COMP 341 Intro to AI
Making Sequential Decisions under Action Uncertainty

Asst. Prof. Barış Akgün

Koç University

Maximum Expected Utility

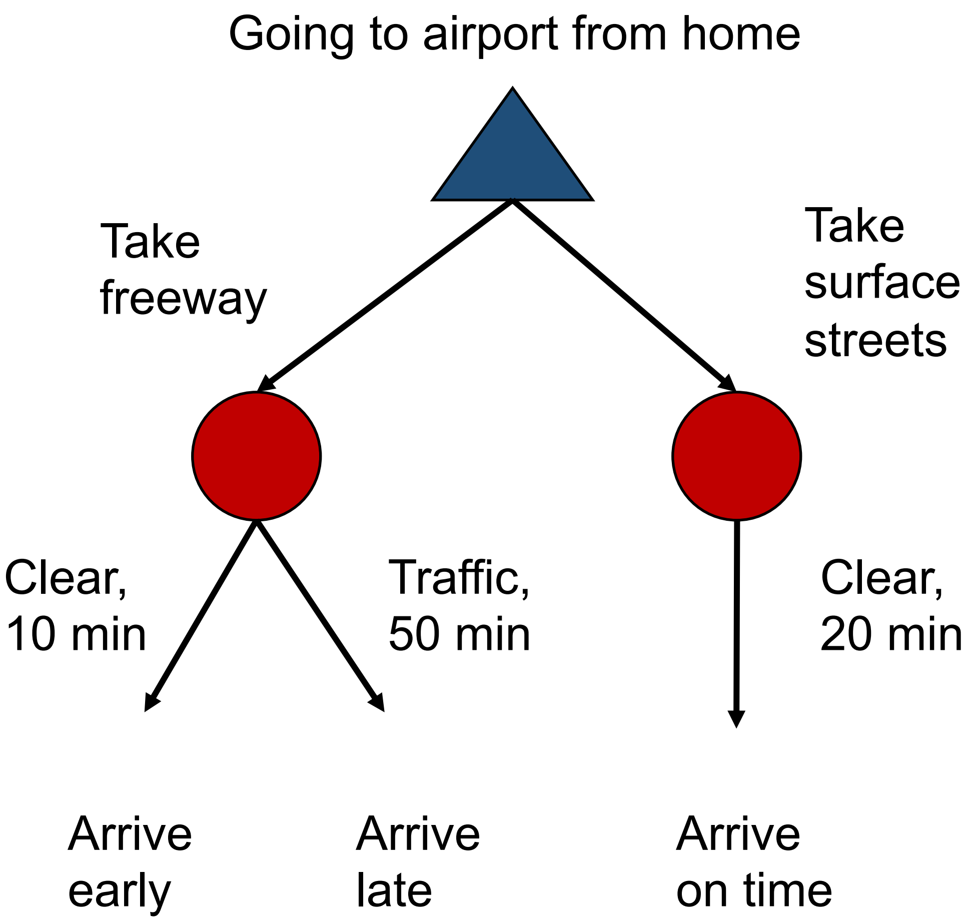
- A rational agent should choose the action which maximizes its expected utility, given its knowledge

- Questions:
  - Where do utilities come from?
  - What do utilities represent?
  - Why expected utility?

Utilities and Unknown Outcomes

- One way has a chance to be better or worse
- How to decide?
- Which would you pick if you are catching a flight?
- Which if you are picking up a friend?

Assigning relative value to outcomes = Utilities



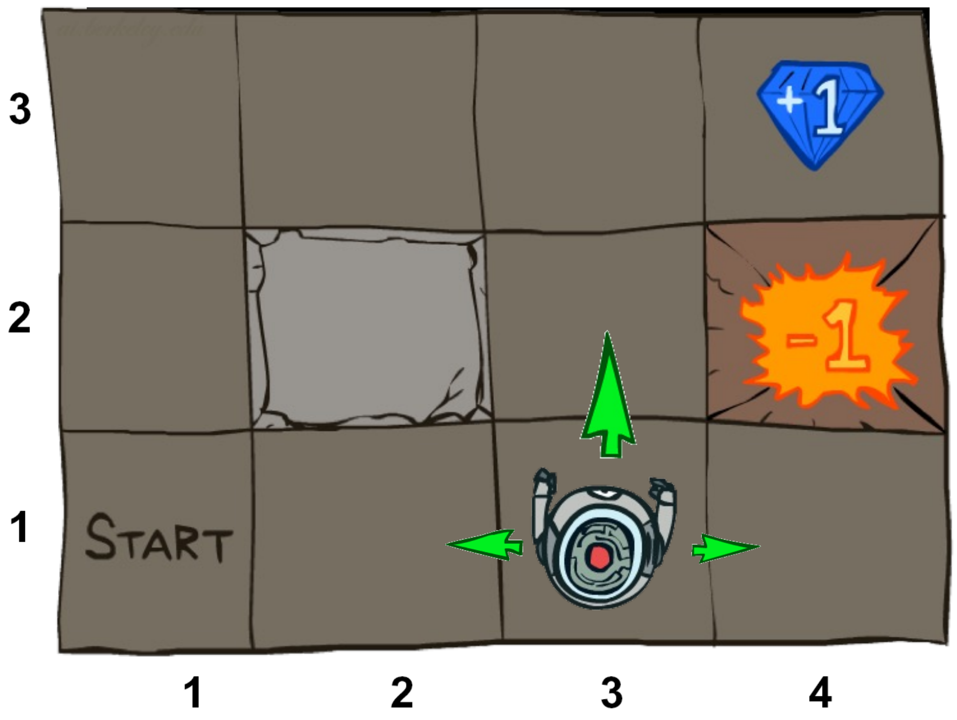
Agent Rational Decisions

- Representing decisions and maximizing utility:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Act to maximize expected rewards over time
  - Can learn how to maximize rewards via Reinforcement Learning
- Examples:
  - Playing a game, reward at the end for winning / losing
  - Vacuuming a house, reward for each piece of dirt picked up
  - Automated taxi, reward for each passenger delivered

- Decision networks: Single (or temporally unrelated) decisions
- RL: Sequential Decisions

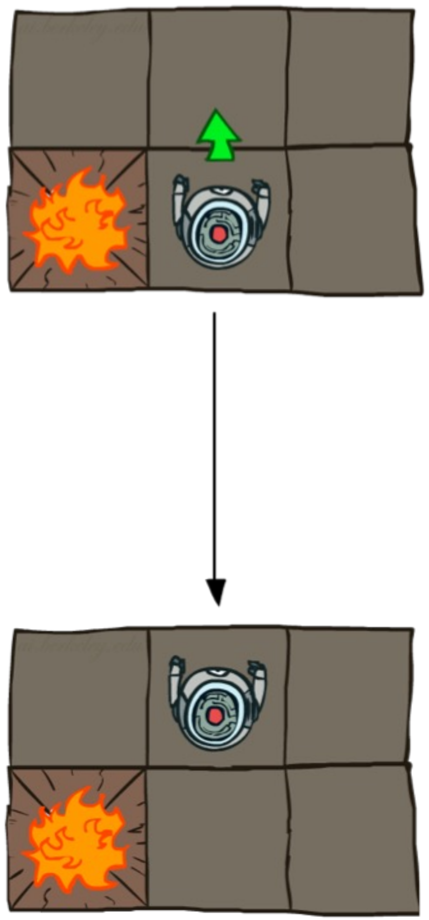
Example: Grid World

- 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)
  - E.g., 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

