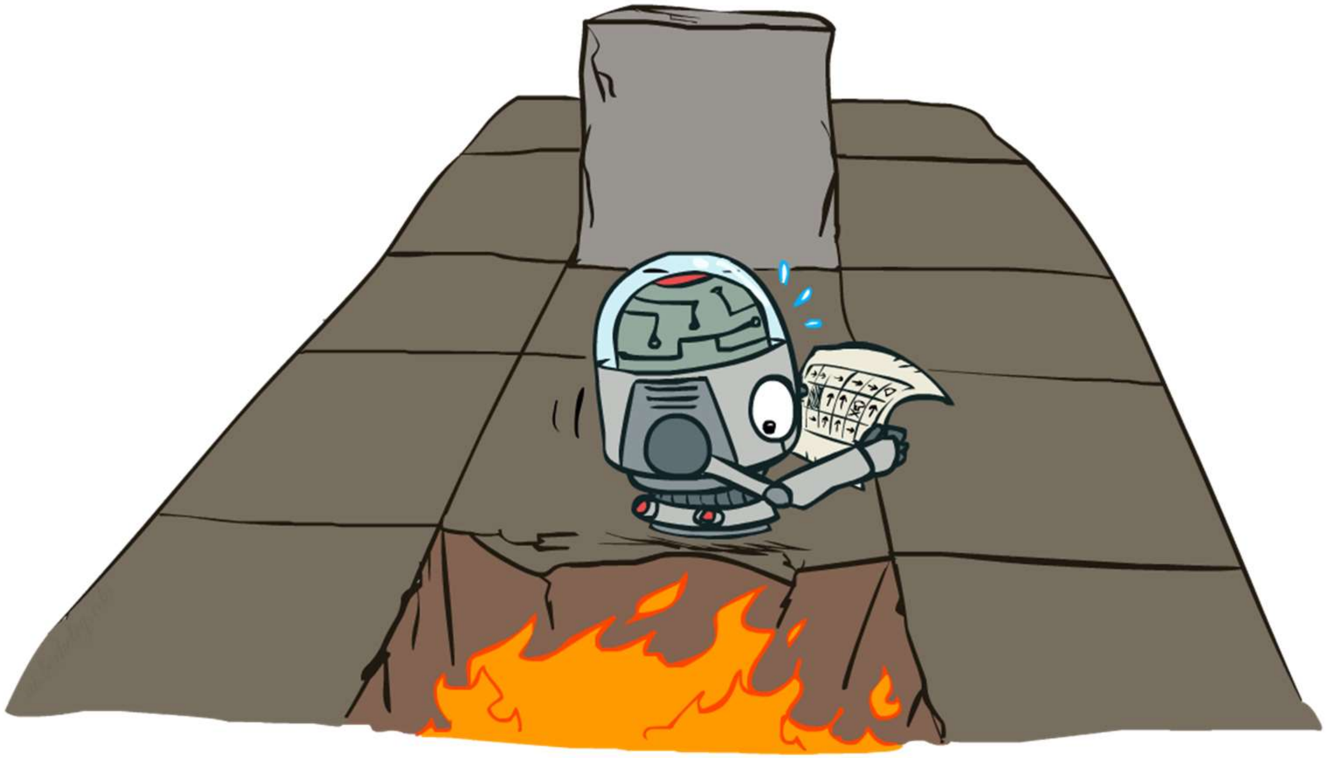
CSE 473: Introduction to Artificial Intelligence

Markov Decision Processes II

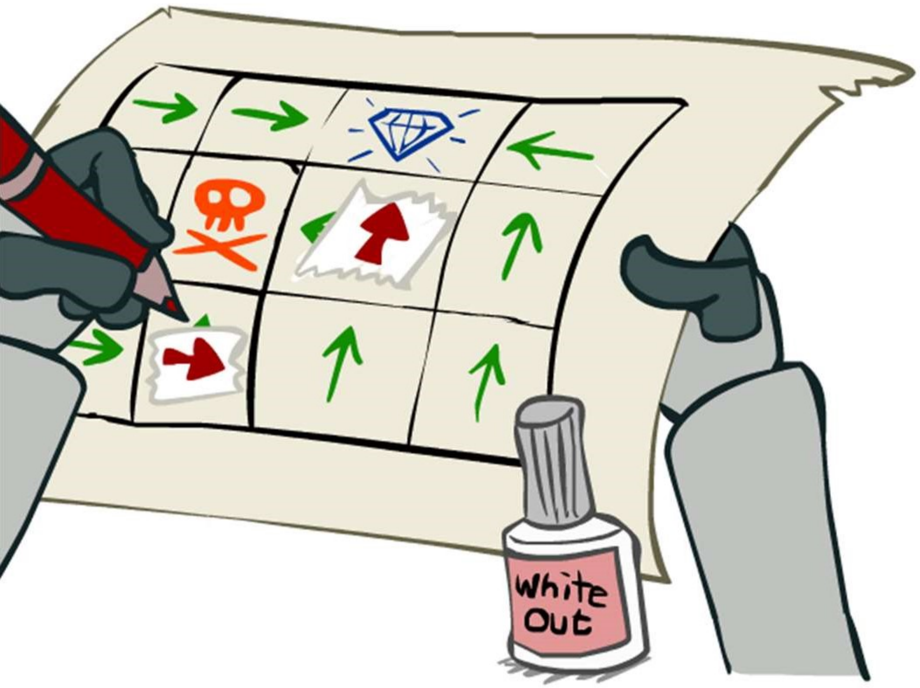


Steve Tanimoto

Based on slides by: Dan Klein and Pieter Abbeel --- University of California, Berkeley

