Artificial Intelligence, CS 640, Boston University

Markov Decision Processes and Reinforcement Learning

Two Lectures by Margrit Betke
November 9 & 14, 2023

Unless noted otherwise, the slides were created by Dan Klein and Pieter Abbeel with illustrations by Ketrina Yim, for CS188, Intro to AI, University of California, Berkeley, 2014, http://ai.berkeley.edu. All materials are used with permission.

Learning Outcomes

You should be able to explain:

- MDPs vs RL
- Planning (Offline) vs Learning (Online)
- RL
  - Model-based (sample actions to learn T and R)
  - Model-free
    - Passive: Policy evaluation (direct evaluation vs. TD learning)
    - Active: Q-learning (converges to optimal policy, but agent needs to explore enough)
  - Exploration vs. Exploitation
    - Explore: ε-greedy, exploration function
  - Approximate Q-learning with linear value functions
  - Policy Search

CS 188: Artificial Intelligence

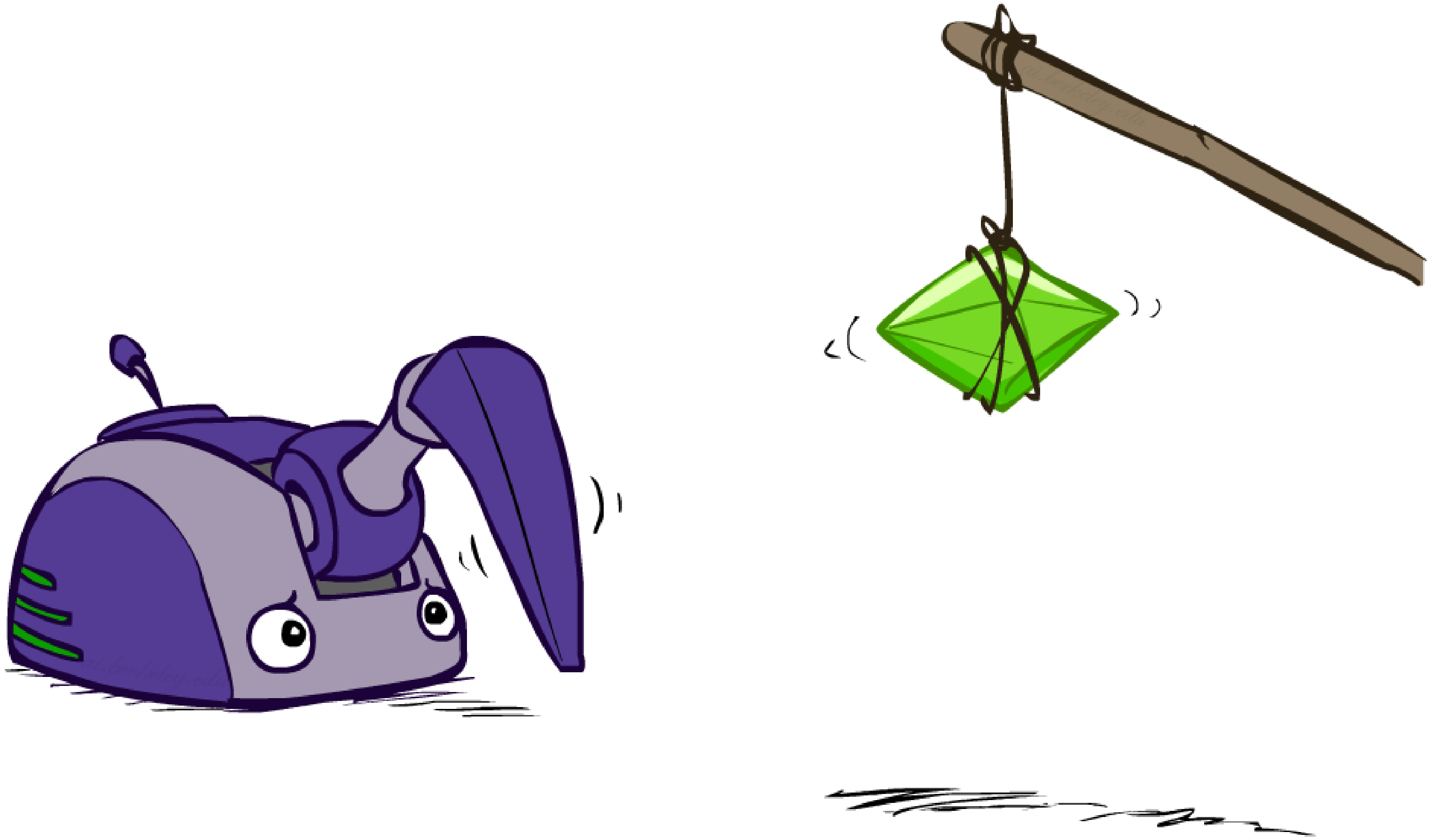
Reinforcement Learning

Instructors: Dan Klein and Pieter Abbeel

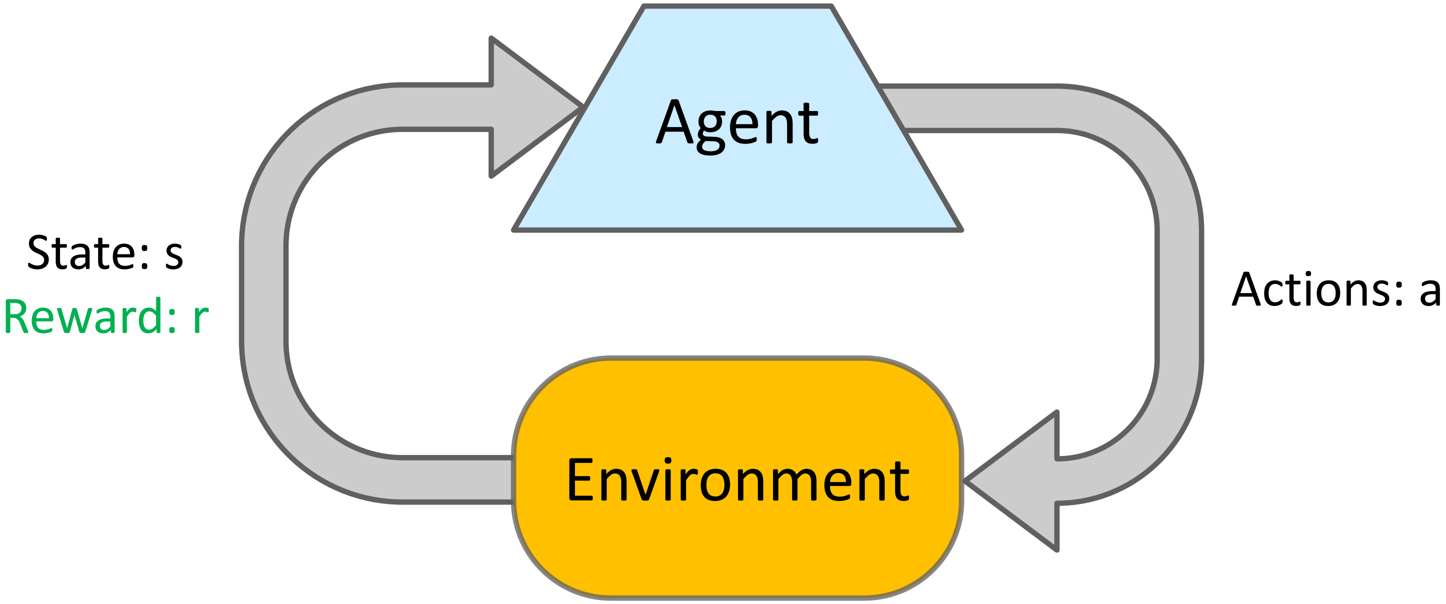
University of California, Berkeley

[These 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.berkeley.edu.]

Reinforcement Learning



Reinforcement Learning



Basic idea:

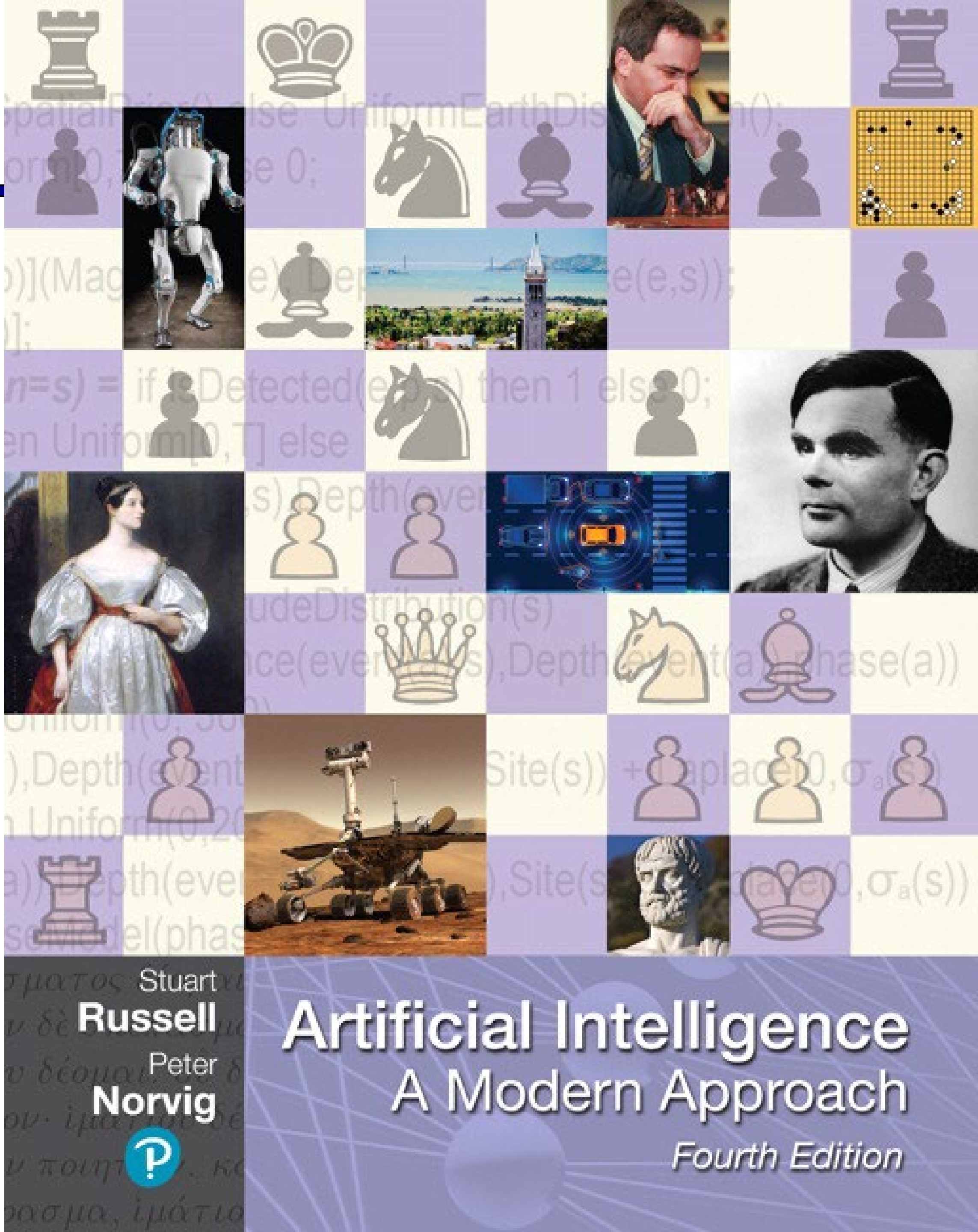
- Receive feedback in the form of rewards
- The agent's utility at a state is defined by the sum of rewards agent obtained when at that state
- Agent must learn to act so as to maximize expected rewards
- All learning is based on observed samples of outcomes!

Slide additions by Betke

AIMA

Chapter 22 – Reinforcement Learning
Sections 22.1-22.5

Chapter 17.3 – Bandit Problems



Example: Learning to Walk

Initial

A Learning Trial

After Learning [1K Trials]

[Kohl and Stone, ICRA 2004]

Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Training

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – training]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

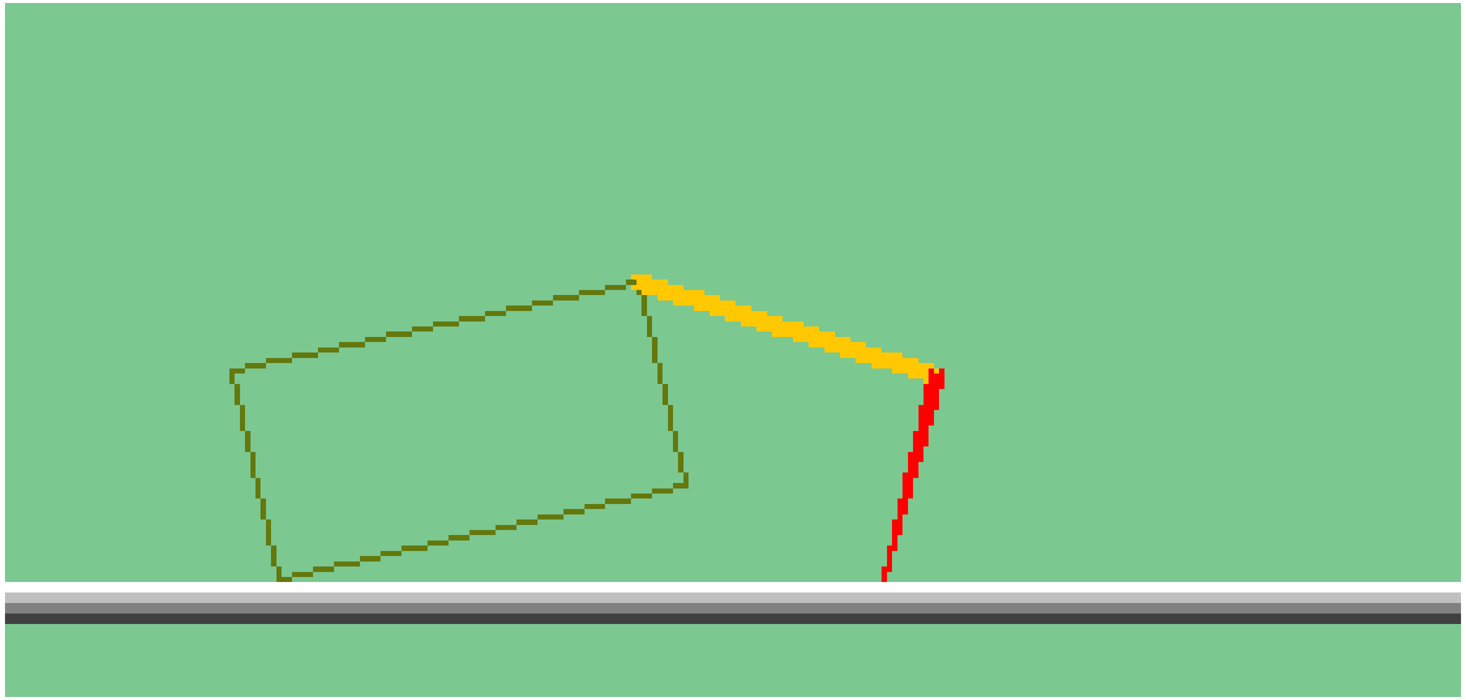
[Video: AIBO WALK – finished]

Example: Toddler Robot

[Tedrake, Zhang and Seung, 2005]

[Video: TODDLER – 40s]

The Crawler!

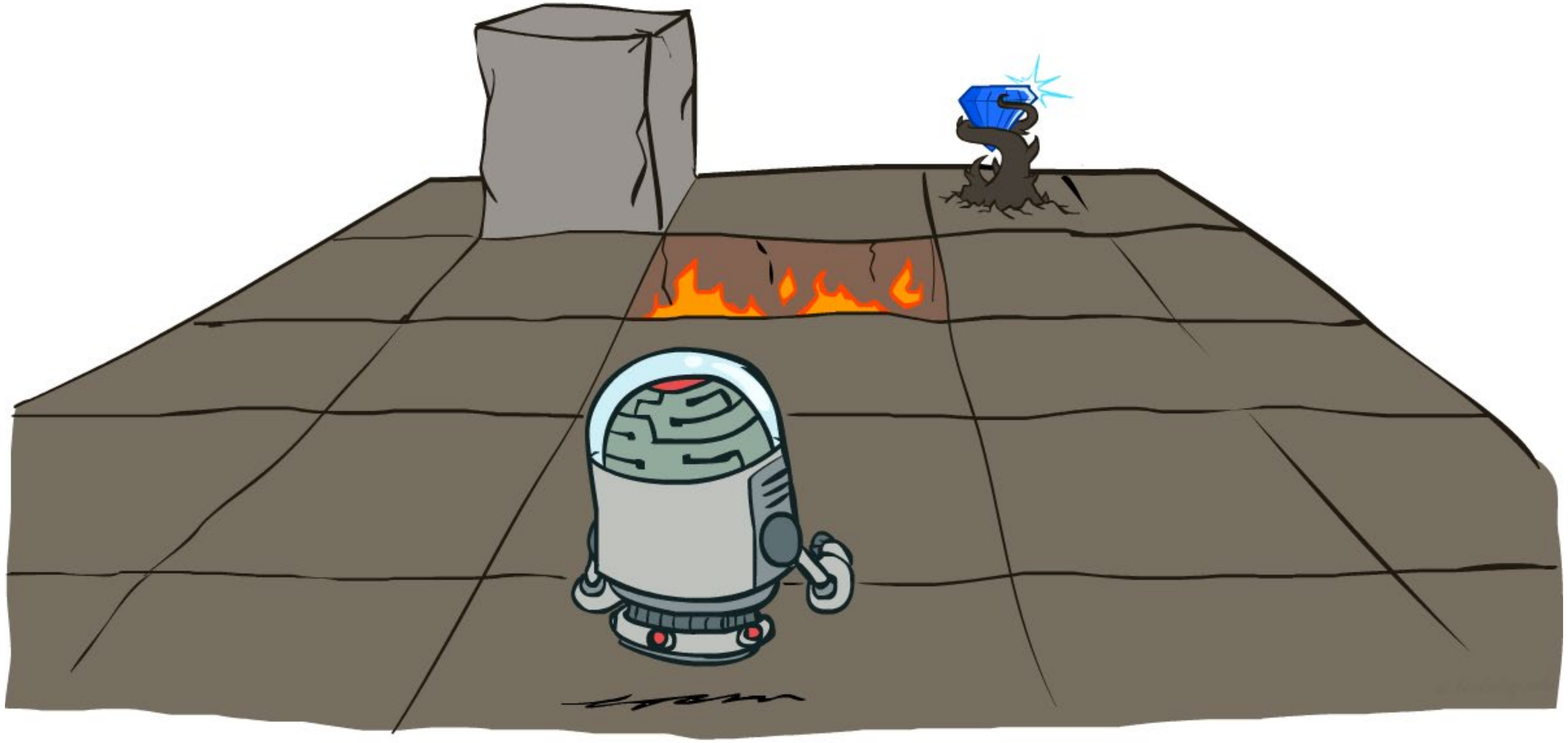


[Demo: Crawler Bot (L10D1)] [You, in Project 3]

Video of Demo Crawler Bot

CS 188: Artificial Intelligence

Markov Decision Processes

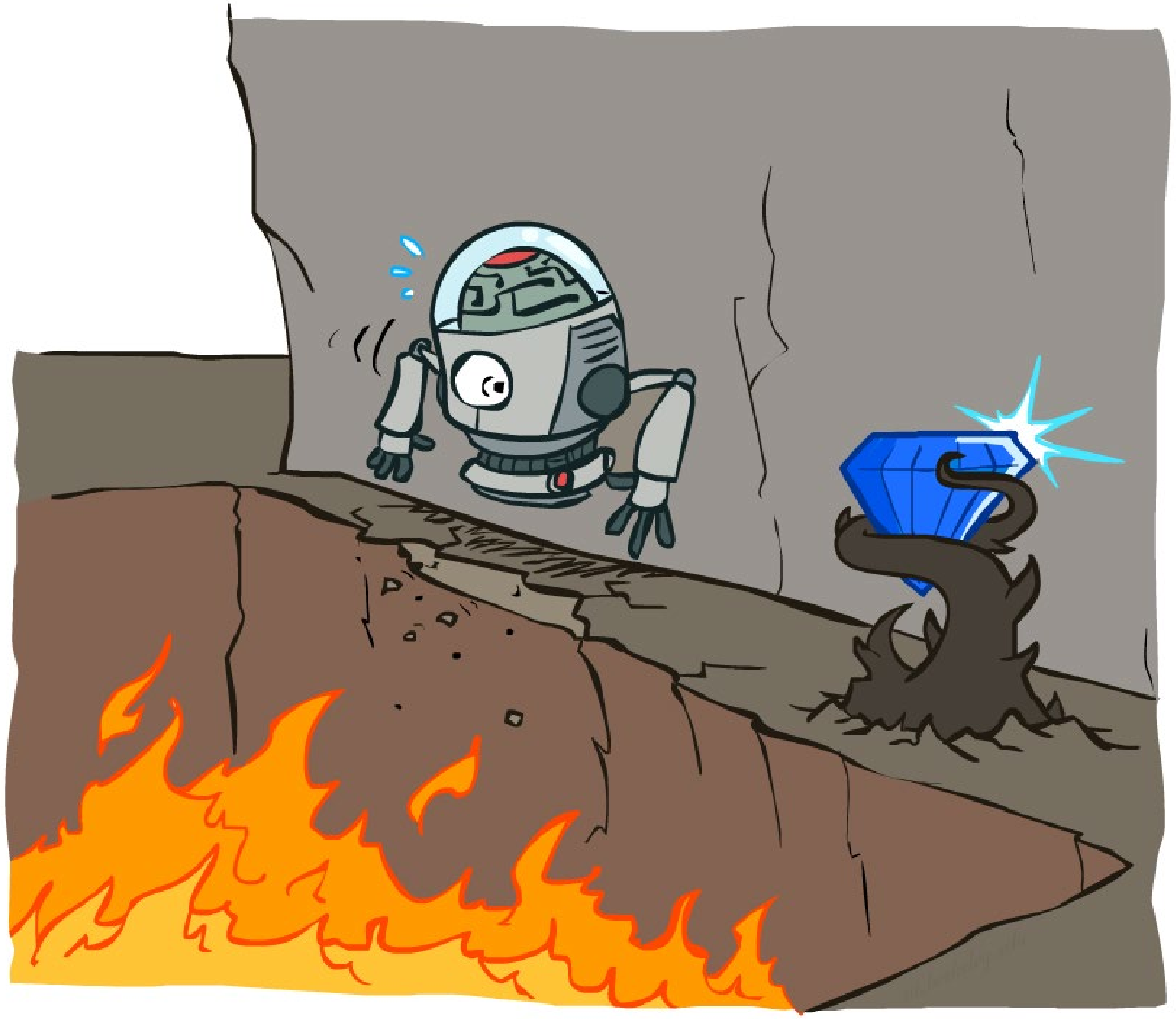


Instructors: Dan Klein and Pieter Abbeel

University of California, Berkeley

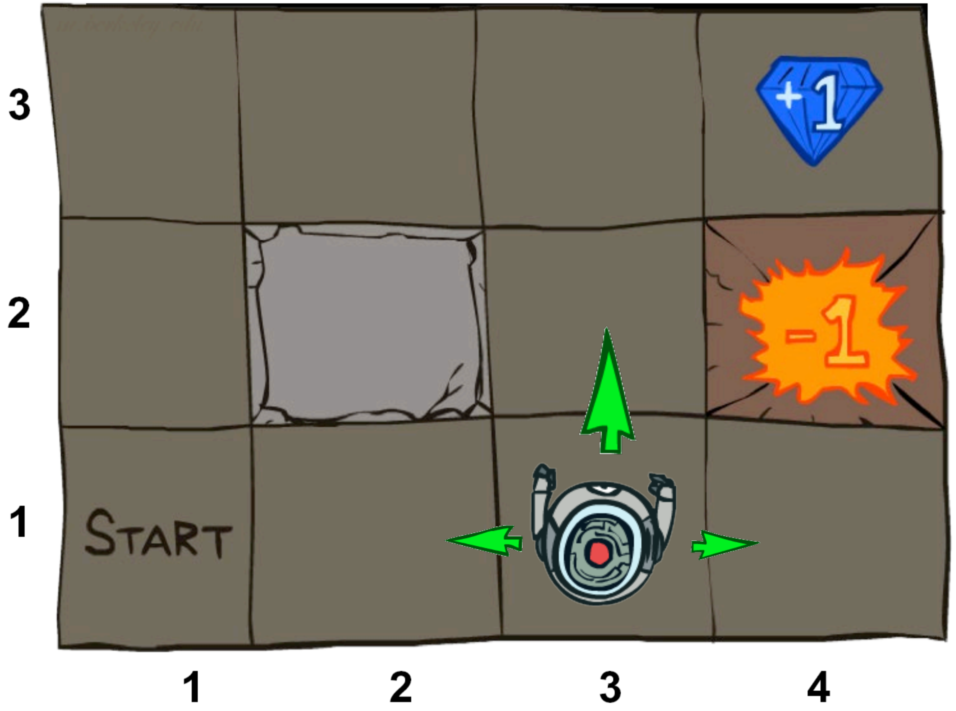
[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Non-Deterministic Search



