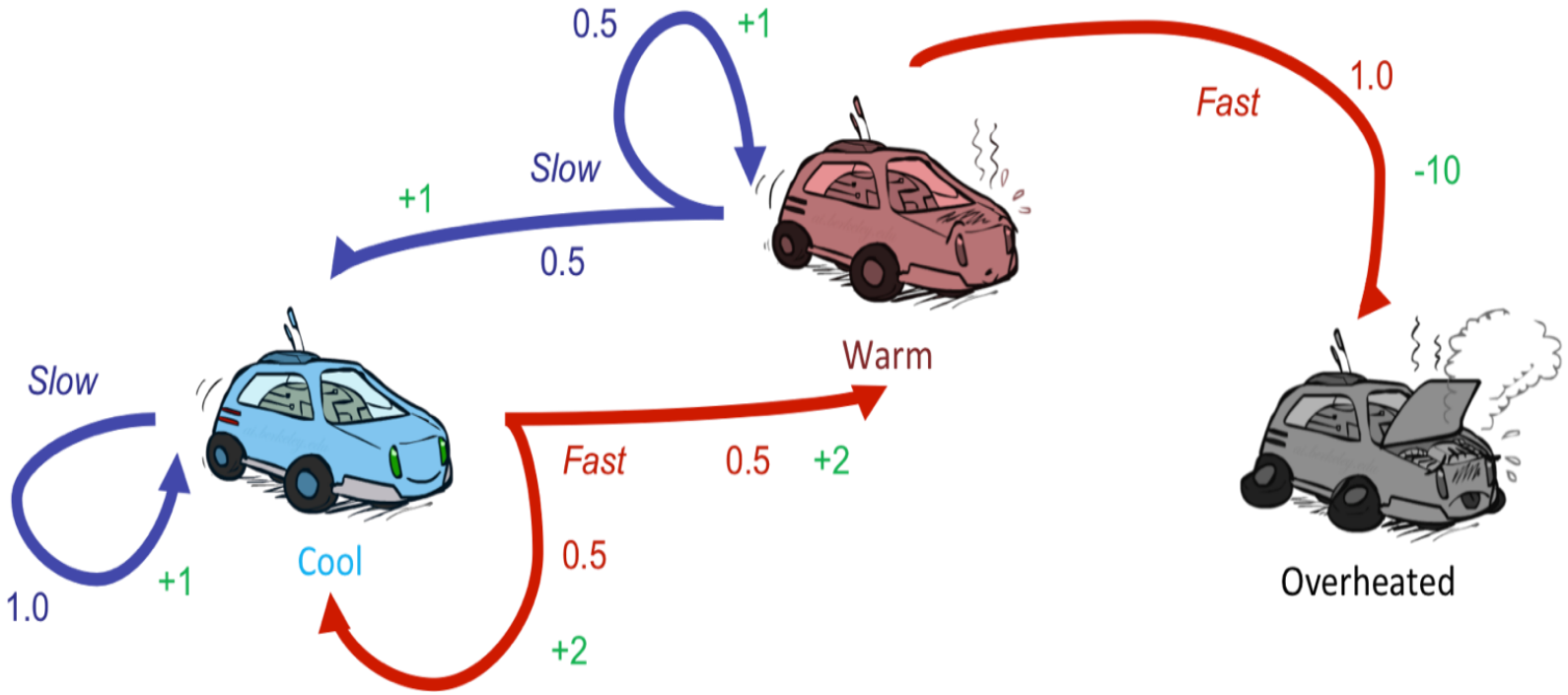
	cool	warm	overheated
V_0	0	0	0
V_1	2	1	0



Assume no discount!

Value iteration: example

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



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

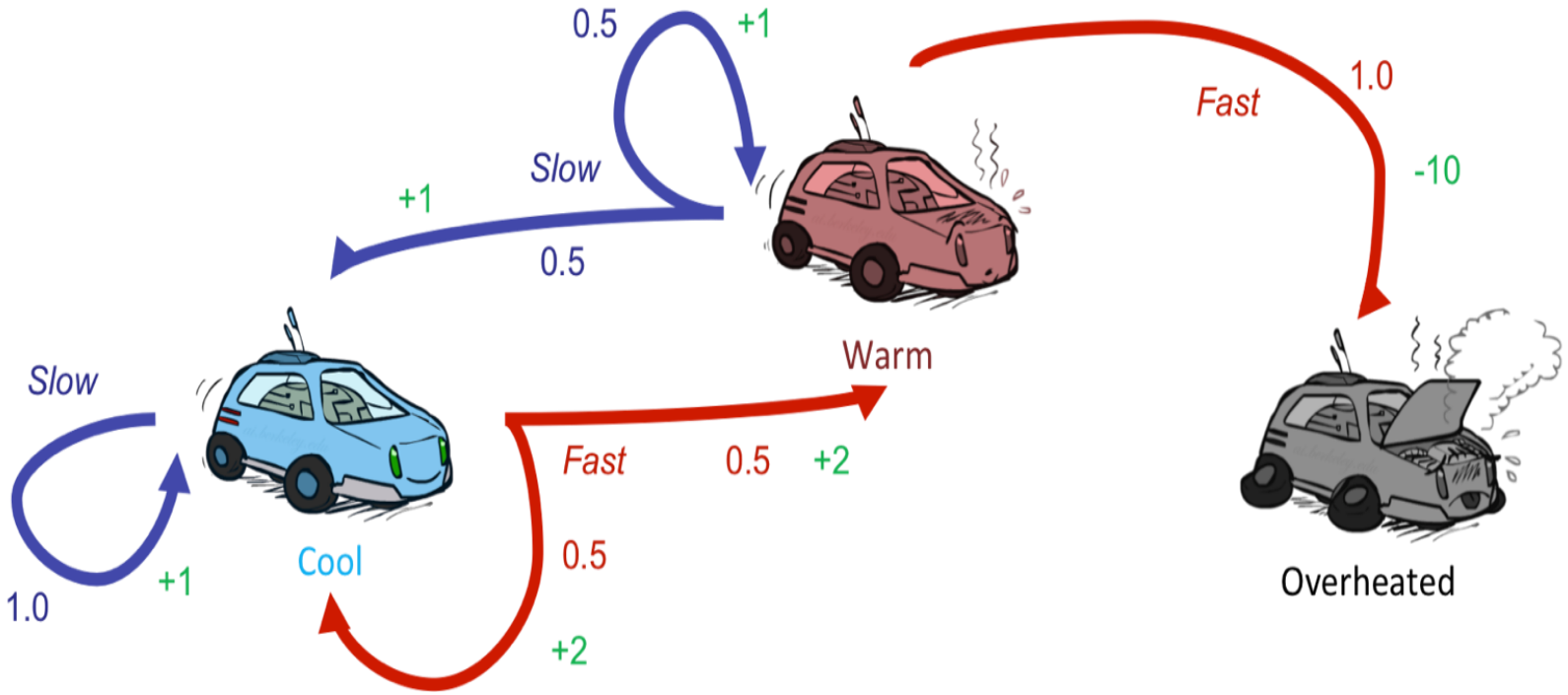
Assume no discount!

Value iteration: example

V_2

V_1

V_0

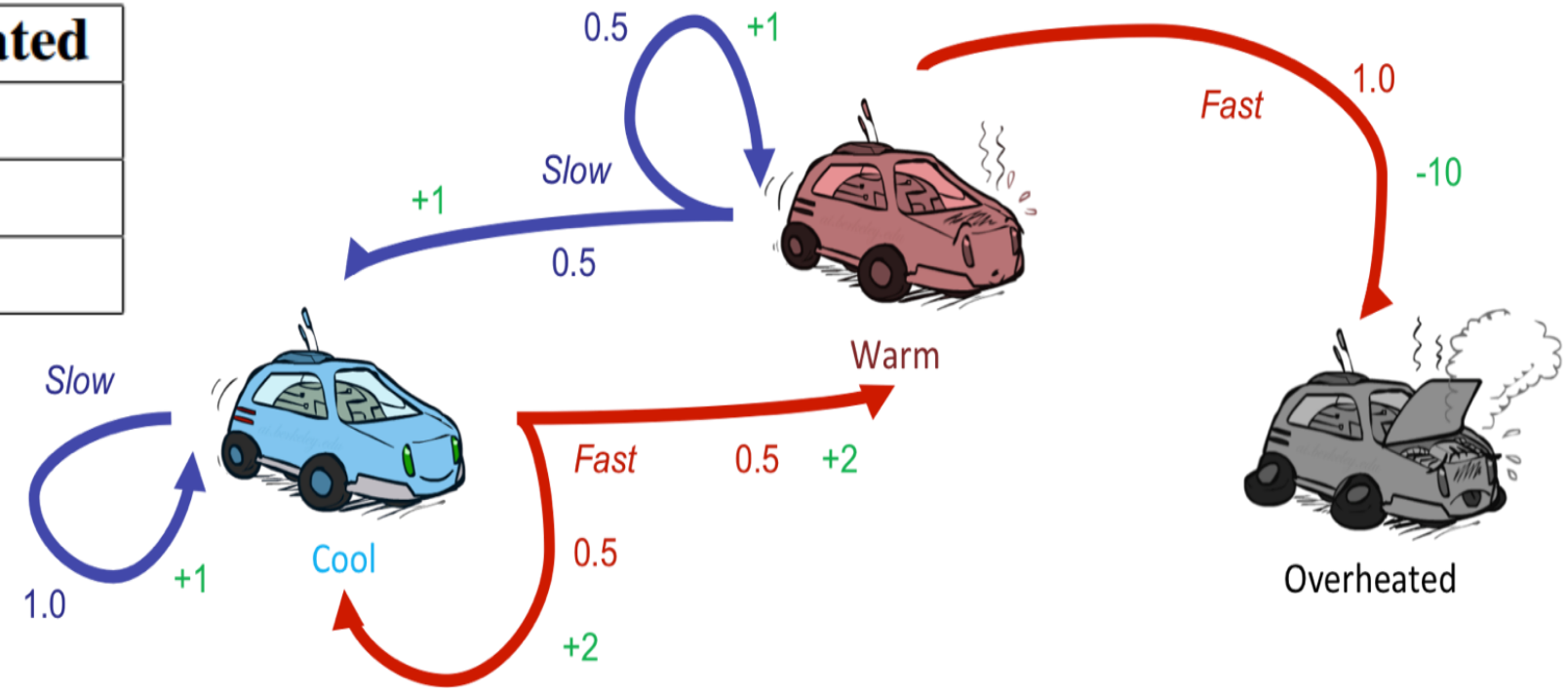


$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Repeat the same process for discount = 0.5