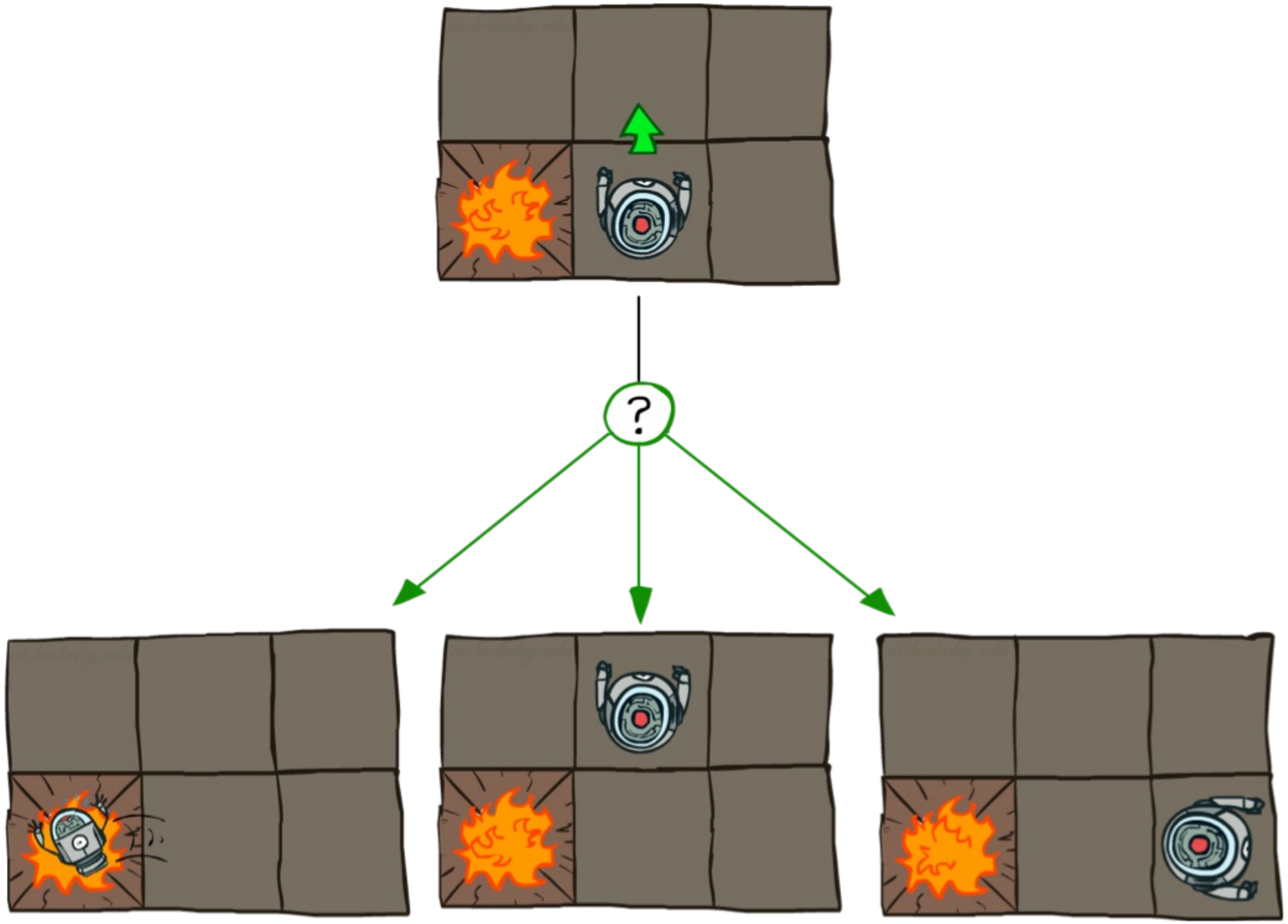


Grid World Actions

Deterministic Grid World

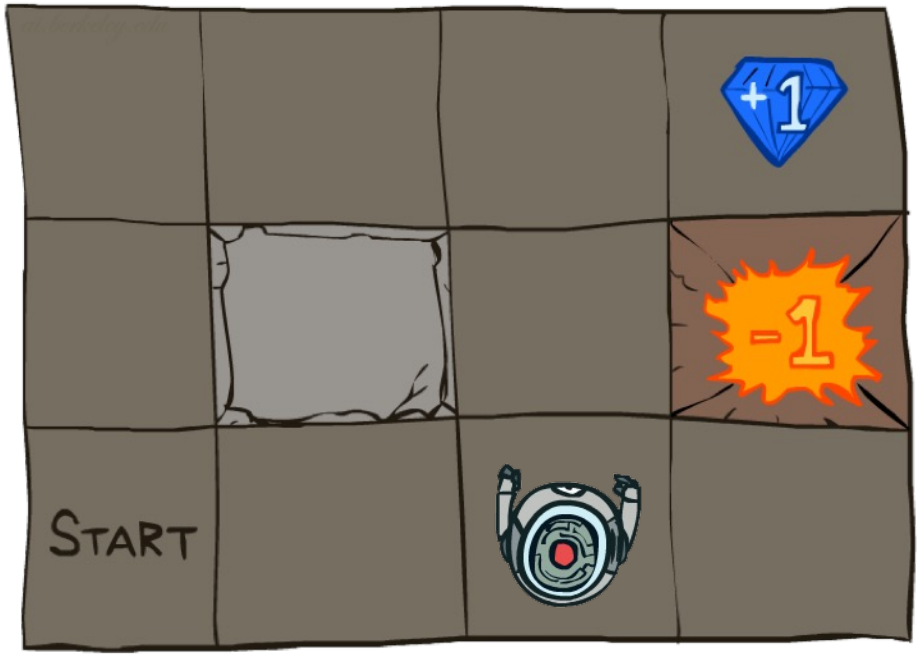


Stochastic Grid World



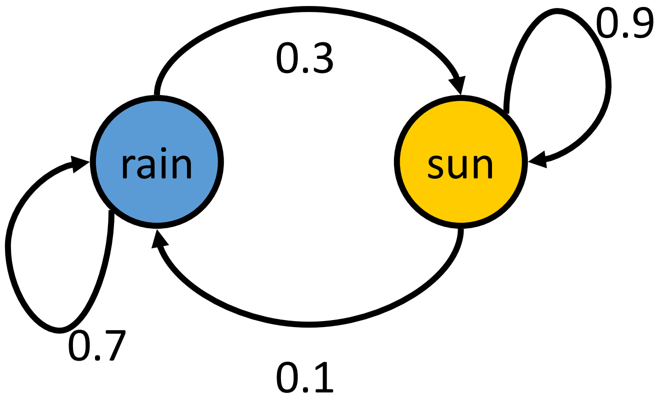
Grid World

- Why not just search?
  - Stochastic Action Outcomes
- Why not use expectimax and re-plan at each state?
  - A valid idea but…
- Computational burden, repeated states, infinite search tree…
- Markov Decision Processes are a good general way to attack this problem
- The solution will be a sort of “search with memory”



Recall Markov Chains

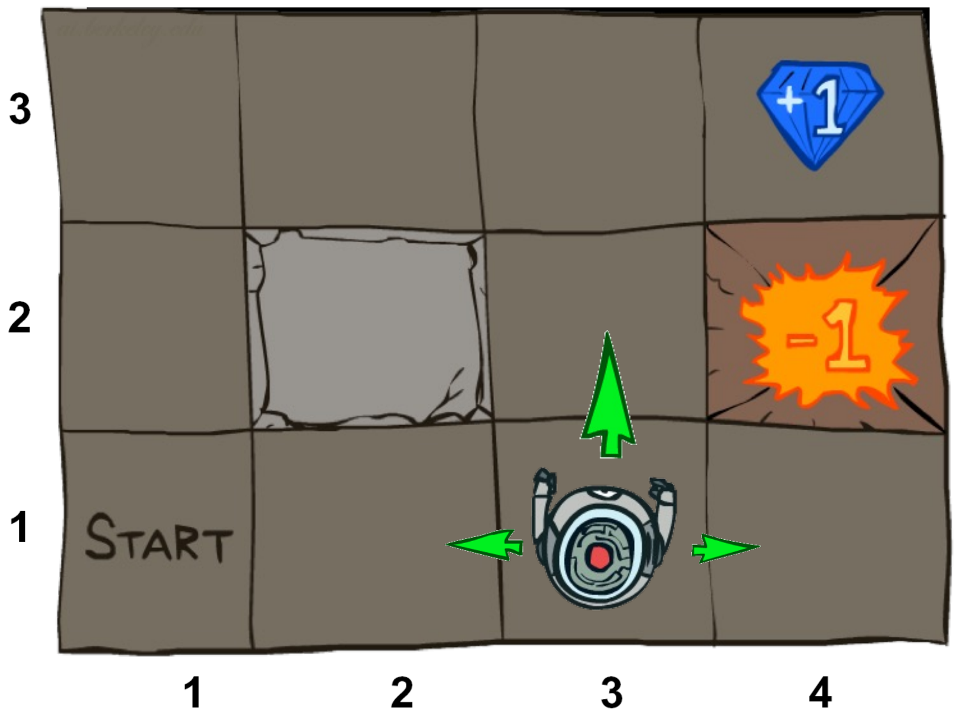
- States, transition model and initial distribution



- Assume you have invented a weather machine
  - The states change, with some uncertainty, based on your actions
- You have energy costs but want to keep it sunny
  - You make action decisions based on rewards

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')
    - Can be stochastic (P(s' | s, a)) or deterministic
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s, a) or R(s') (all are equivalent)
  - 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 Markovian about MDPs?

- Recall: Markov property generally means that given the present state, past and the future are independent
- For MDPs, action outcomes only depend on the current state

$$
\begin{aligned}
\mathrm{P}(\mathrm{S}_{t+1} = s' | \mathrm{S}_t = s_t, \mathrm{A}_t = a_t, \mathrm{S}_{t-1} = s_{t-1}, \mathrm{A}_{t-1} = a_{t-1}, \ldots, \mathrm{S}_0 = s_0) \\
= \mathrm{P}(\mathrm{S}_{t+1} = s' | \mathrm{S}_t = s_t, \mathrm{A}_t = a_t)
\end{aligned}
$$

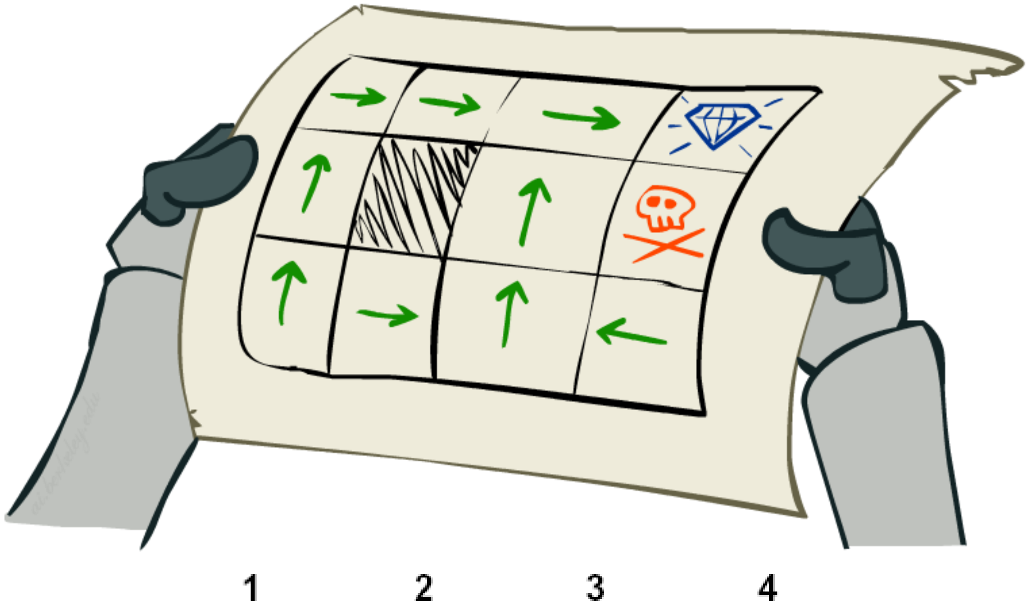
- Just like search where the successor function only depends on the current state

