# CS 3368 Introduction to Artificial Intelligence

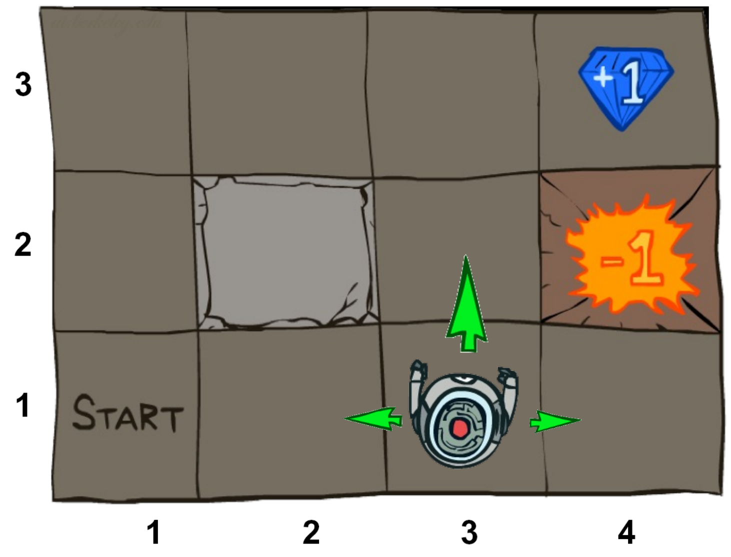
## Markov Decision Process

Department of Computer Science
Texas Tech University

- Instructor: Jingjing Yao
- Email: jingjing.yao@ttu.edu
- Office: EC 306F
- Office hours: 10 - 11 am, Tuesday and Thursday

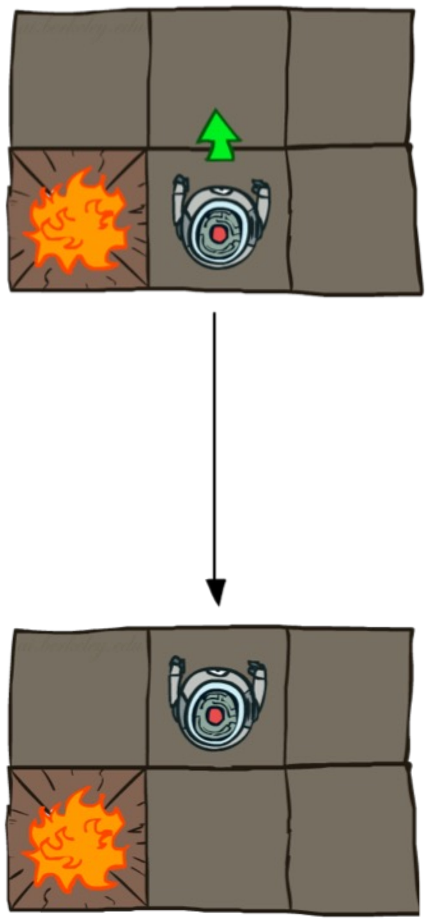
# Example: Grid World

- 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 reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards

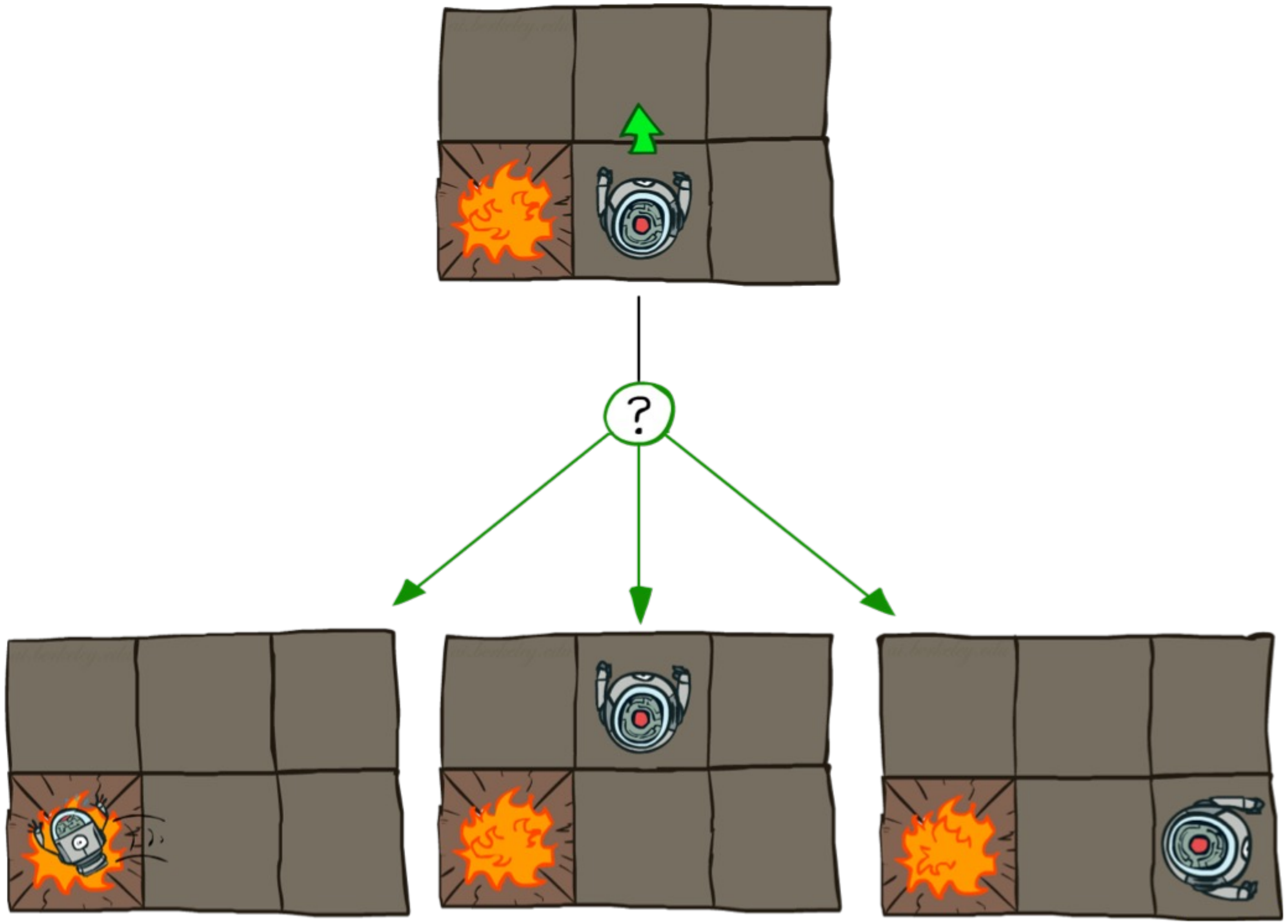


# Grid World Actions

## Deterministic Grid World

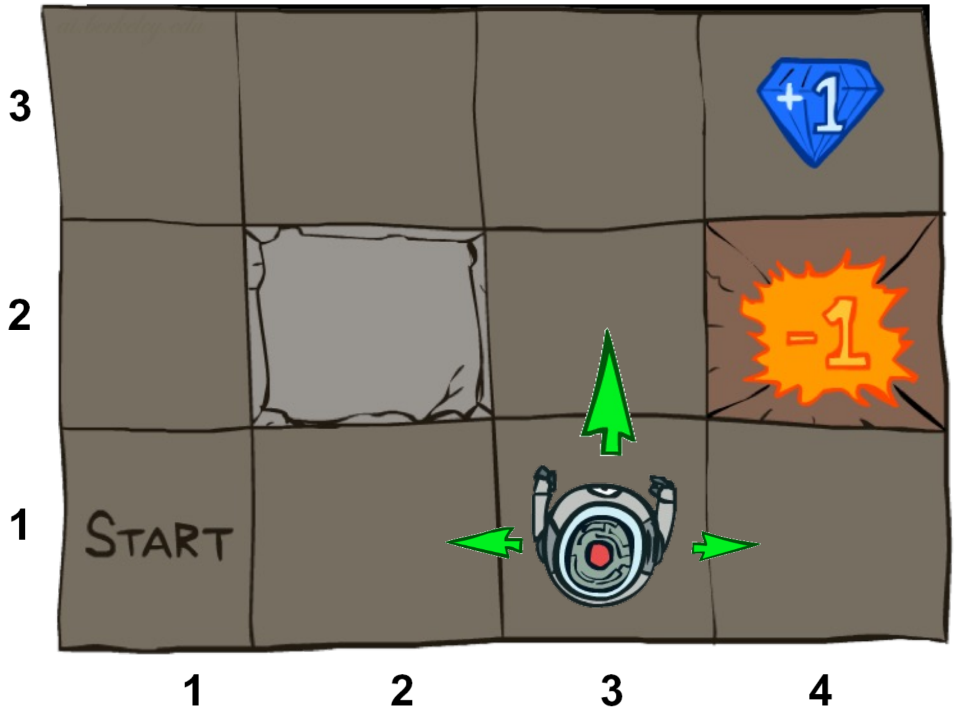


## Stochastic Grid World



# Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s' | s, a)$
    - 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

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

$$=$$

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

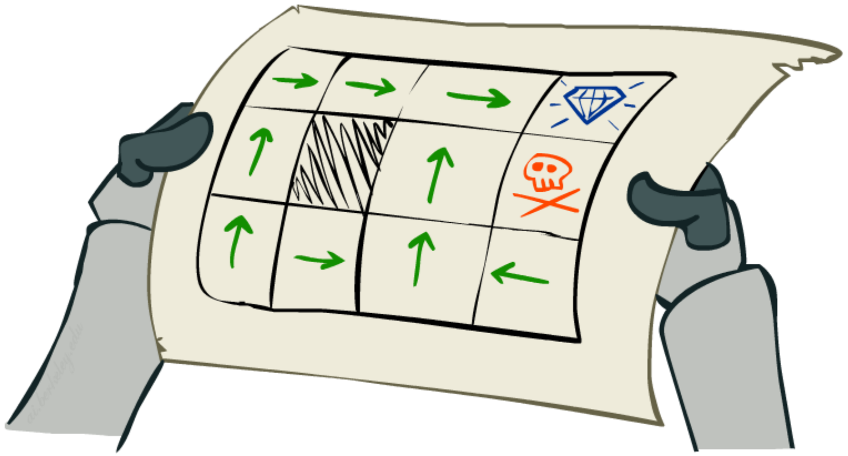
- This is just like search, where the successor function could only depend on the current state (not the history)

Andrey Markov
(1856-1922)

# Policies

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

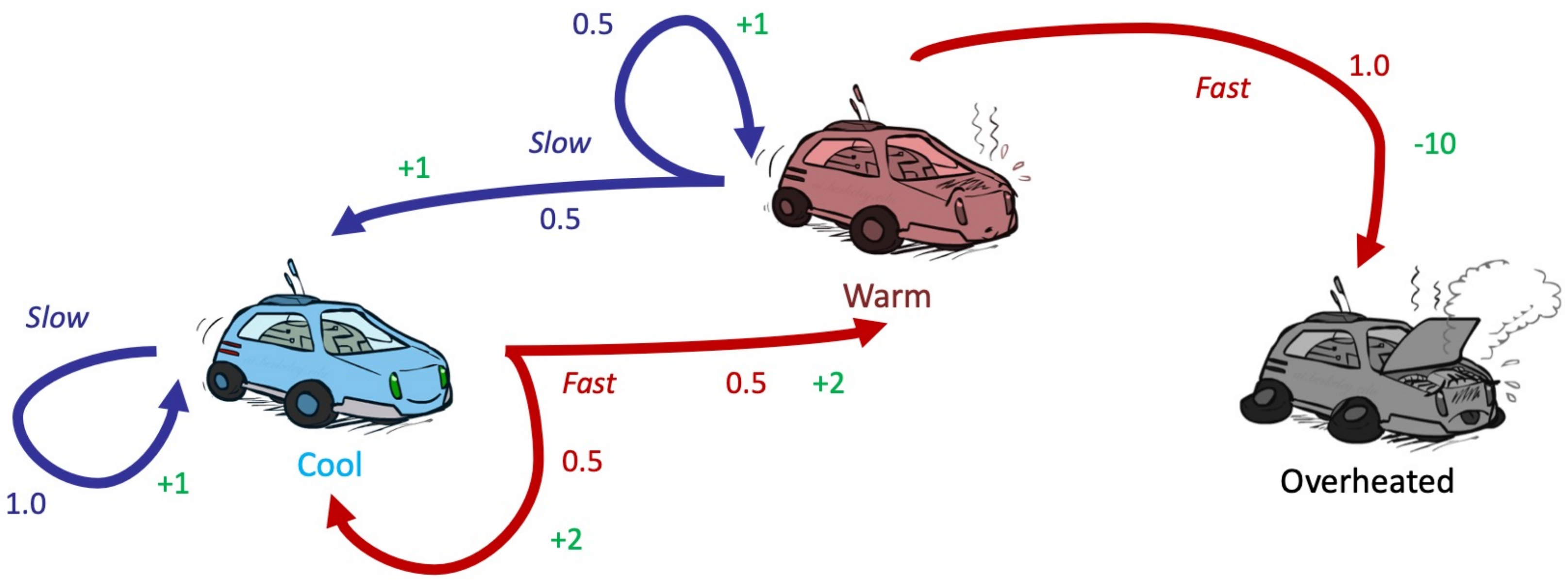
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy $\pi$ gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed



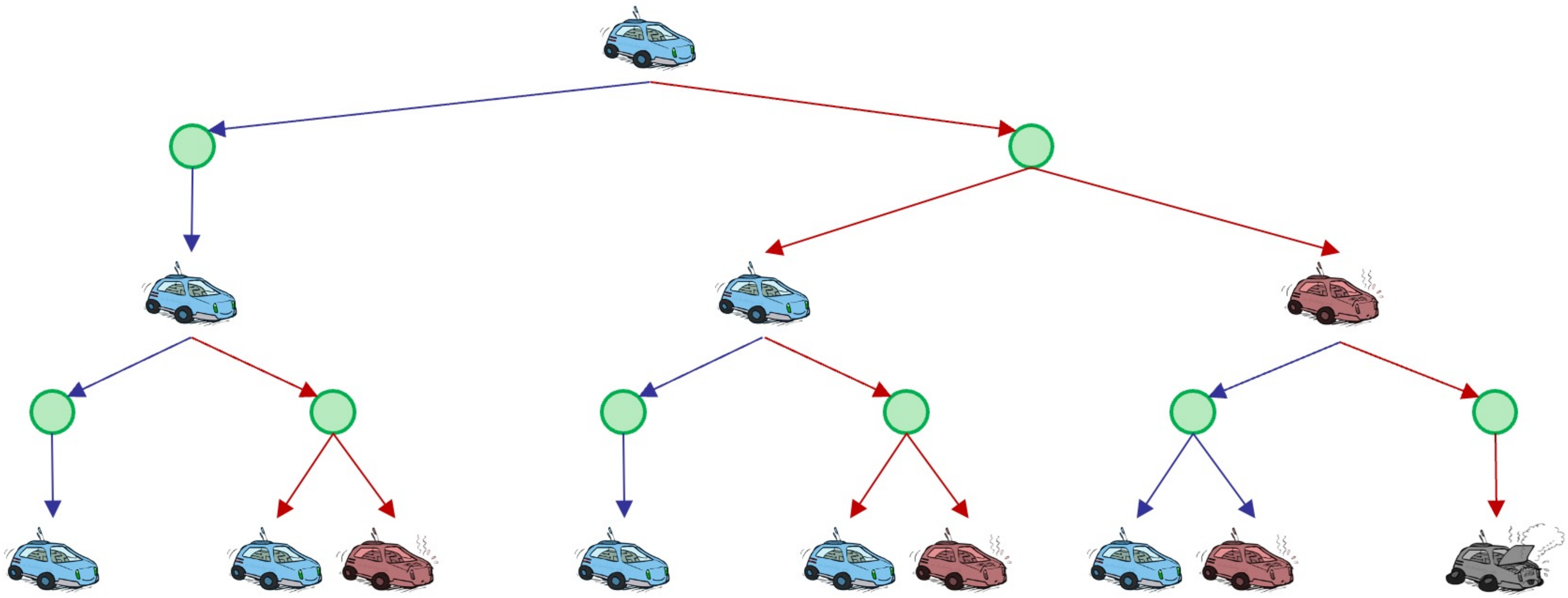
Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