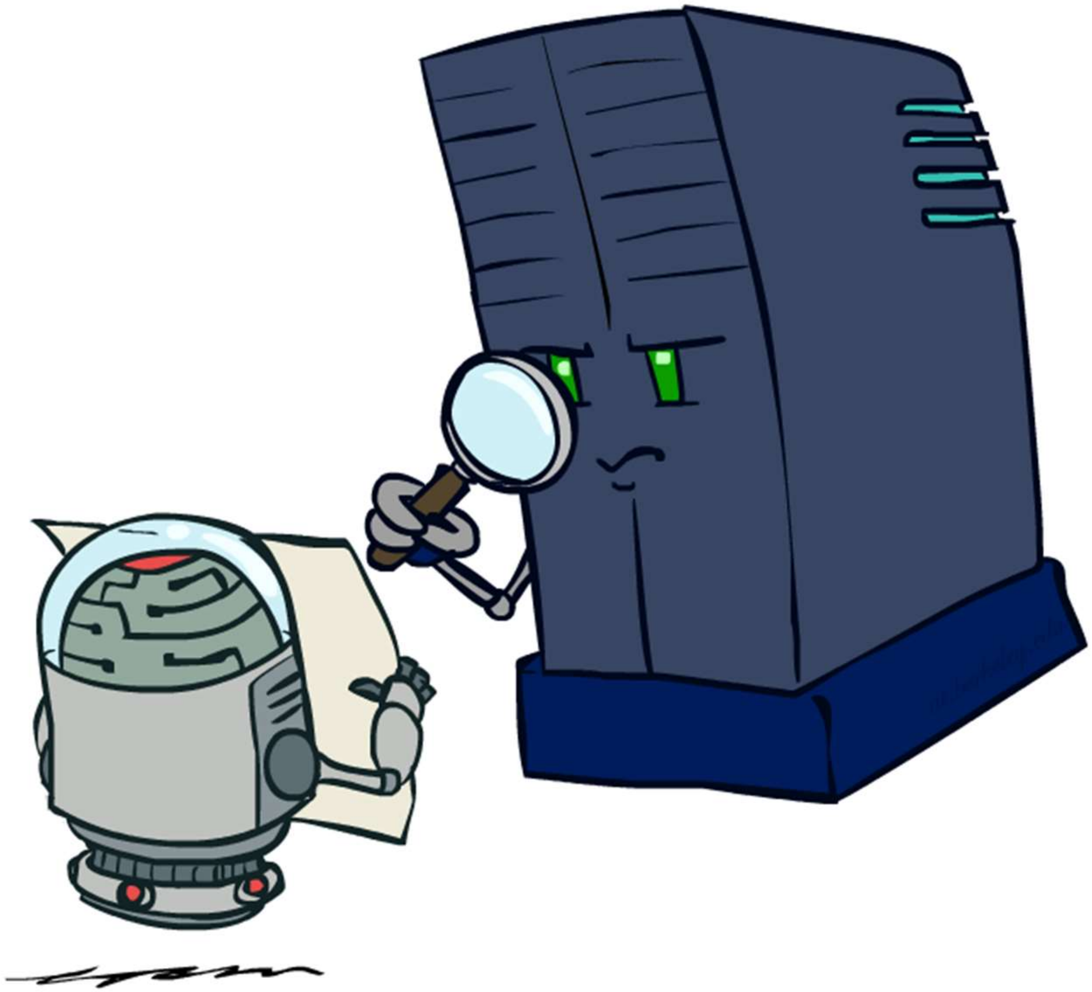
slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkele

Solving MDPs



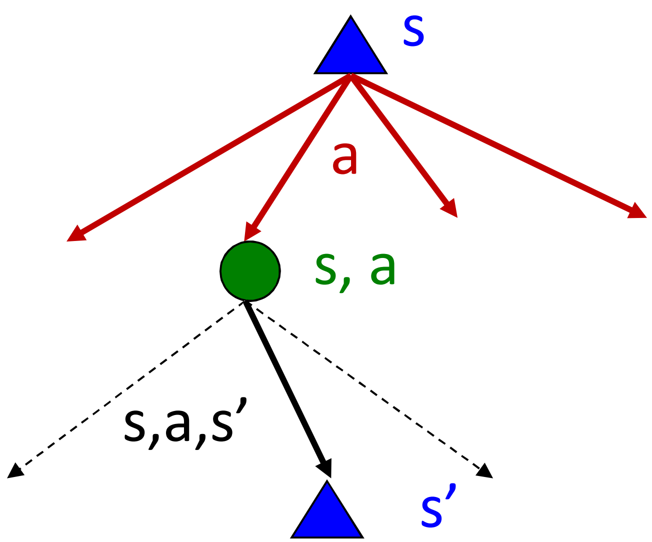
- Value Iteration
- Policy Iteration
- Reinforcement Learning

Policy Evaluation

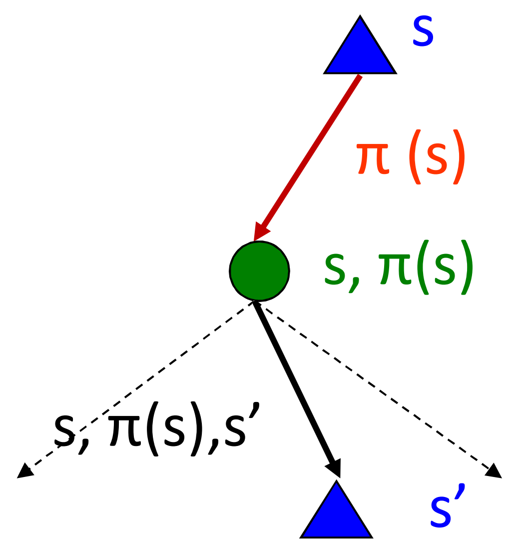


Fixed Policies

Do the optimal action



Do what π says to do



xpectimax trees max over all actions to compute the optimal values

we fixed some policy π (s), then the tree would be simpler – only one action per s