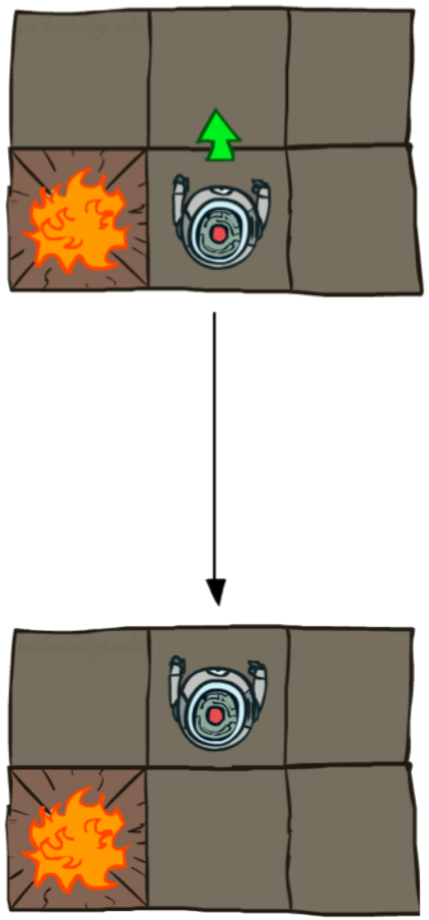
Example: Grid World

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

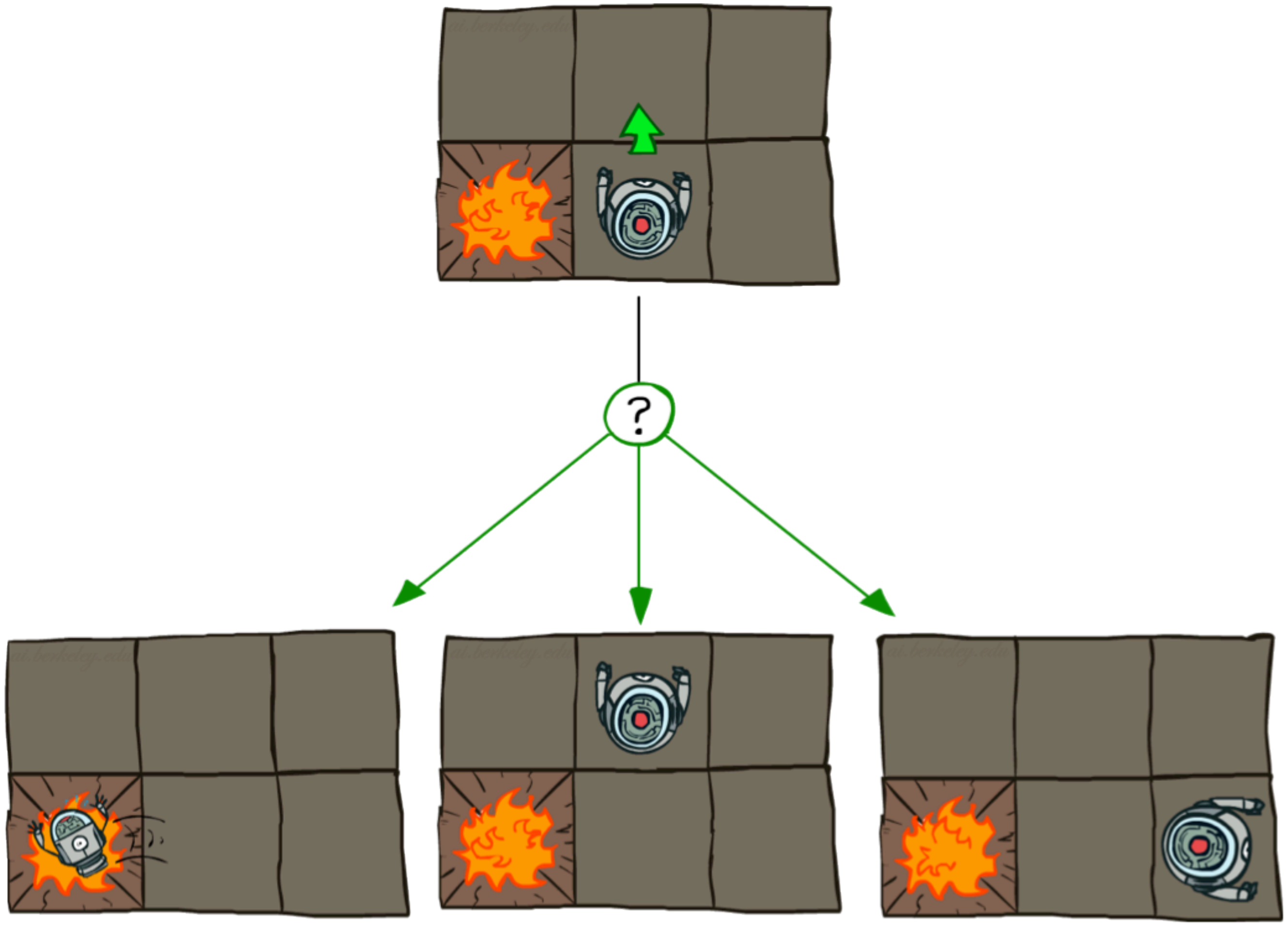


Grid World Actions

Deterministic Grid World

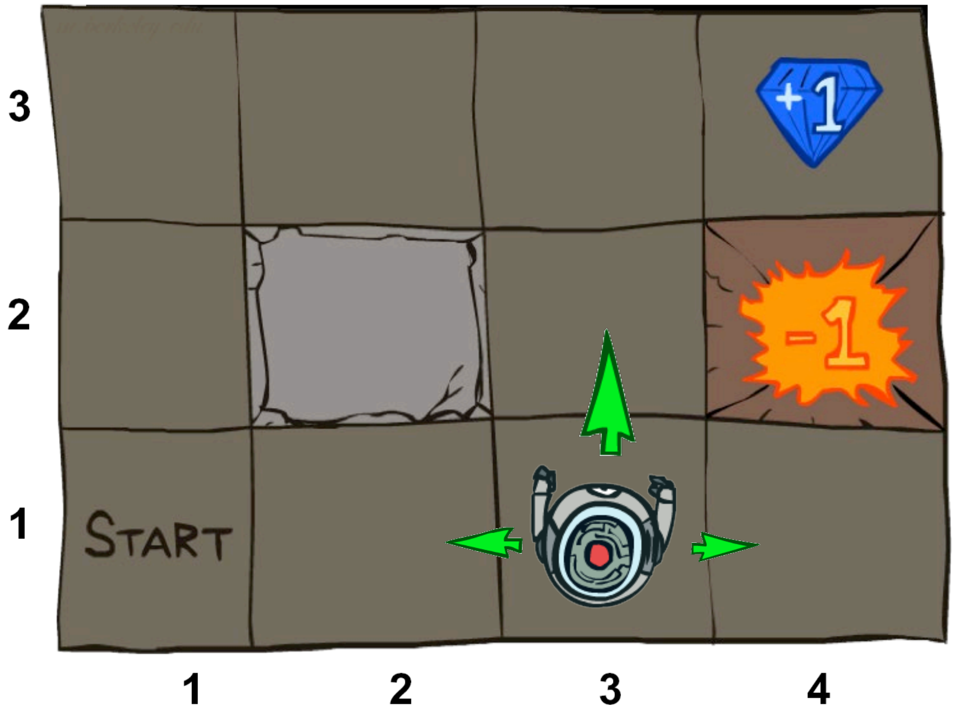


Stochastic Grid World



Markov Decision Processes

- An MDP is defined by:
  - A set of states s ∈ S
  - A set of actions a ∈ 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

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

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

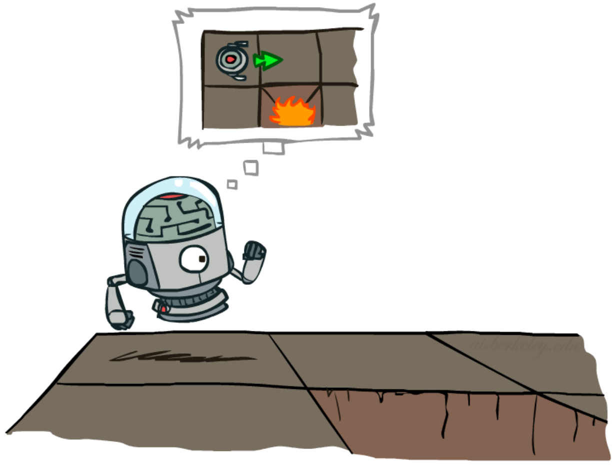
$$=$$

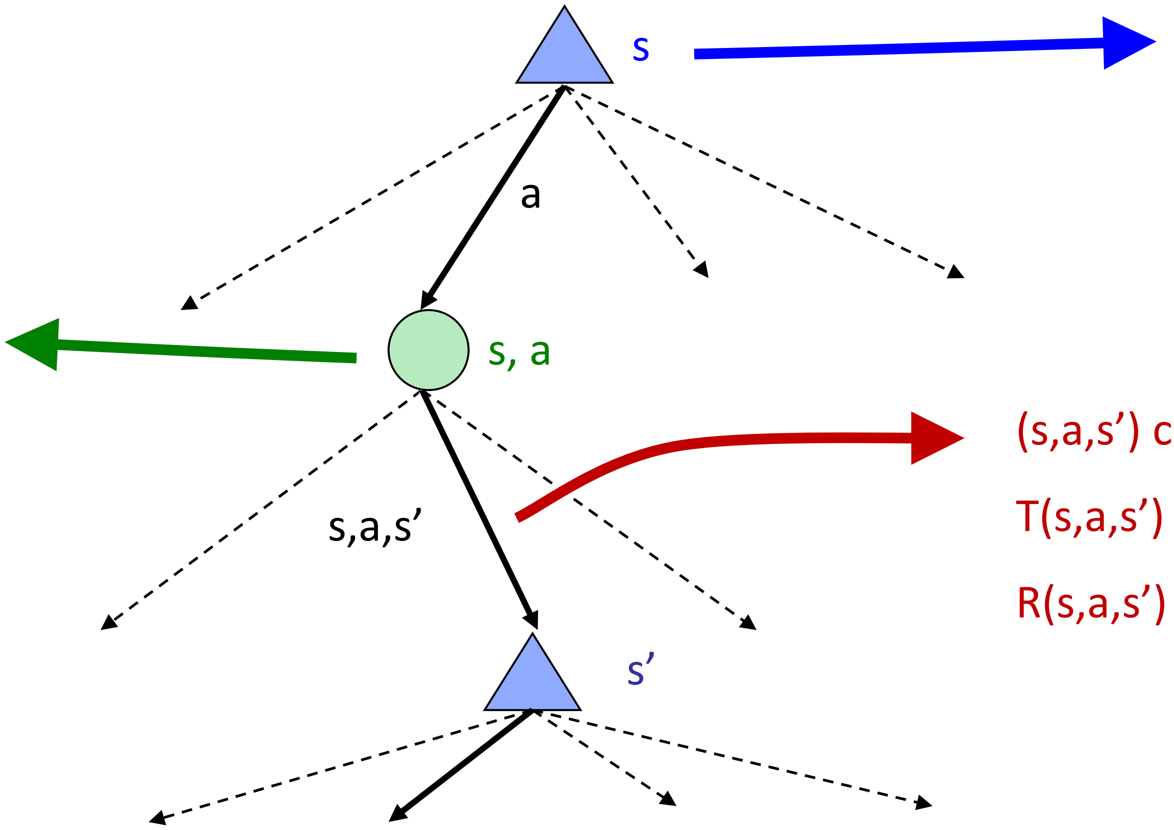
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- The successor function only depends on the current state, not the history

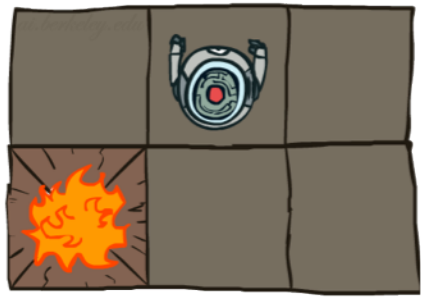
Andrey Markov
(1856-1922)

MDP Search Trees





s is a *state*

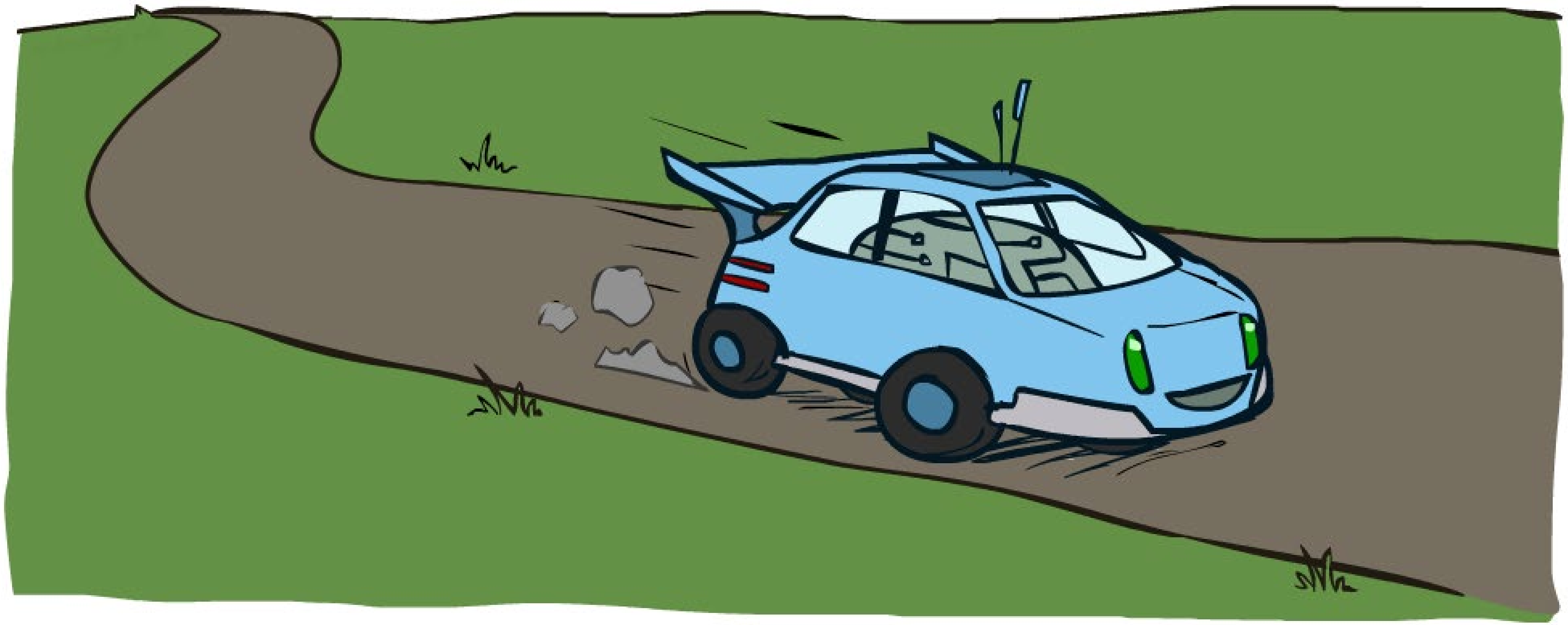


(s,a,s’) called a *transition*

T(s,a,s’) = P(s’|s,a)

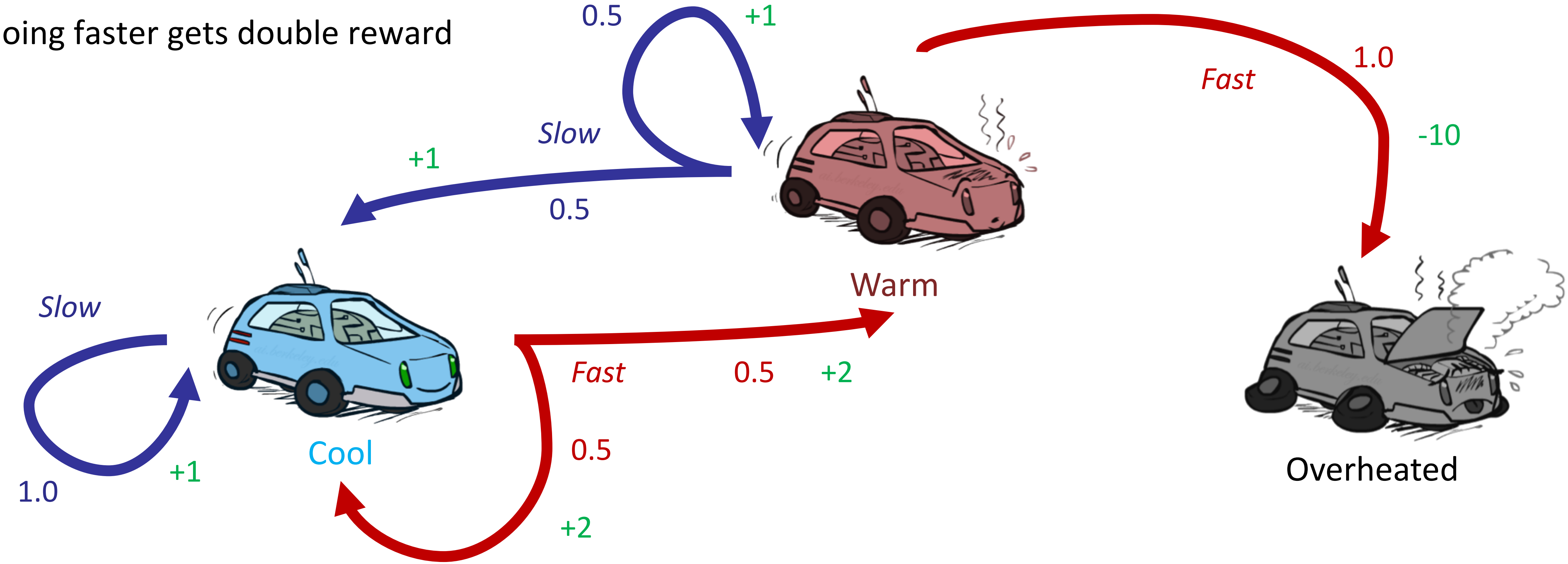
R(s,a,s’)

Example: Racing

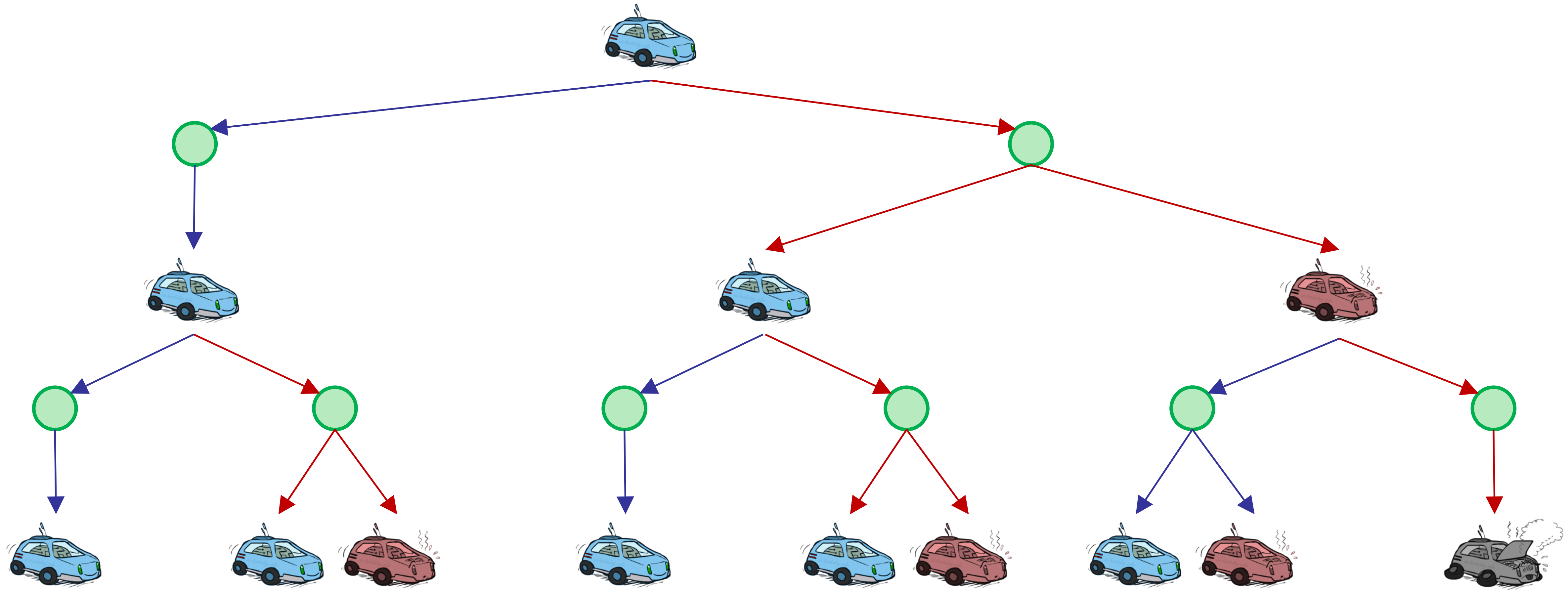


Example: Racing

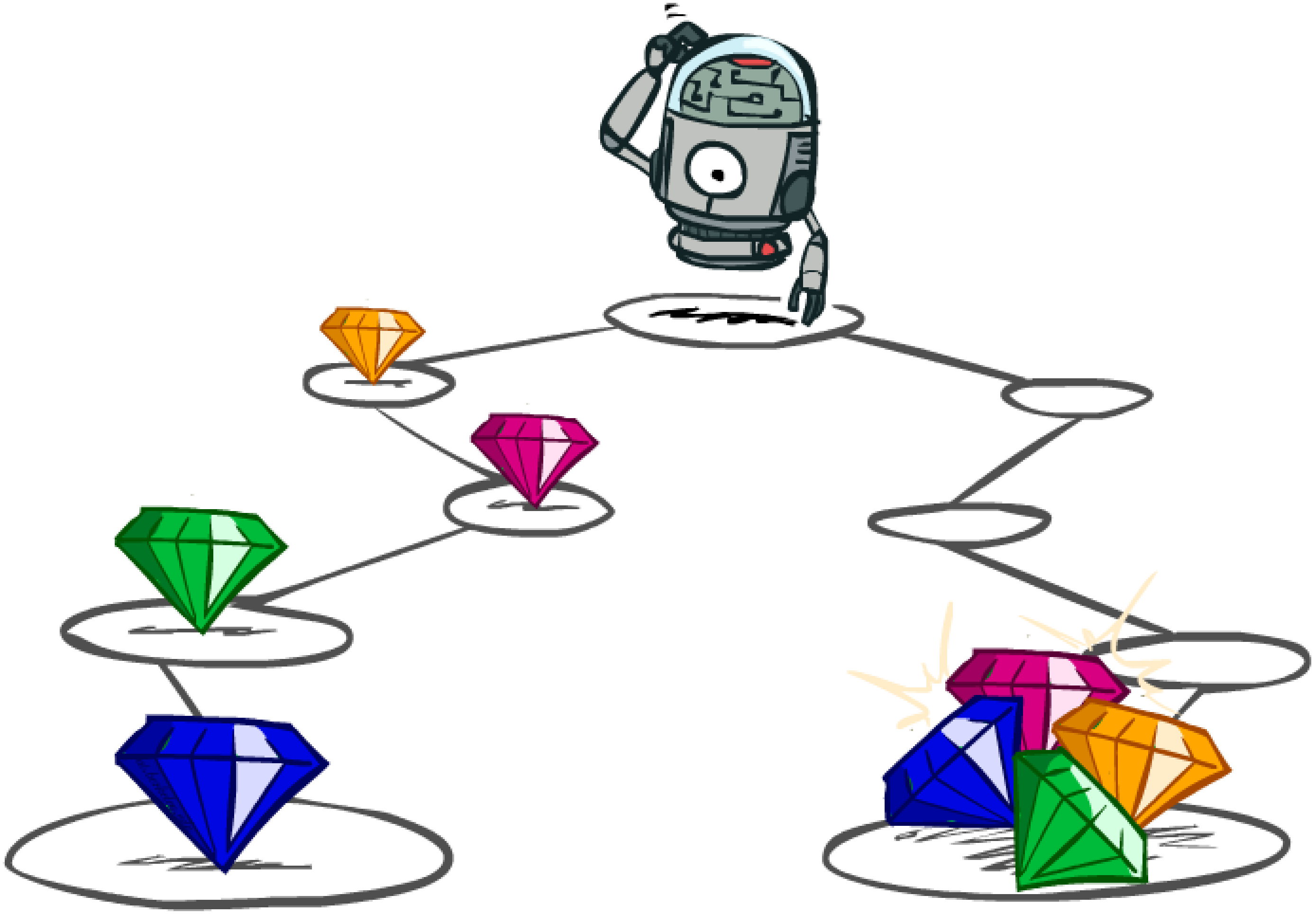
- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Racing Search Tree



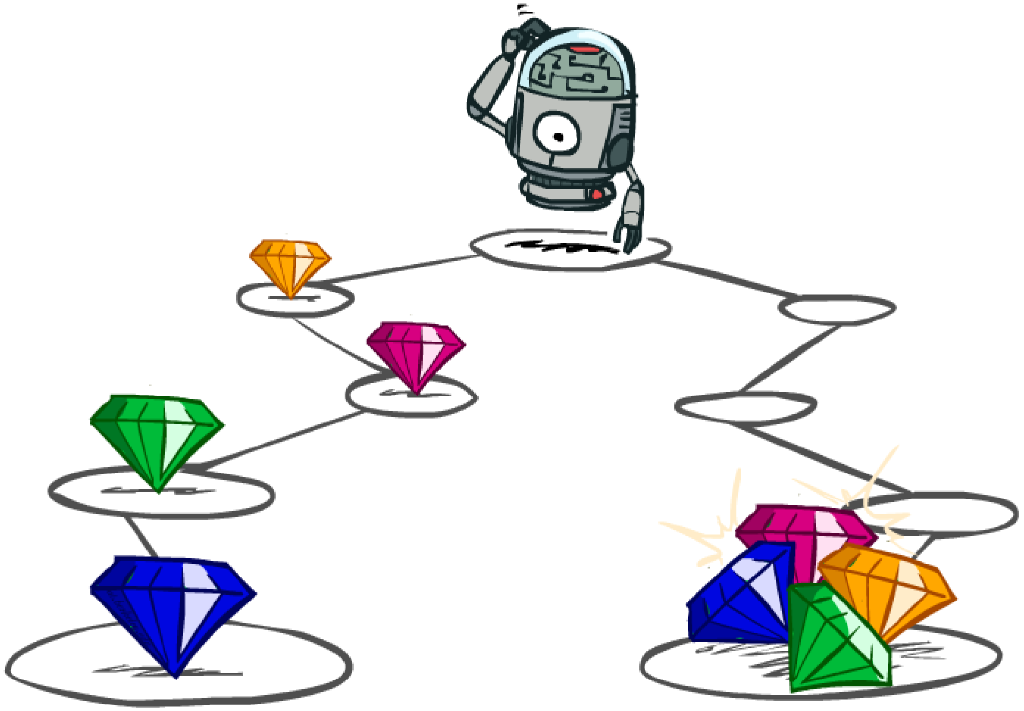
Utilities of Sequences



Utilities of Sequences

What preferences should an agent have over reward sequences?

- More or less?
- Now or later?