# Example: Racing

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

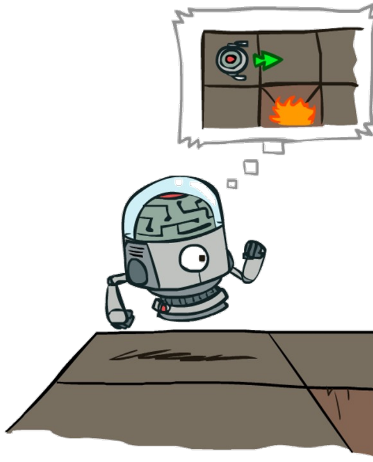


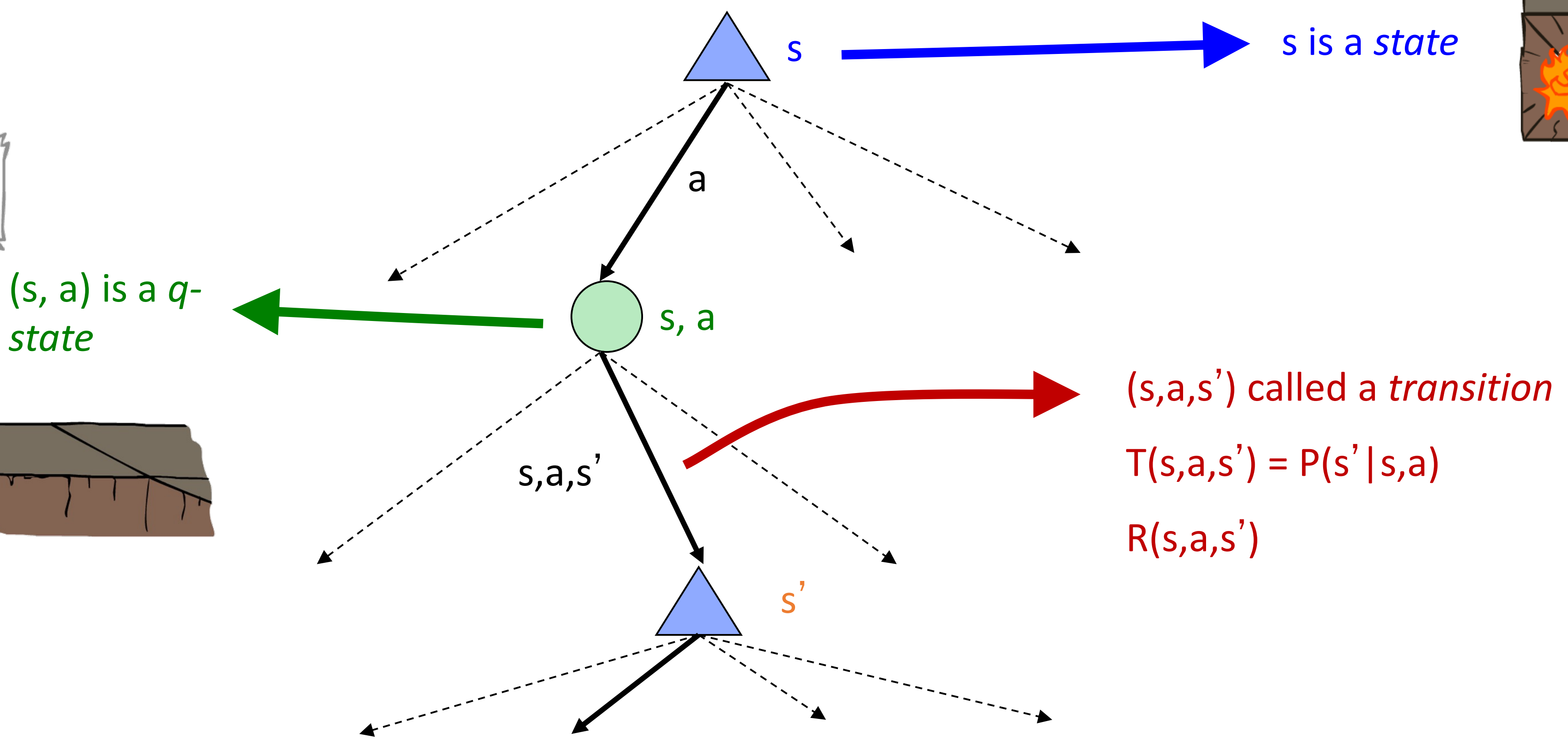
# Racing Search Tree

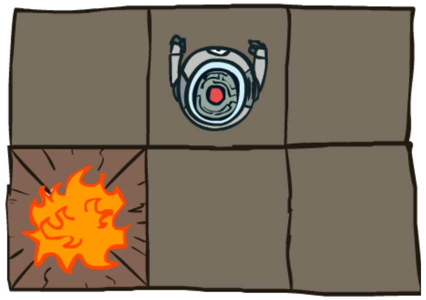


# MDP Search Trees

- Each MDP state projects an expectimax-like search tree





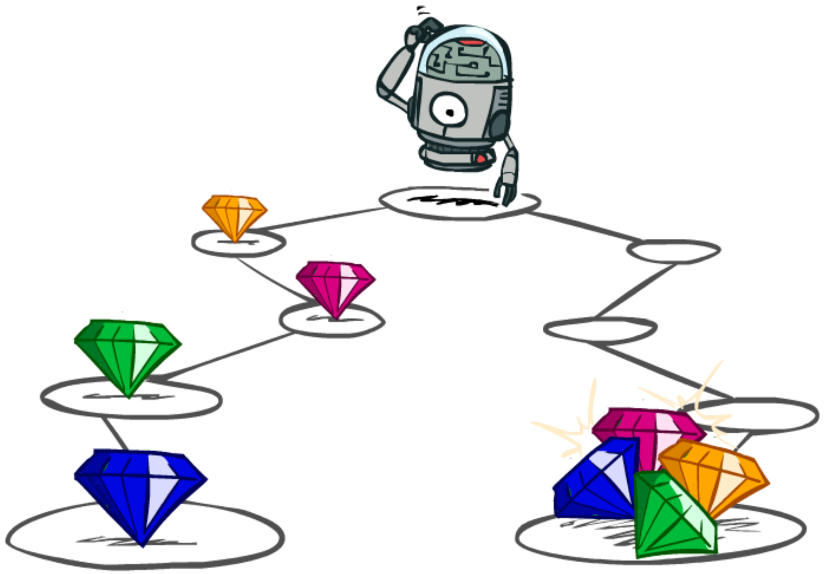



# Utilities of Sequences

- 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

$1$

Worth Now

$\gamma$

Worth Next Step

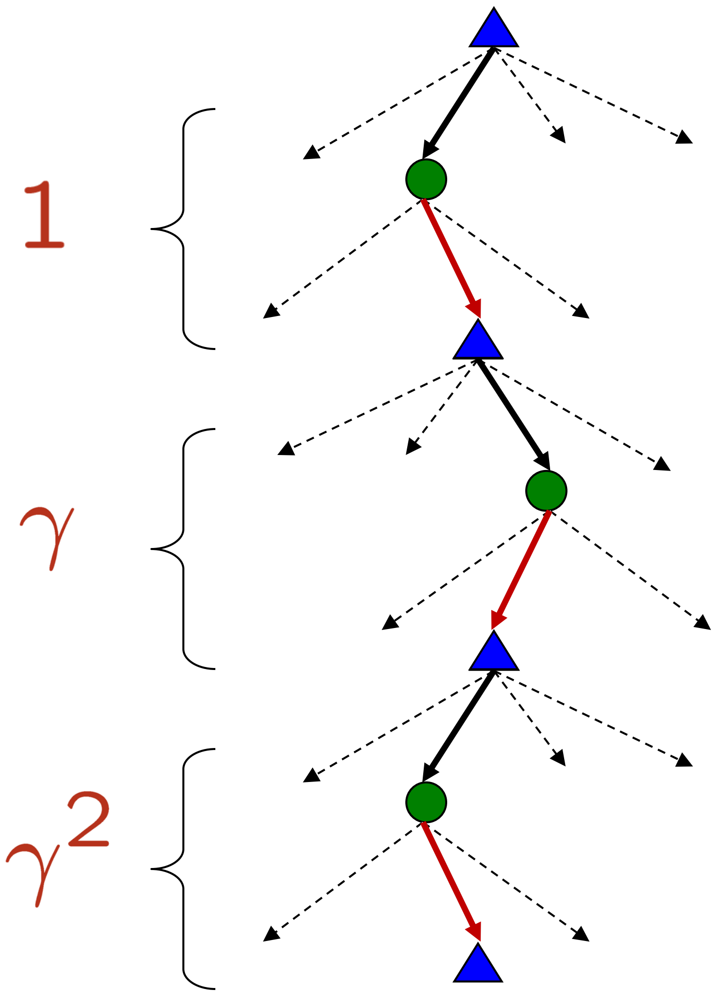
$\gamma^2$

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])
- Discounted utility:

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

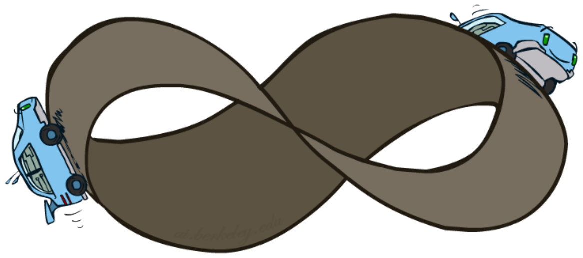


# Infinite Utilities?

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

- Solutions:
  - Finite horizon: Terminate episodes after a fixed T steps (e.g. life)
  - Discounting: use $0 < \gamma < 1$

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

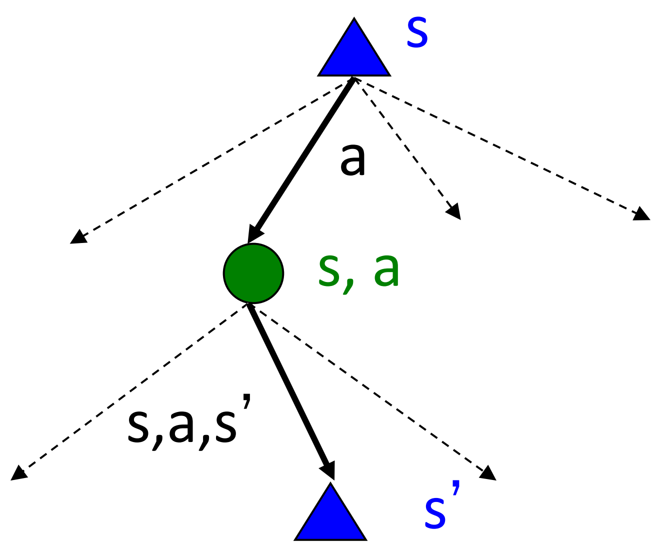