… though the tree's value would depend on which policy we fixed

Utilities for a Fixed Policy

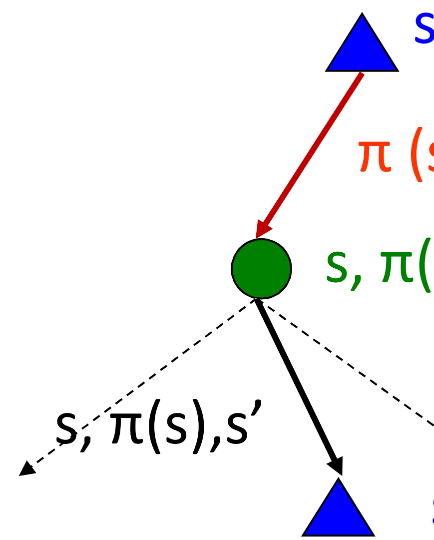
nother basic operation: compute the utility of a state s
nder a fixed (generally non-optimal) policy

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

/π (s) = expected total discounted rewards starting in s and following π

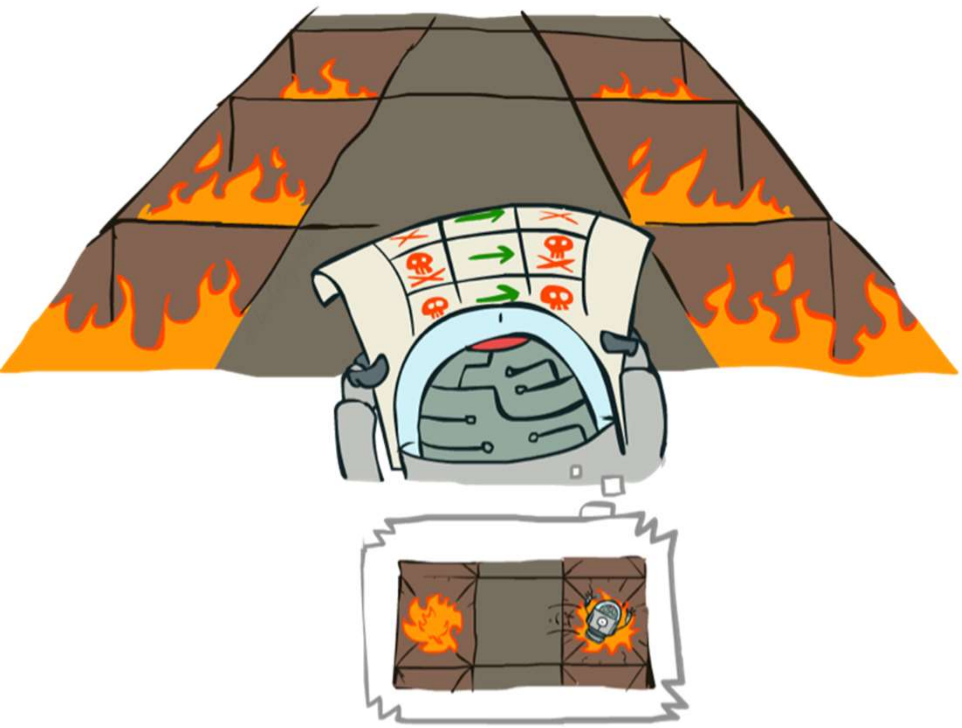
ecursive relation (one-step look-ahead / Bellman equation):