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

γ

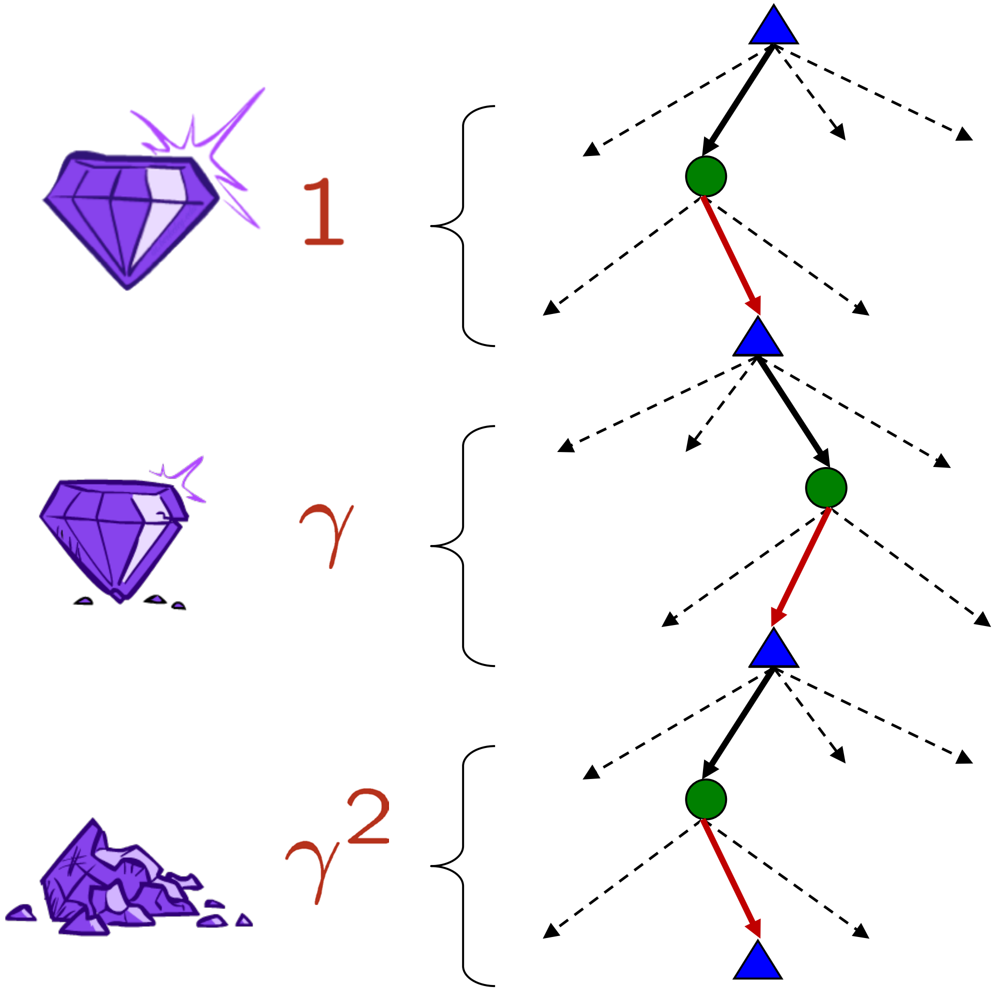
Worth Next Step

γ^2

Worth In Two Steps

Discounting

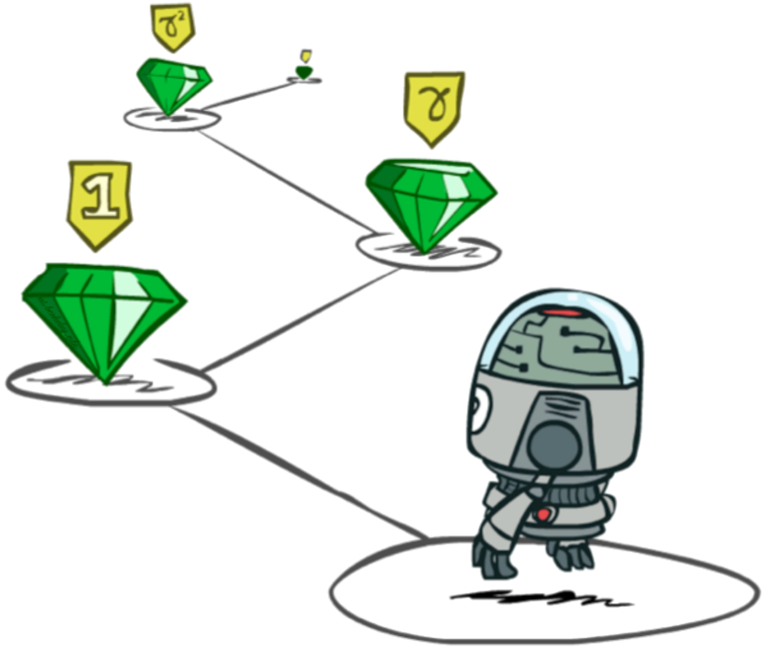
- How to discount?
  - Each time we descend a level, we multiply in the discount once
- Why discount?
  - Sooner rewards probably do have higher utility than later rewards
- Example: discount of 0.5
  - U([1,2,3]) = 1*1 + 0.5*2 + 0.25*3
  - U([1,2,3]) < U([3,2,1])



Stationary Preferences

- Theorem: if we assume stationary preferences:

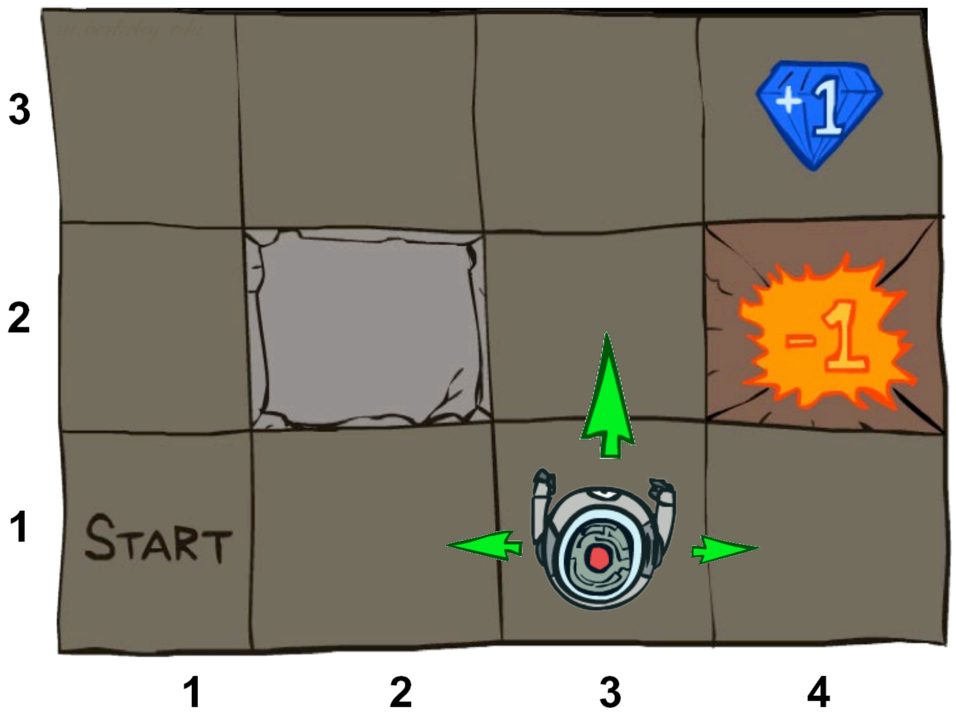
$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$

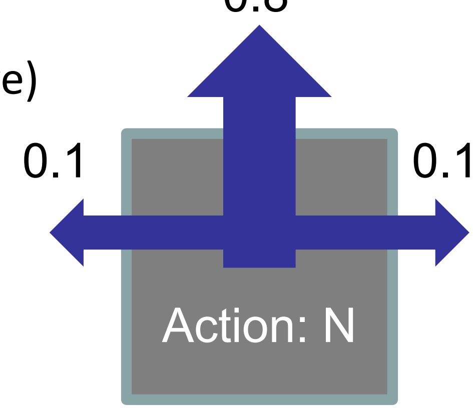


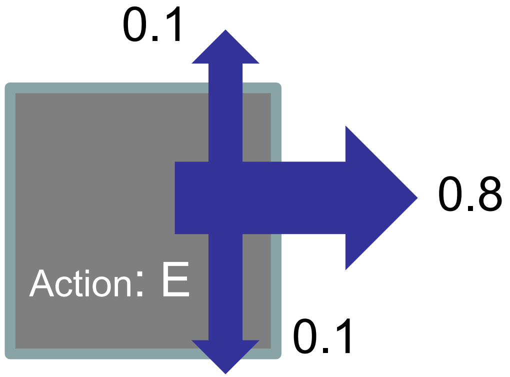
- Then: there are only two ways to define utilities
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

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



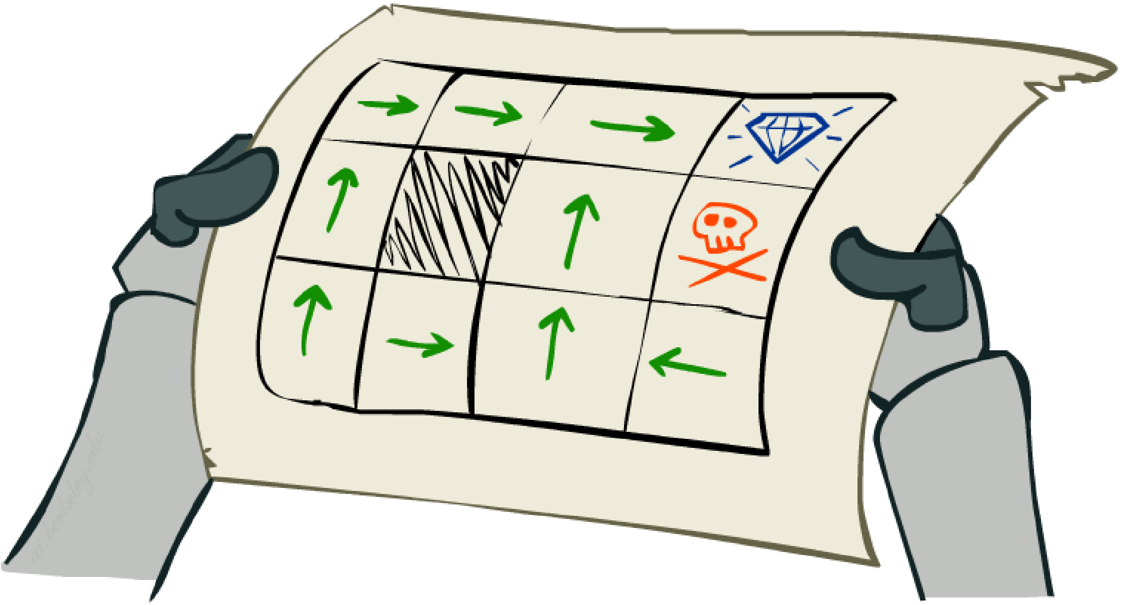




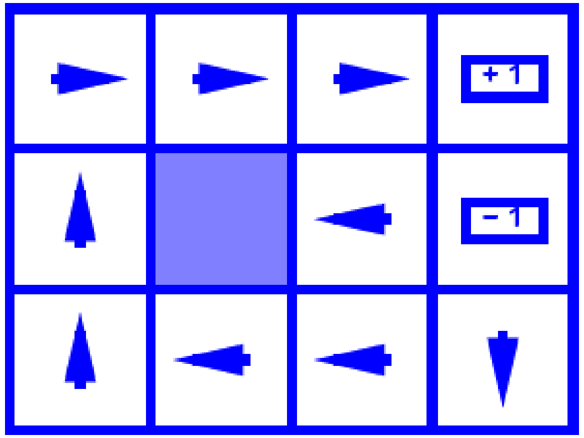
Policies

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

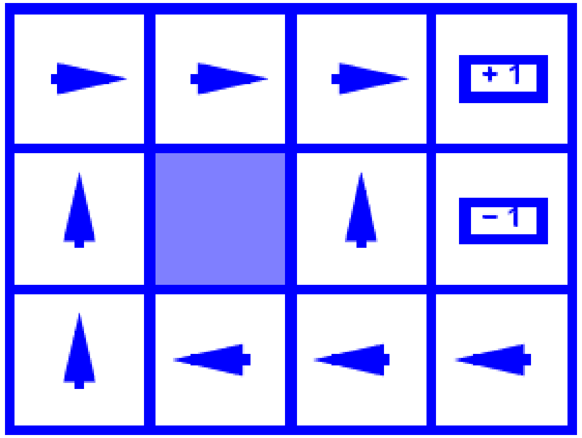
- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed



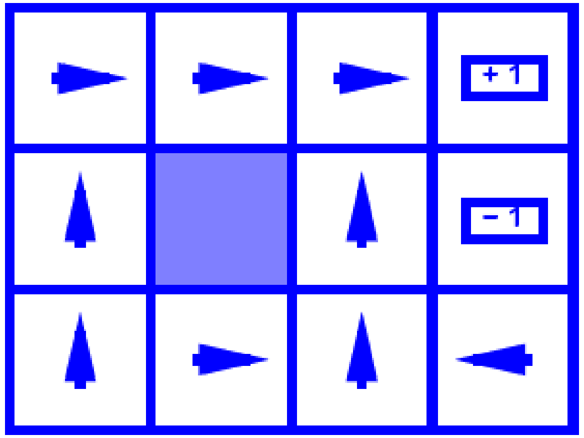
Optimal Policies



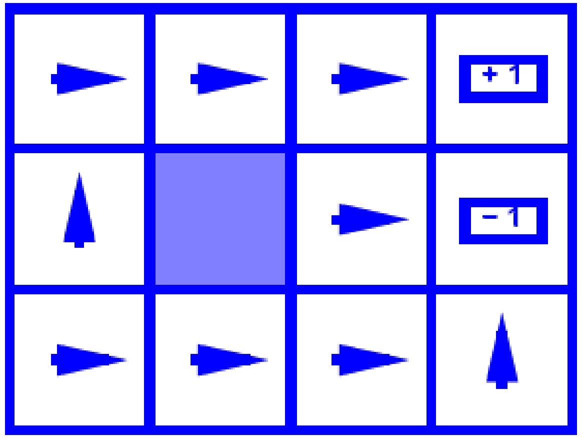
R(s) = -0.01



R(s) = -0.03



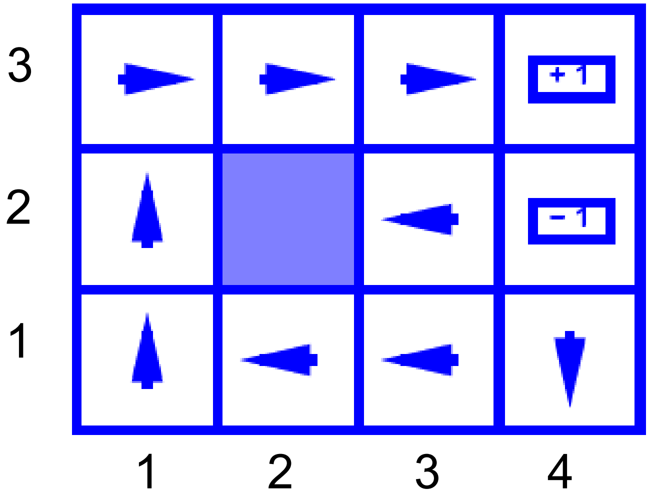
R(s) = -0.4



R(s) = -2.0

Optimal Policy

R(s) = -0.01



3	→	→	→	+1
2	↑	▨	←	- 1
1	↑	←	←	↓
	1	2	3	4

R(s) is the “living reward” – each time the agent makes a step, it costs 0.01.

Here, the agent tries to avoid falling into the terminal state with the “-1” reward.

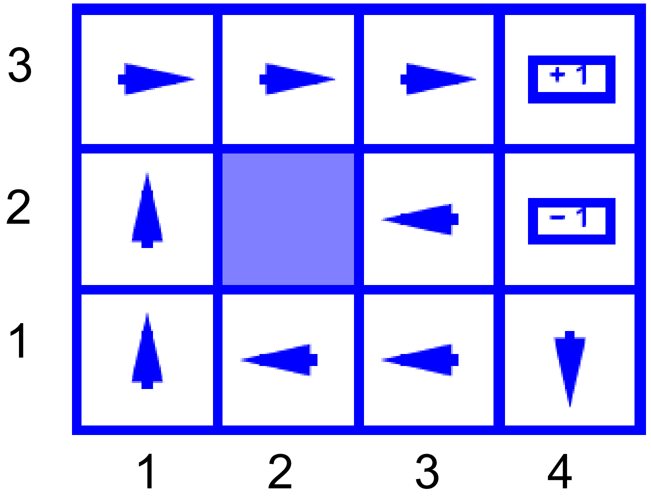
When following the action arrows in each of the non-terminal grid states, there is NO risk of ever falling into the “-1” end state.

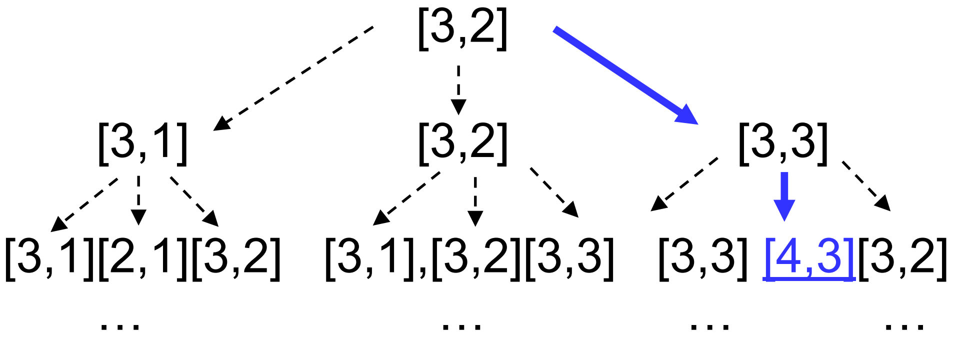
When next two the “-1” end state, the agent heads in the opposite direction, even if the likelihood of banging against the wall that is directly ahead is 80%.

Slide by Betke

Optimal Policy

R(s) = -0.01





R(s) is the "living reward" – each time the agent makes a step, it costs 0.01.

Here, the agent tries to avoid falling into the terminal state with the "-1" reward.

When following the action arrows in each of the non-terminal grid states, there is NO risk of ever falling into the "-1" end state.

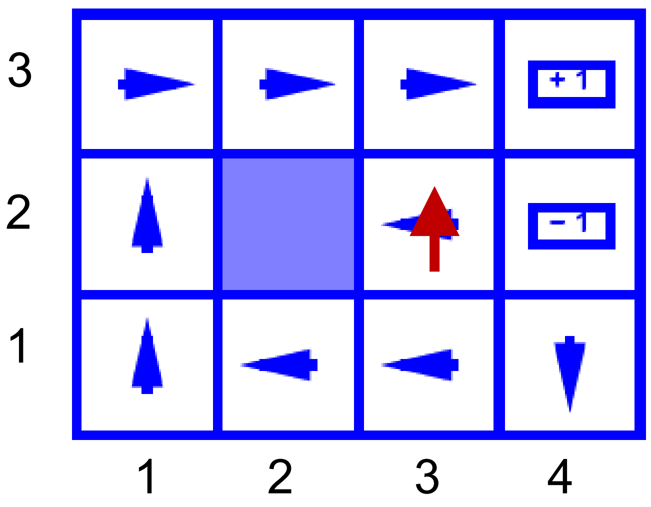
When next two the "-1" end state, the agent heads in the opposite direction, even if the likelihood of banging against the wall that is directly ahead is 80%.

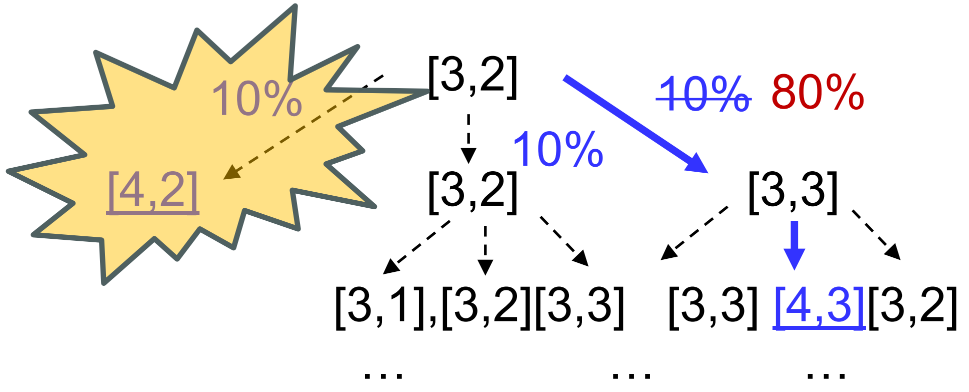
Sum of rewards of all 2-step state sequences starting at [3,2]:
-0.01-0.01
Except:
-0.01-0.01+1

Slide by Betke

Policy would not be optimal if: ↑

R(s) = -0.01





R(s) is the “living reward” – each time the agent makes a step, it costs 0.01.

Here, the agent tries to avoid falling into the terminal state with the “-1” reward.

When following the action arrows in each of the non-terminal grid states, there is NO risk of ever falling into the “-1” end state.

When next two the “-1” end state, the agent heads in the opposite direction, even if the likelihood of banging against the wall that is directly ahead is 80%.

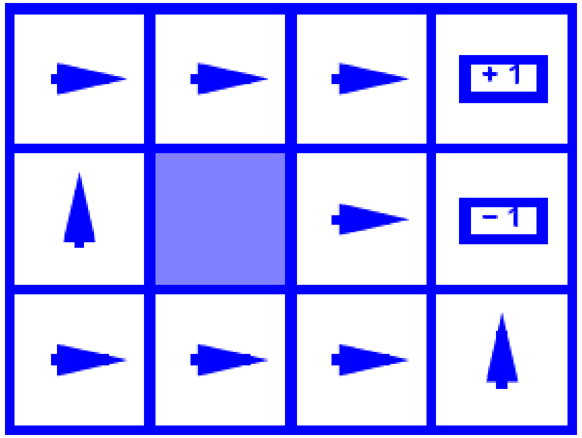
Sum of rewards of all 2-step state sequences starting at [3,2]:
-0.01-0.01=-0.02
Except:
-0.01-0.01+1 = 0.98

Now also 1 step sequence ending in [4,2]:
-0.01-1 = -1.01

Slide by Betke

Optimal Policy

Life is so dreadfully expensive that the agent heads to the nearest exit from all tiles, even if the nearest exit is [4,2], which will cost -1.



R(s) = -2.0

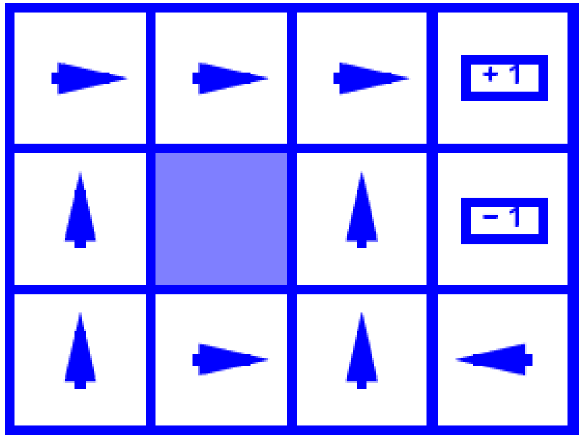
Slide additions by Betke

Optimal Policy

Agent's life is quite unpleasant: Each step costs 0.4.

Agent is willing to take the risk of falling into [4,2] from [3,2].

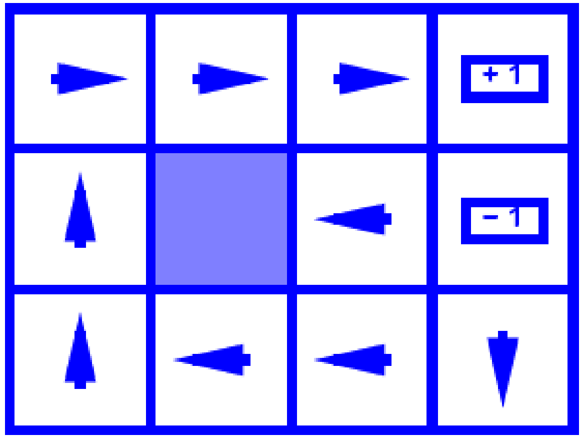
Agent tries to reach [4,3] on a shortest path from all grid tiles, even taking the shortcut from [3,1] to [3,2] (rather than keeping away from grid tiles neighboring [4,2].