    - Smaller $\gamma$ means smaller "horizon" – shorter term focus
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

# Summary: Defining MDPs

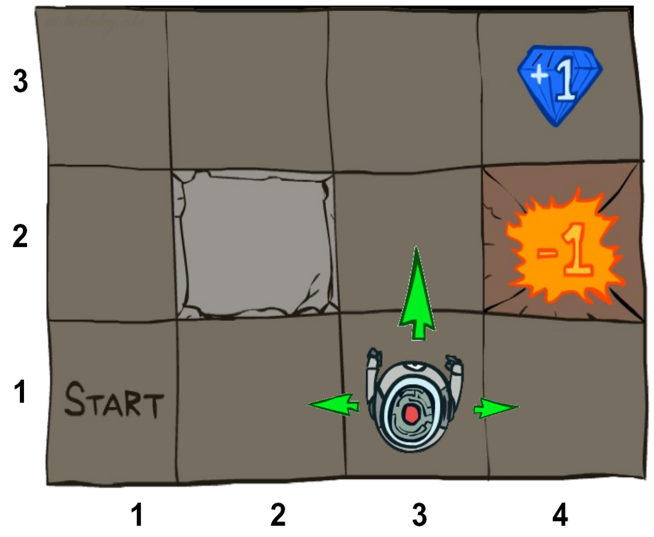
- Markov decision processes:
  - Set of states S
  - Start state $s_0$
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount $\gamma$)

- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards



# MDP for Grid World

- Actions: up, down, left, and right
  - Every action is possible in every state
- The transition model P(s'|s,a)
  - An action achieves its intended effect with probability 0.8
  - An action leads to a 90-degree left turn with probability 0.1
  - An action leads to a 90-degree right turn with probability 0.1
  - If the robot bumps into a wall, it stays in the same square
- The reward function R(s) is the reward of entering state s
  - R(s24) = -1
  - R(s34) = 1
  - Otherwise, R(s) = -0.04

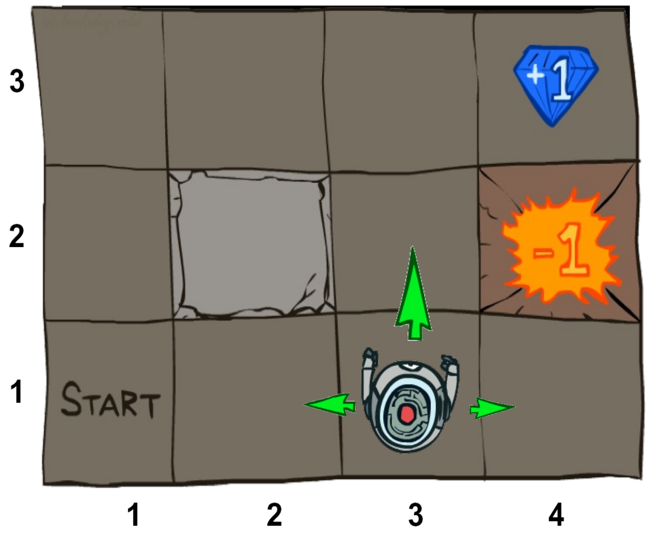


# Grid World Problem

- What should the robot do to maximize its rewards?