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

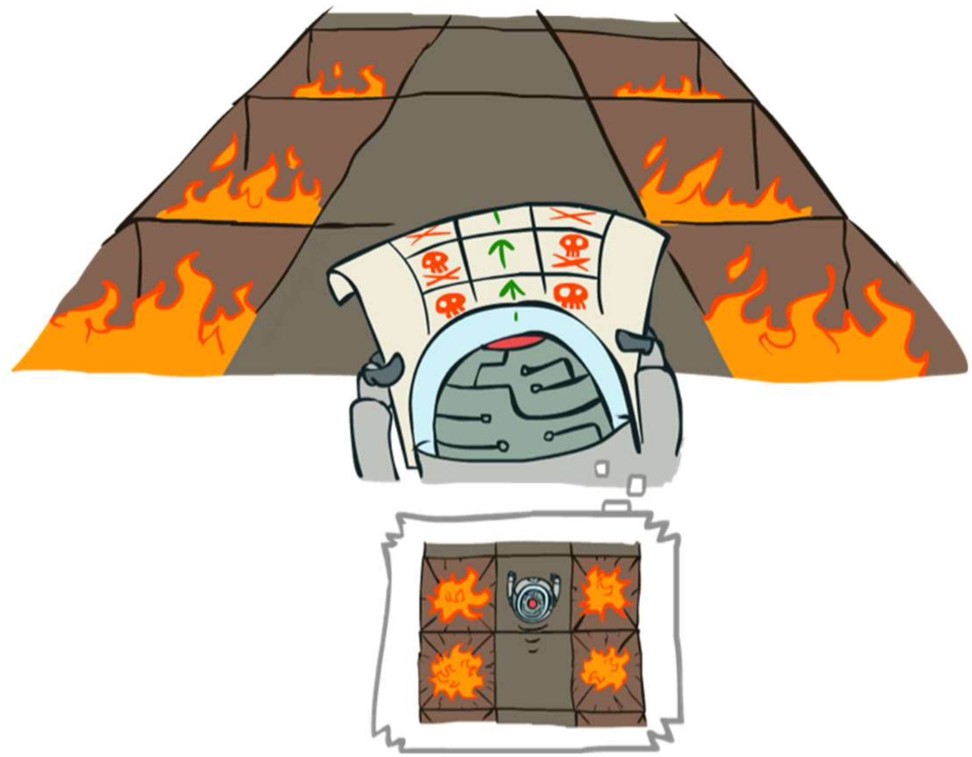


Example: Policy Evaluation

Always Go Right



Always Go Forward



Example: Policy Evaluation

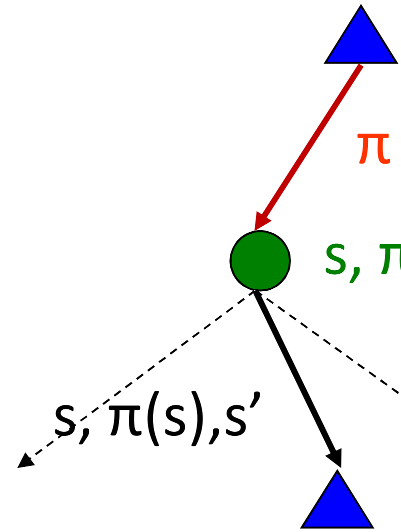
Always Go Right

Always Go Forward

Policy Evaluation

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

ea 1: Turn recursive Bellman equations into updates
ke value iteration)



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

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

fficiency: $O(S^2)$ per iteration

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

Policy Iteration

lternative approach for optimal values:

- Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence

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

- Step 2: Policy improvement: update policy using one-step look-ahead with resul converged (but not optimal!) utilities as future values

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

- Repeat steps until policy converges

his is policy iteration

- It's still optimal! Can converge (much) faster under some conditions

Comparison

- oth value iteration and policy iteration compute the same thing (all optimal value

- n 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

- n policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because w 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)

- oth are dynamic programs for solving MDPs

Summary: MDP Algorithms

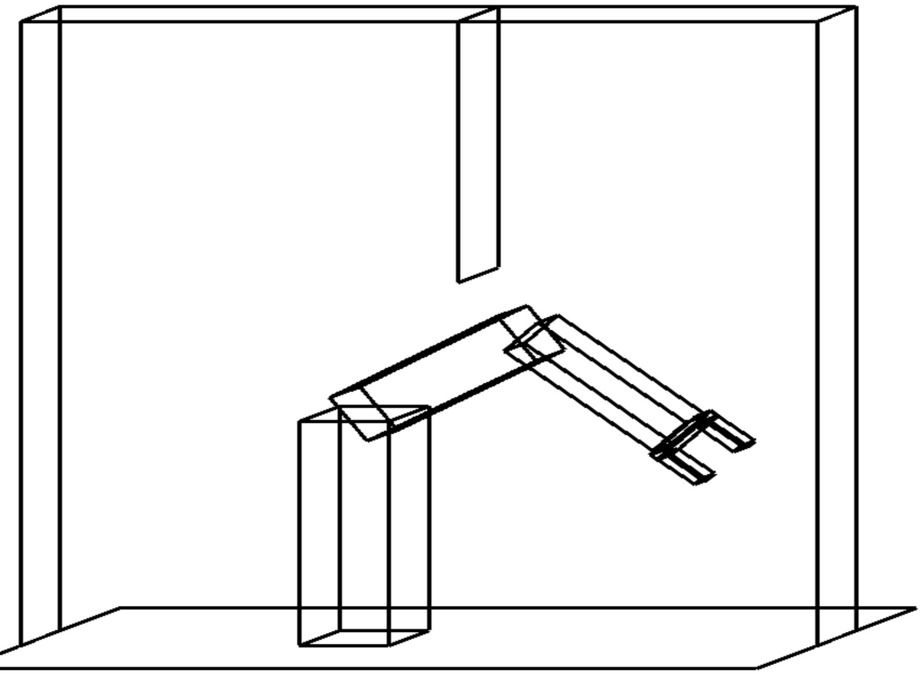
o you want to….

- 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 (one-step lookahead)

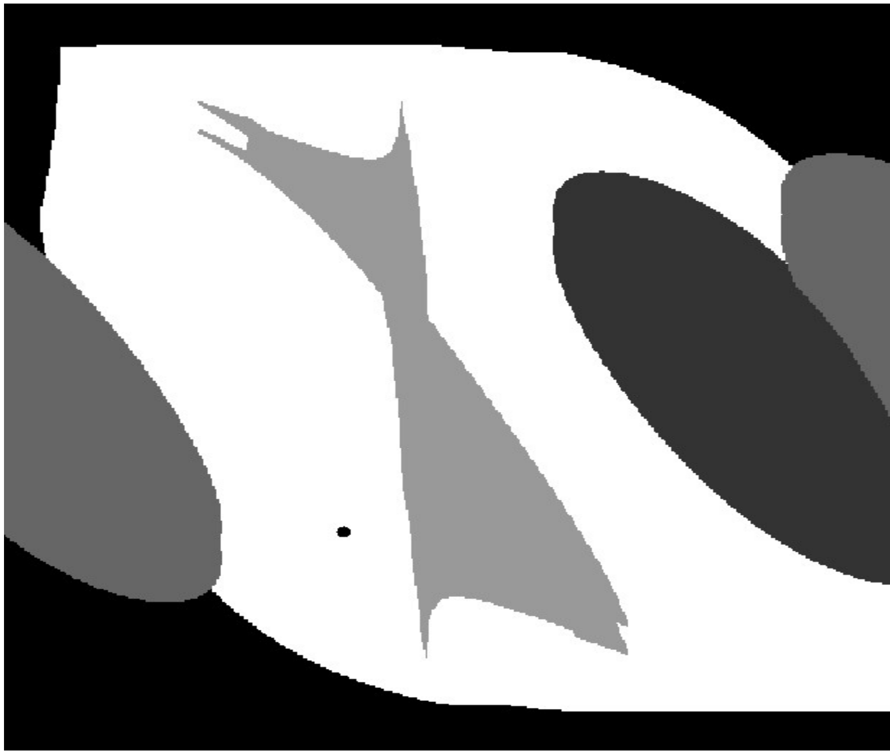
hese all look the same!