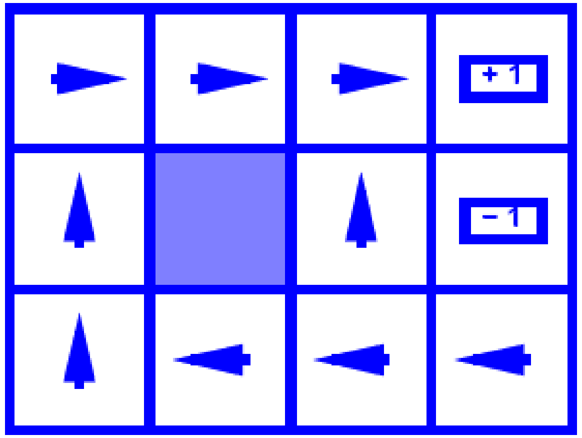
R(s) = -0.4

Slide by Betke

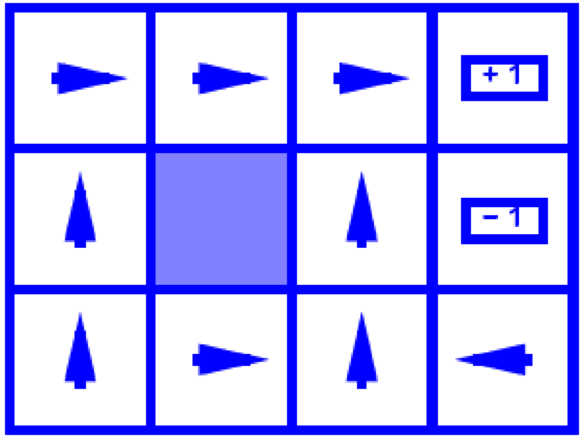
For different R(s): Different Optimal Policy



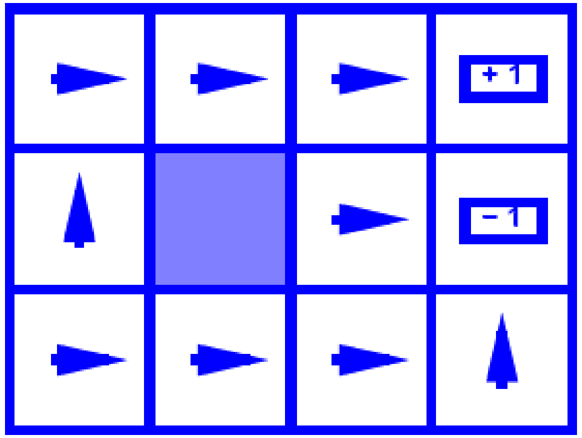
R(s) = -0.01



R(s) = -0.03



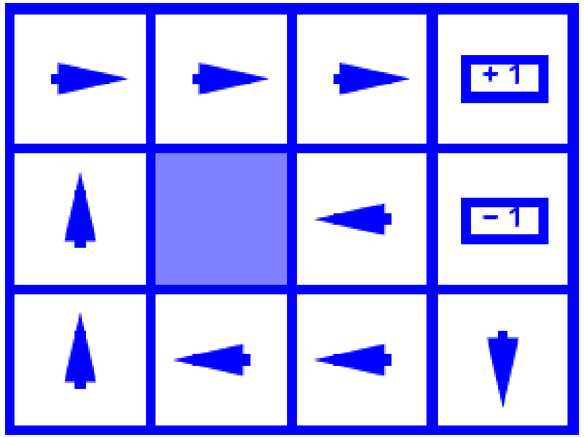
R(s) = -0.4



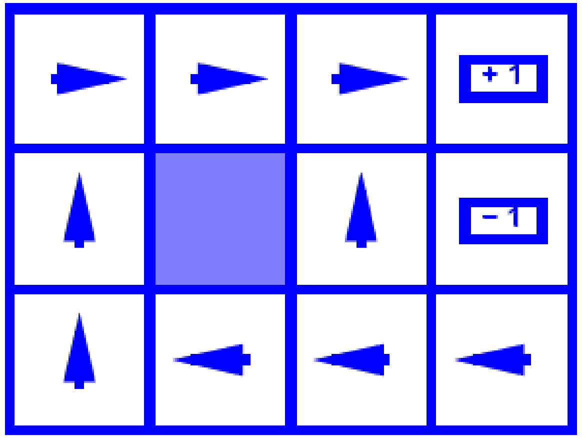
R(s) = -2.0

Slide additions by Betke

For different R(s): Different Optimal Policy



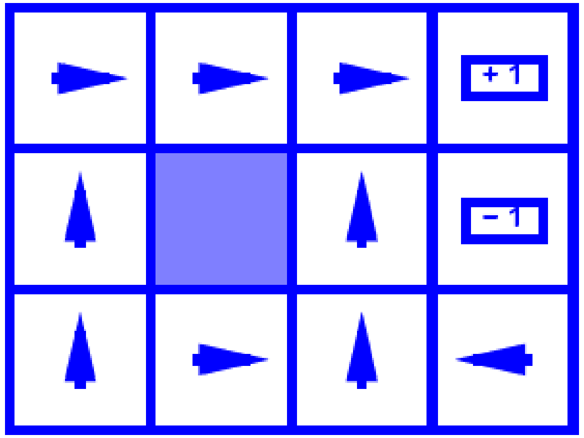
R(s) = -0.01



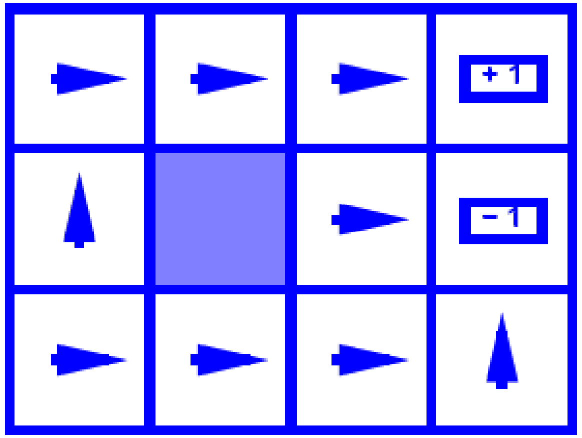
R(s) = -0.03

How can we determine the optimal policy?

That is: Which direction to move at each tile?



R(s) = -0.4



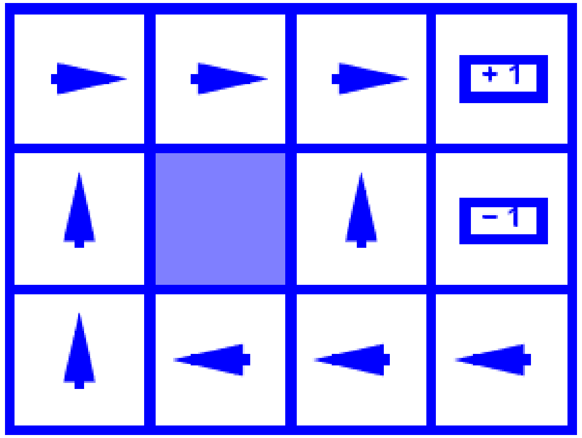
R(s) = -2.0

Slide additions by Betke

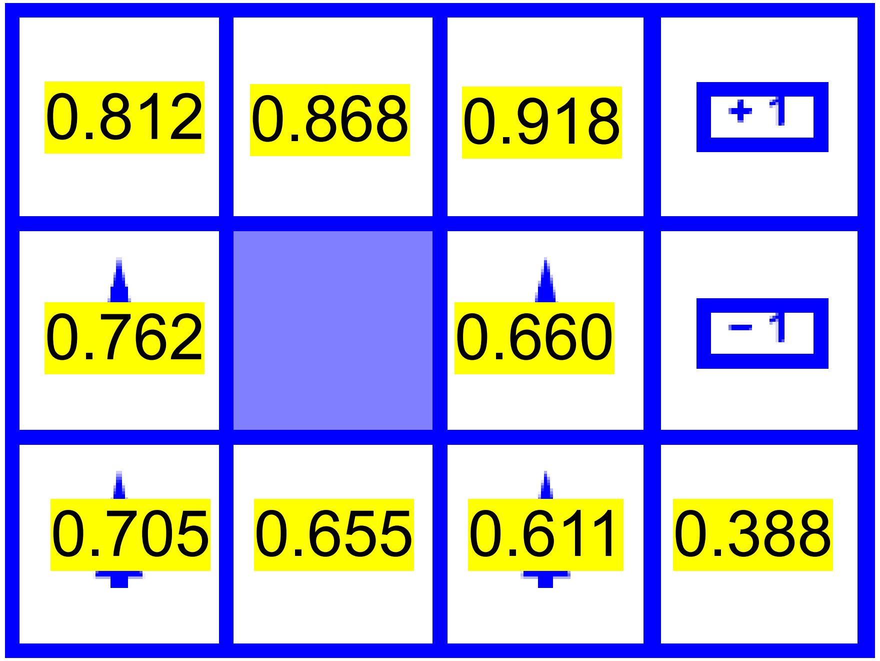
Utility of States, given Policy π

$U^{\pi}(s) = E\,[\,\sum_{t=0}^{\infty} \gamma^{t}\ R(s_t) \mid \pi, s_0 = s\,]$

If agent executes optimal policy π^*: Utility of a state $U(s) = U^{\pi^*}(s)$



R(s) = -0.04



0.812	0.868	0.918	+ 1
0.762		0.660	- 1
0.705	0.655	0.611	0.388

R(s) = -0.04 $\gamma = 1$

Slide by Betke

Utility of States, given Policy π

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid \pi, s_0 = s \right]$$

If agent executes optimal policy π*: Utility of a state $U(s) = U^{\pi^*}(s)$

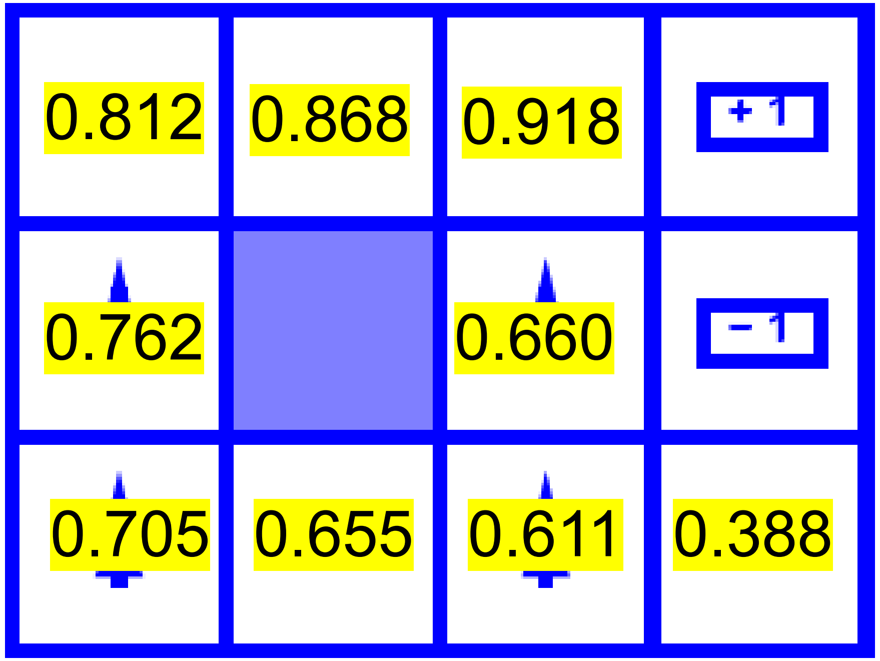
$$\pi^*(s) = \text{argmax}_a \sum_{s'} T(s,a,s')\, U(s')$$

Choose action that maximizes the expected utility of the subsequent state s'.

$$U(s) = R(s) + \underbrace{\gamma \max_a \sum_{s'} T(s,a,s')\, U(s')}$$

Immediate reward at s

Expected discounted utility of next state, assuming agent chooses optimal action



0.812	0.868	0.918	+1
0.762	▨	0.660	−1
0.705	0.655	0.611	0.388

R(s) = -0.04 $\gamma = 1$

Slide by Betke

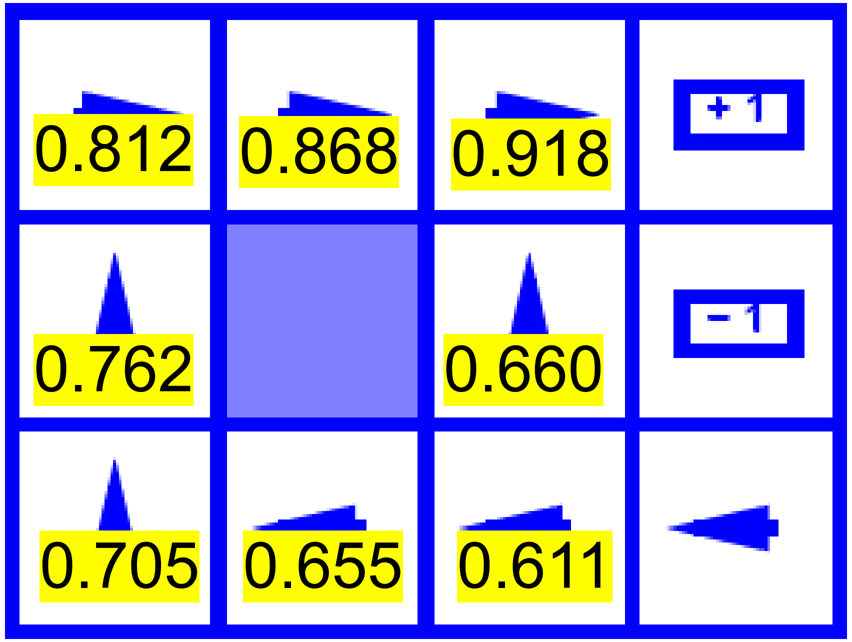
Utility of States, given Policy π

$$U(s) = R(s) + \underbrace{\gamma \max_a \sum_{s'} T(s,a,s')\, U(s')}_{\text{Expected discounted utility of next state, assuming agent chooses optimal action}}$$

Immediate reward at s

$U(1,1) = -0.04 + \gamma \max\{$

$0.8\,U(1,2)+0.1U(2,1)+0.1U(1,1),$	Up
$0.9\,U(1,1)+0.1U(1,2),$	Left
$0.9\,U(1,1)+0.1U(2,1),$	Down
$0.8\,U(2,1)+0.1U(1,1)+0.1U(1,2)$	Right



→ 0.812	→ 0.868	→ 0.918	+ 1
↑ 0.762	▨	↑ 0.660	− 1
↑ 0.705	← 0.655	← 0.611	←

R(s) = -0.04 $\gamma = 1$

Slide by Betke

Utility of States, given Policy π

$$U(s) = R(s) + \gamma \underbrace{\max_a \sum_{s'} T(s,a,s')\, U(s')}$$

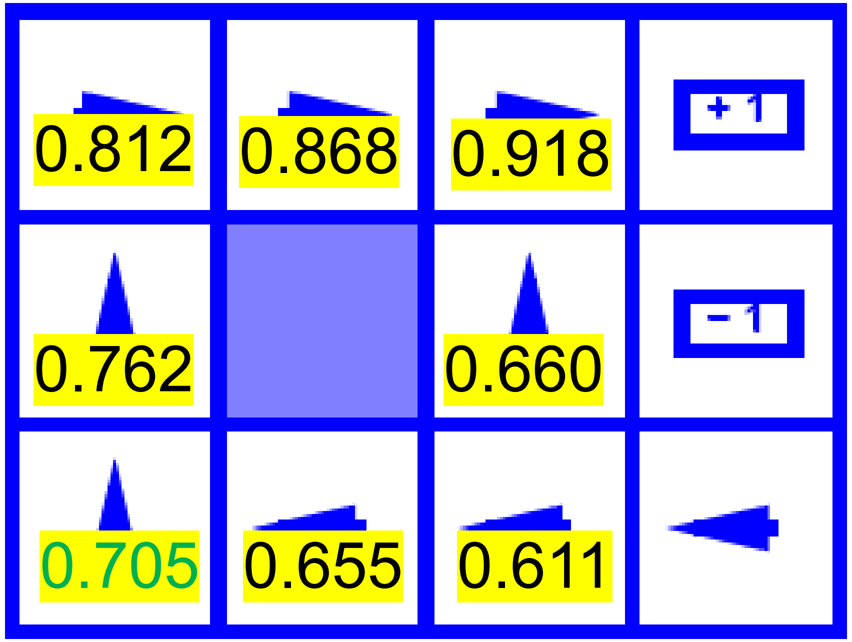
Immediate
reward at s

Expected discounted utility of next state,
assuming agent chooses optimal action

$U(1,1) = -0.04 + \gamma \max\{$

$0.8\,U(1,2)+0.1U(2,1)+0.1U(1,1),$	Up
$0.9\,U(1,1)+0.1U(1,2),$	Left
$0.9\,U(1,1)+0.1U(2,1),$	Down
$0.8\,U(2,1)+0.1U(1,1)+0.1U(1,2)$	Right

$= -0.04 + 0.8\times0.762+0.1\times0.655+0.1\times0.705 = 0.7056$



→ 0.812	→ 0.868	→ 0.918	+1
↑ 0.762	▨	↑ 0.660	−1
↑ 0.705	← 0.655	← 0.611	←

R(s) = -0.04 $\gamma = 1$

Slide by Betke

How to Compute Utility of States

Bellman Equation:

$U(s) = R(s) + \gamma \max_a \Sigma_{s'} T(s,a,s') U(s')$

Value Iteration Algorithm:

Initial U's = zero

Update: $U_{i+1}(s) = R(s) + \gamma \max_a \Sigma_{s'} T(s,a,s') U_i(s')$

Terminate upon convergence

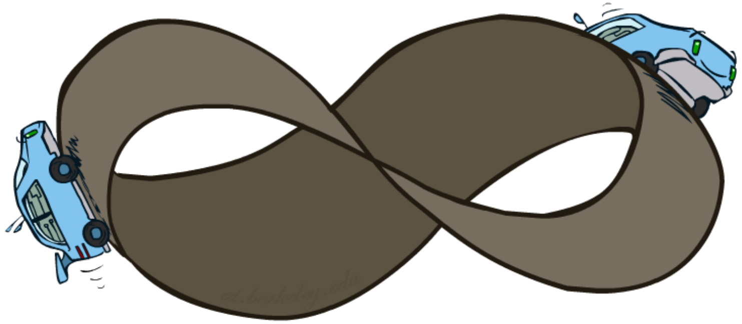
Infinite Utilities?!

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

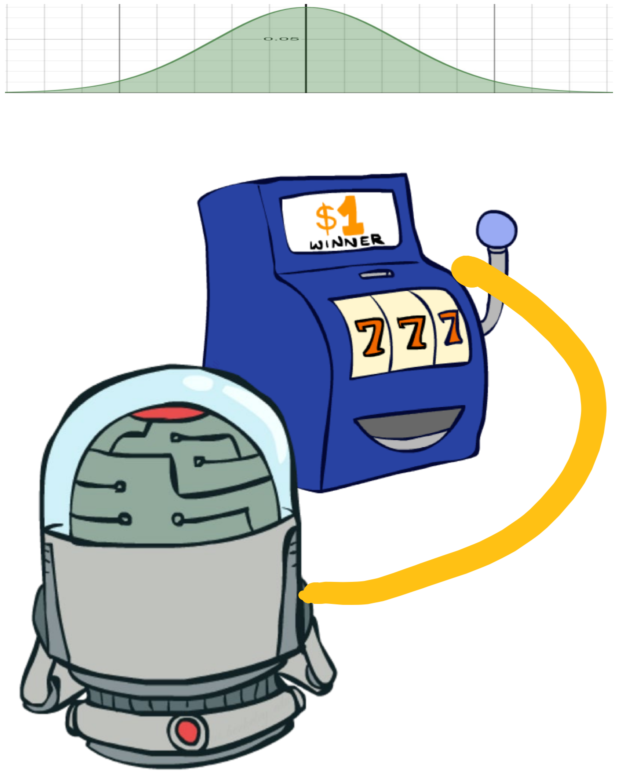
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Discounting: use $0 < \gamma < 1$

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

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



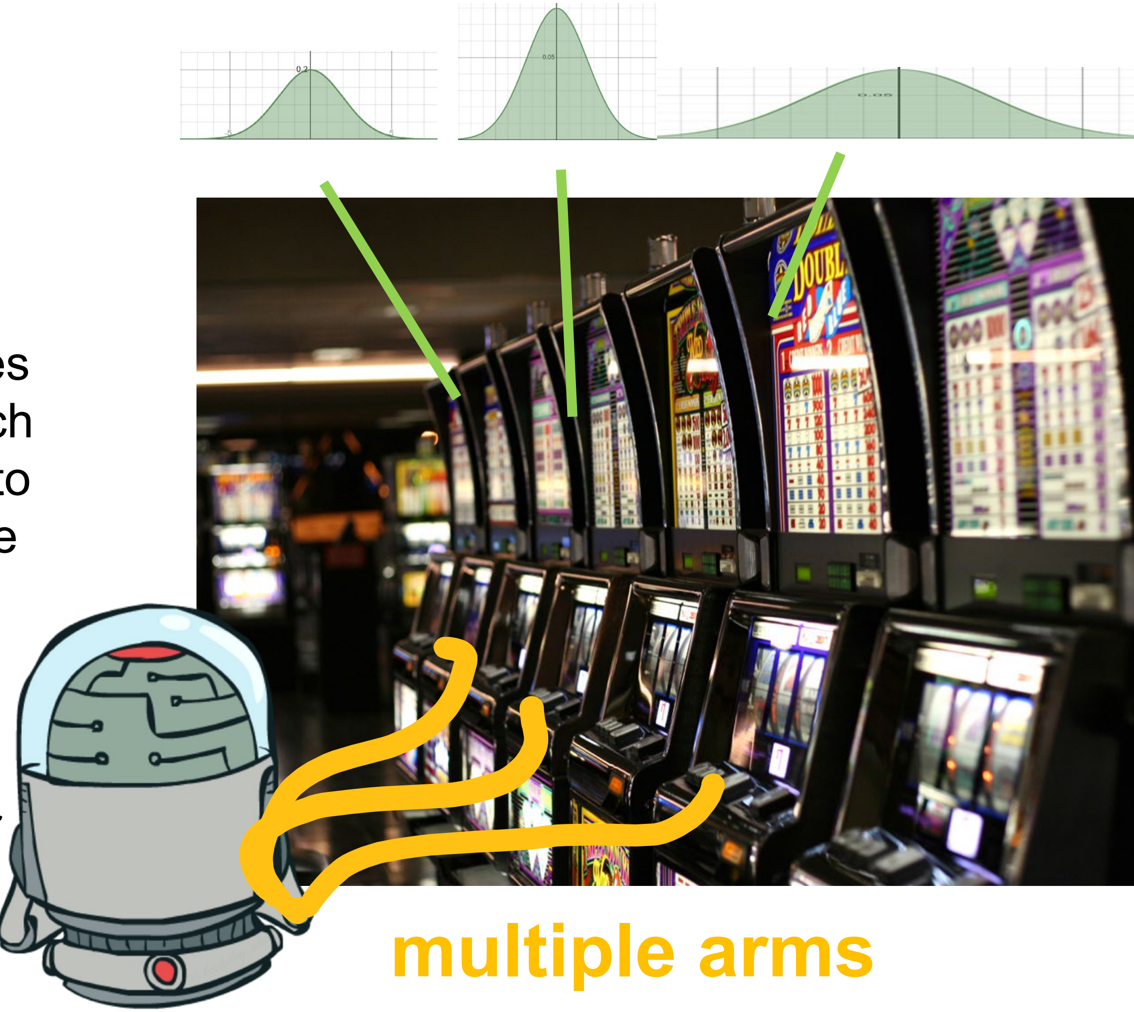
Single-arm bandit versus multi-armed bandits



A slot machine is called a bandit because it typically robs you of your money.

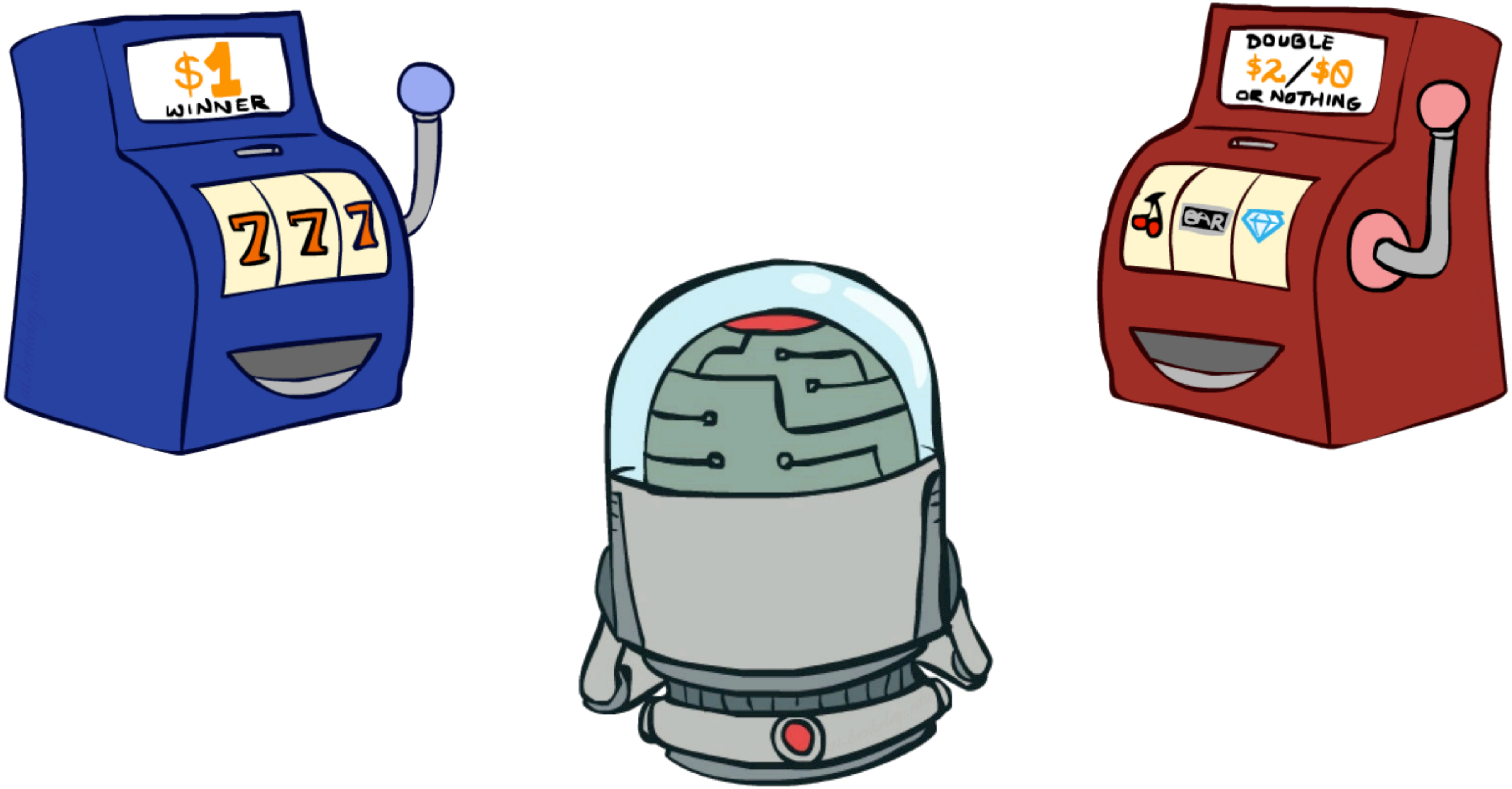
The gambler must decide which machines to play, how many times to play each machine and in which order to play them, and whether to continue with the current machine or try a different machine.

Each slot machine has its own distribution of wins. When the gambler pulls the lever down, it samples the bandit's distribution.