Policies

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

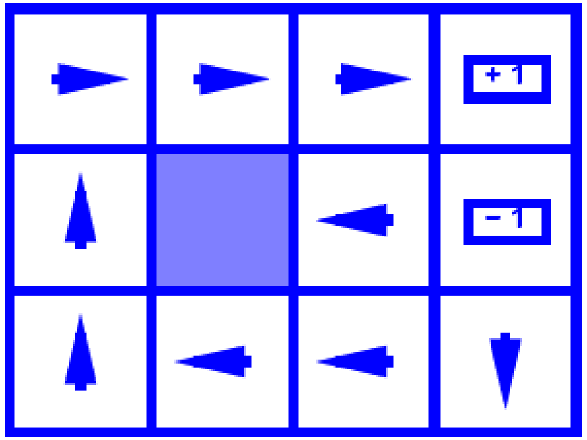
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent

- Expectimax didn't compute entire policies
  - It computed the action for a single state only
  - Doing it at each step would be inefficient and sometimes not possible

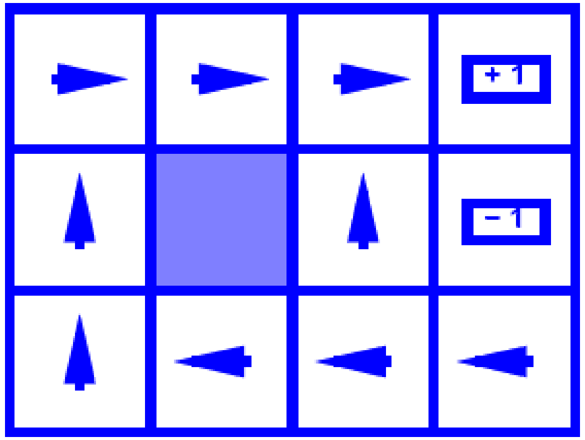


Optimal policy when R(s, a, s') = -0.03 for all non-terminals s

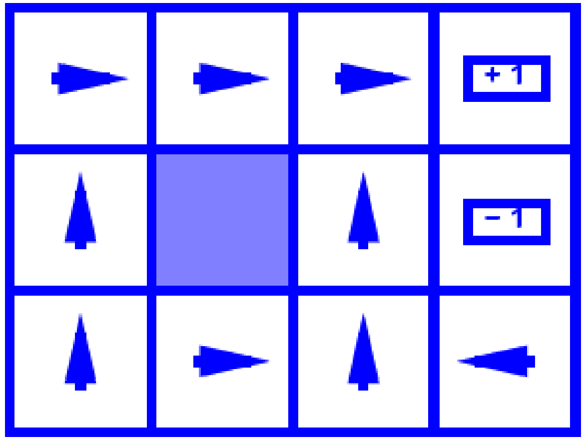
Optimal Policies



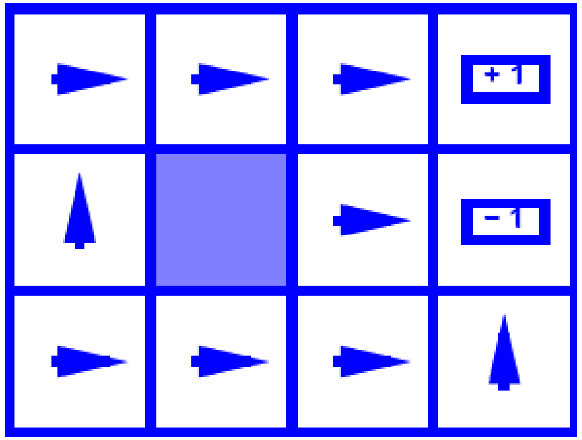
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

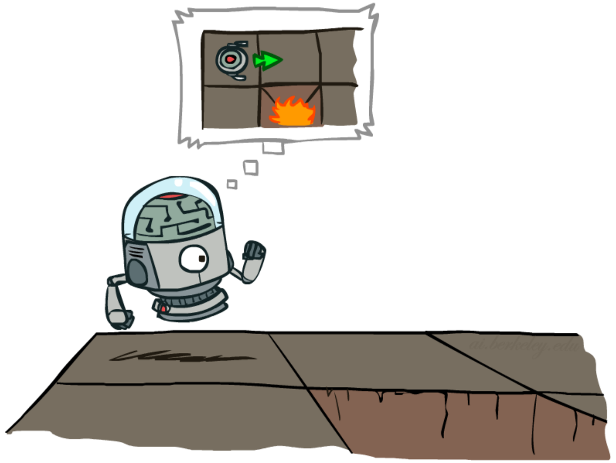
Utilities and Policies

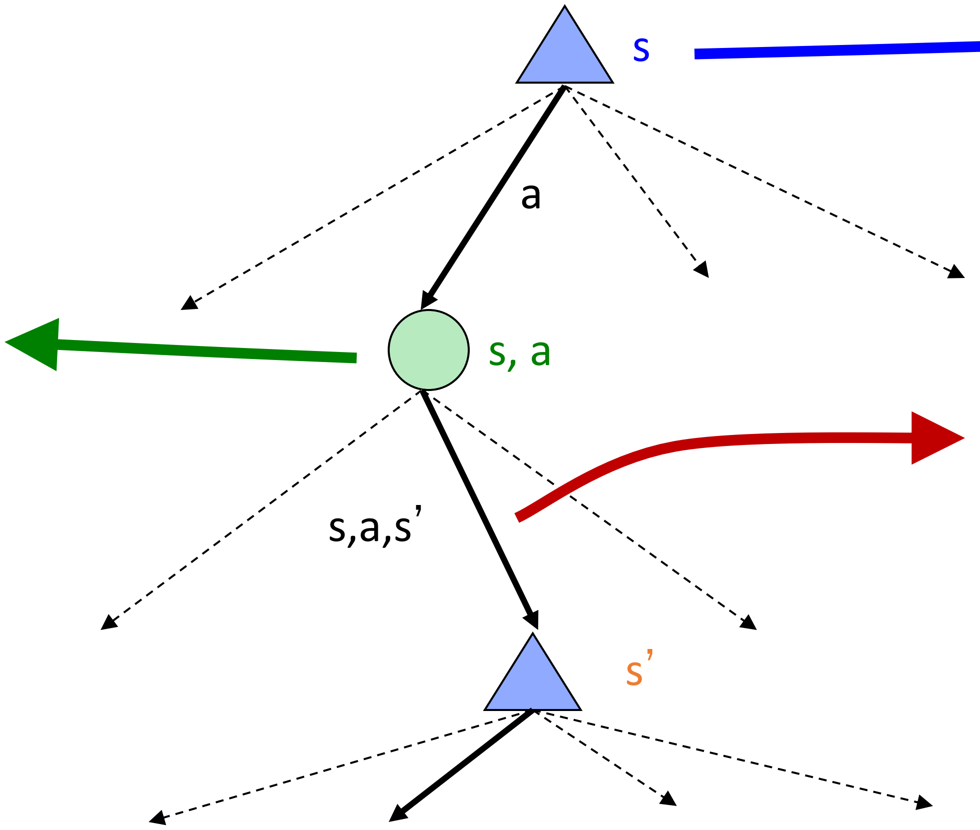
- Utility: Defined by the reward function
- Solving MDPs: Finding a policy
- Policy: What action to take in each state?
- Optimal Policy: Has the highest expected utility

- How to calculate the utility of a policy?