Value iteration: example

	cool	warm	overheated
V_0	0	0	0
V_1	2	1	0
V_2	2.75	1.75	0

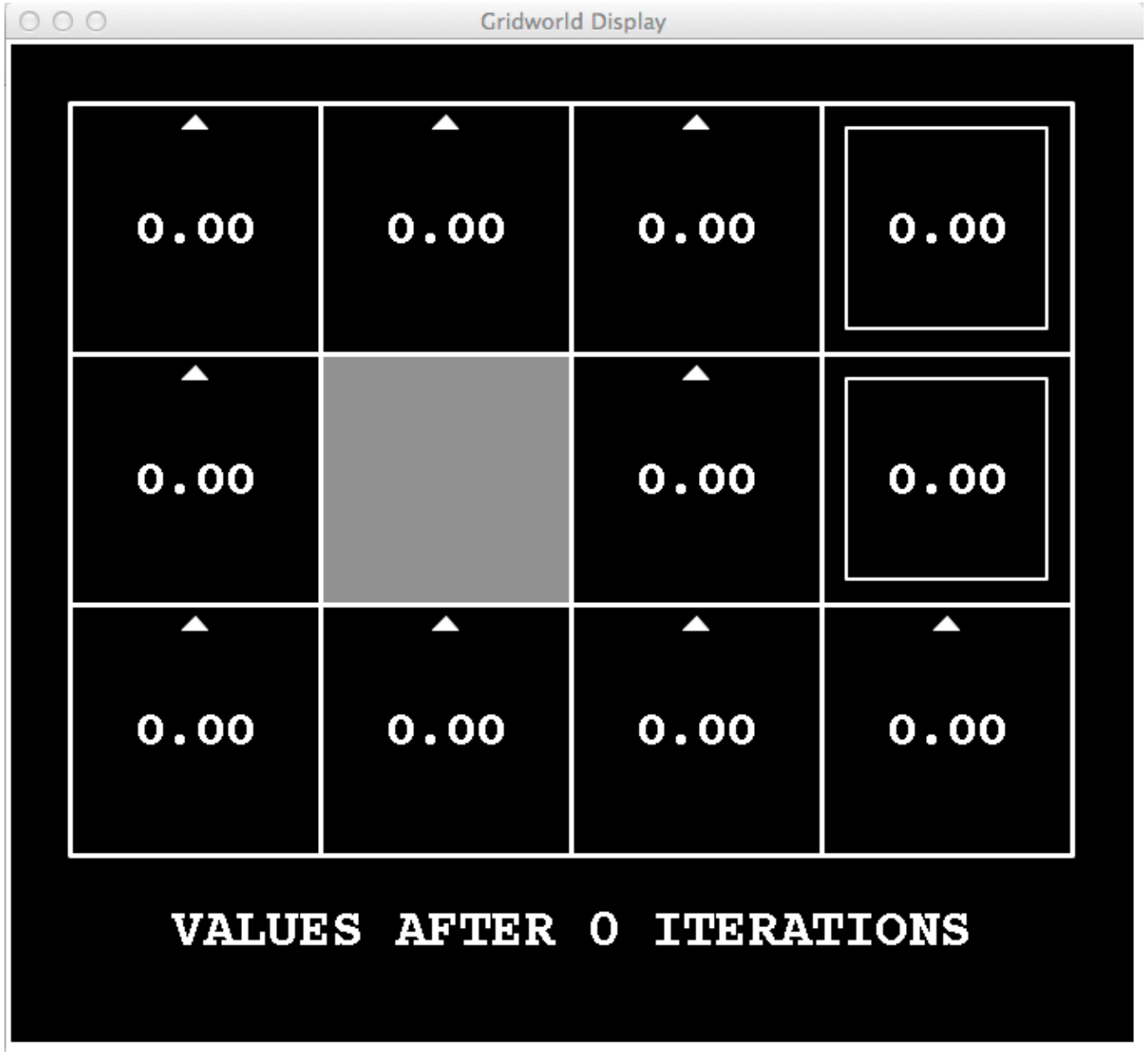


$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Repeat the same process for discount = 0.5

Values of States: time limited values

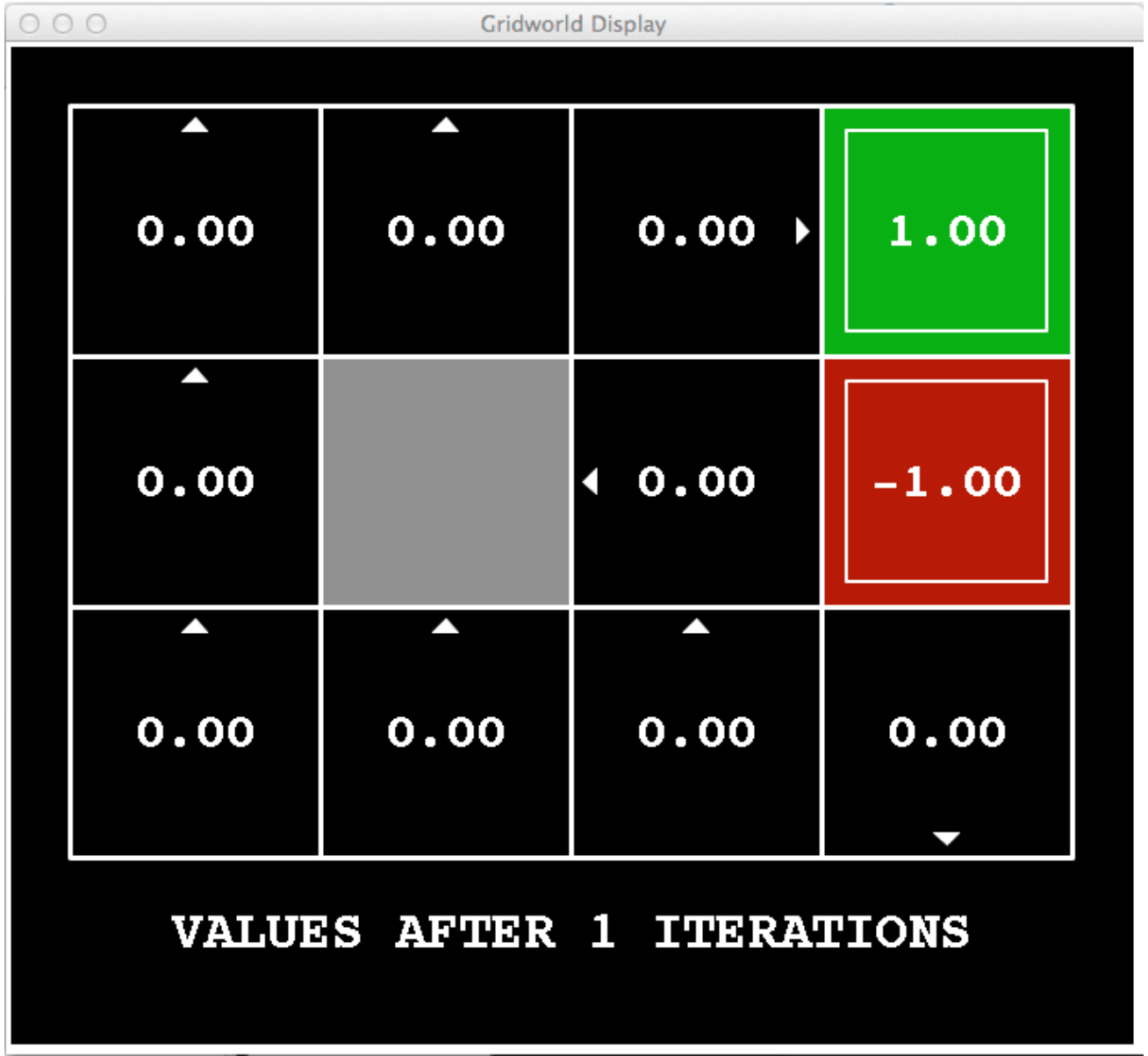
Find values ?



Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States: time limited values

Find values ?



Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States: time limited values

Find values ?

Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States: time limited values

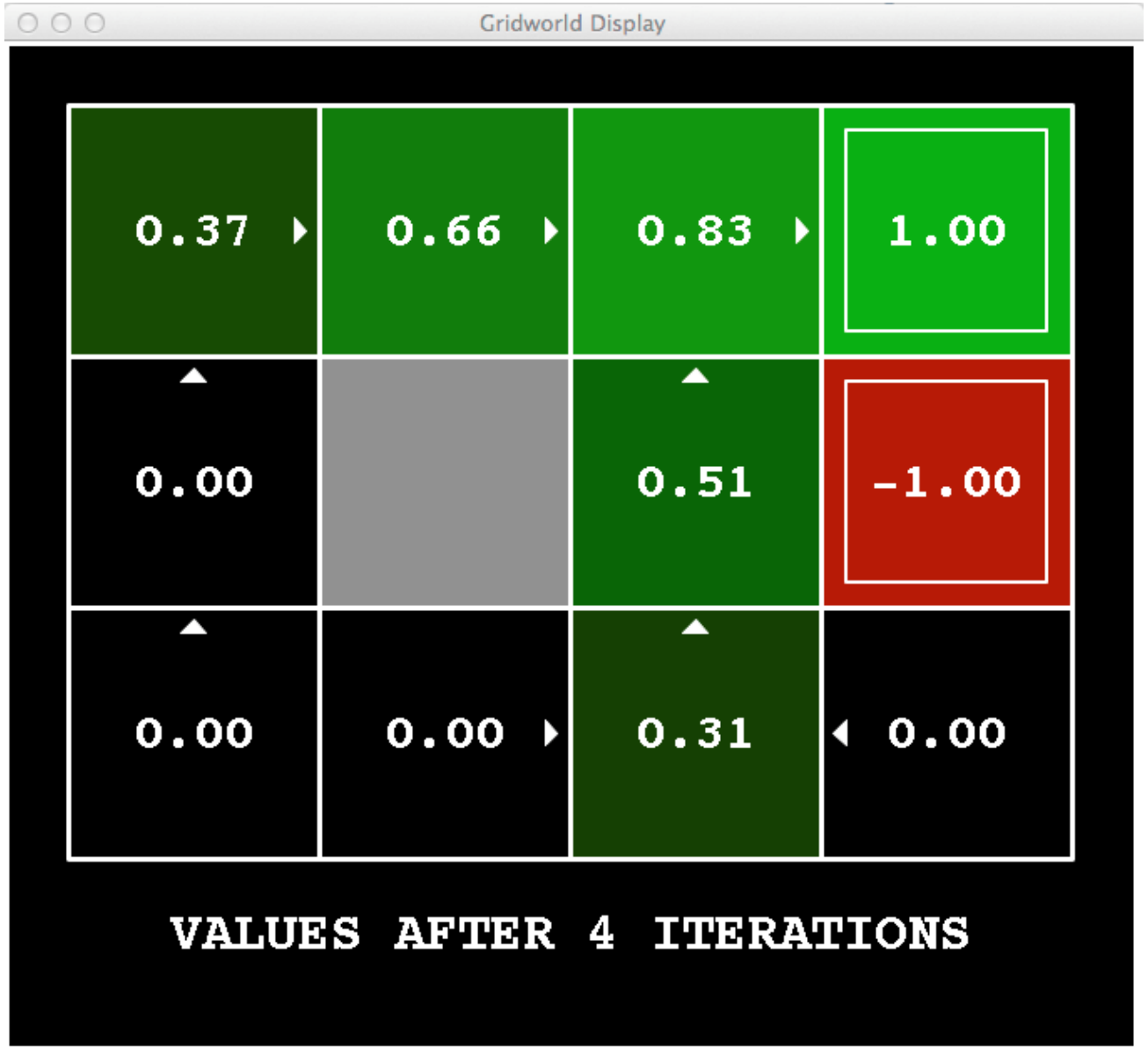
Find for 4 iteration

Find values ?

Noise = 0.2
Discount = 0.9
Living reward = 0

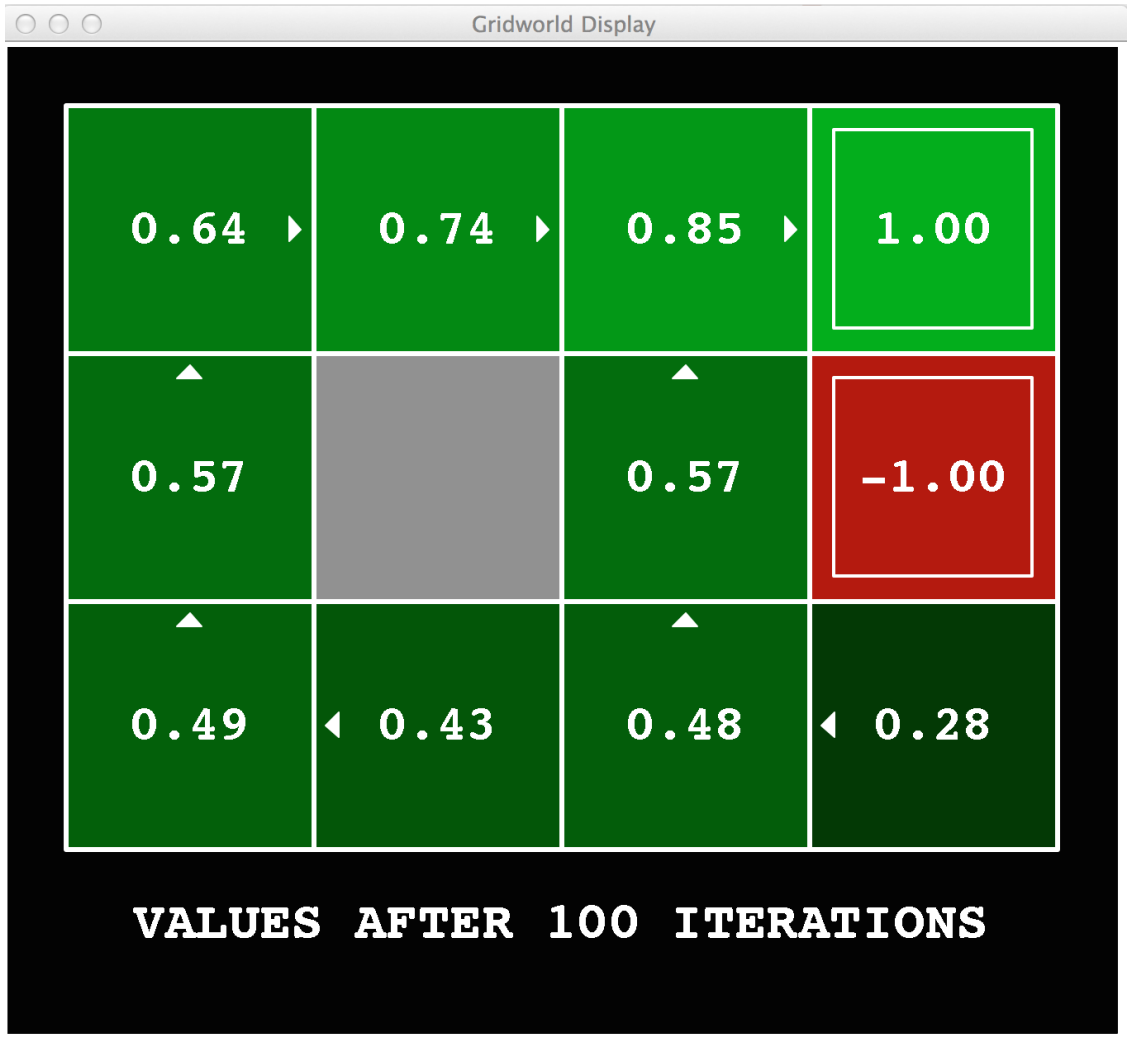
Values of States: time limited values

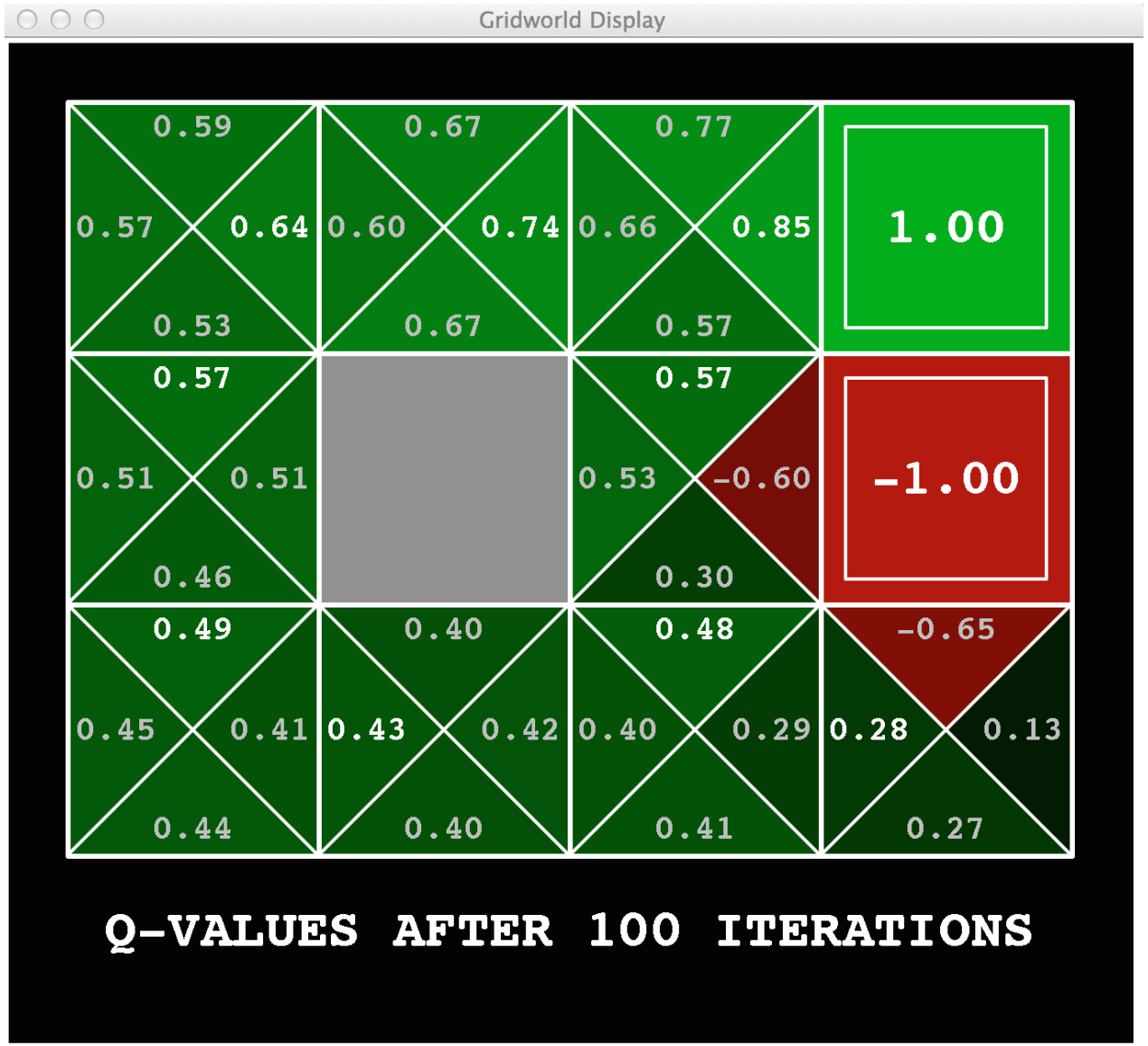
And so on …..