Slide by Betke, image by Yamaguchi先生, illustrations by Yim

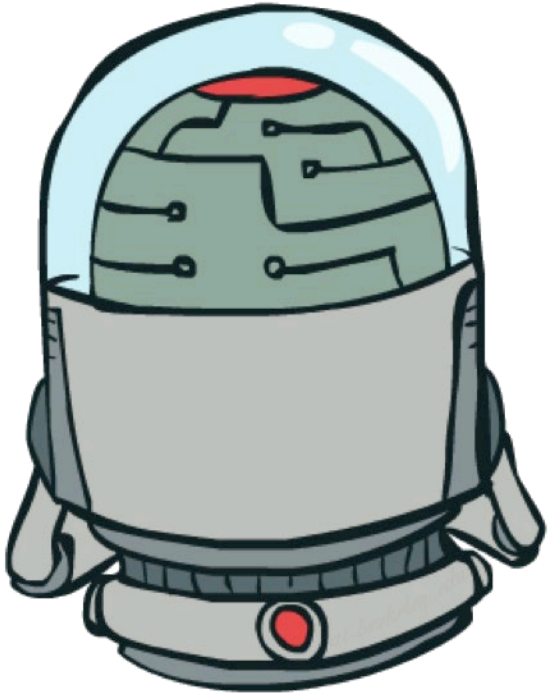
Double Bandits



Double Bandits

Gambler:
Which slot machine
should I play?

Cost of a single play:
$1



Blue:
A deterministic bandit.
Each time you play,
you are guaranteed to win $1.

Red:
A fair bandit:
Half the time you play, you win $2
the other half you win $0.

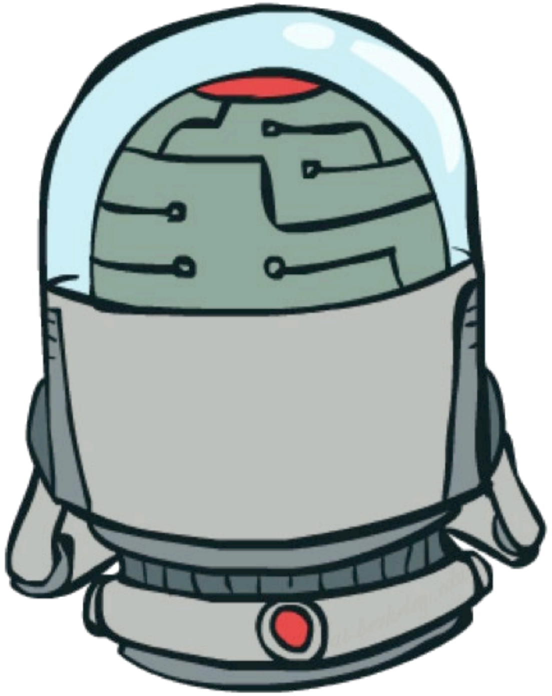
Slide additions by Betke

Double Bandits

Blue:
A deterministic bandit.
Each time you play,
you are guaranteed to win $1.

Gambler:
Which slot machine
should I play?

Cost of a single play:
$1



Red:
A generous non-bandit:
Each time you play, you win $2
¾ of the time and $0 ¼ of the time.

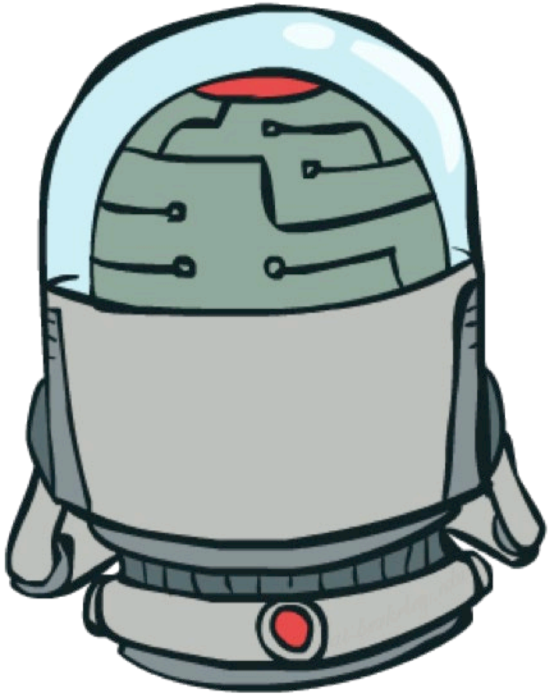
How much money are you expected
to win if you play 10 times?

Slide additions by Betke

Double Bandits

Gambler:
Which slot machine
should I play?

Cost of a single play:
$1



Blue:
A deterministic bandit.
Each time you play,
you are guaranteed to win $1.

Red:
A generous non-bandit:
Each time you play, you win $2 ¾ of the time and $0 ¼ of the time.

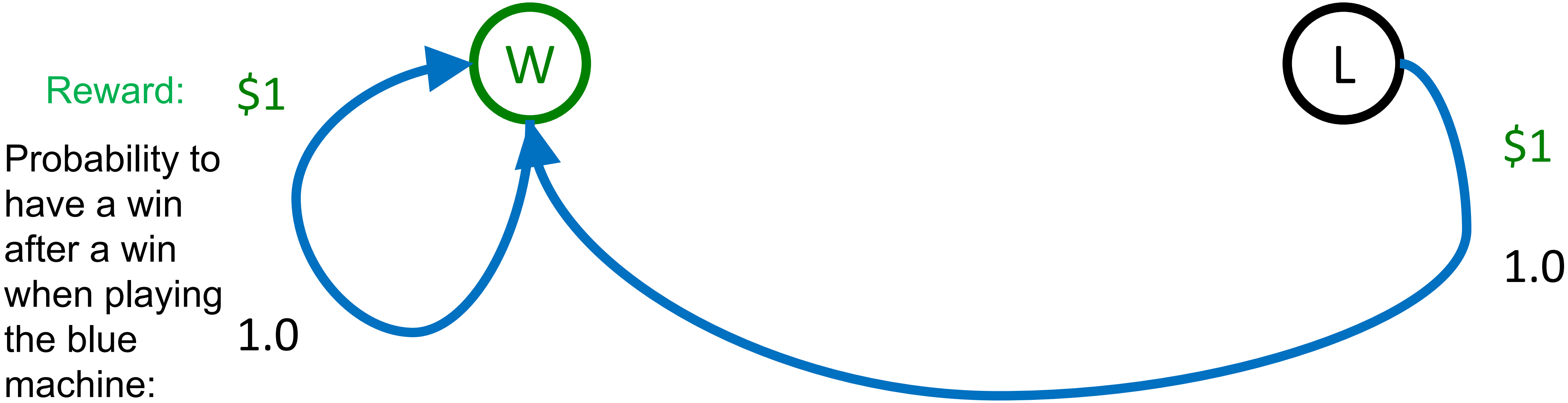
How much money are you expected to win if you play 10 times?

10 x $2 x ¾ + 10 x $0 x ¼ = $15

Slide additions by Betke

Double-Bandit Markov Decision Process

- Actions: *Blue*
- States: Win, Lose


W
L
Reward:
$1
Probability to have a win after a win when playing the blue machine:
1.0
$1
1.0

Slide additions by Betke

Double-Bandit Markov Decision Process

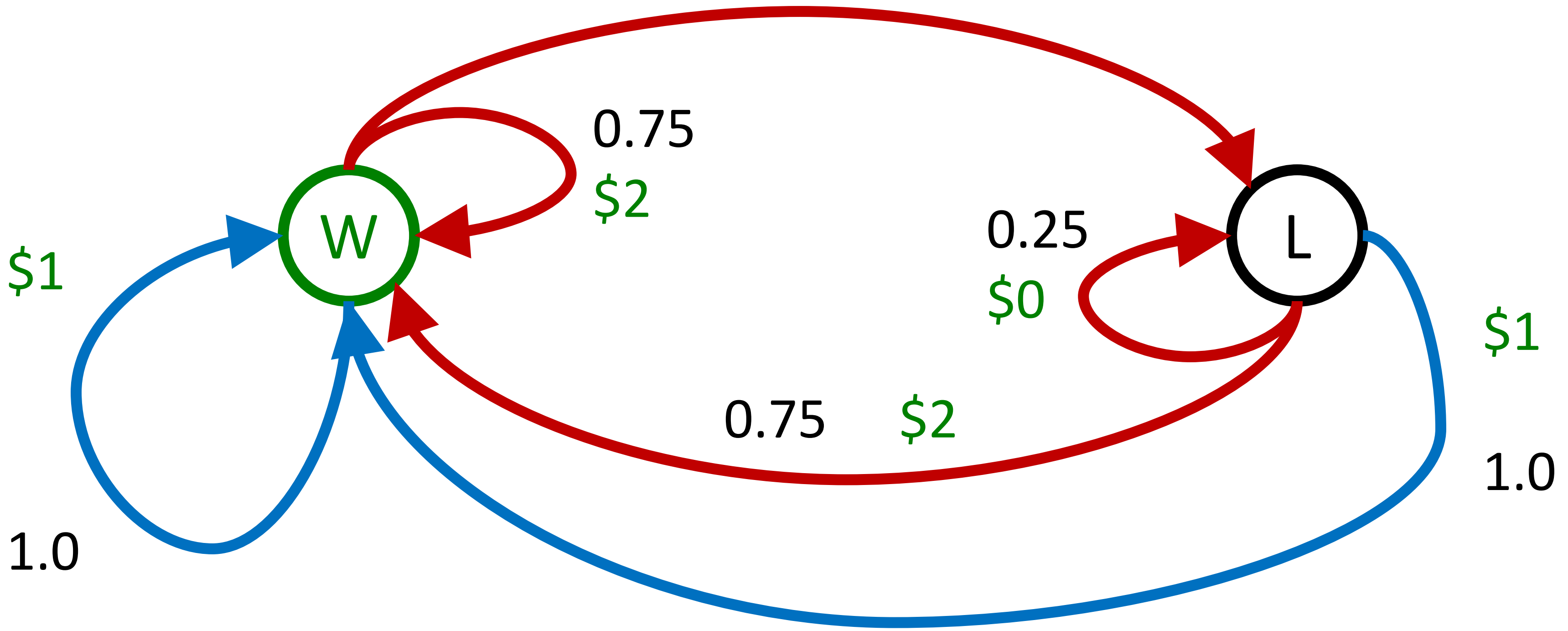
- Actions: *Blue*, *Red*
- States: Win, Lose

Probability to have a loss after a win when playing the red machine: 0.25

Reward: $0

Reward: $1

Probability to have a win after a win when playing the blue machine: 1.0



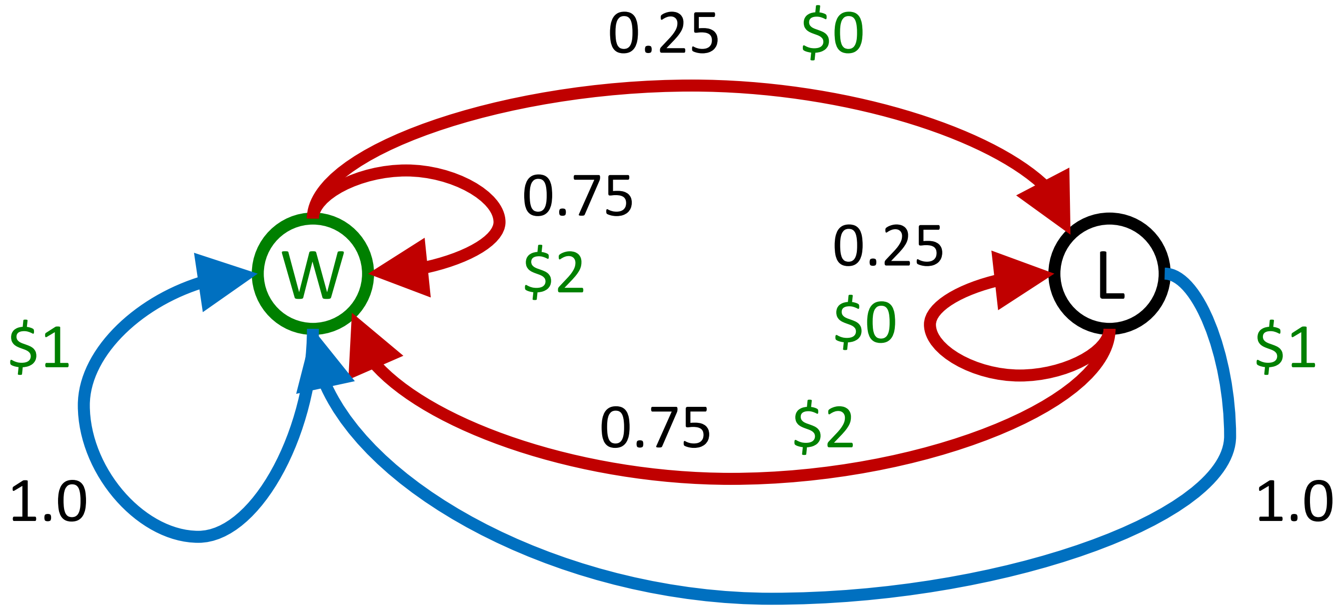
Slide additions by Betke

Offline Planning

- Solving Markov Decision Processes 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!

Value after playing 100 times:

Play Red	$150
Play Blue	$100



Let’s Play!

$2 $2 $0 $2 $2

$2 $2 $0 $0 $0

Let’s Play!

Your rewards after playing red 10 times:
$2 $2 $0 $2 $2 $2 $2 $0 $0 $0

= $12

Slide additions by Betke

Let’s Play!

Your rewards after playing red 10 times:
$2 $2 $0 $2 $2 $2 $2 $0 $0 $0

= $12

Contrary to the offline situation, you actually won money, rather than just thinking about it!

Slide additions by Betke

Let’s Play!

$1
WINNER
777

DOUBLE
$2/$0
OR NOTHING
BAR

Your rewards after playing red 10 times:
$2 $2 $0 $2 $2 $2 $2 $0 $0 $0

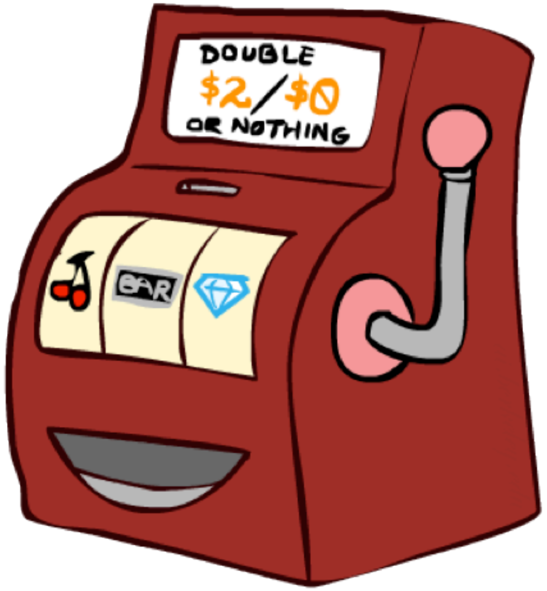
= $12

In this online situation, you are “**exploiting**” your observations that red is giving you “**rewards**.”

Slide additions by Betke

Let’s Play!

$1
WINNER
777



Your rewards after playing red 10 times:
$2 $2 $0 $2 $2 $2 $2 $0 $0 $0

= $12

But wait – shouldn’t you have received $15?

Slide additions by Betke

Let’s Play!

$1
WINNER
777

DOUBLE
$2 / $0
OR NOTHING
BAR

Your rewards after playing red 10 times:
$2 $2 $0 $2 $2 $2 $2 $0 $0 $0

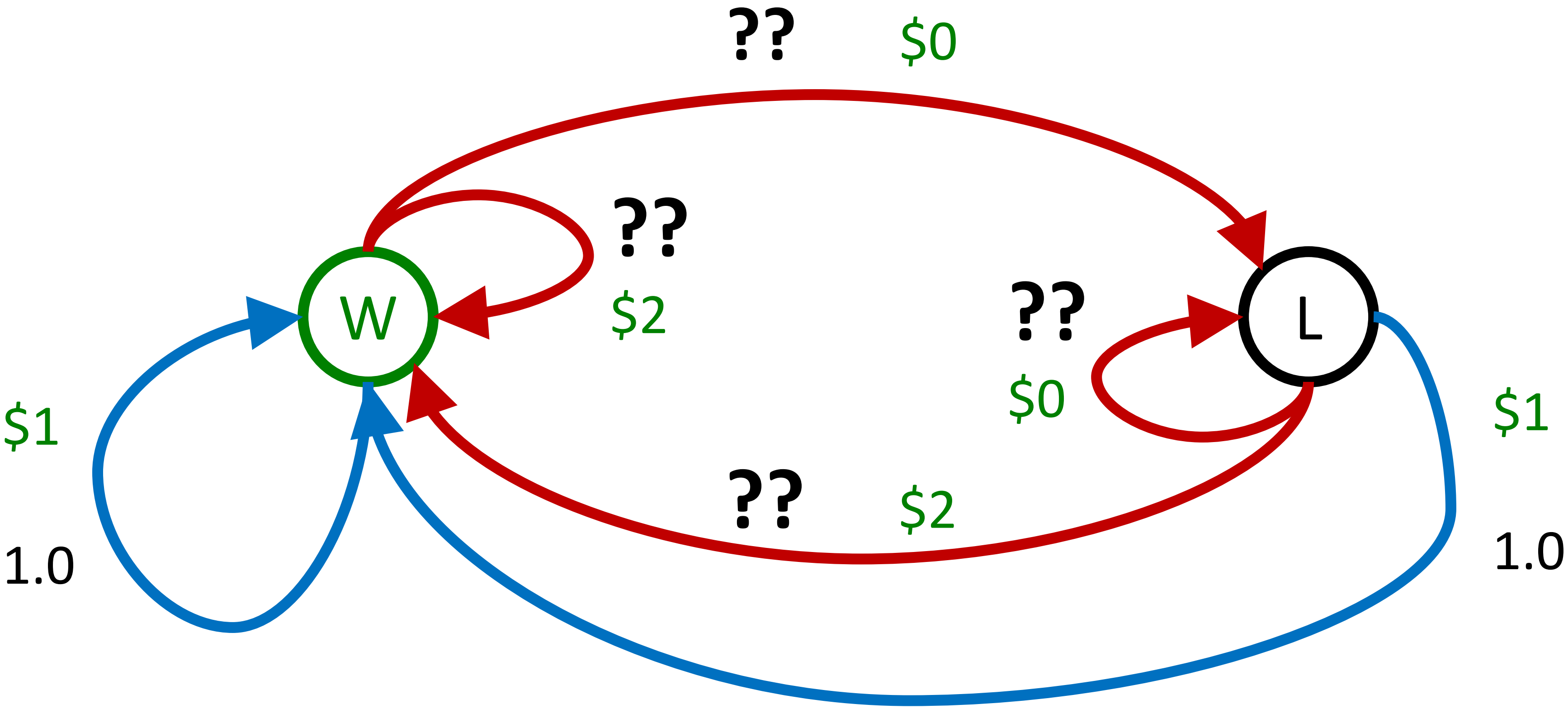
= $12

But wait – shouldn’t you have received $15?
Yes, but maybe it’s just a **sampling** issue. You won 6/10 times.
Just keep on playing some more, and you’ll get closer to winning ¾ of the time.

Slide additions by Betke

Online Planning

But no, maybe the rules changed, and red's win chance is different!!!



Slide changes by Betke

Let’s Play!

$0 $0 $0 $2 $0

$2 $0 $0 $0 $0

Let's Play!

Your rewards after playing red 10 times:
$0 $0 $0 $2 $0 $2 $0 $0 $0 $0
= $4

Red is a real bandit – you only won 2/10 times, i.e., you lost $8 !
Cut your losses or exploring more? “**Exploration**” may yield “**regret**.”

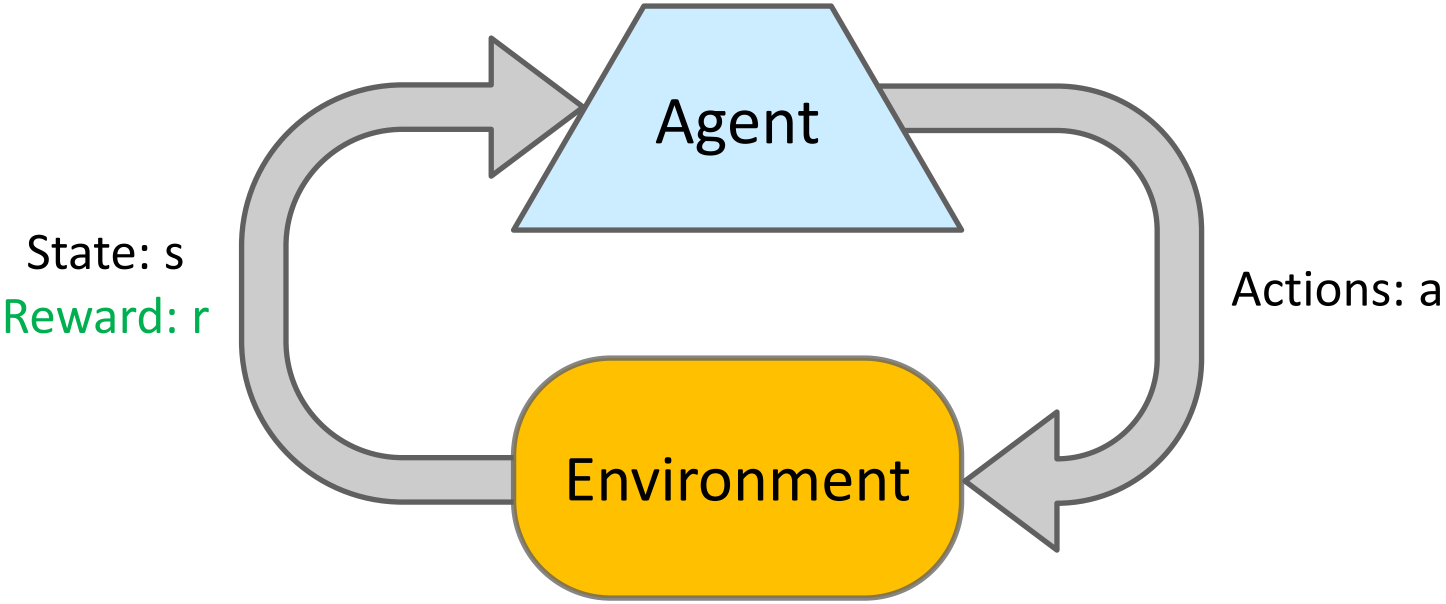
Slide additions by Betke

What Just Happened?

- That 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

- Important 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

Reinforcement Learning



- Basic idea:
  - Receive feedback in the form of rewards
  - Agent’s utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model T(s,a,s')
  - A reward function R(s,a,s')
- Still looking for a policy $\pi(s)$

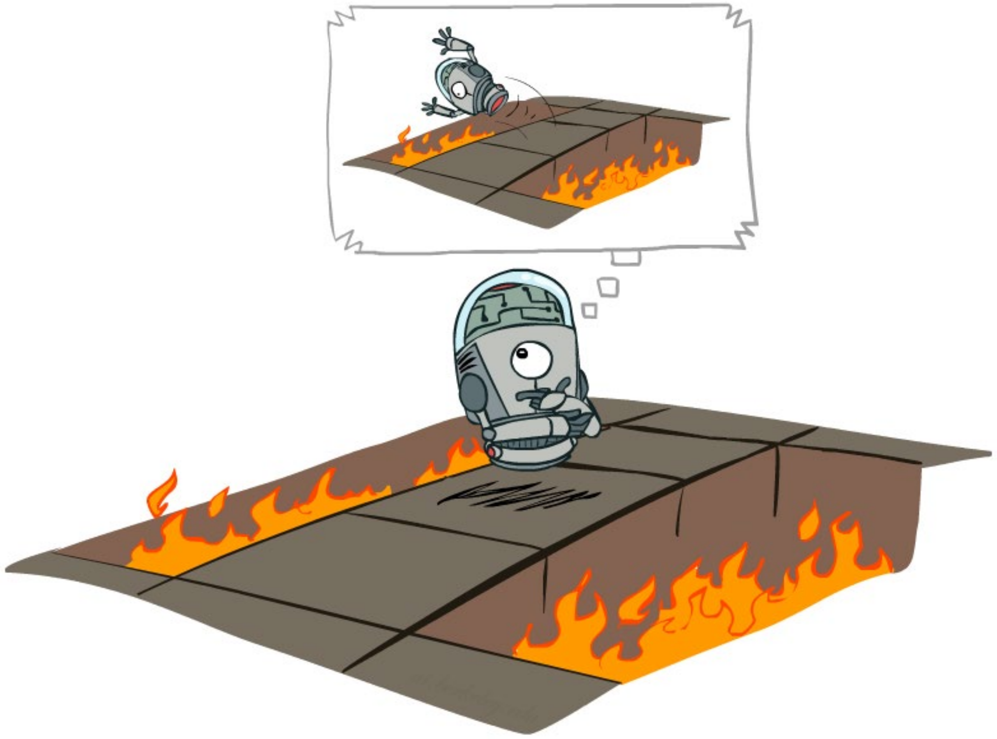
Cool

Warm

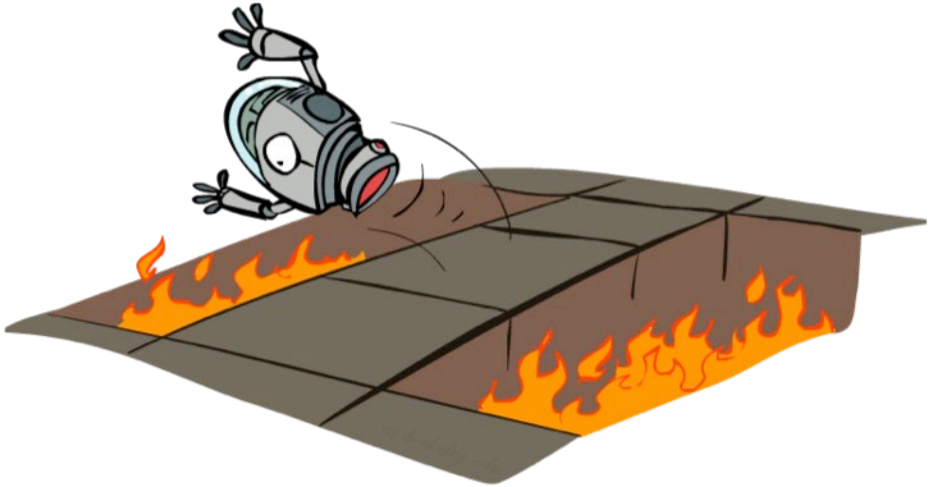
Overheated

- New twist: don't know T or R
  - We don't know which states are good or what the actions do
  - To learn, we must actually try out actions and states