- 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

Manipulator Control

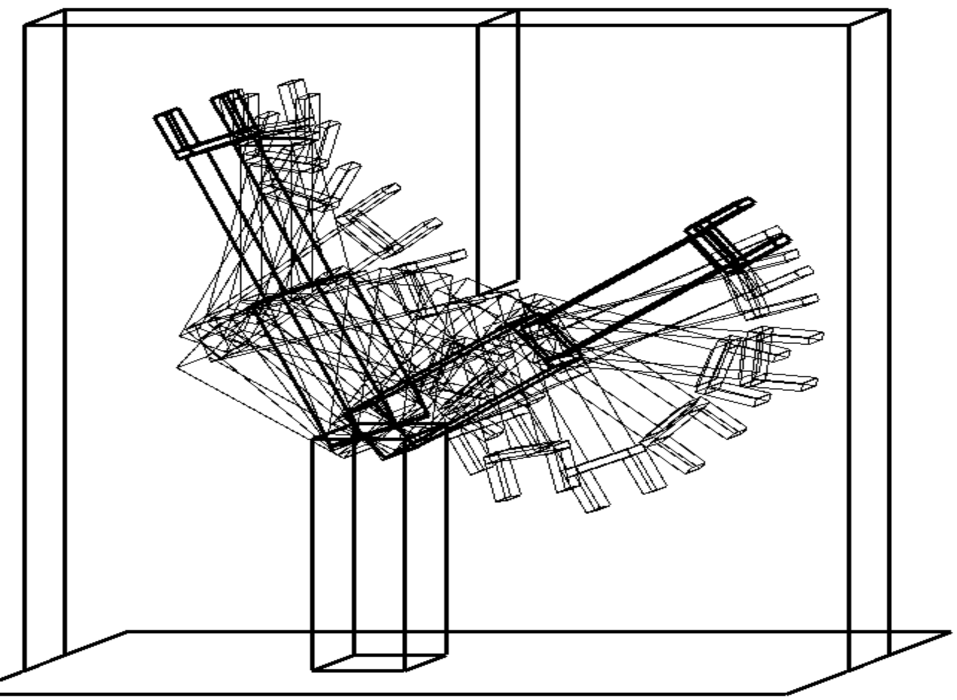


Arm with two joints (workspace)