Noise = 0.2
Discount = 0.9
Living reward = 0

Values of States: time limited values





The Bellman Equations

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

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

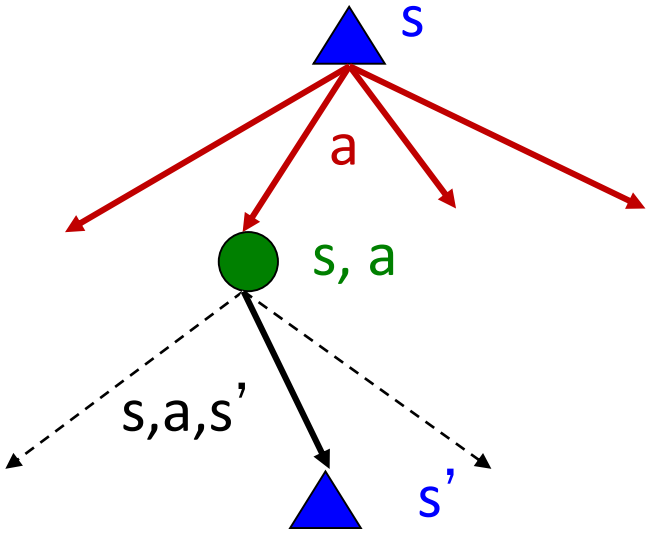
$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

Value iteration: Convergence*

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

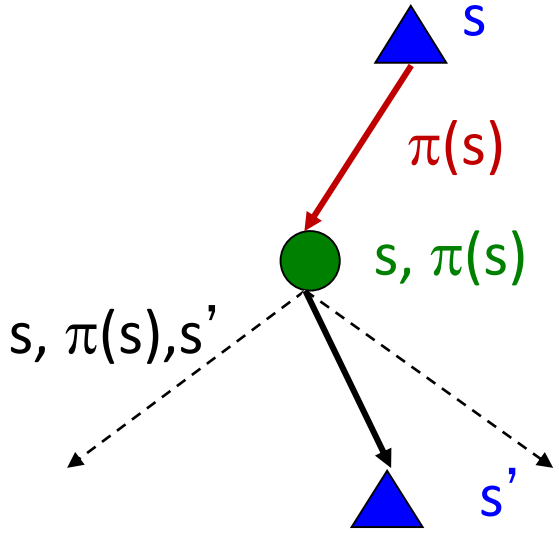
Policy Evaluation: fixed policy

Do the optimal action



- Expectimax trees max over all actions to compute the optimal values

Do what π says to do



- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state.

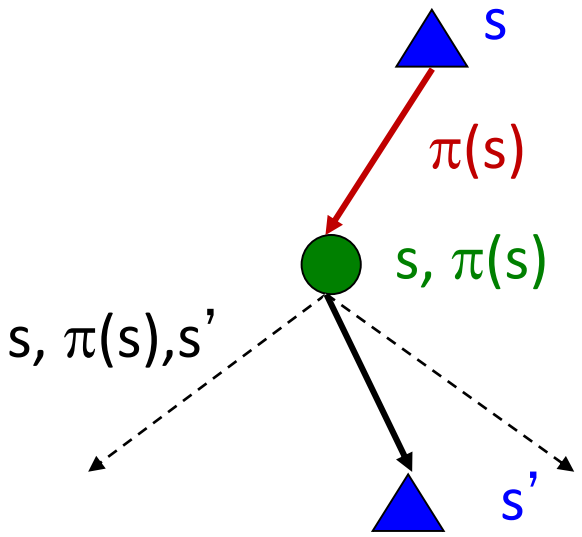
Utilities for a Fixed Policy

- Define the utility of a state s, under a fixed policy π:

  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π

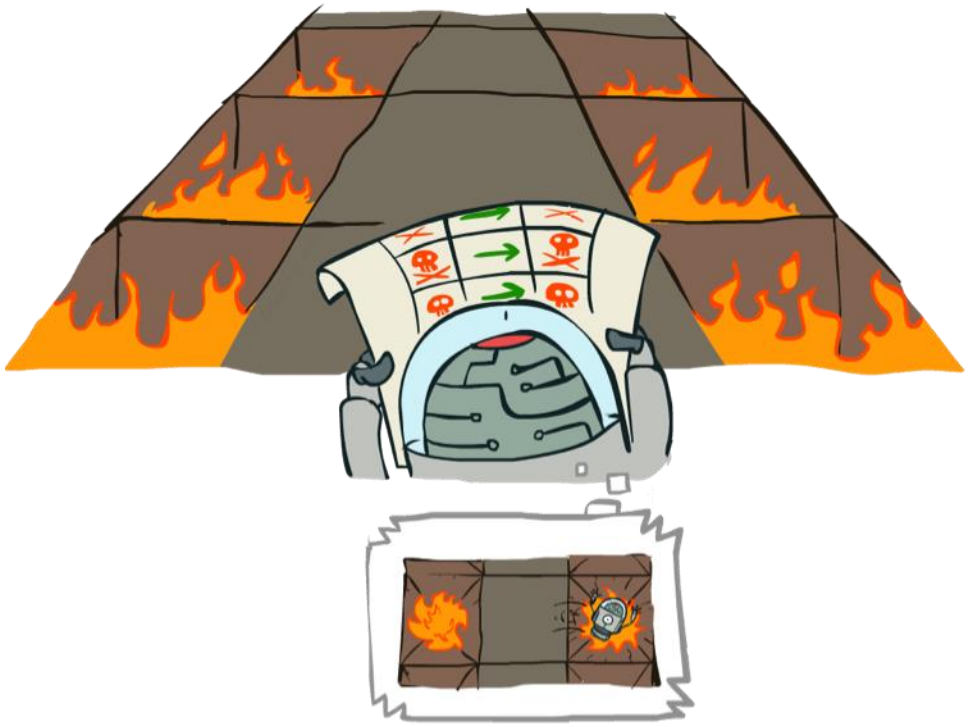
- Recursive relation (one-step look-ahead / Bellman equation):

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

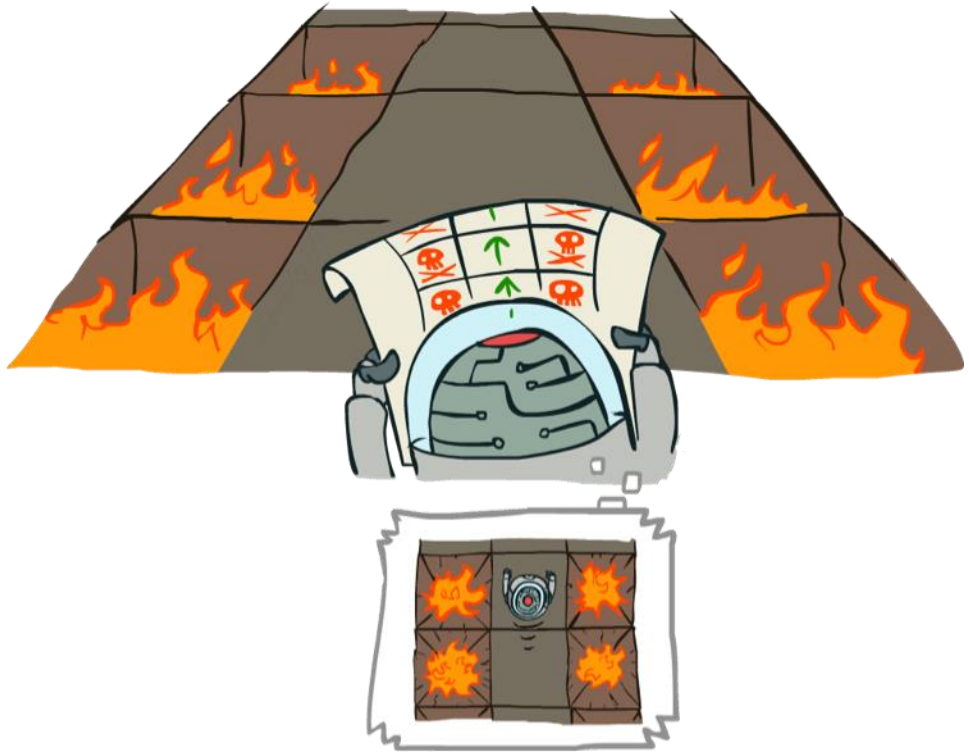


Example: Policy Evaluation

Always Go Right



Always Go Forward



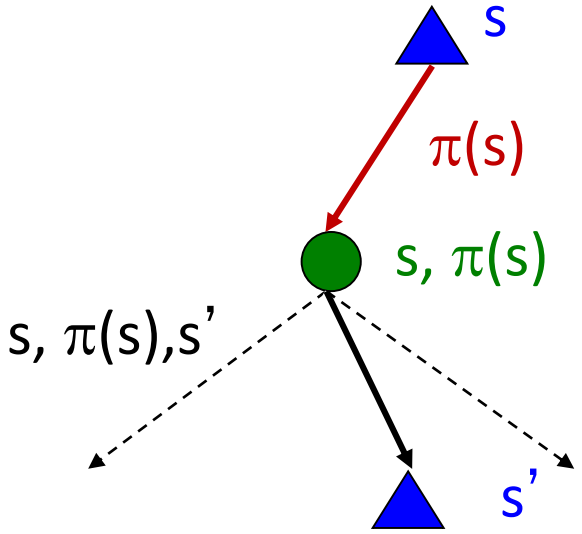
Policy Evaluation

- How do we calculate the V's for a fixed policy π?