Offline (MDPs) vs. Online (RL)

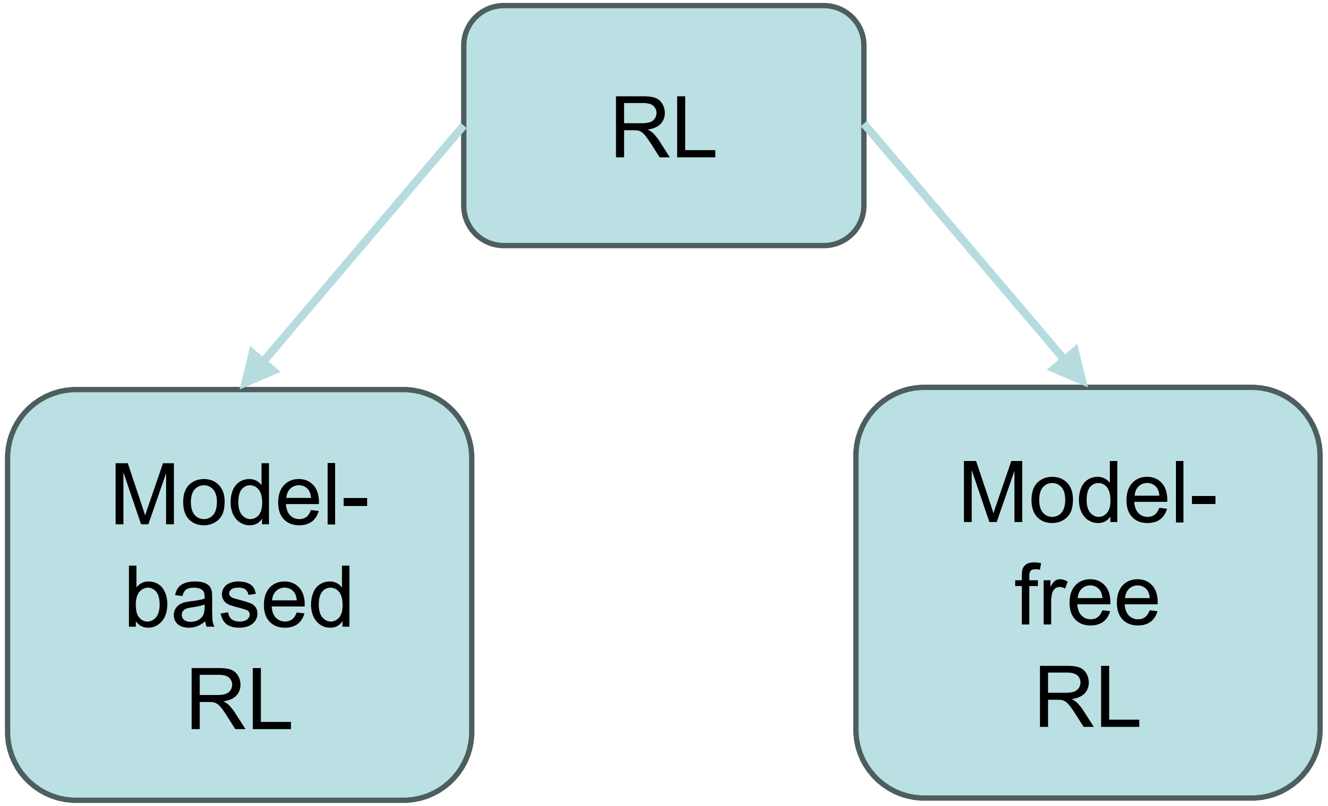


Offline Solution



Online Learning

Taxonomy of RL Methods



RL

Model-based RL

Model-free RL

Slide by Betke

Example: Expected Age

Goal: Compute expected age of CS 640 students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

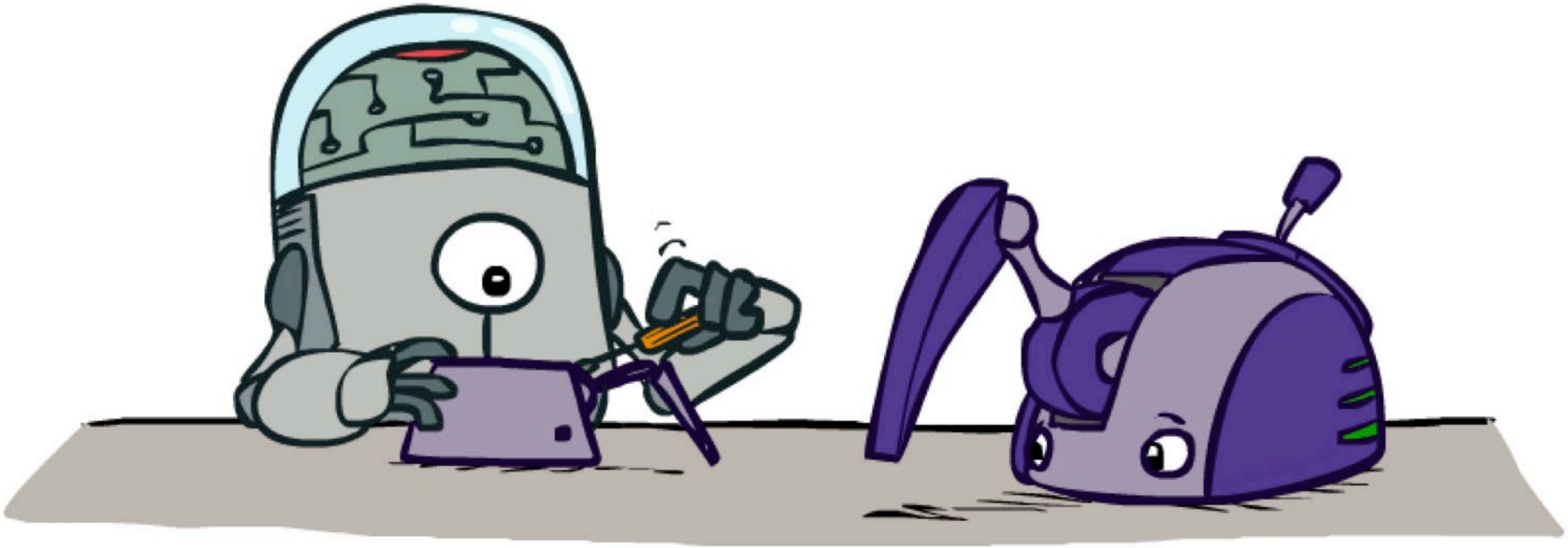
Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

Model-Based Learning



Try to figure out the model =
Learn existing transition and reward functions

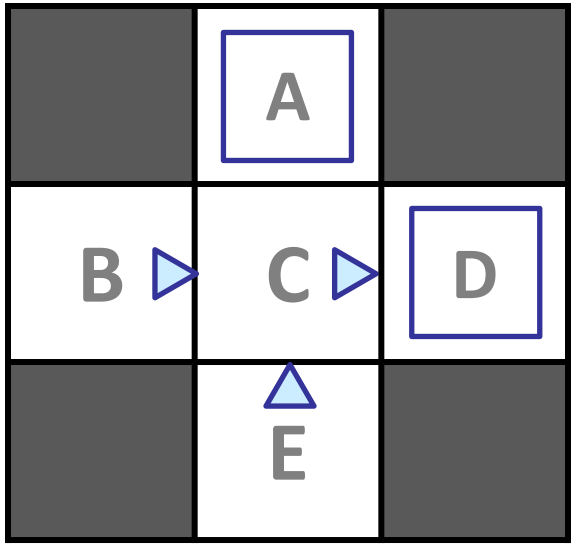
Slide additions by Betke

Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes s’ for each s, a
  - Normalize to give an estimate of $\widehat{T}(s, a, s')$
  - Discover each $\widehat{R}(s, a, s')$ when we experience (s, a, s’)
- Step 2: Solve the learned MDP
  - For example, use value (utility) iteration, as before
    (Bellman update equation)

Example: Model-Based Learning

Input Policy π



Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

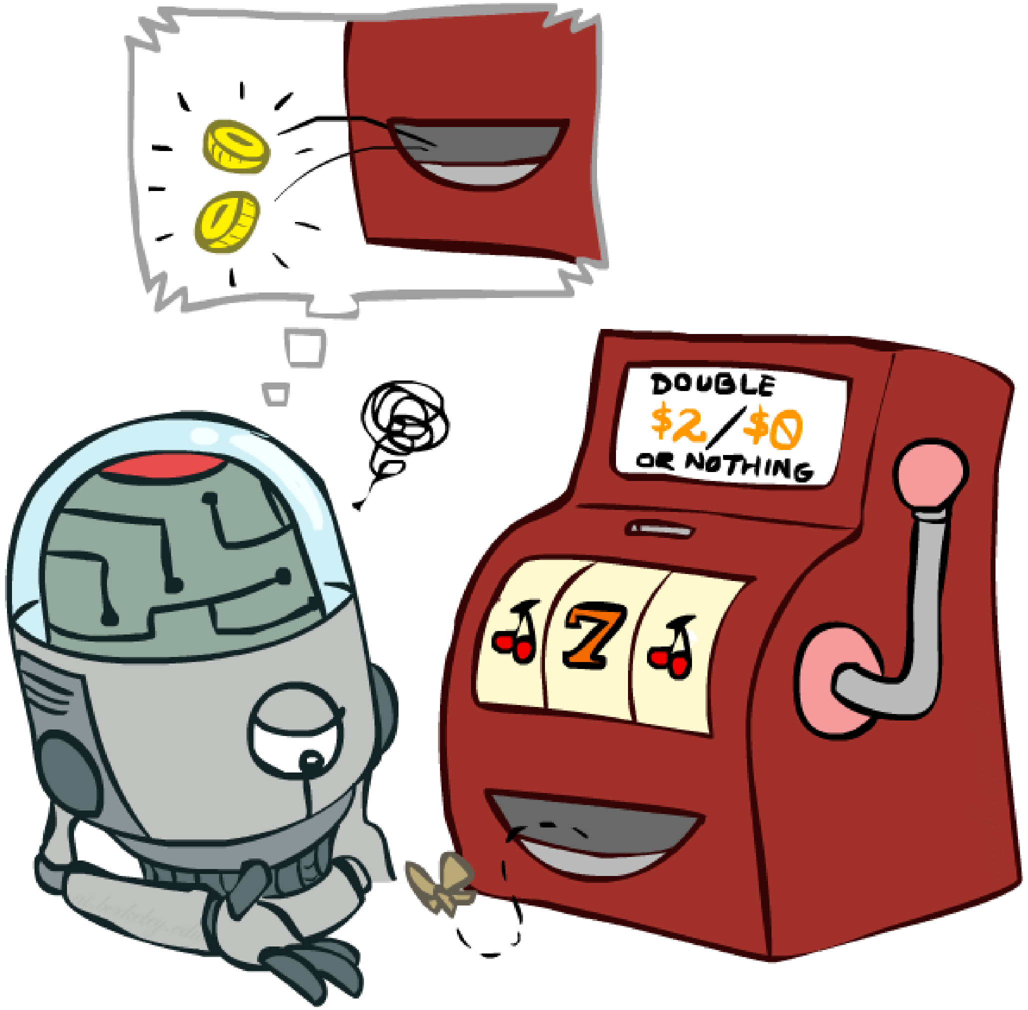
$$\hat{T}(s, a, s')$$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

$$\hat{R}(s, a, s')$$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

Model-Free Learning

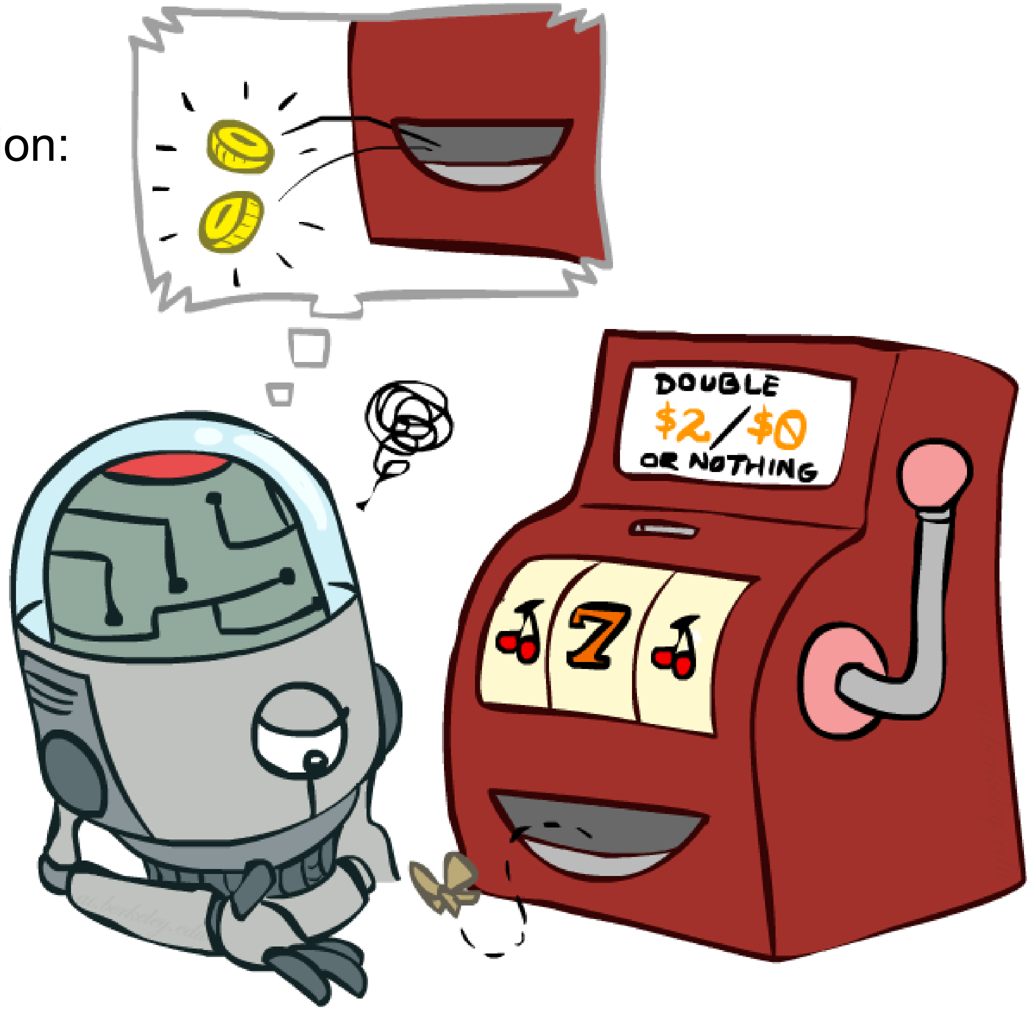


Model-Free Learning

Compare

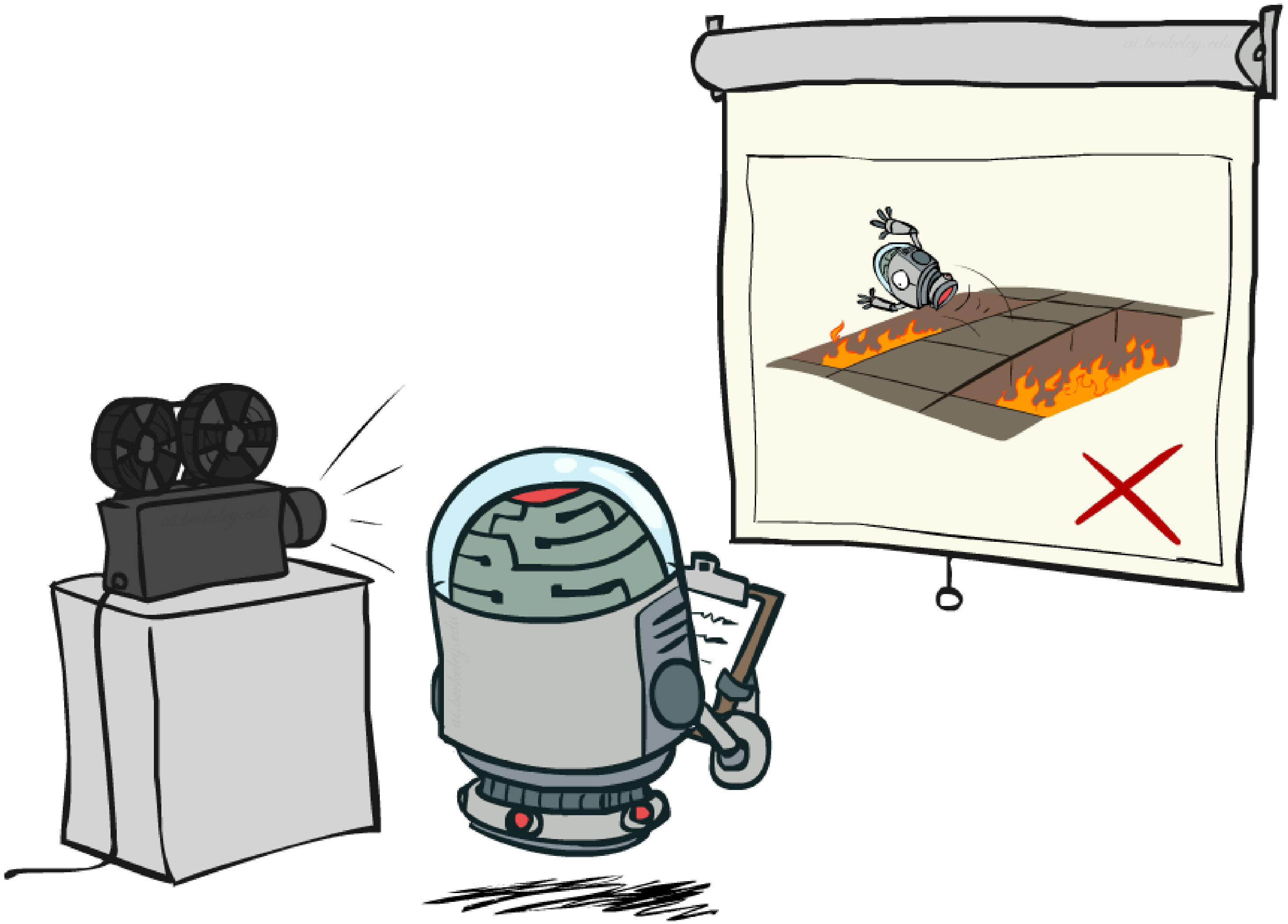
Expected result of action:
2 gold coins

with true result:
a worthless bow



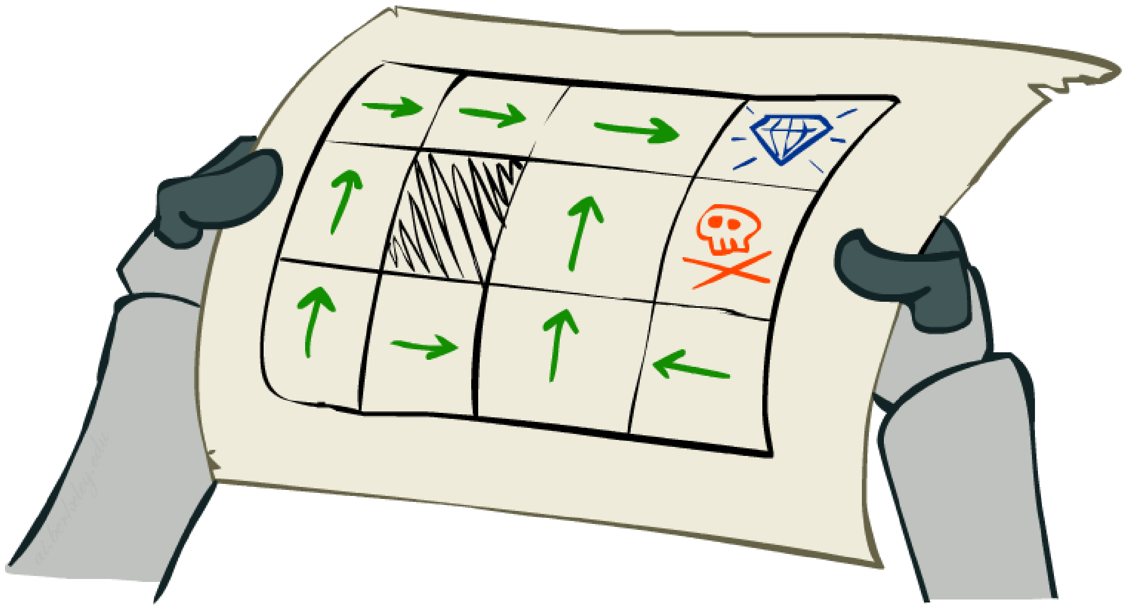
Slide additions by Betke

Passive Reinforcement Learning



Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values



- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

Passive Reinforcement Learning: Direct Utility Evaluation

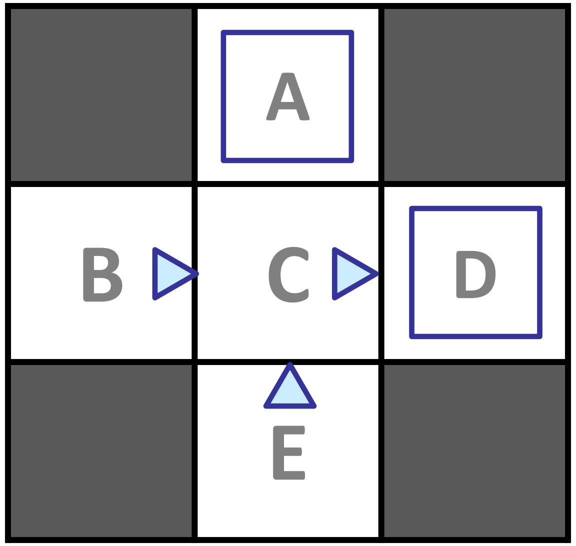
- Goal: Compute values for each state under π

- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples

- This is called direct utility evaluation

Example: Direct Utility Evaluation

Input Policy π



Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

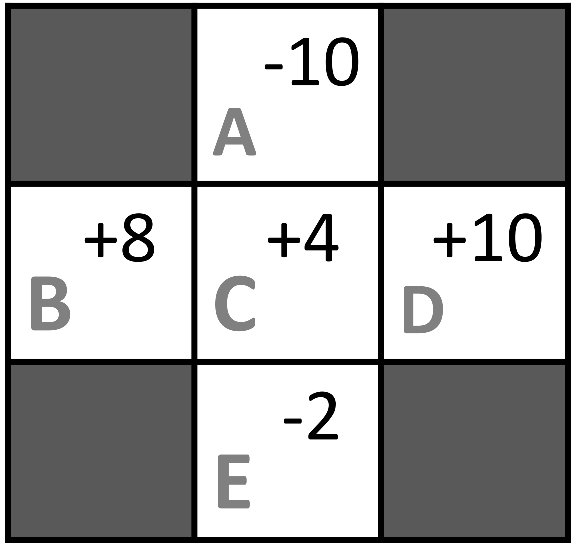
Episode 3

E, north, C, -1
C, east,   D, -1
D, exit,   x, +10

Episode 4

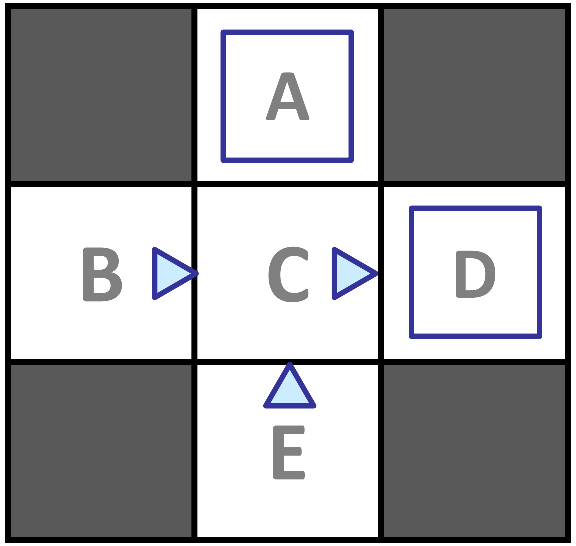
E, north, C, -1
C, east,   A, -1
A, exit,   x, -10

Output Values



Example: Direct Utility Evaluation

Input Policy π



Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

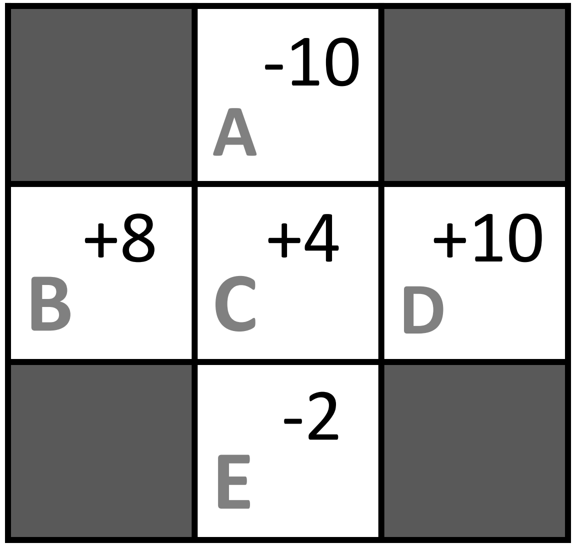
Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Utility Values:



Why 8 for B? Episodes 1 & 2: -1-1+10 + 8 = 16.
Average: 16/2 = 8

Why 4 for C? 9+ 9 + 9 – 11 = 16.
16/4 = 4.