MDP Seach Trees

Each MDP state projects an expectimax-like search tree





s is a *state*

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

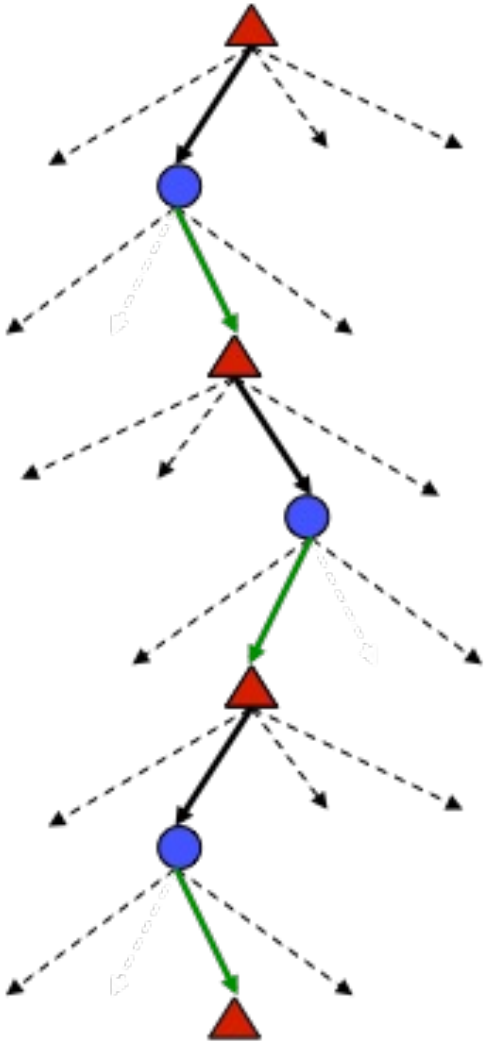
T(s,a,s') = P(s'|s,a)

R(s,a,s')

Solution Horizon

- Finite:
  - Agent must solve the problem in a finite amount of time/steps
  - The state sequences must be finite
  - The right action in a state depends on how much time left

- Infinite:
  - Agent does not have a time/step limit
  - Optimal action depends only on the state

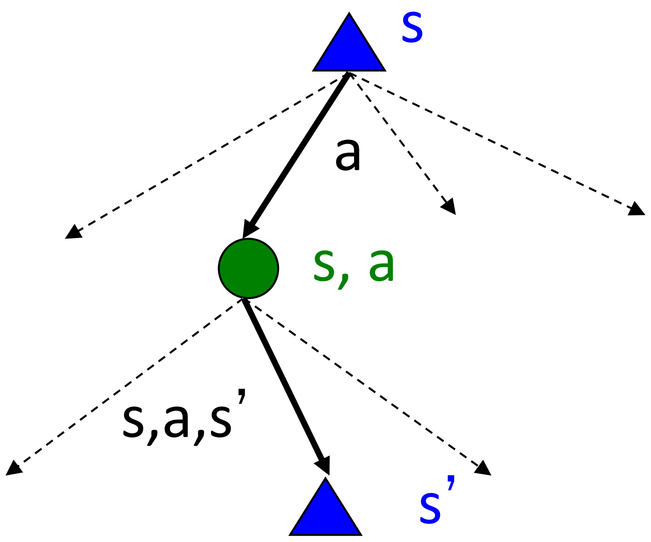


Non-Stationary

Stationary

Recap: Defining MDPs

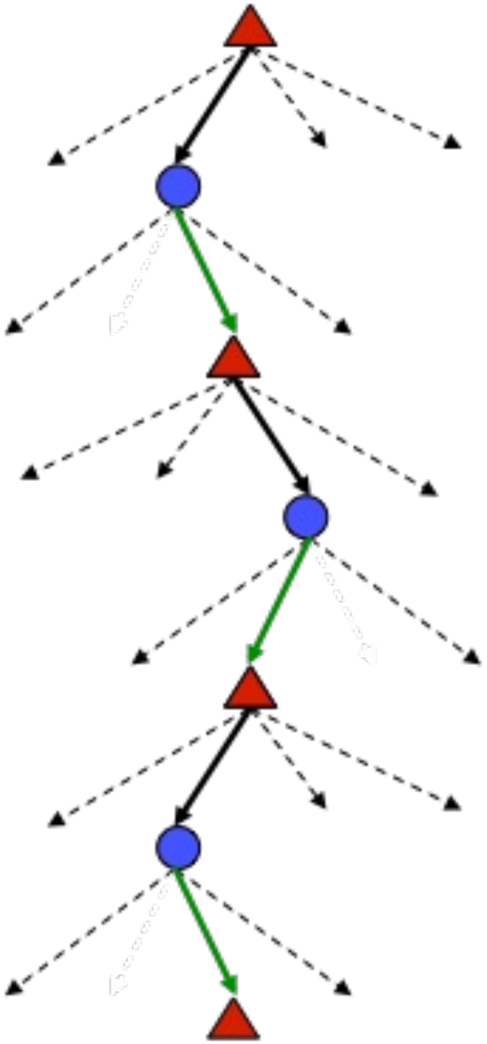
- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



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

Utilities of State Sequences

- A path in a tree gives a sequence
- We receive rewards at each state!
- What is the utility of a sequence?
  - Sum of rewards?
  - Average rewards?
  - Reward now is better than later?
  - What about infinite sequences?
- Idea: Sum of discounted rewards



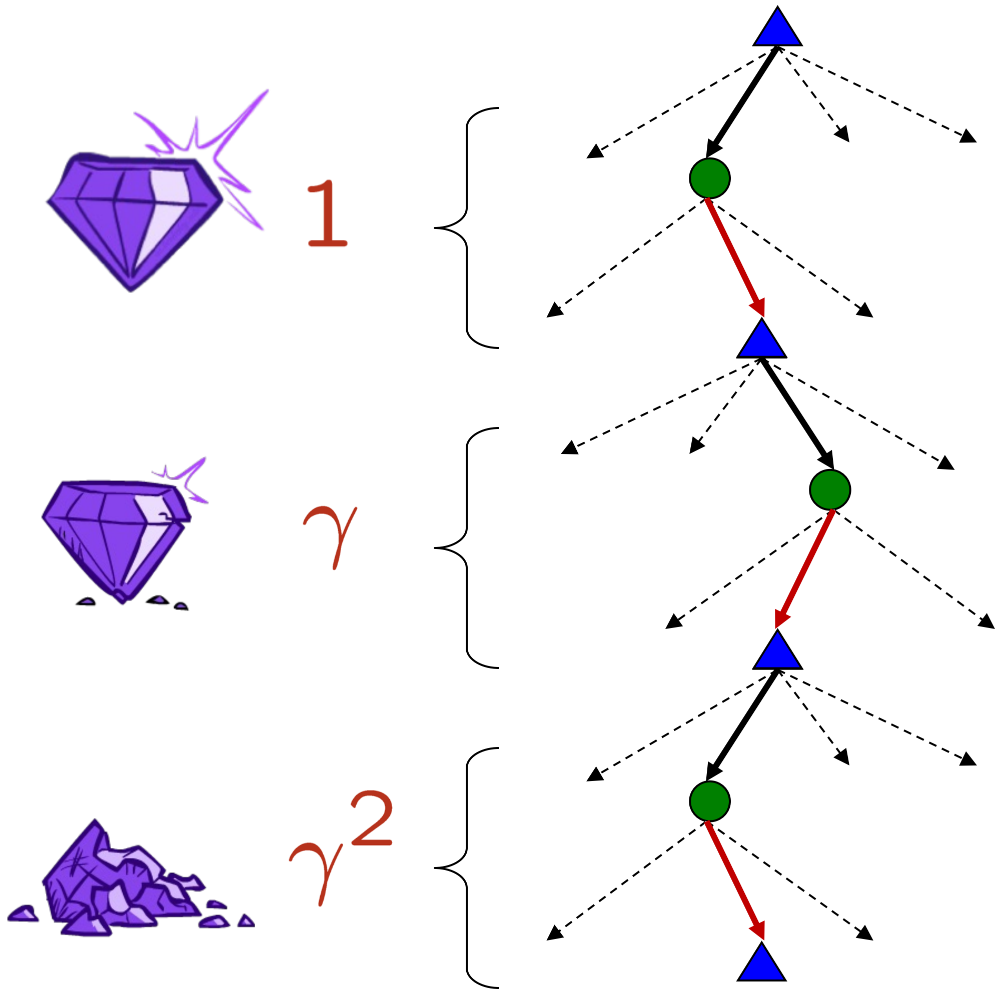
Discounting

$\gamma \in [0,1]$: discount factor

- 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

$$U([1,2,3]) = 1 \cdot 1 + 0.5 \cdot 2 + 0.25 \cdot 3$$

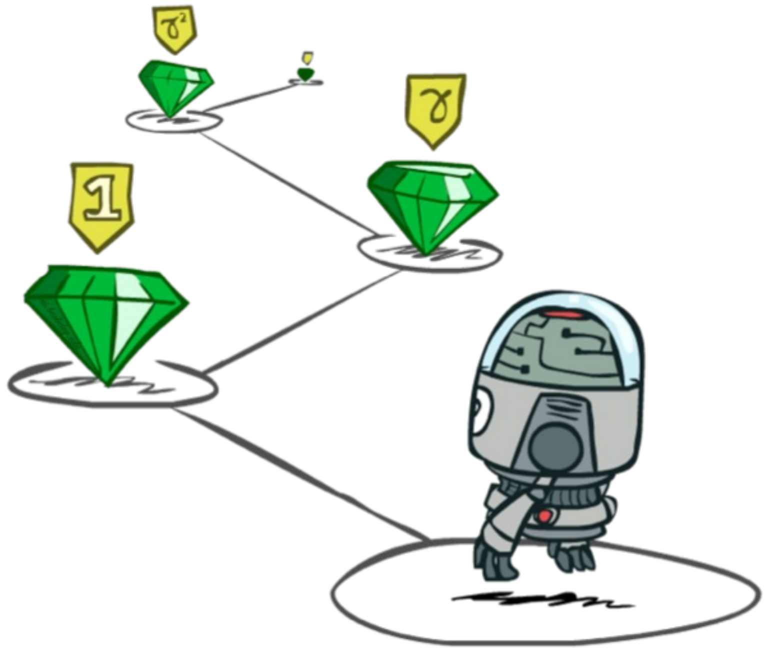
$$U([1,2,3]) < U([3,2,1])$$



Stationary Preferences

- Theorem: if we assume stationary preferences:

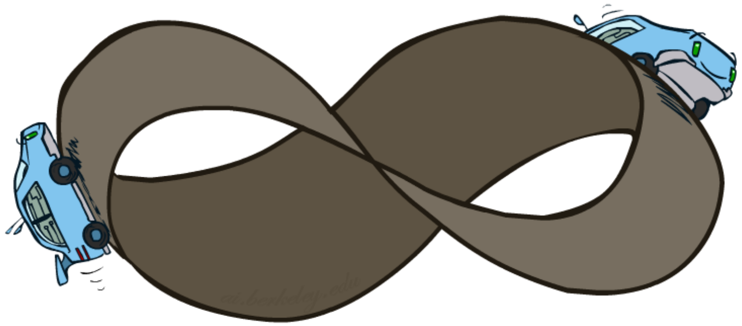
$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$



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

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (like depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives non-stationary policies (π depends on time left)
  - Discounting: use $0 < \gamma < 1$



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

    - Smaller γ means smaller "horizon" – shorter term focus
    - The discount factor favors "shorter" solutions (unless there is a high living reward!)
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached

Calculating Policies

- MDPs end up being a good representation for many real-world problems.