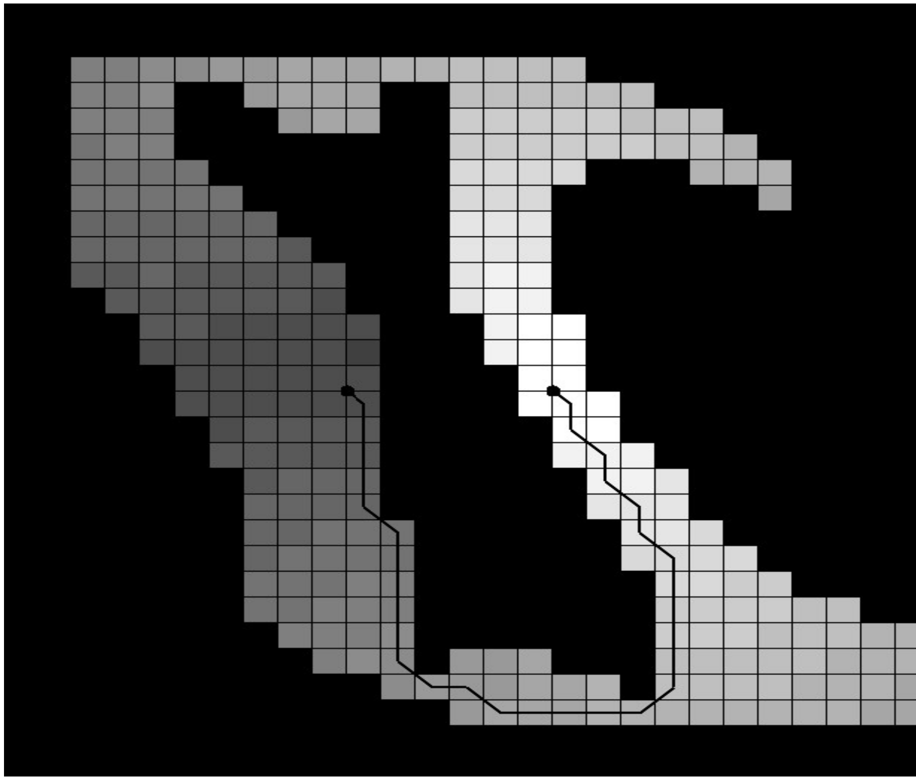


Configuration space

Manipulator Control Path

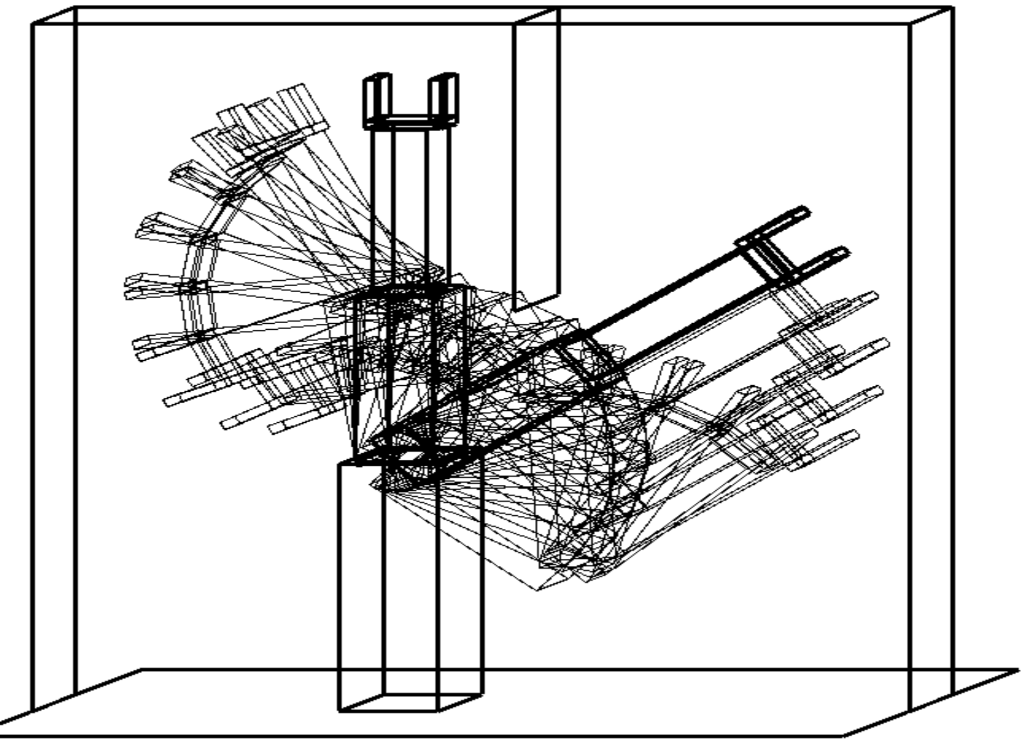


Arm with two joints (workspace)

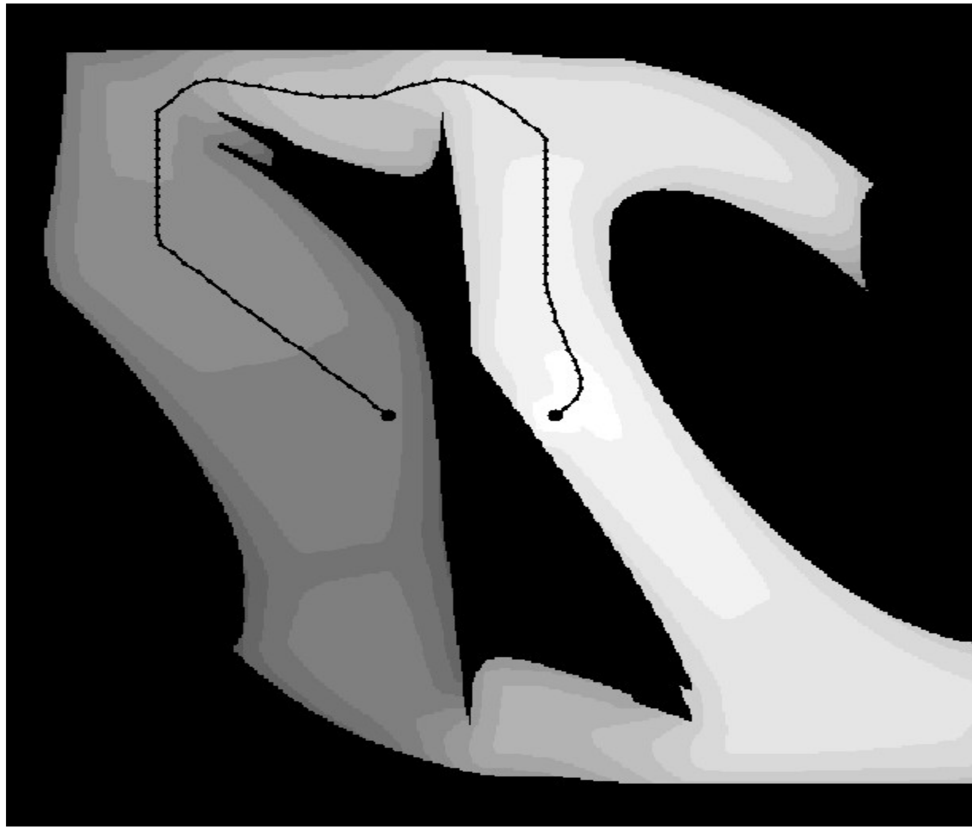


Configuration space

Manipulator Control Path

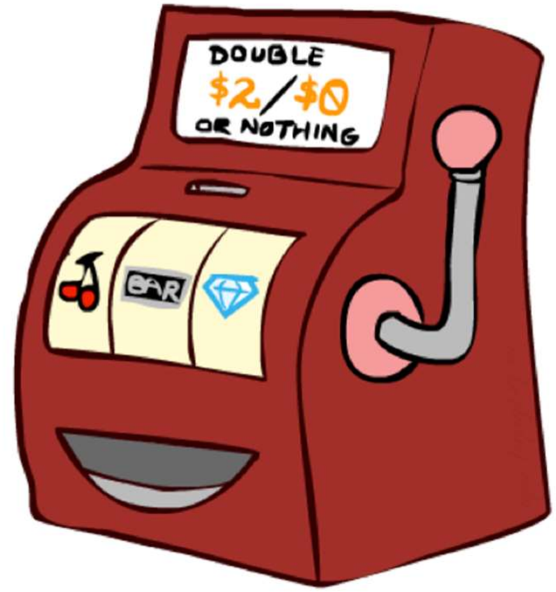


Arm with two joints (workspace)