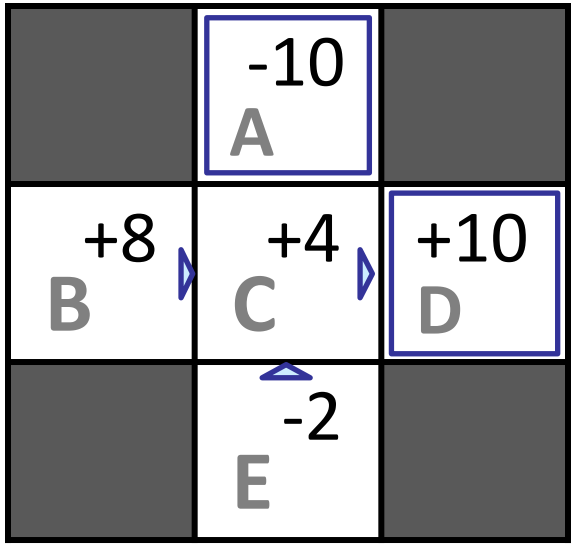
Slide additions by Betke

Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any knowledge of T, R
  - It eventually computes the correct average values, using just sample transitions

- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values



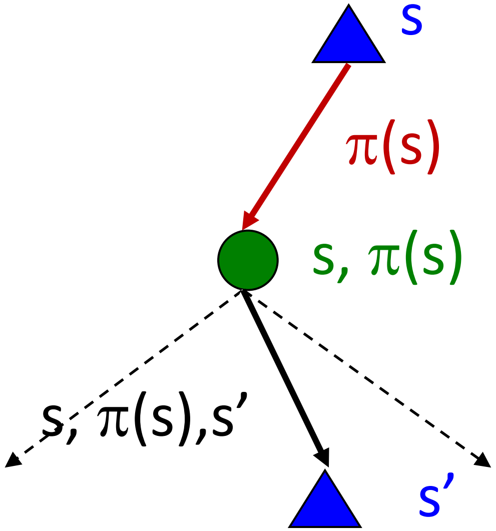
If B and E both go to C under this policy, how can their values be different?

Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate V for a fixed policy:
  - Each round, replace V with a one-step-look-ahead layer over V

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

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



  - This approach fully exploited the connections between the states
  - Unfortunately, we need T and R to do it!

- Key question: how can we do this update to V without knowing T and R?
  - In other words, how to we take a weighted average without knowing the weights?

Betke notes: Value V = Utility U

Sample-Based Policy Evaluation?

- We want to improve our estimate of V by computing these averages:

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

- Idea: Take samples of outcomes s’ (by doing the action!) and average

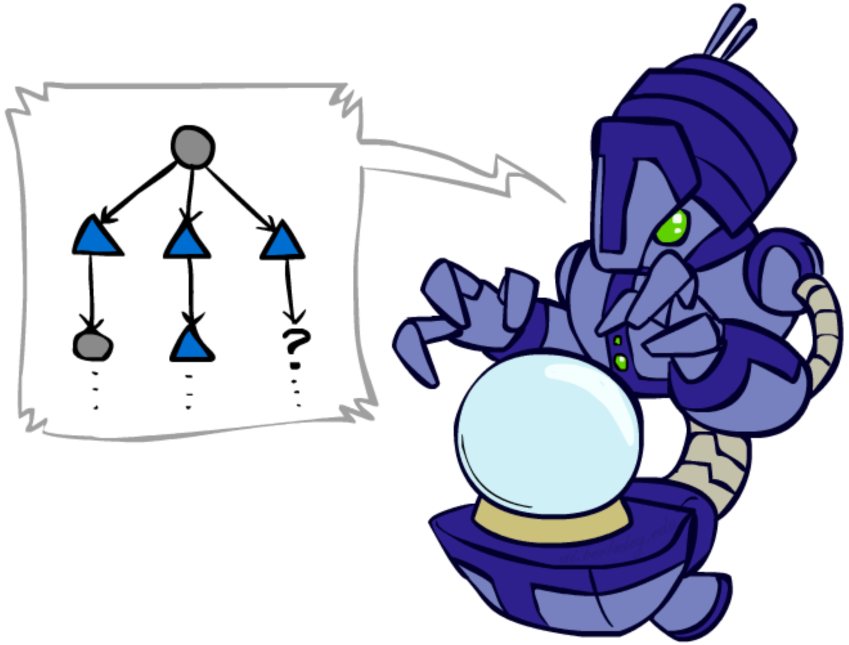
$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$

$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$

$$\cdots$$

$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n}\sum_i sample_i$$



Betke notes: Value V = Utility U

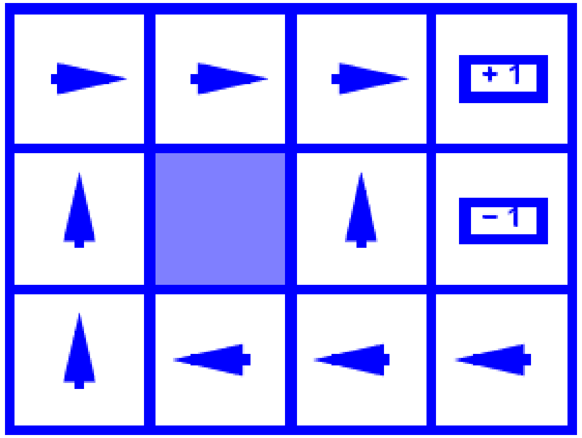
Example: Sample-Based Policy Evaluation

Set of trials in the environment using policy π:

Trail 1: [1,1][1,2],[1,3][1,2][1,3][2,3][3,3][4,3]

Trail 2: [1,1][1,2][1,3][2,3][3,3][3,2][3,3][4,3]

…



R(s) = -0.04

Example: Sample-Based Policy Evaluation

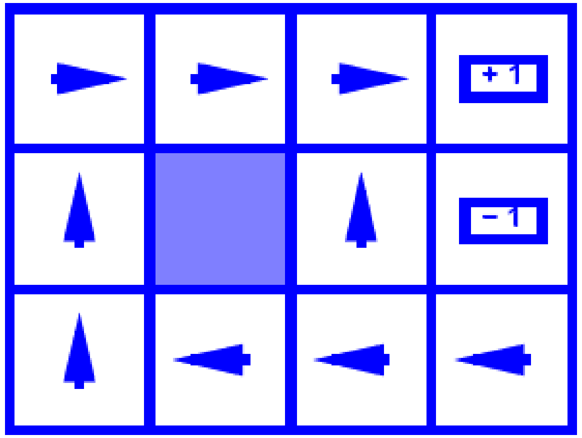
Set of trials in the environment using policy π:

Trail 1: [1,1][1,2] [1,3] [1,2] [1,3] [2,3][3,3][4,3]

Trail 2: [1,1][1,2] [1,3] [2,3][3,3][3,2][3,3][4,3]

…

At state [1,3]: 2xright, 1xdown: T([1,3],Right,[2,3])=2/3



R(s) = -0.04

Example: Sample-Based Policy Evaluation

Set of trials in the environment using policy π:

Trail 1: [1,1][1,2],[1,3][1,2][1,3][2,3][3,3][4,3]

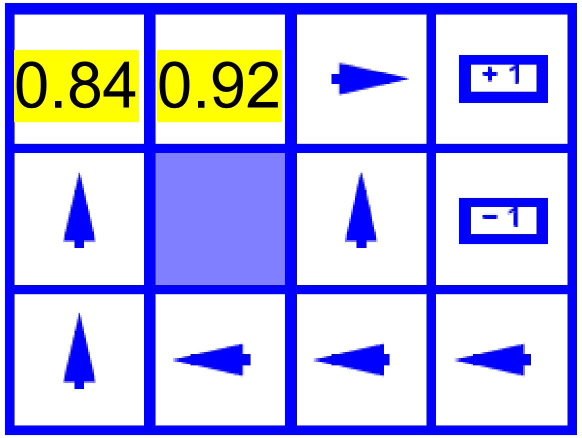
Trail 2: [1,1][1,2][1,3][2,3][3,3][3,2][3,3][4,3]

…

Say, after trail 1: U(1,3)=0.84, U(2,3)=0.92

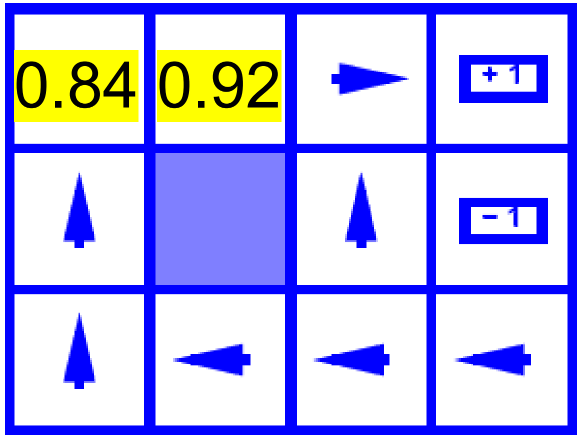
Update: U(1,3)=-0.04+U(2,3) = -0.04+0.92= 0.88

Current estimate 0.84 is a little low.



R(s) = -0.04

Example: Sample-Based Policy Evaluation



R(s) = -0.04

Set of trials in the environment using policy π:

Trail 1: [1,1][1,2],[1,3][1,2][1,3][2,3][3,3][4,3]

Trail 2: [1,1][1,2][1,3][2,3][3,3][3,2][3,3][4,3]

…

Say, after trail 1: U(1,3)=0.84, U(2,3)=0.92

Update: U(1,3)=-0.04+U(2,3) = -0.04+0.92= 0.88

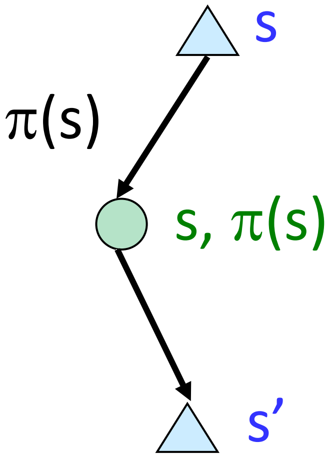
Current estimate 0.84 is a little low. Use learning rate α.

Update: $U(s) \leftarrow U(s) + \alpha\ [\ R(s) + \gamma\ U(s') – U(s)\ ]$

This update rule is the Temporal-difference (TD) equation.

Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s’, r)
  - Likely outcomes s’ will contribute updates more often



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha) sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Betke notes: Value V = Utility U

Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1 - \alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:

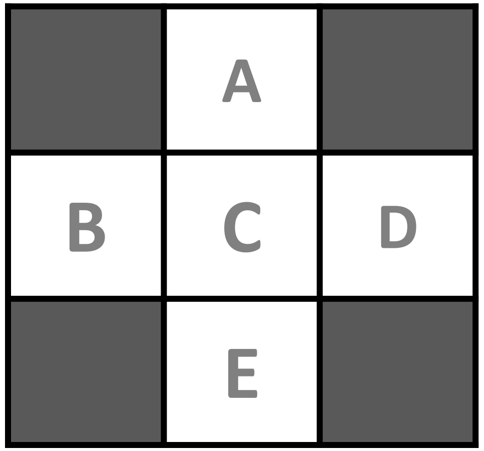
$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)

- Decreasing learning rate (alpha) can give converging averages

Example: Temporal Difference Learning

States

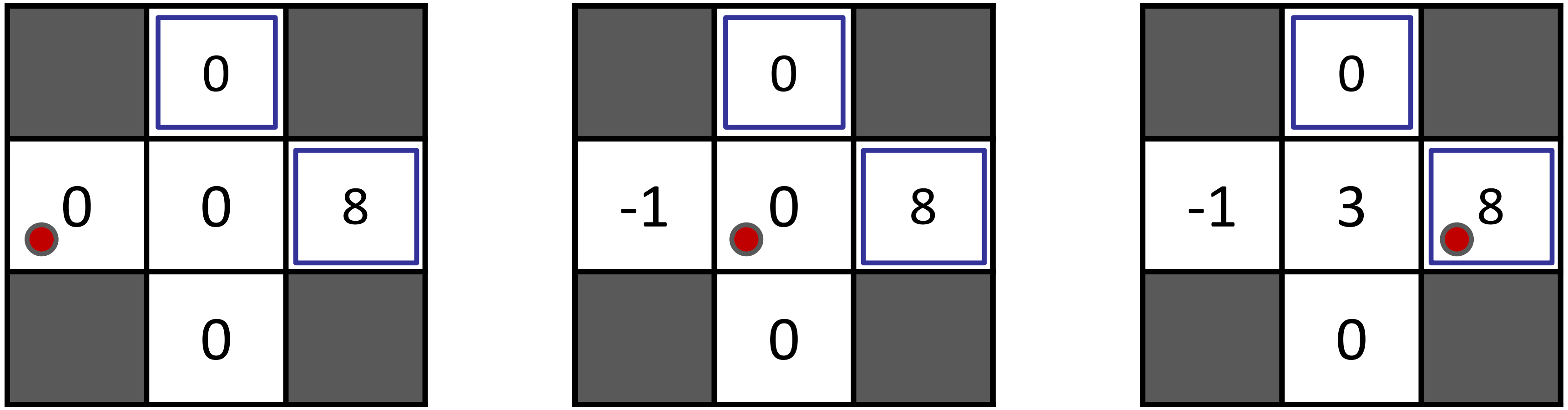


Assume: γ = 1, α = 1/2

Observed Transitions

B, east, C, -2

C, east, D, -2



$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

B: 0.5 x 0 + 0.5 [-2 + 0] = -1

C: 0.5 x 0 + 0.5 [-2 + 8] = 3

Betke additions

Problems with TD Value (Utility) Learning

- TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we're sunk:

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

  Learning values (=utilities) for each state is not enough

- Alternative: The "Q-learning" method, which learns an

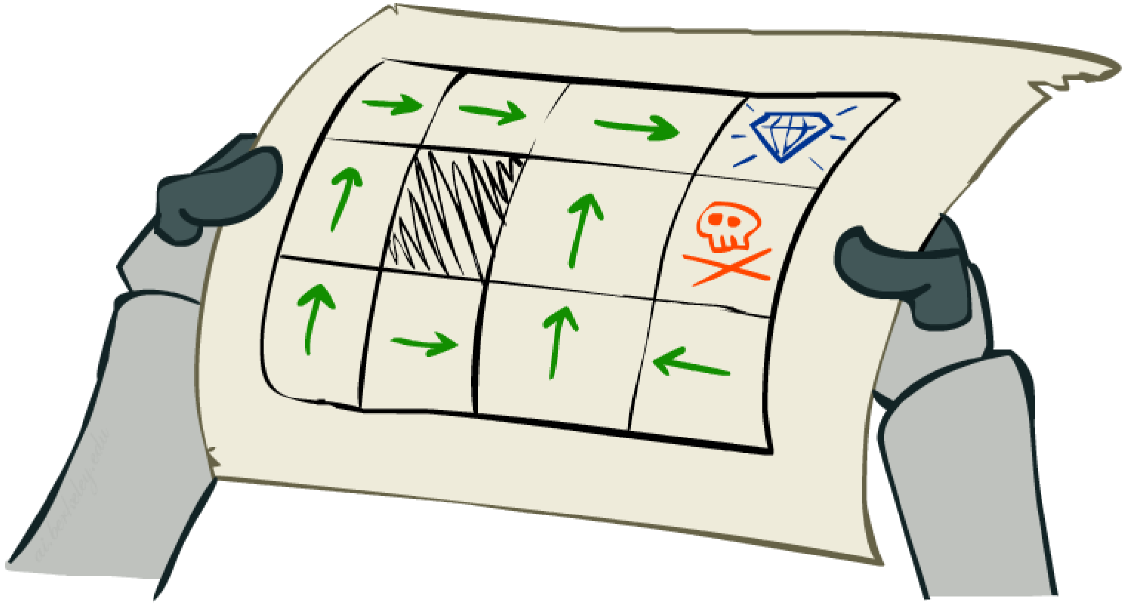
  action-utility representation (instead of just a utility representation)

  $Q(a,s)$ = value of doing action a in state s. $U(s) = \max_a Q(s,a)$

Betke

Reminder: Passive Reinforcement Learning

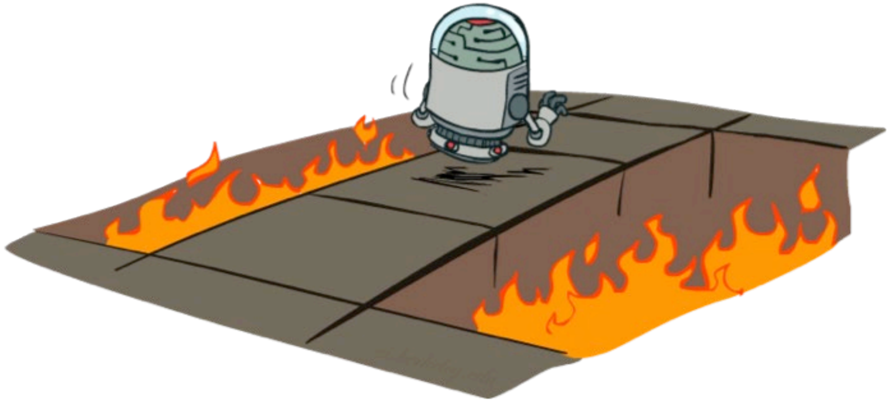
- Simplified task: policy evaluation
  - Input: a fixed policy $\pi(s)$
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - Goal: learn the state values



- In this case:
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.

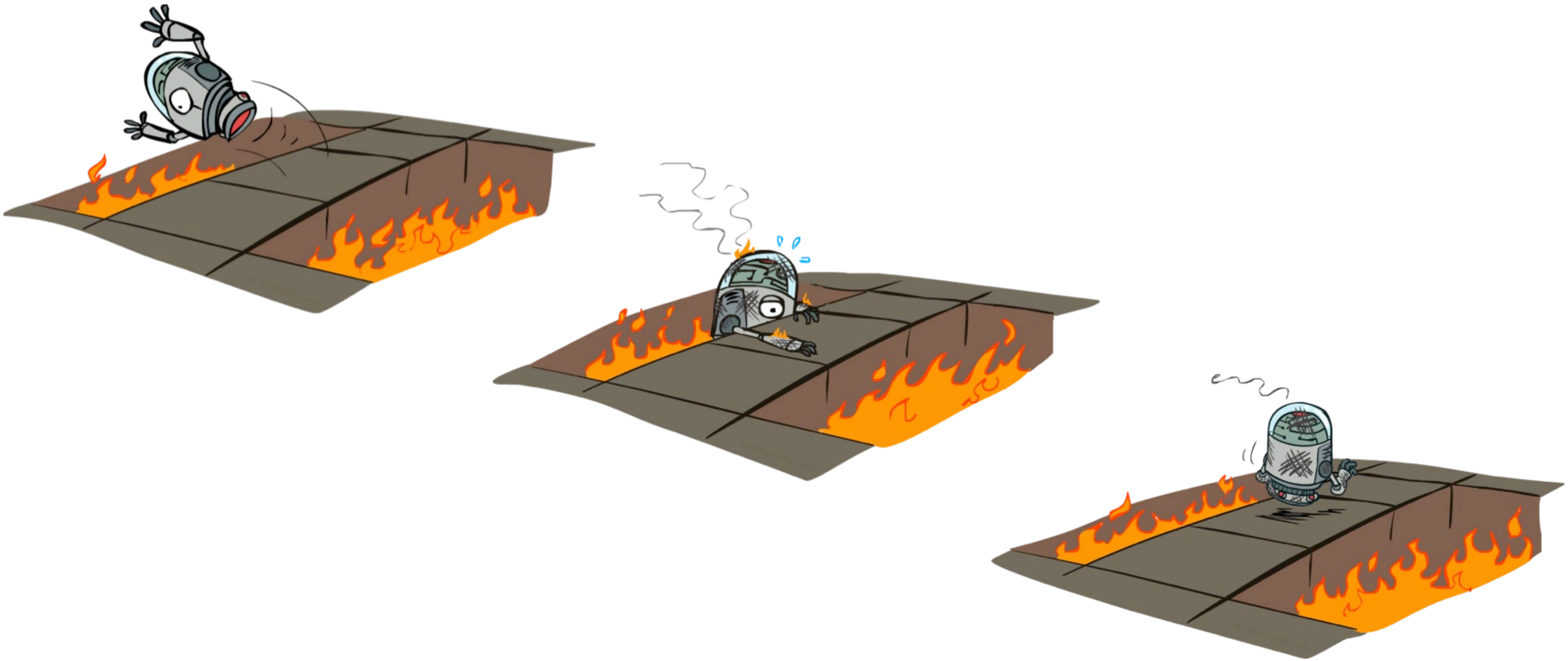
Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don’t know the transitions T(s,a,s’)
  - You don’t know the rewards R(s,a,s’)
  - You choose the actions now
  - Goal: learn the optimal policy / values

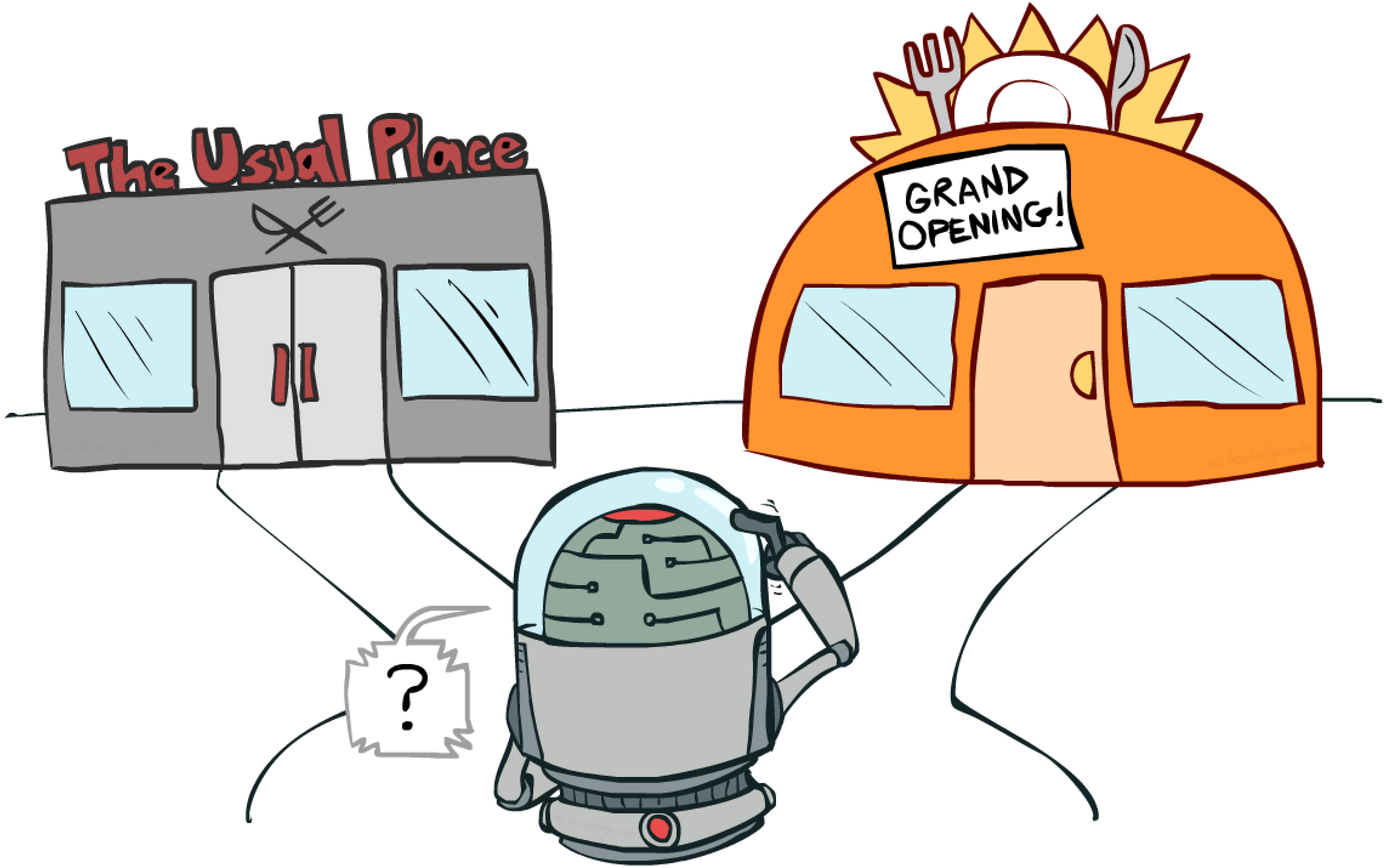


- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

Active Reinforcement Learning



Exploration vs. Exploitation



Bandit Problems are Active RL Problems

DOUBLE
$2/$0
OR NOTHING
BAR

Your rewards after playing red 10 times:
$0 $0 $0 $2 $0 $2 $0 $0 $0 $0
= $4

Red is a real bandit – you only won 2/10 times, i.e., you lost $8 !
Cut your losses or exploring more? **“Exploration”** may yield “**regret**.”