- Given an MDP description, we will see several algorithms for solving for the optimal policy

- Two General Class of Algorithms:
  - Value Iteration
  - Policy Iteration

Optimal Quantities

- The value (utility) of a state s:

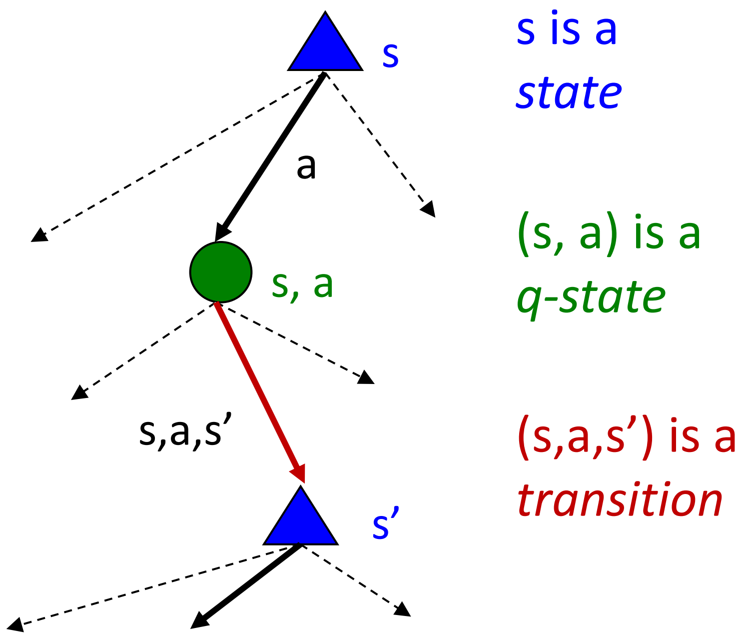
  $V^*(s)$ = expected utility starting in s and acting optimally

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

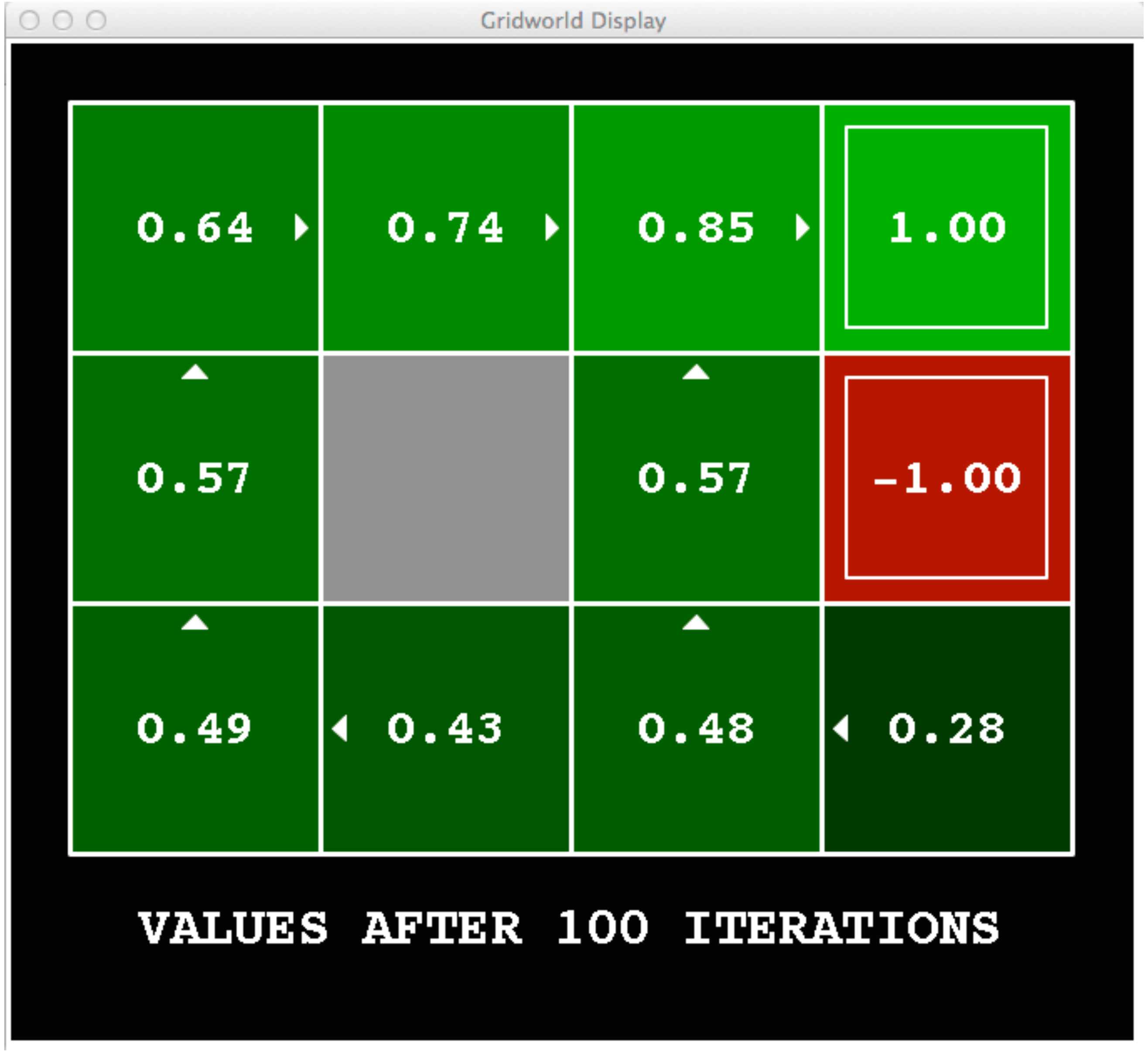
  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

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



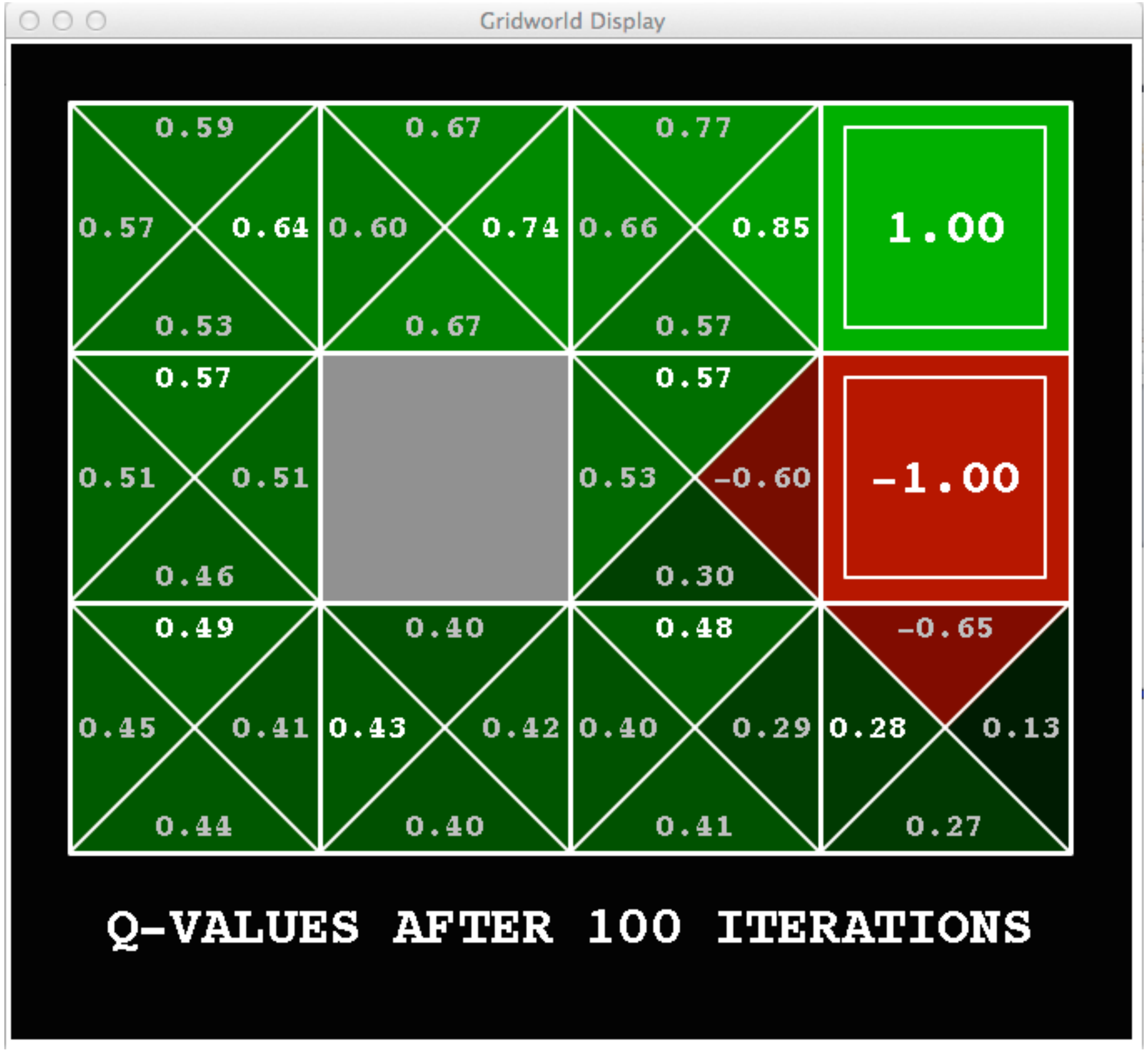
Snapshot of Demo – Gridworld Values



Value of a state is a more "long term" quantity whereas reward is an immediate quantity.

Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo – Gridworld Q Values



Q-Value is also a long term quantity

Noise = 0.2
Discount = 0.9
Living reward = 0

A Mathematical Remark

- Given a potentially infinite state sequence:

$$\sigma = \{s_0, s_1, s_2, \dots\}$$

- The utility of the path is

$$\mathrm{U}(\sigma) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = r_0 + \gamma \mathrm{U}(\sigma'), \sigma' = \{s_1, s_2, \dots\}$$

- Then the expected utility, or the value, of the state s_1 given a policy π is:

$$\mathrm{V}(s_0) = \sum_{s'} \mathrm{P}(s'|s_0, \pi(s_0))(\mathrm{R}(s_0, \pi(s_0), s') + \gamma \mathrm{V}(s'))$$

- or if $\mathrm{R}(s, a, s') = r_0$

$$\mathrm{V}(s_0) = r_1 + \gamma \sum_{s'} \mathrm{P}(s'|s_0, \pi(s_0))\, \mathrm{V}(s')$$

Bellman Equations

- $V^*(s)$: Expected utility starting in s and acting optimally:

  $$V^*(s) = \sum_{s'} (P(s'|s,\pi^*(s))\big(R(s,\pi^*(s),s') + \gamma V^*(s')\big))$$

- $\pi^*(s)$: For a single state, pick the action that maximizes the expected utility

  $$\pi^*(s) = \arg_a \max(\sum_{s'} (P(s'|s,a)\big(R(s,a,s') + \gamma V^*(s')\big)))$$

- $Q^*(s,a)$: Expected utility of starting in s with action a and action optimally

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

- These are related:

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

  $$V^*(s) = \max_a(\sum_{s'} \Big(P(s'|s,a)\big(R(s,a,s') + \gamma V^*(s')\big)\Big))$$

Some Observations

Immediate reward

$$V^*(s) = \max_a \left(\sum_{s'} \left(P(s'|s,a) \left(\boxed{R(s,a,s')} + \gamma \boxed{V^*(s')} \right) \right) \right)$$

Discounted neighbor value

- If we calculate the values of states, we immediately get the policy
- These quantities are defined recursively
- The value/utility of a state is related to its neighboring states

- Why not iteratively calculate the values of states, using current estimates of the neighboring states?

Bellman Updates

- General Case:

$$V_{t+1}(s) = \max_a(\sum_{s'} \left(P(s'|s,a)\left(R(s,a,s') + \gamma V_t(s')\right)\right))$$

- If the rewards are only for states:

$$V_{t+1}(s) = R(s) + \gamma \max_a(\sum_{s'} (P(s'|s,a)V_t(s')))$$

- The idea is to find the unknown function $V(s)$ given the MDP
- Note that the policy is not explicitly calculated

Value Iteration

- Start with $V_0(s) = 0$
- Use the Bellman update to recursively calculate $V(s)$

  $$V_{t+1}(s) = \max_a \left(\sum_{s'} \left(P(s'|s,a)\left(R(s,a,s') + \gamma V_t(s')\right)\right)\right)$$

  - At each step, go over all the states
- Repeat until convergence
- Complexity of one iteration: $O(|S|^2|A|)$- Why?

- Theorem: $V(s)$ will converge to optimal values, $V^*(s)$
  - The book has a proof – read it!
- Note: The resulting policy may converge long before the values! – Why?

Value Iteration

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, transition model T, reward function R, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'; \delta \leftarrow 0$
    for each state s **in** S **do**
      $\boxed{U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s')\ U[s']}$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\boxed{\delta < \epsilon(1-\gamma)/\gamma}$
  return U

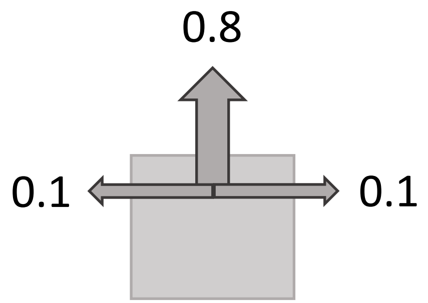
Bellman update equation
Could also use the general version here

Comes from the convergence analysis!

Exercise: Value Iteration

- MDP:
  - States: Represented by the grid
  - Actions: Up, Down, Left, Right
  - Rewards: As seen on the grid
  - Transition Model: 80-10-10 (see the figure)
  - Discount: $\gamma = 0.5$
  - (2,2) is a terminal state

-0.04	+1.0
-0.04	-0.04

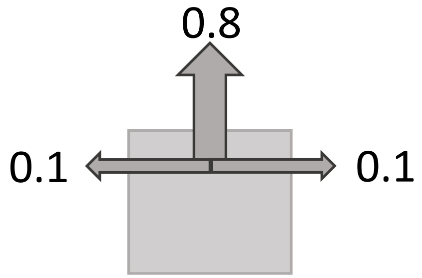


- Question: Starting from $\mathrm{V}(s_{11}) = \mathrm{V}(s_{12}) = \mathrm{V}(s_{21}) = 0.1$ and $\mathrm{V}(s_{22}) = 1.0$, do one step of value iteration

R

-0.04	+1.0
-0.04	-0.04

T



$\gamma = 0.5$

V$_0$

0.1	+1.0
0.1	0.1

$$V_{t+1}(s) = R(s) + \gamma \max_a \left(\sum_{s'} (P(s'|s,a) V_t(s')) \right)$$

$$\begin{aligned} V_1(s_{21}) = -0.04 + 0.5 \max_a (& P(s_{11}|s_{21},UP)V_0(s_{11}) + P(s_{21}|s_{21},UP)V_0(s_{21}) + P(s_{22}|s_{21},UP)V_0(s_{22}), \\ & P(s_{11}|s_{21},DN)V_0(s_{11}) + P(s_{21}|s_{21},DN)V_0(s_{21}) + P(s_{22}|s_{21},DN)V_0(s_{22}), \\ & P(s_{11}|s_{21},LT)V_0(s_{11}) + P(s_{21}|s_{21},LT)V_0(s_{21}) + P(s_{22}|s_{21},LT)V_0(s_{22}), \\ & P(s_{11}|s_{21},RT)V_0(s_{11}) + P(s_{21}|s_{21},RT)V_0(s_{21}) + P(s_{22}|s_{21},RT)V_0(s_{22})) \end{aligned}$$

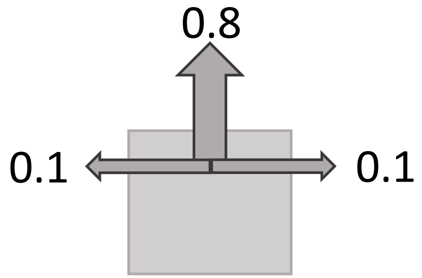
$$\begin{aligned} V_1(s_{21}) = -0.04 + 0.5 \max_a (& \boxed{a = UP\colon\ 0.1 \cdot 0.1 + 0.1 \cdot 0.1 + 0.8 \cdot 1.0 = 0.82,} \\ & a = DN\colon\ 0.1 \cdot 0.1 + 0.9 \cdot 0.1 + 0.0 \cdot 1.0 = 0.1, \\ & a = LT\colon\ 0.8 \cdot 0.1 + 0.1 \cdot 0.1 + 0.1 \cdot 1.0 = 0.19, \\ & a = RT\colon\ 0.0 \cdot 0.1 + 0.9 \cdot 0.1 + 0.1 \cdot 1.0 = 0.19) \end{aligned}$$

$$= -0.04 + 0.5 \cdot 0.82 = 0.37$$

R

-0.04	+1.0
-0.04	-0.04

T



$\gamma = 0.5$

V_0

0.1	+1.0
0.1	0.1

$$V_{t+1}(s) = R(s) + \gamma \max_a \left(\sum_{s'} \left(P(s'|s,a) V_t(s') \right) \right)$$

$$\begin{aligned} V_1(s_{11}) = -0.04 + 0.5 \max_a(& P(s_{12}|s_{11},UP)V_0(s_{12}) + P(s_{21}|s_{11},UP)V_0(s_{21}) + P(s_{11}|s_{11},UP)V_0(s_{11}), \\ & P(s_{12}|s_{11},DN)V_0(s_{12}) + P(s_{21}|s_{11},DN)V_0(s_{21}) + P(s_{11}|s_{11},DN)V_0(s_{11}), \\ & P(s_{12}|s_{11},LT)V_0(s_{12}) + P(s_{21}|s_{11},LT)V_0(s_{21}) + P(s_{11}|s_{11},LT)V_0(s_{11}), \\ & P(s_{12}|s_{11},RT)V_0(s_{12}) + P(s_{21}|s_{11},RT)V_0(s_{21}) + P(s_{11}|s_{11},RT)V_0(s_{11})) \end{aligned}$$