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

- Let $s_{ij}$ be the position in row $i$ and column $j$.
- $s_{11}$ is the initial state.
- There is a wall at $s_{22}$.
- $s_{24}$ and $s_{34}$ are goal states.
  The robot escapes the world at either goal state.



# Understand Transition Model

**CQ:** The robot is in $s_{14}$ and tries to move to our right, what is the probability that the robot stays in $s_{14}$?

(A) 0.1

(B) 0.2

(C) 0.8

(D) 0.9

(E) 1.0

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

# A fixed sequence of actions

**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 |

# A fixed sequence of actions

**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 optimal policies of the grid world

- 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 |

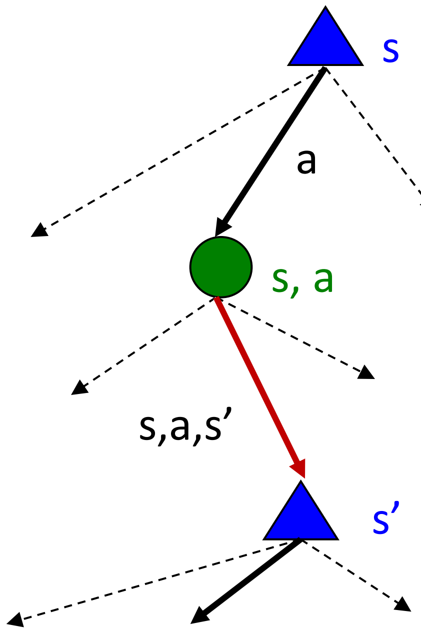
# The optimal policy

- When the reward function is
- $R(s) < -1.6284$
- $R(s) > -0.02$

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

# 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*

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

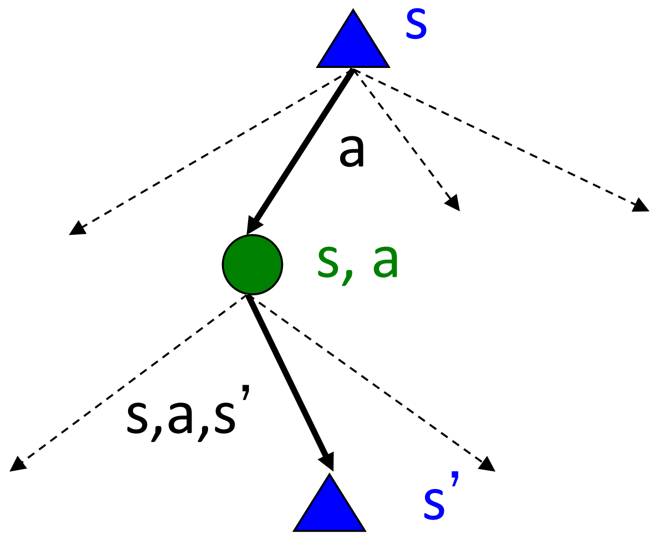
# The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

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

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

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




- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

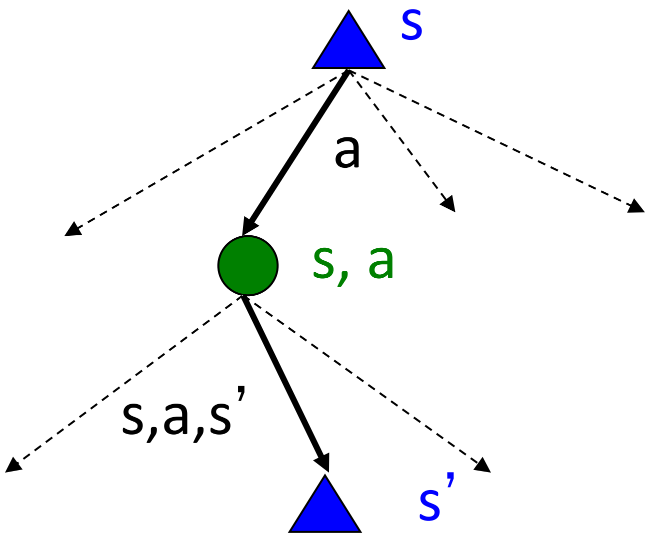
# Values of States

- 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 (Bellman equations)

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

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

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



# Calculate the Optimal Policy

- What is my expected utility if I am in state s and take action a?

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

- In state s, choose an action that maximizes my expected utility

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

# Determine optimal action

**CQ:** What is the optimal action for state $s_{13}$?
(A) Up (B) Down (C) Left (D) Right

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) V^*(s')$$

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

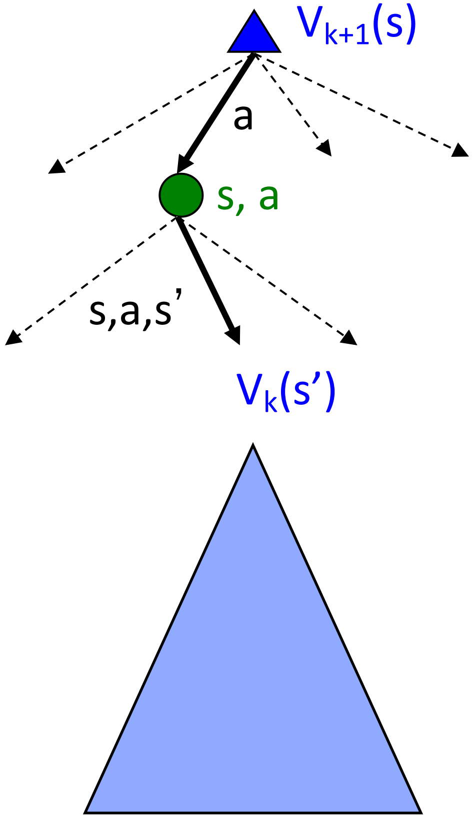
The values of $V^*(s)$ are given below.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |
| 2 | 0.762 | X | 0.660 | -1 |
| 3 | 0.812 | 0.868 | 0.918 | +1 |