- Idea 1: Turn recursive Bellman equations into updates
  (like value iteration)

$$V_0^\pi(s) = 0$$

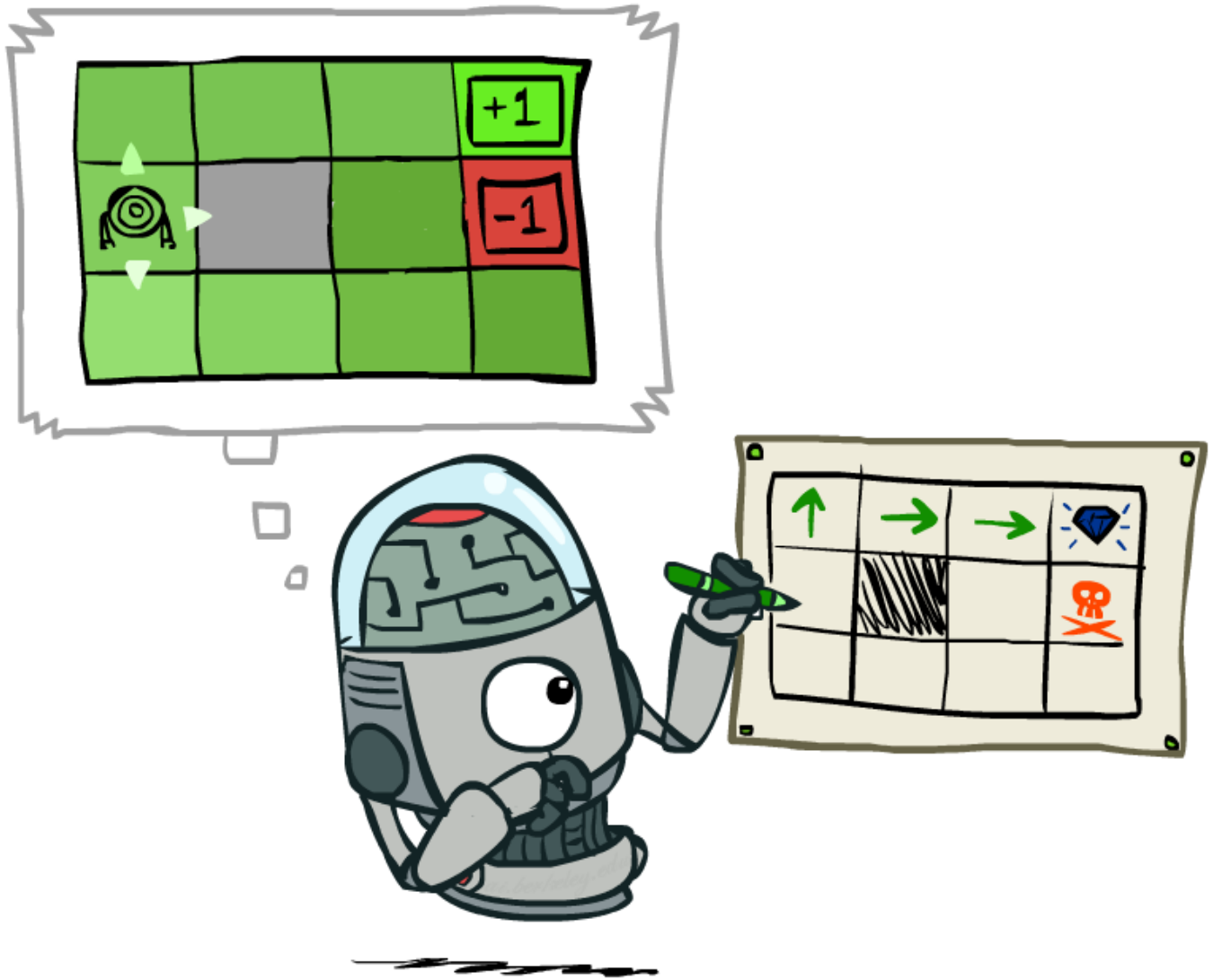
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



- Efficiency: $O(S^2)$ per iteration

- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

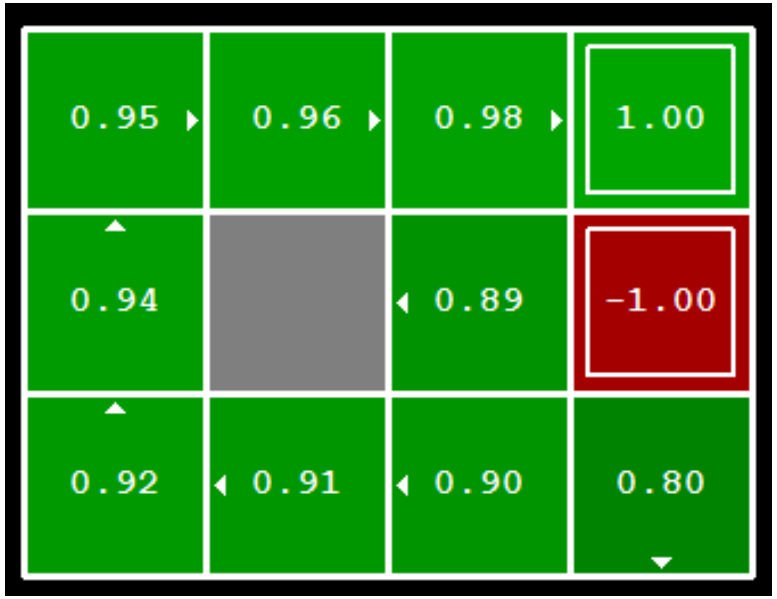
Policy extraction



Computing Actions from V-Values

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

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



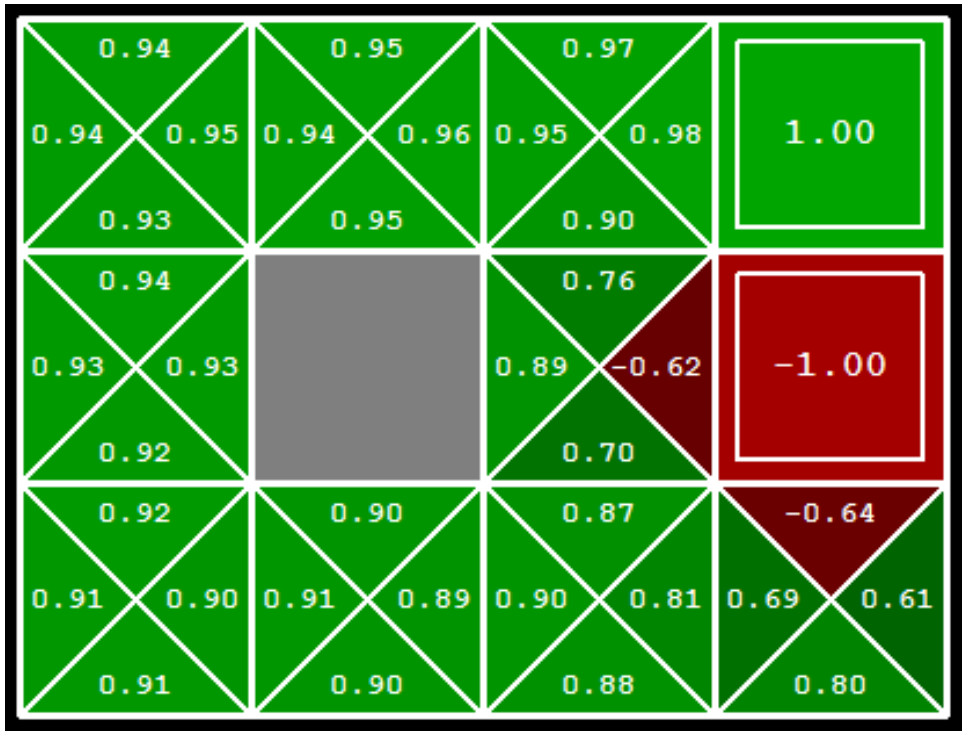
$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Computing Actions from Q-Values

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

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

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



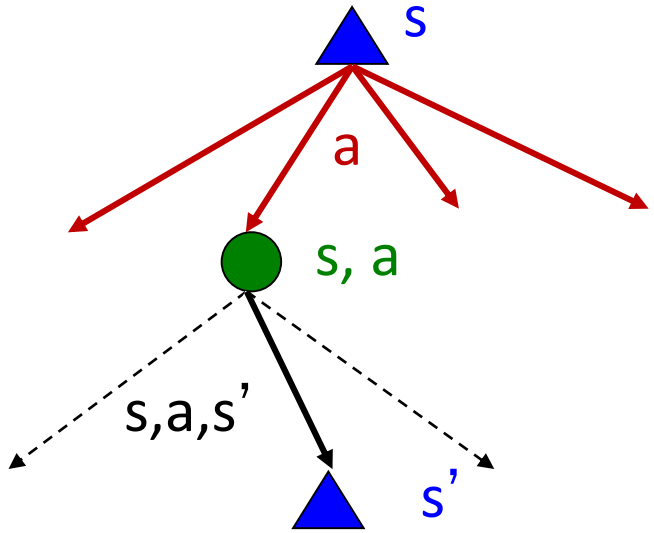
Policy Iteration

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the 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
- This is policy iteration
  - ✓ It's still optimal!
  - ✓ Can converge (much) faster under some conditions