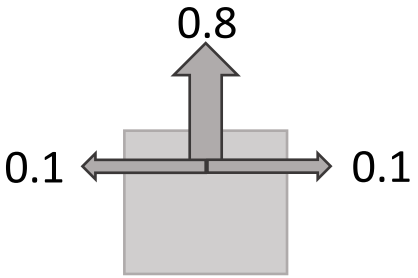
$$\begin{aligned} V_1(s_{11}) = -0.04 + 0.5 \max_a(& a = UP\colon\ 0.8 \cdot 0.1 + 0.1 \cdot 0.1 + 0.1 \cdot 0.1 = 0.1, \\ & a = DN\colon\ 0.0 \cdot 0.1 + 0.1 \cdot 0.1 + 0.9 \cdot 0.1 = 0.1, \\ & a = LT\colon\ 0.1 \cdot 0.1 + 0.0 \cdot 0.1 + 0.9 \cdot 0.1 = 0.1, \\ & a = RT\colon\ 0.8 \cdot 0.1 + 0.1 \cdot 0.1 + 0.1 \cdot 0.1 = 0.1) \end{aligned}$$

$$= 0.01$$

R

-0.04	+1.0
-0.04	-0.04

T



$\gamma = 0.5$

V_0

0.1	+1.0
0.1	0.1

$$V_{t+1}(s) = R(s) + \gamma \max_a \left(\sum_{s'} \left(P(s'|s,a) V_t(s') \right) \right)$$

$$\begin{aligned} V_1(s_{12}) = -0.04 + 0.5 \max_a (& a = UP{:}\ 0.0 \cdot 0.1 + 0.9 \cdot 0.1 + 0.1 \cdot 1.0 = 0.19, \\ & a = DN{:}\ 0.8 \cdot 0.1 + 0.1 \cdot 0.1 + 0.1 \cdot 1.0 = 0.1, \\ & a = LT{:}\ 0.1 \cdot 0.1 + 0.9 \cdot 0.1 + 0.0 \cdot 1.0 = 0.1, \\ & \boxed{a = RT{:}\ 0.1 \cdot 0.1 + 0.1 \cdot 0.1 + 0.8 \cdot 1.0 = 0.82}) \end{aligned}$$

$$= -0.04 + 0.5 \cdot 0.82 = 0.37$$

V_1

0.37	+1.0
0.01	0.37

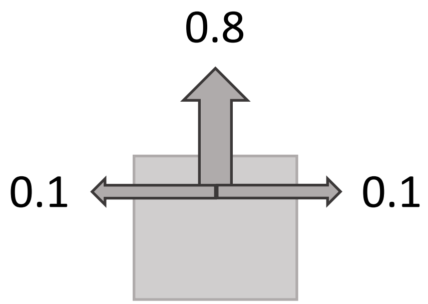
V_5

0.376	+1.0
0.12	0.376

Value Function to Policy (Policy Extraction)

$$\pi^*(s) = \arg_a \max\left(\sum_{s'} \left(P(s'|s,a)\left(R(s,a,s') + \gamma V^*(s')\right)\right)\right)$$

0.90	0.93	0.95	+1.0
0.88		0.79	-1.0
0.85	0.83	0.81	0.13



$\gamma = 0.99, R(s) = -0.01$

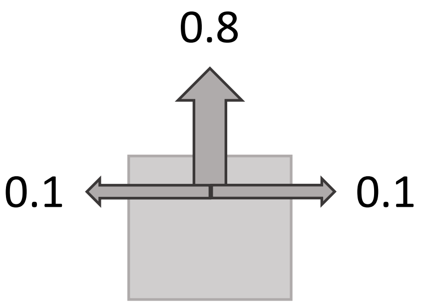
$$\arg_a \max\left(\begin{aligned} a &= UP\text{: } 0.8 \cdot 0.79 + 0.1 \cdot 0.83 + 0.1 \cdot 0.13 \\ a &= DN\text{: } 0.8 \cdot 0.81 + 0.1 \cdot 0.83 + 0.1 \cdot 0.13 \\ &\boxed{a = LT\text{: } 0.8 \cdot 0.83 + 0.1 \cdot 0.81 + 0.1 \cdot 0.79} \\ a &= RT\text{: } 0.8 \cdot 0.13 + 0.1 \cdot 0.81 + 0.1 \cdot 0.79 \end{aligned}\right.$$

With some manipulation
(γ and $R(s)$ same for all)

Value Function to Policy (Policy Extraction)

$$\pi^*(s) = \arg_a \max\left(\sum_{s'}\left(P(s'|s,a)\left(R(s,a,s') + \gamma V^*(s')\right)\right)\right)$$

0.90	0.93	0.95	+1.0
0.88		0.79	-1.0
0.85	0.83	0.81	0.13



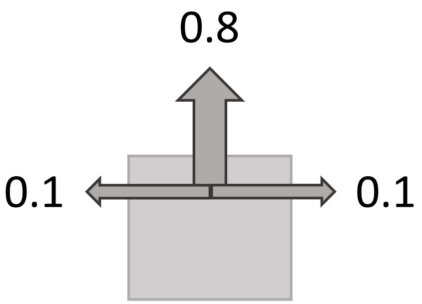
$\gamma = 0.99, R(s) = -0.01$

$$\begin{aligned}
\arg_a \max(\; & a = UP\!: \; 0.8 \cdot 0.95 + 0.1 \cdot 0.79 - 0.1 \cdot 1.0 = 0.739 \\
& a = DN\!: 0.8 \cdot 0.81 + 0.1 \cdot 0.83 - 0.1 \cdot 1.0 \\
& \boxed{a = LT\!: \; 0.8 \cdot 0.79 + 0.1 \cdot 0.81 + 0.1 \cdot 0.95 = 0.808} \\
& a = RT\!: -0.8 \cdot 1.0 + 0.1 \cdot 0.81 + 0.1 \cdot 0.95
\end{aligned}$$

Exercise at Home

$$\pi^*(s) = \arg_a \max(\textstyle\sum_{s'}(P(s'|s,a)\big(R(s,a,s') + \gamma V^*(s')\big)))$$

0.90	0.93	0.95	+1.0
0.88		0.79	-1.0
0.85	0.83	0.81	0.13

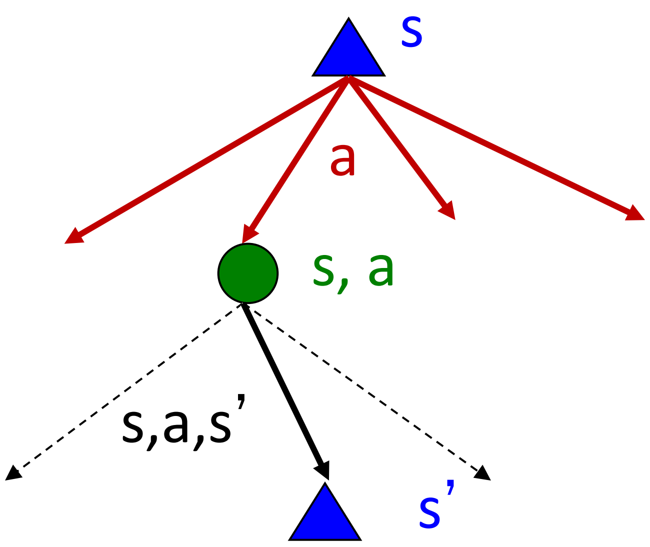


$\gamma = 0.99, R(s) = -0.01$

→	→	→	+1.0
↑		←	-1.0
←	←	←	?

Problems with Value Iteration

$$V_{t+1}(s) = \max_a \left(\sum_{s'} \left(P(s'|s,a) \left(R(s,a,s') + \gamma V_t(s') \right) \right) \right)$$



1. It's slow, $O(|S|^2|A|)$ per step
2. The "max" at each state rarely changes after a certain point
3. As a result, the policy converges before the value function!