# Value Iteration

- Start with $V_0(s) = 0$
- Given vector of $V_k(s)$ values, do one step of expectimax from each state:

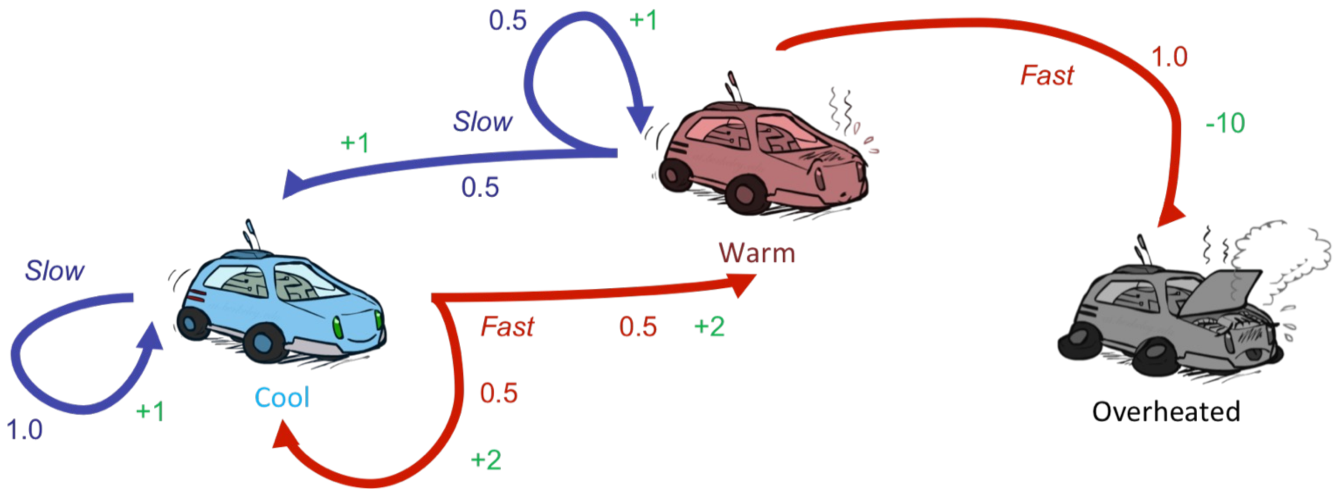
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$




- Repeat until convergence
- 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

| | | | |
|---|---|---|---|
| $V_2$ | 3.5 | 2.5 | 0 |
| $V_1$ | 2 | 1 | 0 |
| $V_0$ | 0 | 0 | 0 |




*Assume no discount*

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

# Example

- Write down the Bellman equation for V*(s11)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |
| 2 | 0.762 | X | 0.660 | -1 |
| 3 | 0.812 | 0.868 | 0.918 | +1 |

# Example

**CQ:** What is $V_1(s_{23})$?

(A) $(-\infty, 0)$ (B) $[0, 0.25)$ (C) $[0.25, 0.5)$
(D) $[0.5, 0.75)$ (E) $[0.75, 1]$

$V_0(s)$:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | X | 0 | -1 |
| 3 | 0 | 0 | 0 | +1 |

# The Values of $V_1(s)$

$V_0(s)$:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | X | 0 | -1 |
| 3 | 0 | 0 | 0 | +1 |

$V_1(s)$:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | X | | -1 |
| 3 | | | | +1 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | −0.04 | −0.04 | −0.04 | −0.04 |
| 2 | −0.04 | X | −0.04 | -1 |
| 3 | −0.04 | −0.04 | 0.76 | +1 |

$V_1(s)$:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $-0.04$ | $-0.04$ | $-0.04$ | $-0.04$ |
| 2 | $-0.04$ | X | $-0.04$ | -1 |
| 3 | $-0.04$ | $-0.04$ | 0.76 | +1 |

$V_2(s)$:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $-0.08$ | $-0.08$ | $-0.08$ | $-0.08$ |
| 2 | $-0.08$ | X | 0.464 | -1 |
| 3 | $-0.08$ | 0.56 | 0.832 | +1 |

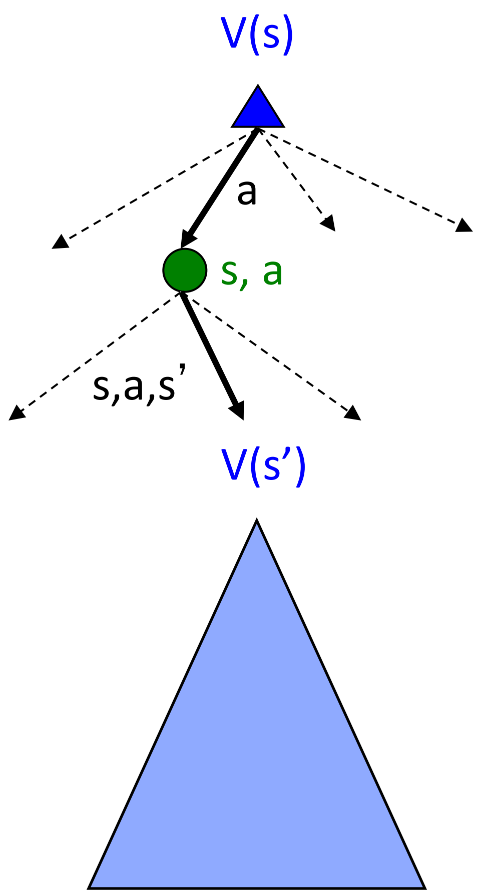
# Value Iteration Summary

- Bellman equations characterize the optimal values:

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

- Value iteration computes them:

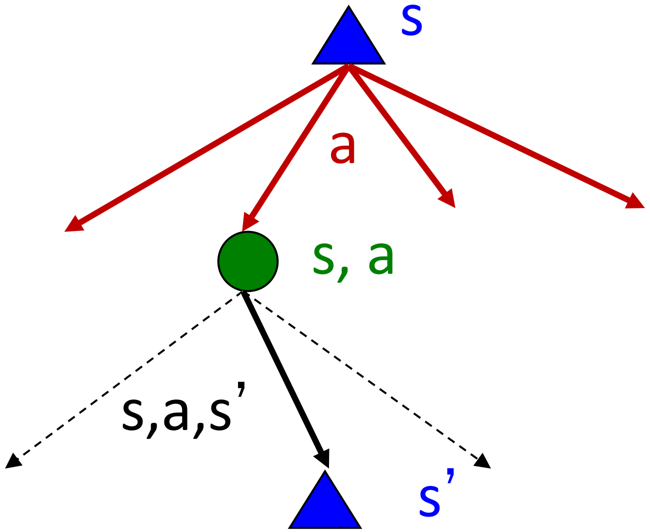
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$



# Problems with Value Iteration

- Value iteration repeats the Bellman updates:

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

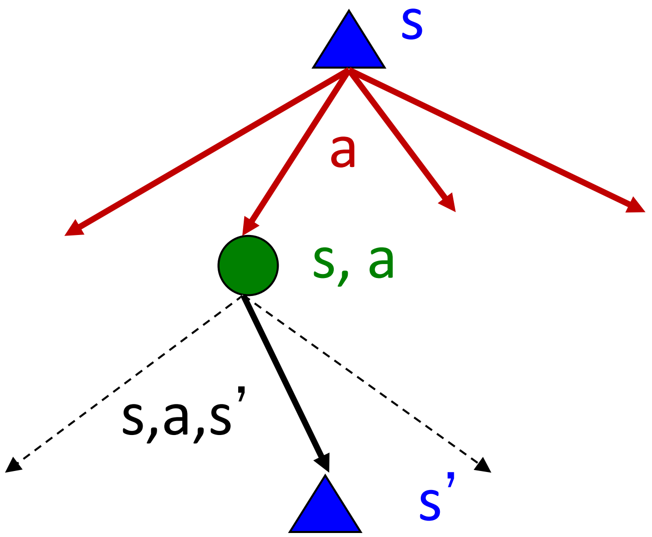



- Problem 1: It's slow – $O(S^2A)$ per iteration

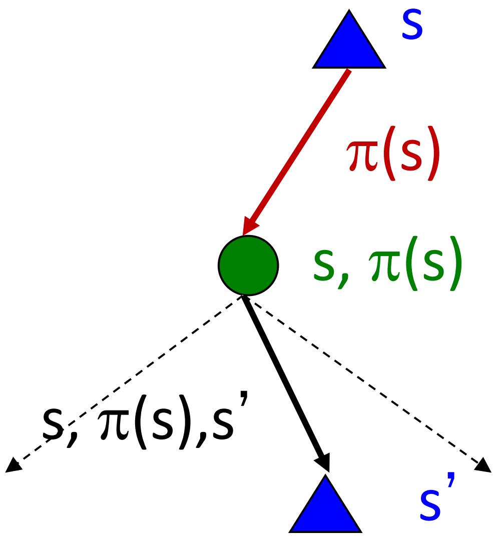
- Problem 2: The policy often converges long before the values

# Fixed Policies

Do the optimal action




Do what $\pi$ says to do




- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - … though the tree's value would depend on which policy we fixed

# Utilities for a Fixed Policy

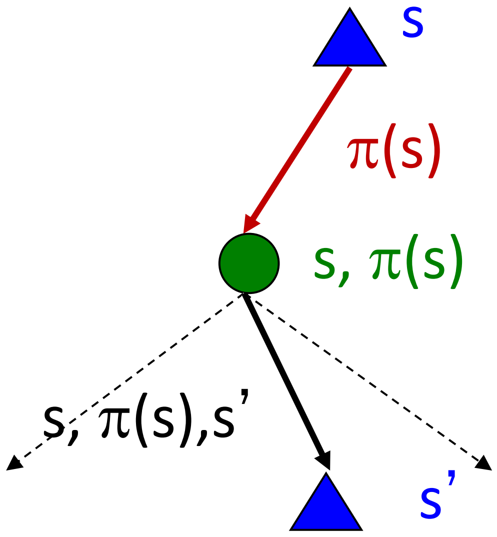
- Another basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy

- Define the utility of a state s, under a fixed policy $\pi$:

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

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

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



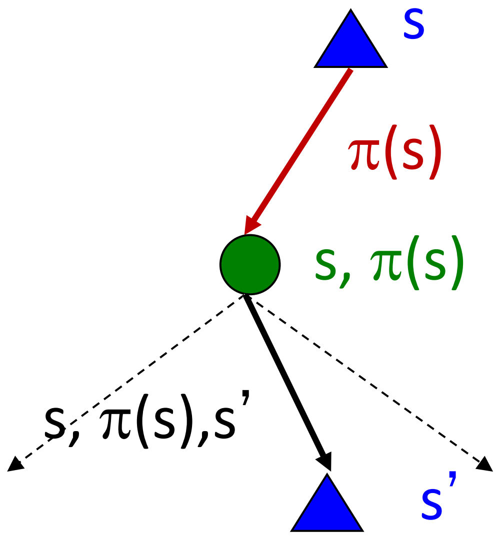
# Policy Evaluation

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

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

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

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




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

# Computing Actions from Values

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

- How should we act?

- We need to do a mini-expectimax (one step)

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

- 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?

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

- 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 until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal
  - Can converge (much) faster

# 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)

# Summary: MDP Algorithms

- So you want to….
  - Compute optimal values: use value iteration or policy iteration
  - Compute values for a particular policy: use policy evaluation
  - Turn your values into a policy: use policy extraction
- These all look the same!
  - They basically are – they are all variations of Bellman updates
  - They all use one-step lookahead expectimax fragments
  - They differ only in whether we plug in a fixed policy or max over actions

# Example

- Reward of entering a non-goal state = -0.04
- Transition probabilities: 0.8 in intended direction, 0.1 to the left and 0.1 to the right
- Execute Policy Iteration
- Initial policy π(s11) = right; π(s21) = right

| | 1 | 2 |
|---|---|---|
| 1 | | **+1** |
| 2 | | -1 |

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

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