Policy Iteration

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

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

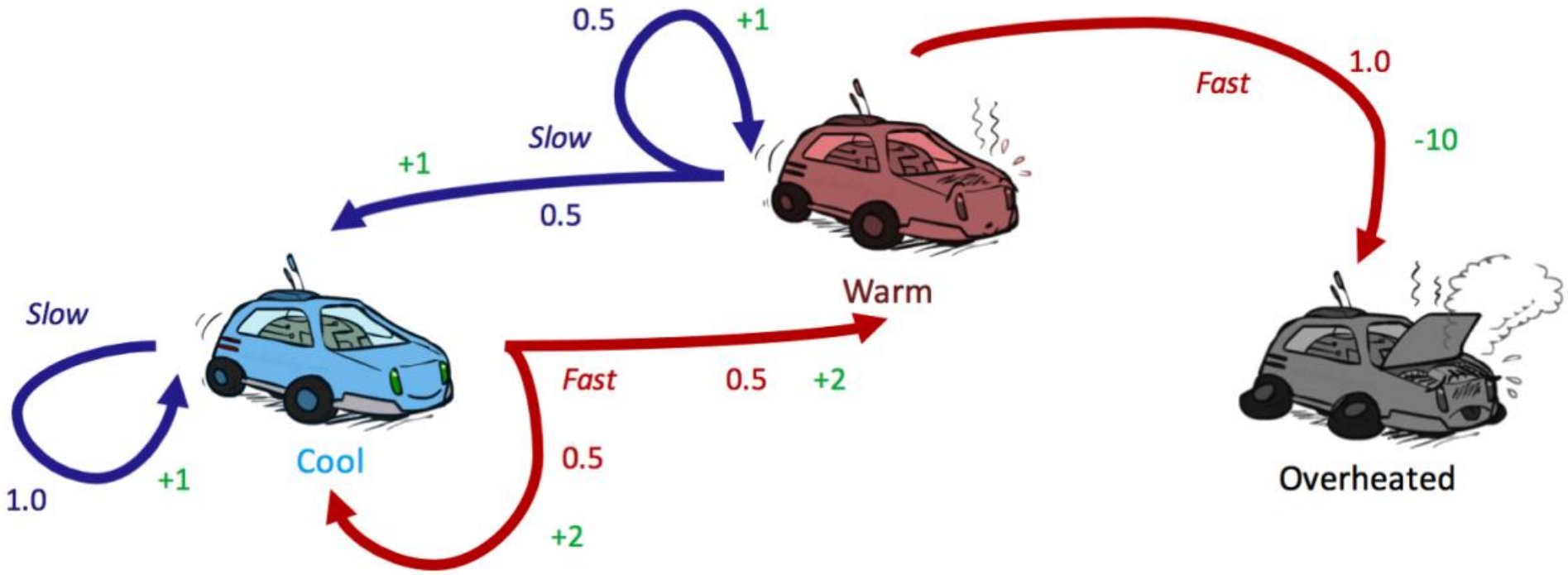
- Improvement: For fixed values, get a better policy using policy extraction
  - ✓ One-step look-ahead:

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

Policy Iteration: example 1

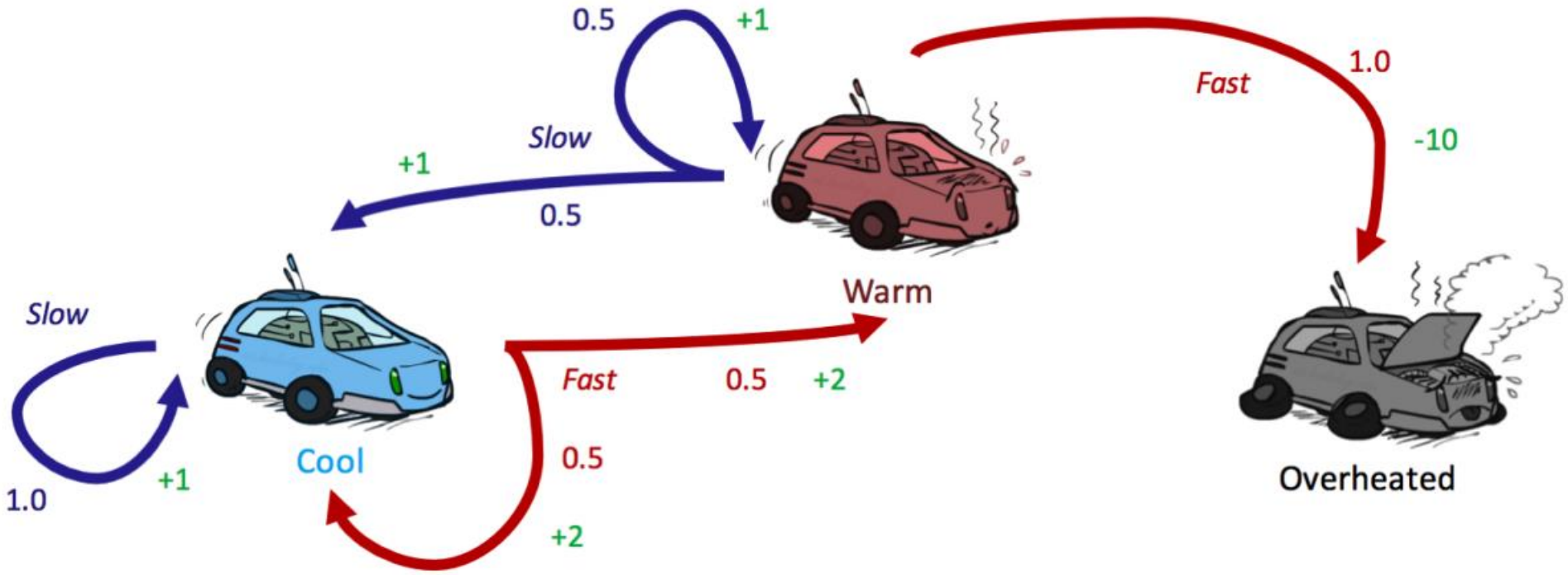
We start with an initial policy of *Always go slow*:

	cool	warm	overheated
π_0	*slow*	*slow*	–



Policy Iteration: example 1

The next step is to run a round of policy evaluation on π_0 :



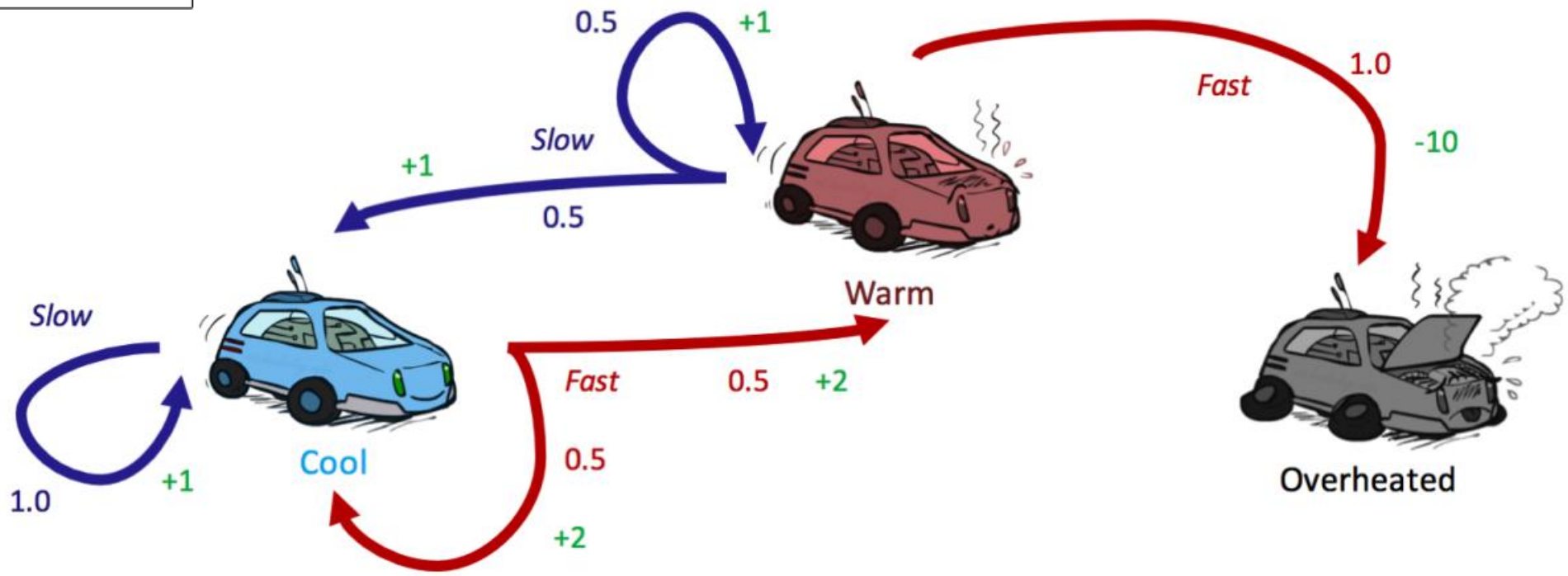
Policy Iteration: example 1

The next step is to run a round of policy evaluation on π_0 :

$$V^{\pi_0}(cool) = 1 \cdot [1 + 0.5 \cdot V^{\pi_0}(cool)]$$
$$V^{\pi_0}(warm) = 0.5 \cdot [1 + 0.5 \cdot V^{\pi_0}(cool)] + 0.5 \cdot [1 + 0.5 \cdot V^{\pi_0}(warm)]$$