Slide additions by Betke

Active RL: Manual Exploration – Bridge Grid

How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy

Video of Demo Q-learning – Epsilon-Greedy – Crawler

How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Q learning

- Q functions store utility information
- Agent that learns Q function does not need a model of the form P(s’|s,a) for learning or action selection.
- Thus, Q-learning is model-free method.
- When Q values are correct, this equation must hold at equilibrium:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

- Update equation for TD Q learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth k+1 values for all states:

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

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth k+1 Q-values for all states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s’,r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Incorporate the new estimate into a running average:

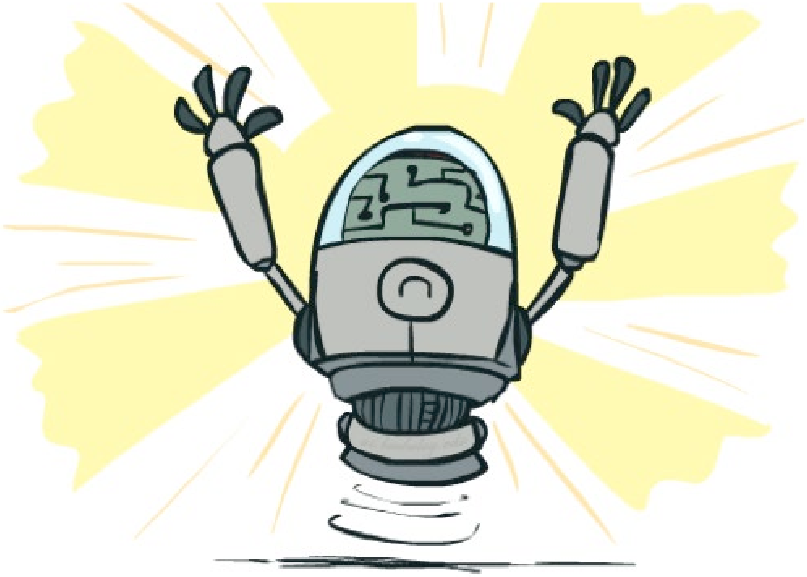
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

- This is called off-policy learning

- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)



Exploration Functions *f(u,n)*

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

- Exploration function

  Takes a utility estimate *u* and a visit count *n*, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

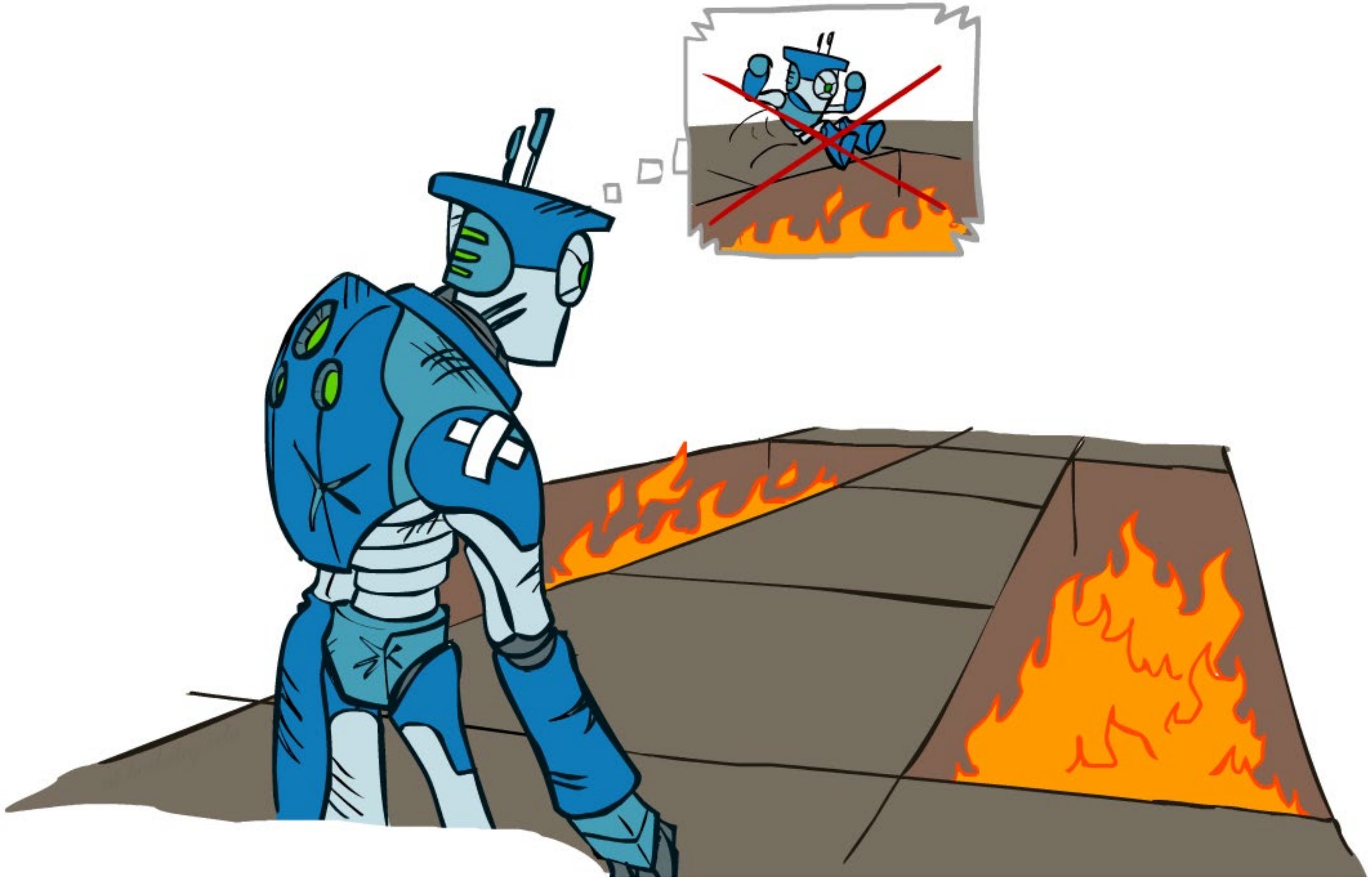
  Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

  Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

  - Note: this propagates the “bonus” back to states that lead to unknown states as well!

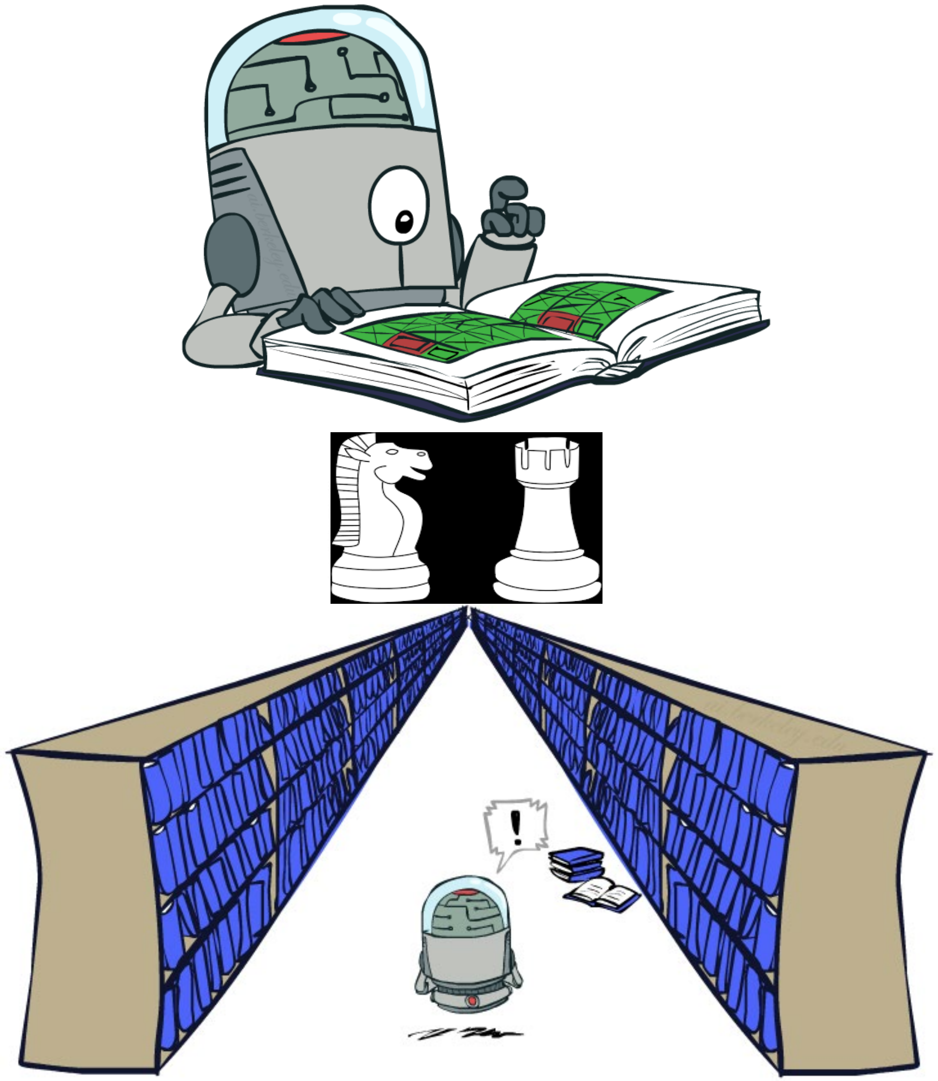
Regret

- Even if you learn the optimal policy, you still make mistakes along the way
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret



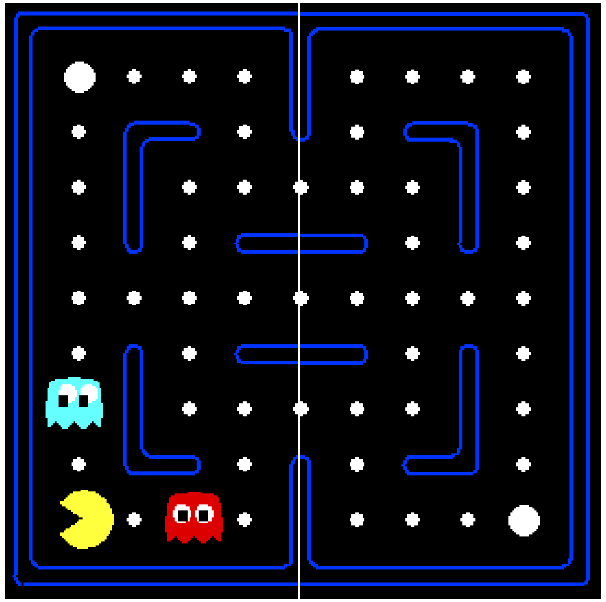
Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the Q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

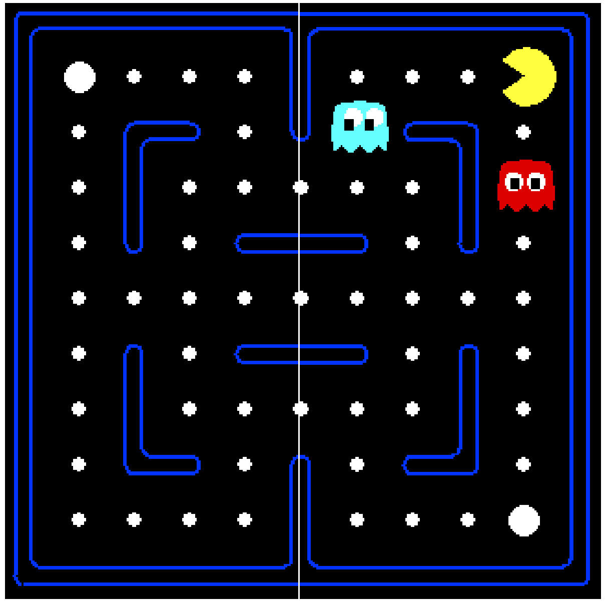


Example: Pacman

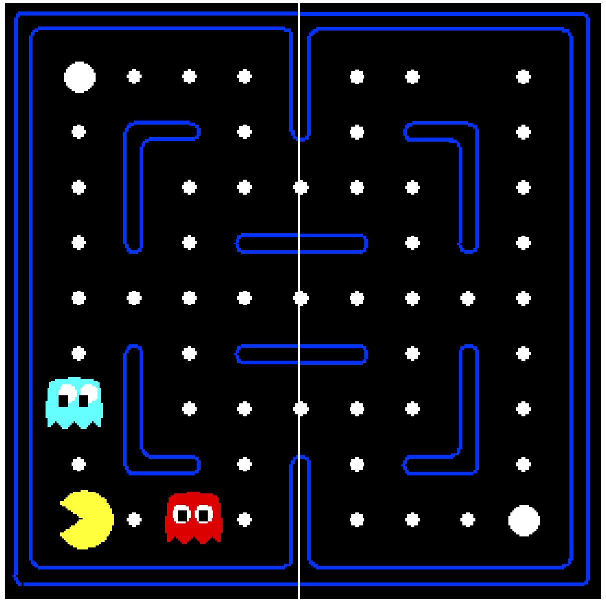
Let's say we discover through experience that this state is bad:



In naïve Q-learning, we know nothing about this state:



Or even this one!

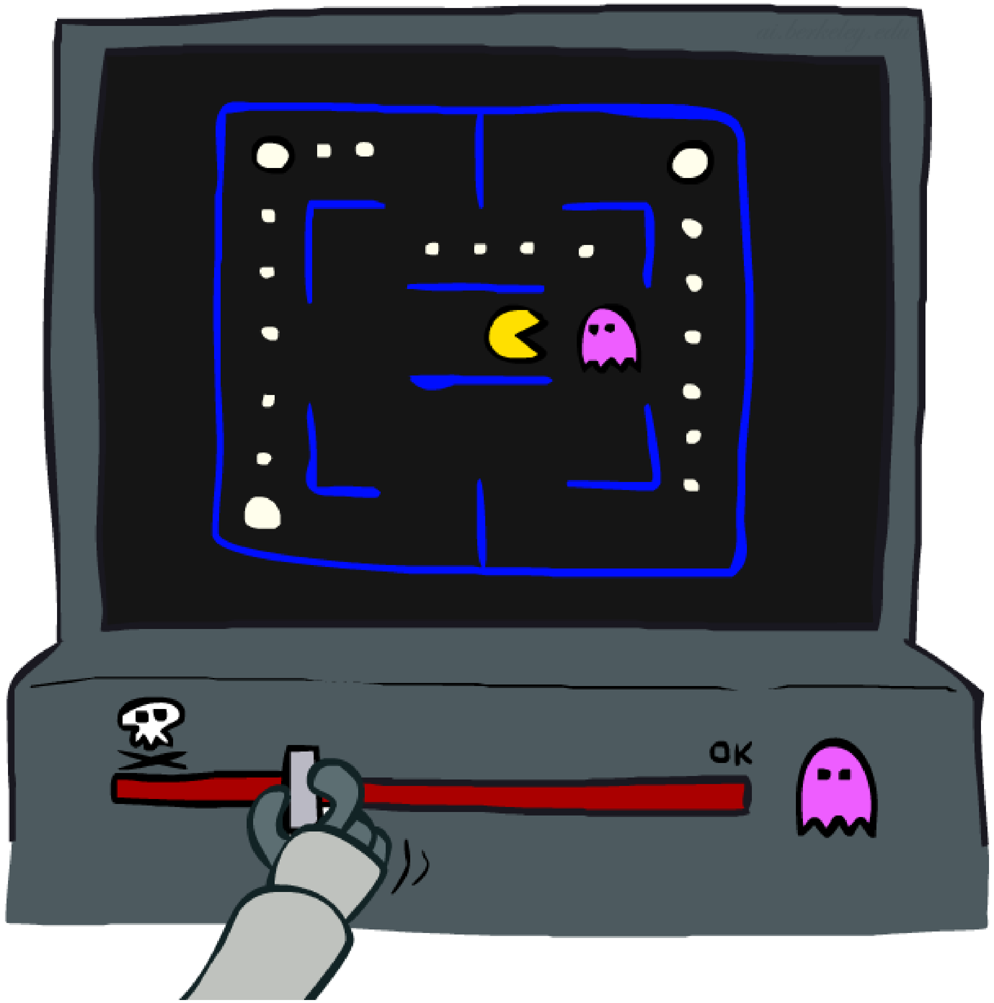


Video of Demo Q-Learning Pacman – Tiny – Watch All

Video of Demo Q-Learning Pacman – Tiny – Silent Train

Video of Demo Q-Learning Pacman – Tricky – Watch All

Approximate Q-Learning



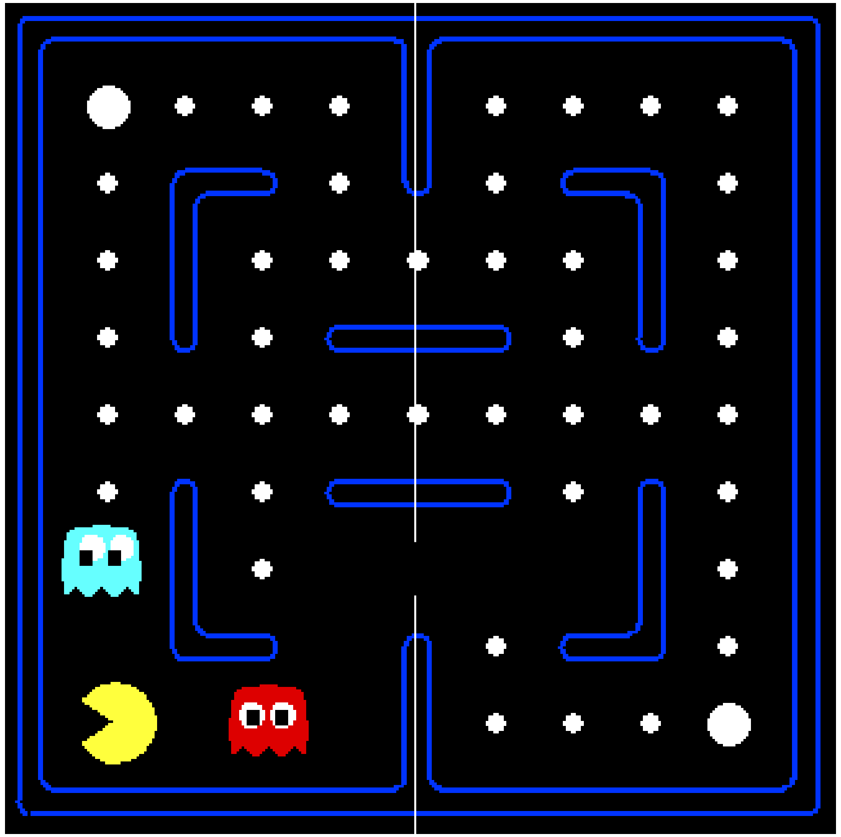
PacMan:

When you learn that a ghost is bad, you should transfer that knowledge to other states that have ghosts

So you don’t actually have to explore all states that have ghosts (Q-learning) and can get away with not exploring all states (approximate Q-learning).

Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Using a feature representation, we can write a Q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

$$\boxed{Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)}$$

- Q-learning with linear Q-functions:

$$\text{transition } = (s,a,r,s')$$

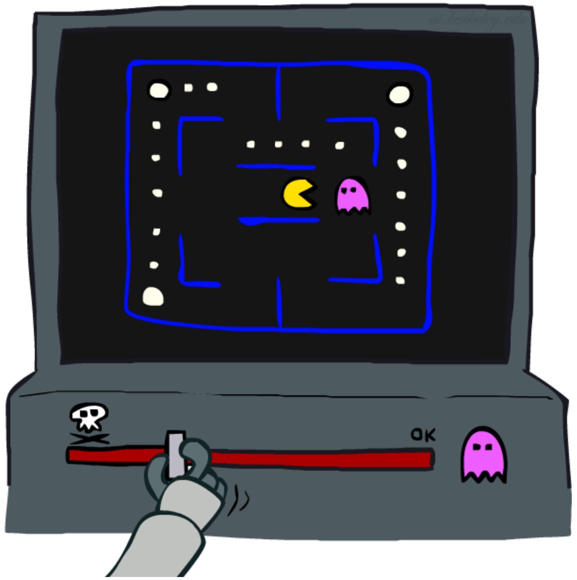
$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$

Exact Q's

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$

Approximate Q's



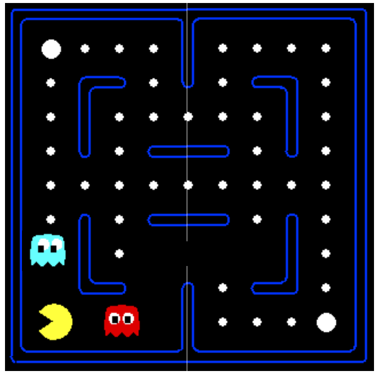
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

s

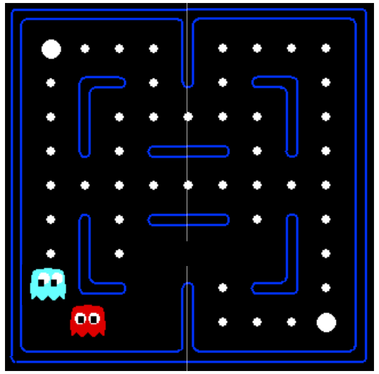


$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$
$r = -500$

s'



$Q(s, \text{NORTH}) = +1$

$r + \gamma \max_{a'} Q(s', a') = -500 + 0$

$Q(s', \cdot) = 0$

difference $= -501$

$w_{DOT} \leftarrow 4.0 + \alpha [-501]\, 0.5$

$w_{GST} \leftarrow -1.0 + \alpha [-501]\, 1.0$

$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Video of Demo Approximate Q-Learning -- Pacman

Linear Value Functions

- Using a feature representation, we can write a Q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a)+w_2 f_2(s,a)+\ldots+w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

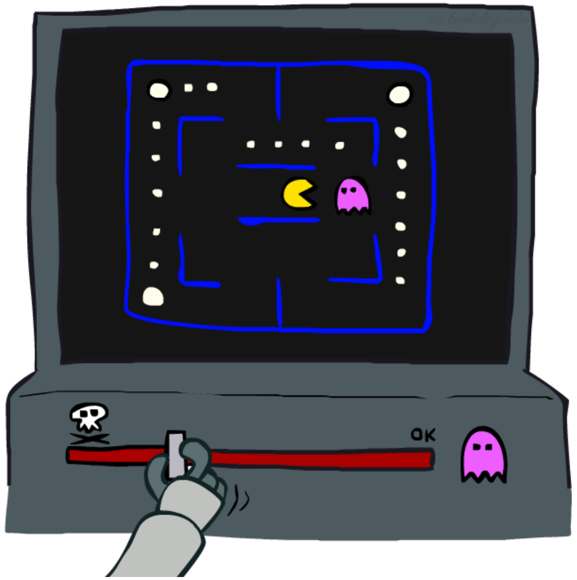
- Q-learning with linear Q-functions:

$$\text{transition } = (s,a,r,s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

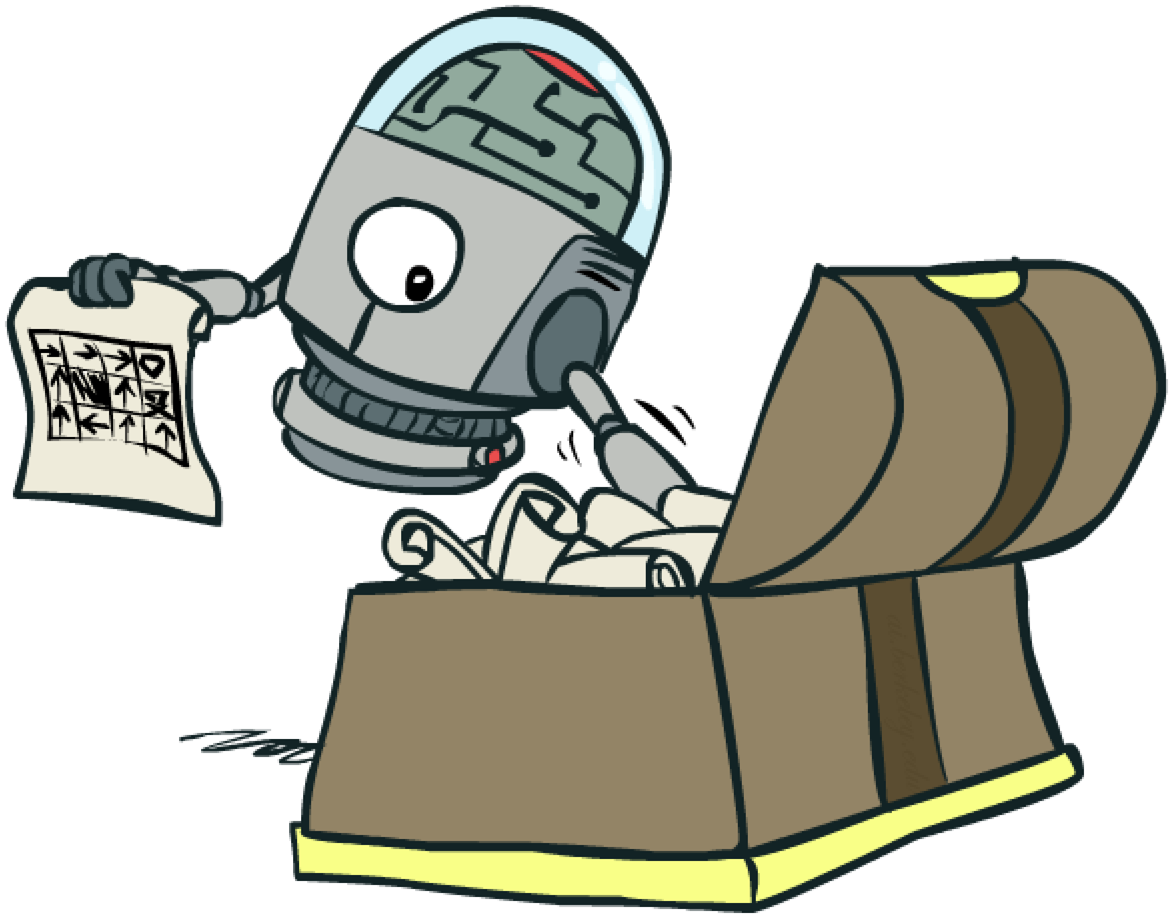
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$ Exact Q's

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$ Approximate Q's



- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V or Q best

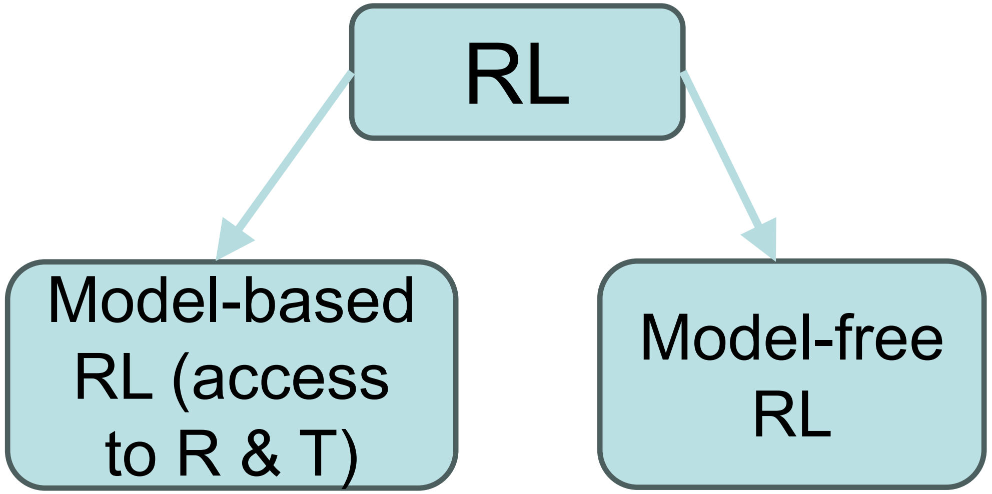
- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before
- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

RL and Deep Learning



Utility Iteration

Utility Iteration where state of environment represented by NN embedding

Q-Learning

Deep Q-Learning

More on this in BU’s Deep Learning class

Learning Outcomes

You should be able to explain:

- MDPs vs RL
- Planning (Offline) vs Learning (Online)
- RL
    - Model-based (sample actions to learn T and R)
    - Model-free
        - Passive: Policy evaluation (direct evaluation vs. TD learning)
        - Active: Q-learning (converges to optimal policy, but agent needs to explore enough)
    - Exploration vs. Exploitation
        - Explore: ε-greedy, exploration function
    - Approximate Q-learning with linear value functions
    - Policy Search