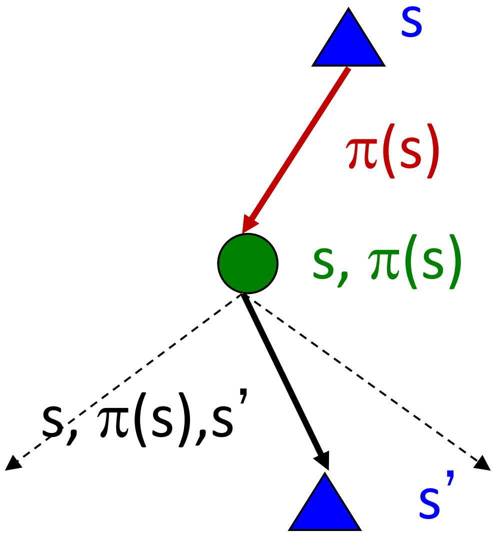
Policy Based Methods

- In value iteration we have used (drum roll) values to get the optimal policy
- Policy iteration is an alternate way to get the optimal policy
- Idea:
  - Start with an initial policy
  - Evaluate the policy (Policy Evaluation)
  - Improve the policy (Policy Improvement)
  - Stop when the policy does not change

Policy Evaluation

- What is the value function for a given policy π?
- In other words, the expected total discounted rewards starting in s and following π, $\mathrm{V}^{\pi}(s) =?$
- Note that π does not have to be optimal

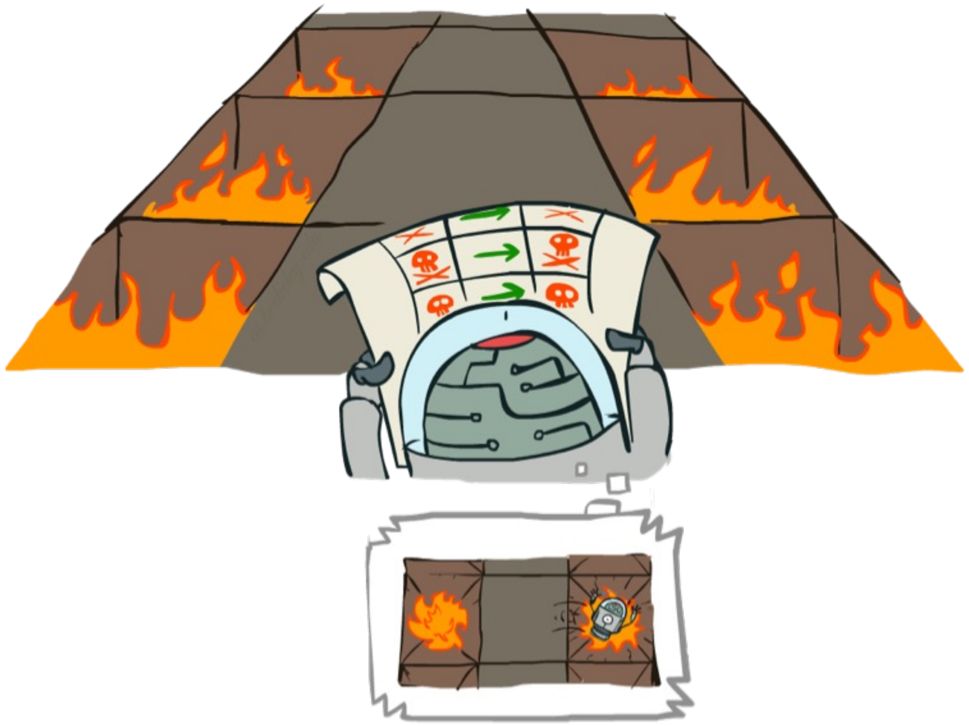
$$\mathrm{V}^{\pi}(s) = \sum_{s'} (\mathrm{P}(s'|s,\pi(s))\big(\mathrm{R}(s,\pi(s),s') + \gamma \mathrm{V}^{\pi}(s')\big))$$



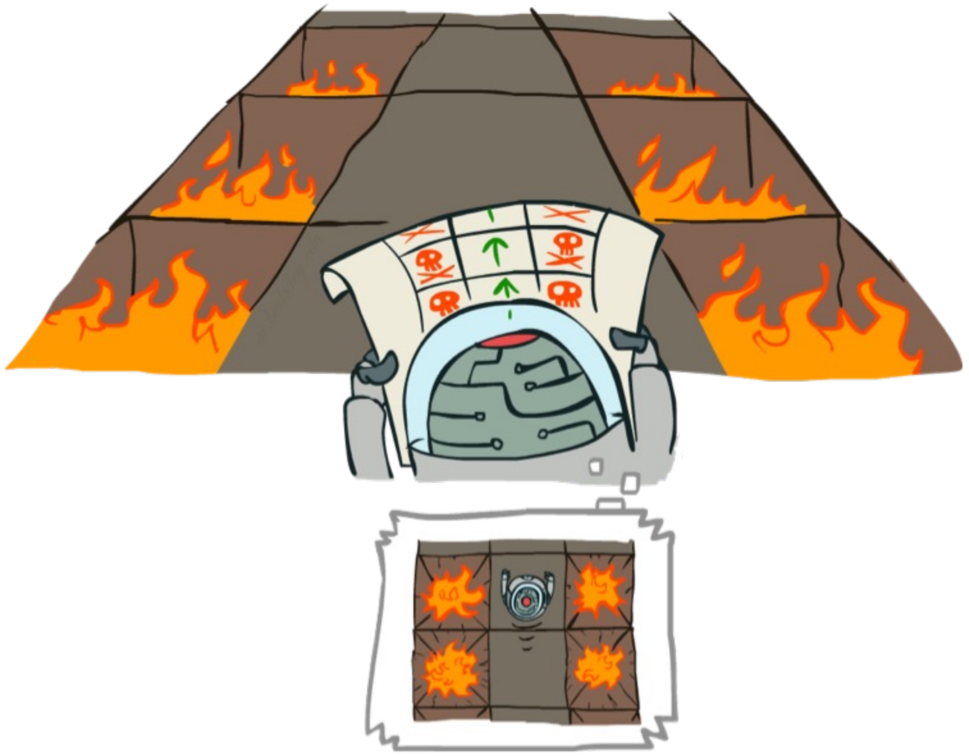
- Like value iteration calculations but simpler since we are not considering all possible actions

Example: Policy Evaluation

Always Go Right



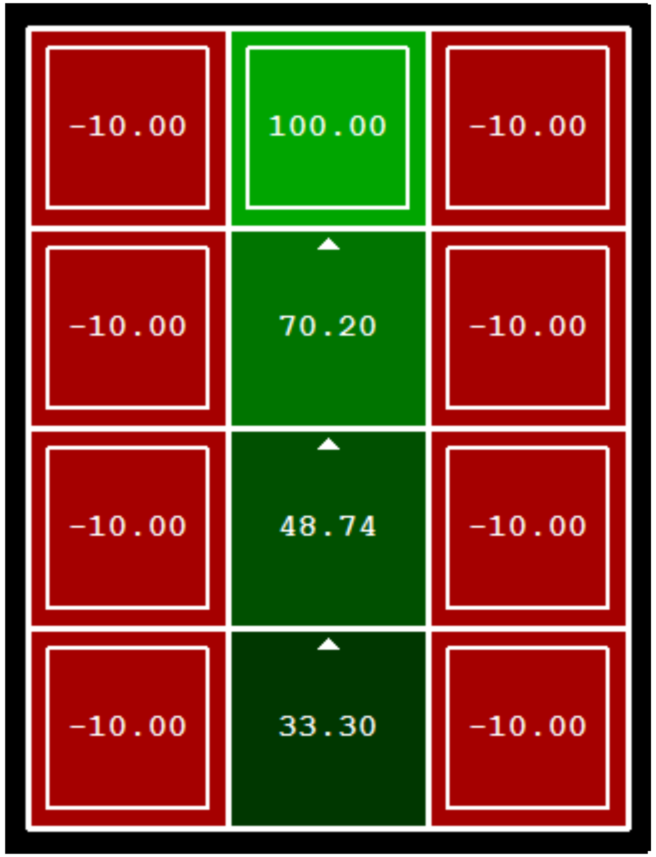
Always Go Forward



Example: Policy Evaluation

Always Go Right

Always Go Forward



Policy Evaluation

$$V^{\pi}(s) = \sum_{s'} (P(s'|s,\pi(s))\big(R(s,\pi(s),s') + \gamma V^{\pi}(s')\big))$$

- How to calculate $V^{\pi}(s) =?$
- Idea 1: Calculate it iteratively, like before ($O(|S|^2)$ per step)

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

$V_{t+1}^{\pi}(s) = \sum_{s'} (P(s'|s,\pi(s))\big(R(s,\pi(s),s') + \gamma V_t^{\pi}(s')\big))$

- Idea 2: Without the max, the Equations are just a linear system!
  - Plug it into your favorite linear system solver!
  - Involves a matrix inversion: $O(|S|^3)$ overall

Policy Improvement

- Just do policy extraction!

$$\pi_{t+1}(s) = \arg_a \max(\sum_{s'}(\mathrm{P}(s'|s,a)\big(\mathrm{R}(s,a,s') + \gamma \mathrm{V}^{\boxed{\pi_t}}(s')\big)))$$

Policy Iteration Summary

- Starting with an initial policy
  - Set $\mathrm{V}^{\pi}(s) = 0$, calculate an initial policy
  - Alternatively start with a random policy
- <u>Evaluation</u>: For the current fixed policy calculate the values $V^{\pi}(s)$

$$\mathrm{V}^{\pi}(s) = \sum_{s'} (\mathrm{P}(s'|s, \pi(s)) \big(\mathrm{R}(s, \pi(s), s') + \gamma \mathrm{V}^{\pi}(s')\big))$$

  - Either through iteration
  - Or through the linear solution
- <u>Improvement</u>: For fixed values, extract a new policy

$$\pi_{t+1}(s) = \arg_a \max(\sum_{s'} (\mathrm{P}(s'|s, a) \big(\mathrm{R}(s, a, s') + \gamma \mathrm{V}^{\pi_t}(s')\big)))$$

- Stop when there is no change

Comparison

- Both value iteration and policy iteration compute the same thing (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

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 differ only in whether we plug in a fixed policy or max over actions

Further Reading

- There are other, more efficient methods to calculate values:
  - Asynchronous Value Iteration
  - Modified Policy Iteration
- Demonstration seeded policy iteration
  - Observe a human or another agent solving the same MDP
  - Use their actions as an initial policy
  - Might not cover the entire state space
- Inverse Problem:
  - Given an optimal policy, what are the rewards/values?
- Policy Search
  - If the state and action spaces are large, improve the policy using local search methods
  - Utility calculations are local
- Partially Observable MDPs (POMDPs)
  - We cannot observe the state directly
  - Use a Dynamic BN to represent probability distribution over states