Solving this system of equations for $V^{\pi_0}(cool)$ and $V^{\pi_0}(warm)$ yields:

	cool	warm	overheated
V^{π_0}			0



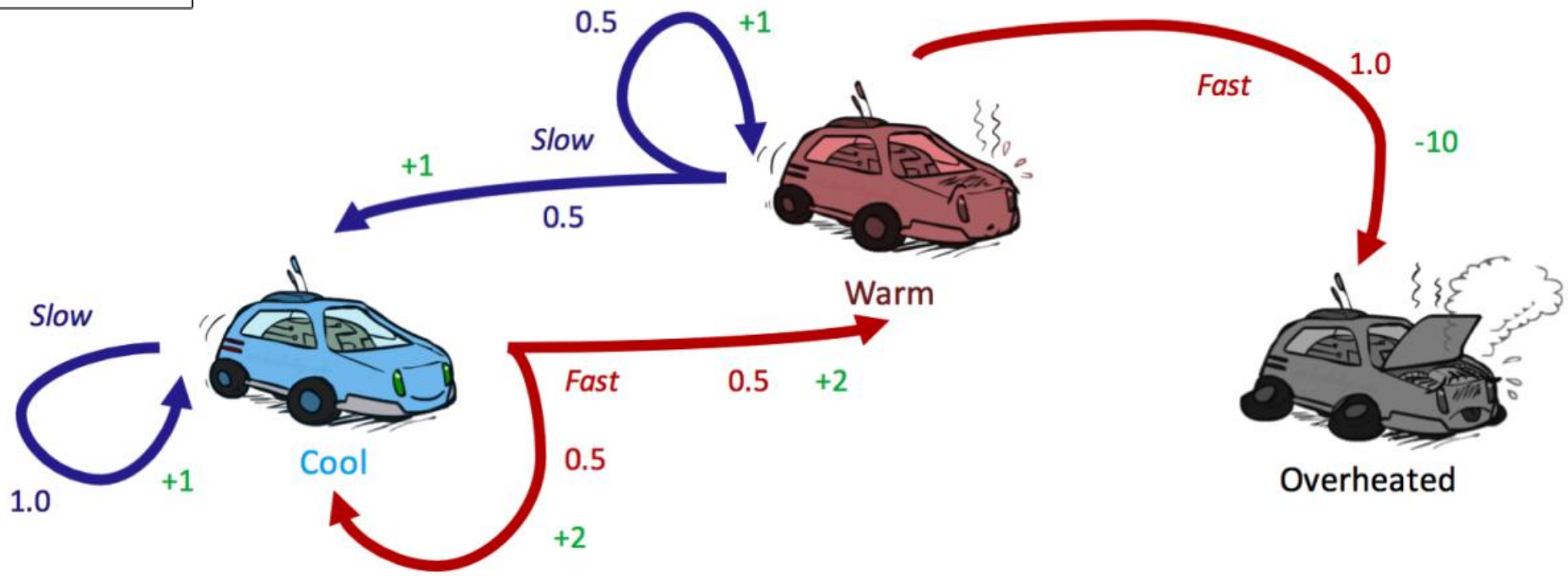
Policy Iteration: example 1

The next step is to run a round of policy evaluation on π_0 :

$$V^{\pi_0}(cool) = 1 \cdot [1 + 0.5 \cdot V^{\pi_0}(cool)]$$
$$V^{\pi_0}(warm) = 0.5 \cdot [1 + 0.5 \cdot V^{\pi_0}(cool)] + 0.5 \cdot [1 + 0.5 \cdot V^{\pi_0}(warm)]$$

Solving this system of equations for $V^{\pi_0}(cool)$ and $V^{\pi_0}(warm)$ yields:

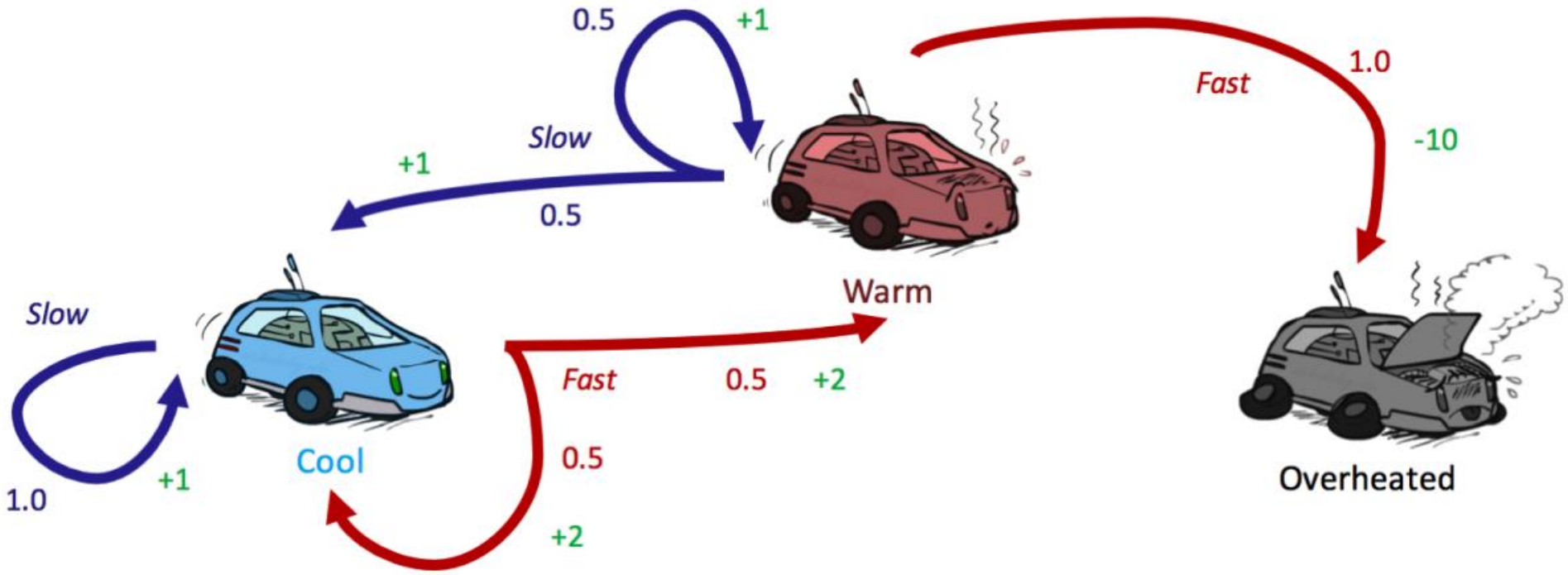
	cool	warm	overheated
V^{π_0}	2	2	0



Policy Iteration: example 1

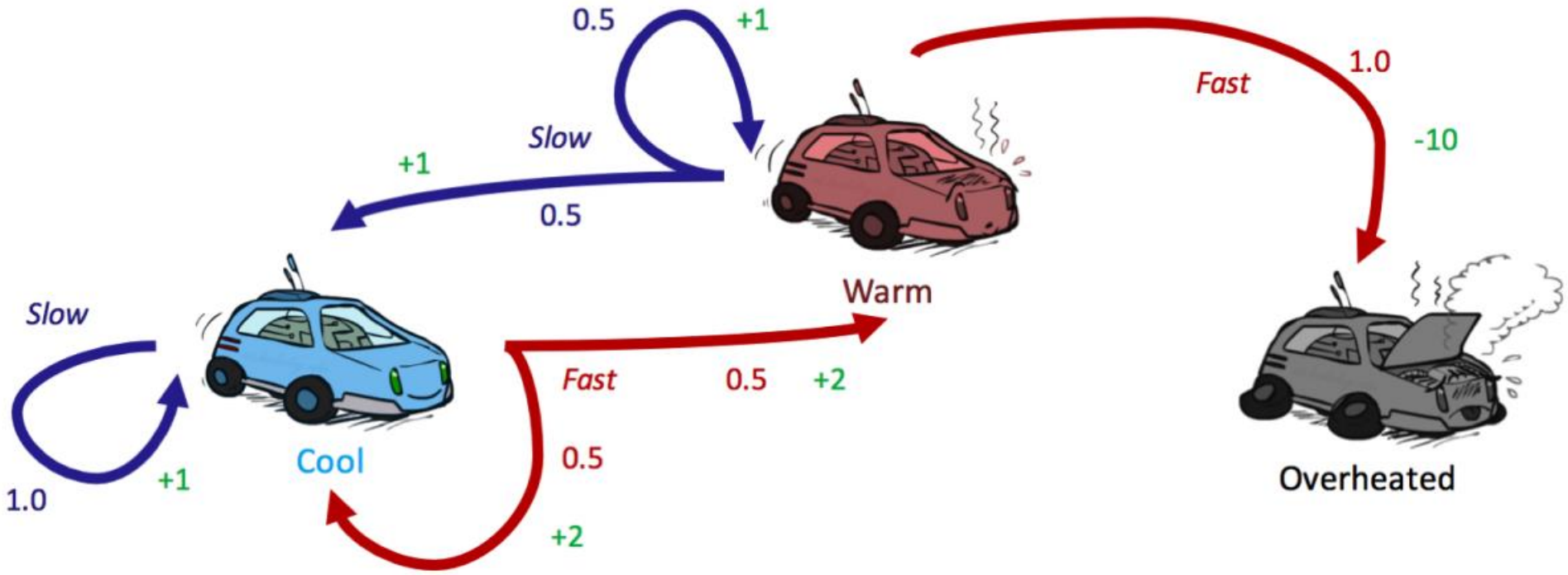
We can now run policy extraction with these values:

$$
\begin{aligned}
\pi_1(cool) &= \text{argmax}\{slow : 1 \cdot [1 + 0.5 \cdot 2],\ fast : 0.5 \cdot [2 + 0.5 \cdot 2] + 0.5 \cdot [2 + 0.5 \cdot 2]\} \\
&= \text{argmax}\{slow : 2,\ fast : 3\} \\
&= \boxed{fast} \\
\pi_1(warm) &= \text{argmax}\{slow : 0.5 \cdot [1 + 0.5 \cdot 2] + 0.5 \cdot [1 + 0.5 \cdot 2],\ fast : 1 \cdot [-10 + 0.5 \cdot 0]\} \\
&= \text{argmax}\{slow : 3,\ fast : -10\} \\
&= \boxed{slow}
\end{aligned}
$$