Configuration space

Double Bandits



Double-Bandit MDP

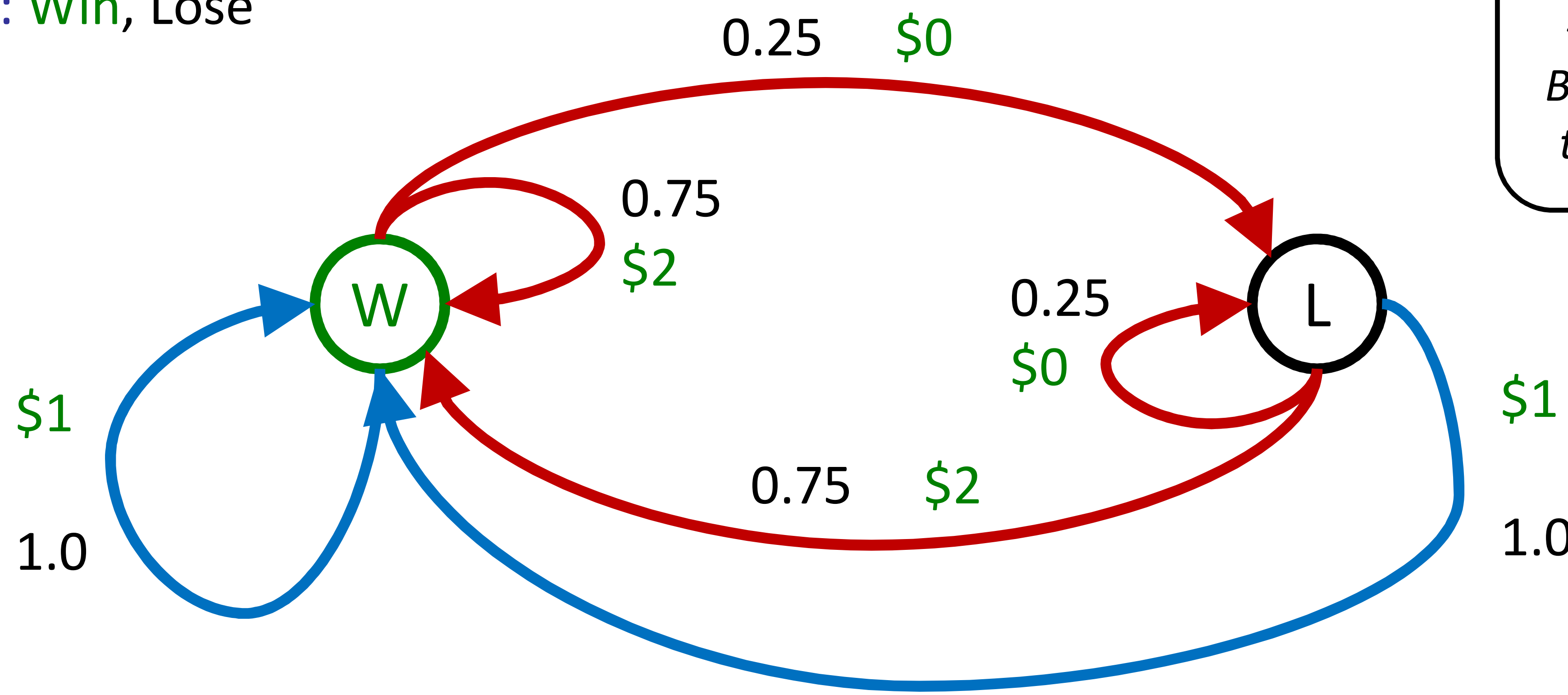
ctions: *Blue*, *Red*

tates: Win, Lose

No discoun

100 time ste

Both states h
the same va



Offline Planning

olving MDPs is offline planning

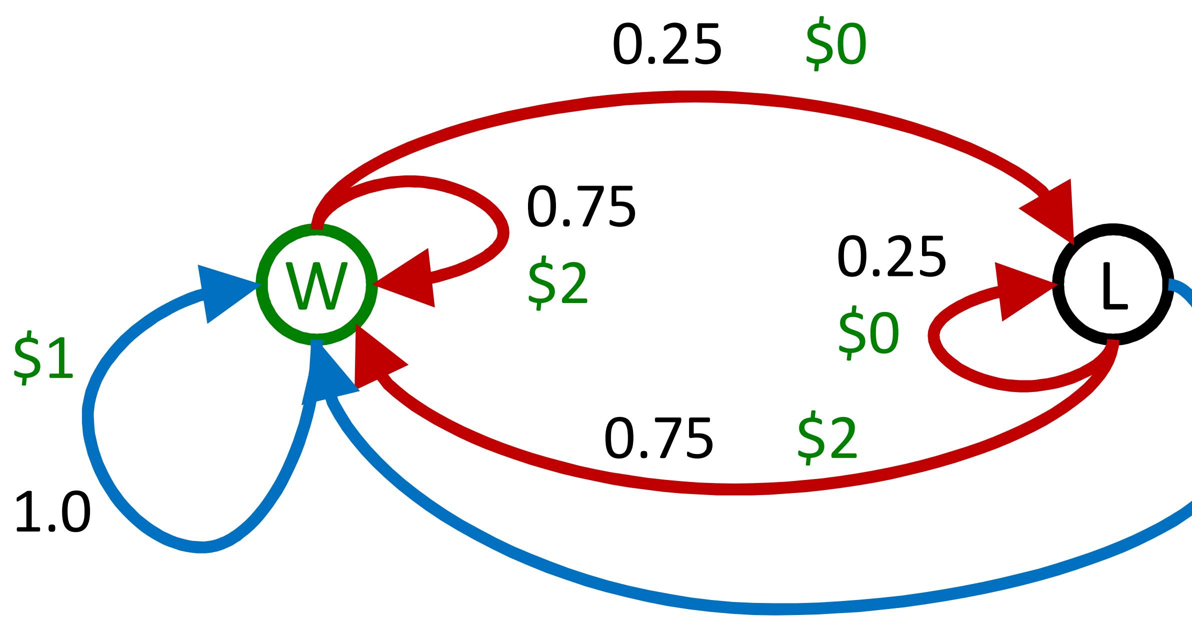
- You determine all quantities through computation
- You need to know the details of the MDP
- You do not actually play the game!