Policy Iteration: example 1

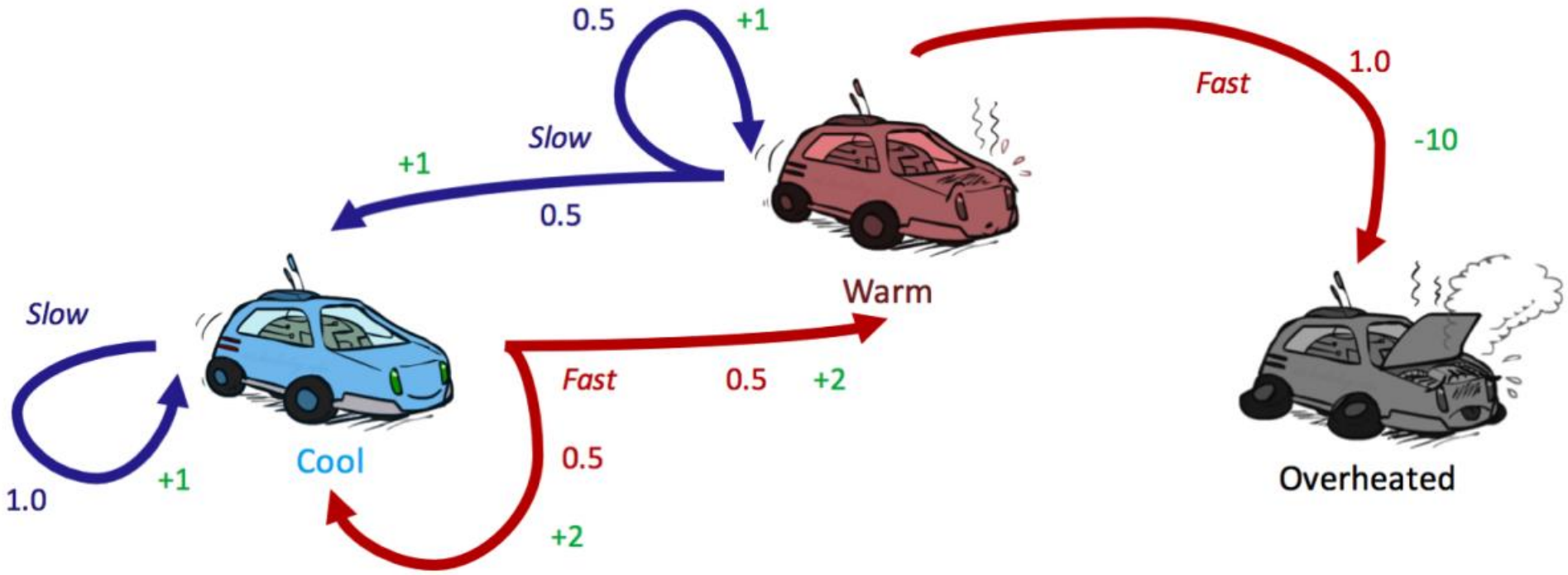
	cool	warm
π_0	*slow*	*slow*
π_1	*fast*	*slow*



Policy Iteration: example 1

Running policy iteration for a second round yields $\pi_2(cool) = fast$ and $\pi_2(warm) = slow$. Since this is the same policy as π_1, we can conclude that $\pi_1 = \pi_2 = \pi^*$. Verify this for practice!

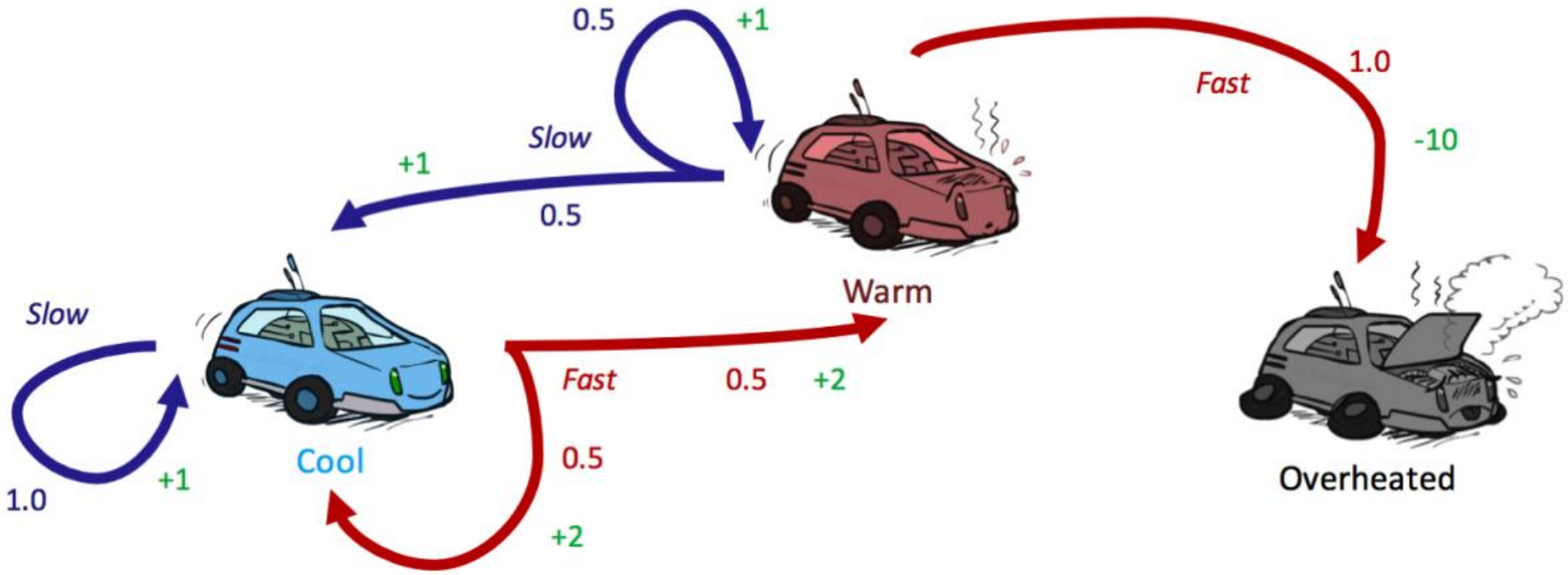
	cool	warm
π_0	*slow*	*slow*
π_1	*fast*	*slow*
π_2	*fast*	*slow*



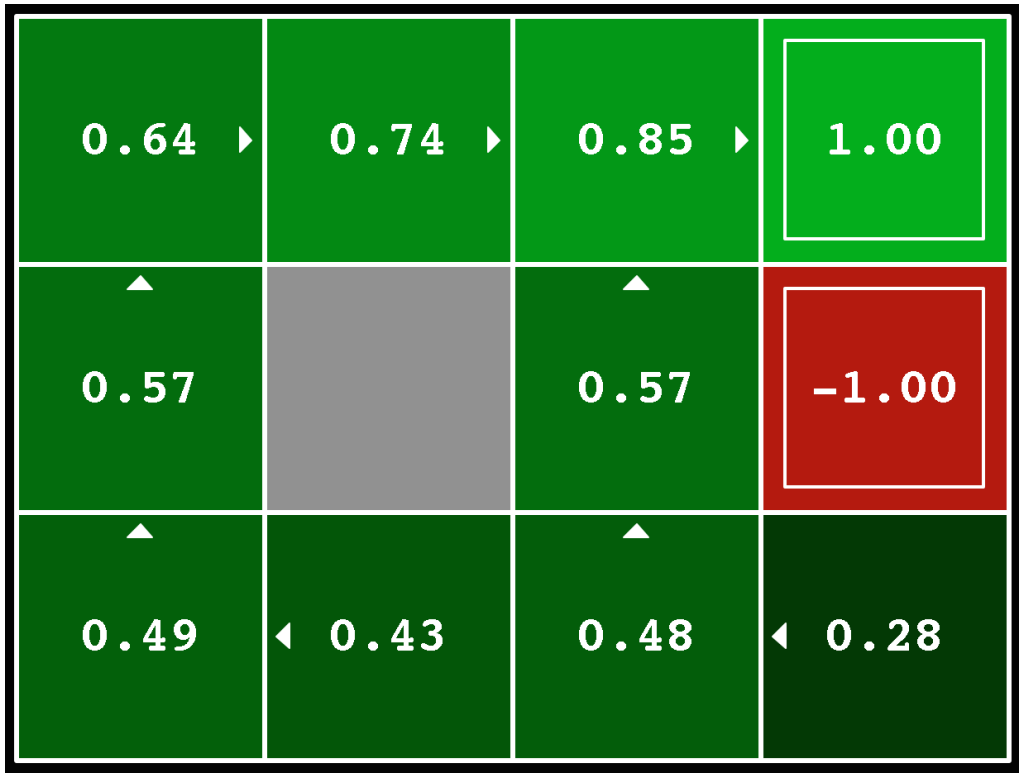
Policy Iteration: example 2

	cool	warm	overheated
π_0	Fast	fast	—

- Given the initial policy
- Use this to obtain optimal policy.



Policy Iteration: example 3



- Given: converged values of each state some policy
- Use this to obtain optimal policy.

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