No discoun

100 time ste

Both states h
the same va

	Value
Play Red	150
Play Blue	100



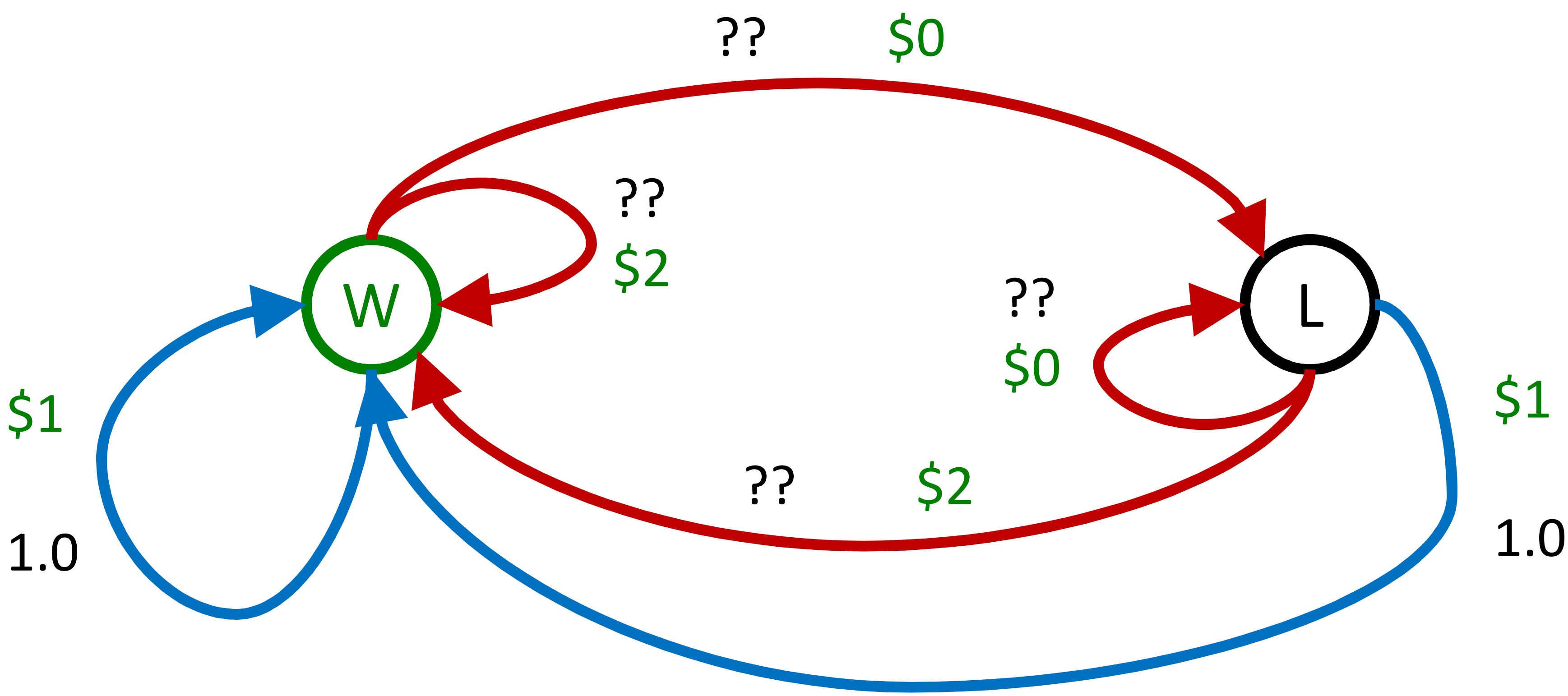
Let’s Play!

$2 $2 $0 $2 $2

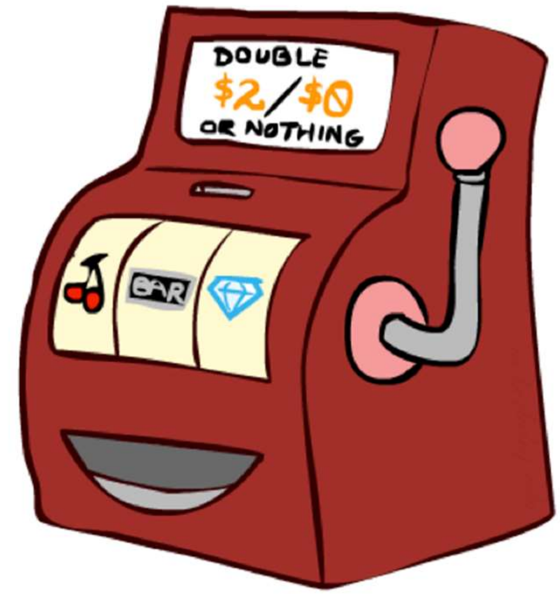
$2 $2 $0 $0 $0

Online Planning

ules changed! Red's win chance is different.



Let’s Play!



$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

What Just Happened?

hat wasn’t planning, it was learning!

- Specifically, reinforcement learning
- There was an MDP, but you couldn’t solve it with just computation
- You needed to actually act to figure it out

nportant ideas in reinforcement learning that came up

- Exploration: you have to try unknown actions to get information
- Exploitation: eventually, you have to use what you know
- Regret: even if you learn intelligently, you make mistakes
- Sampling: because of chance, you have to try things repeatedly
- Difficulty: learning can be much harder than solving a known MDP

Next Time: Reinforcement Learning!