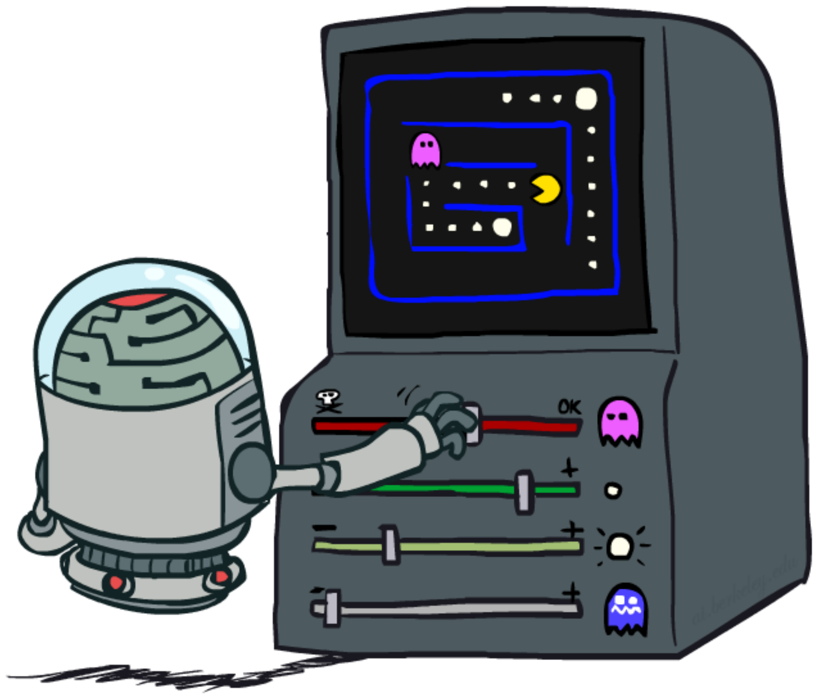
CSE 573: Artificial Intelligence

Reinforcement Learning



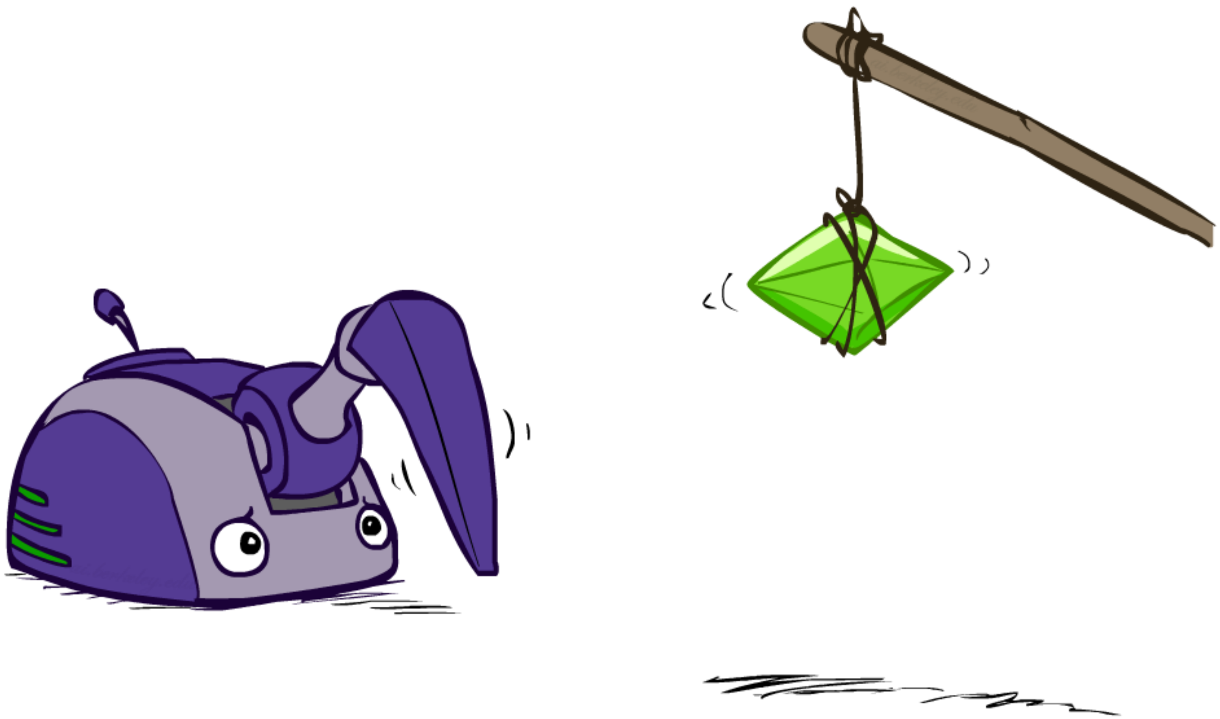
Dan Weld/ University of Washington

[Many slides taken from Dan Klein and Pieter Abbeel / CS188 Intro to AI at UC Berkeley – materials available at http://ai.berkeley.edu.]

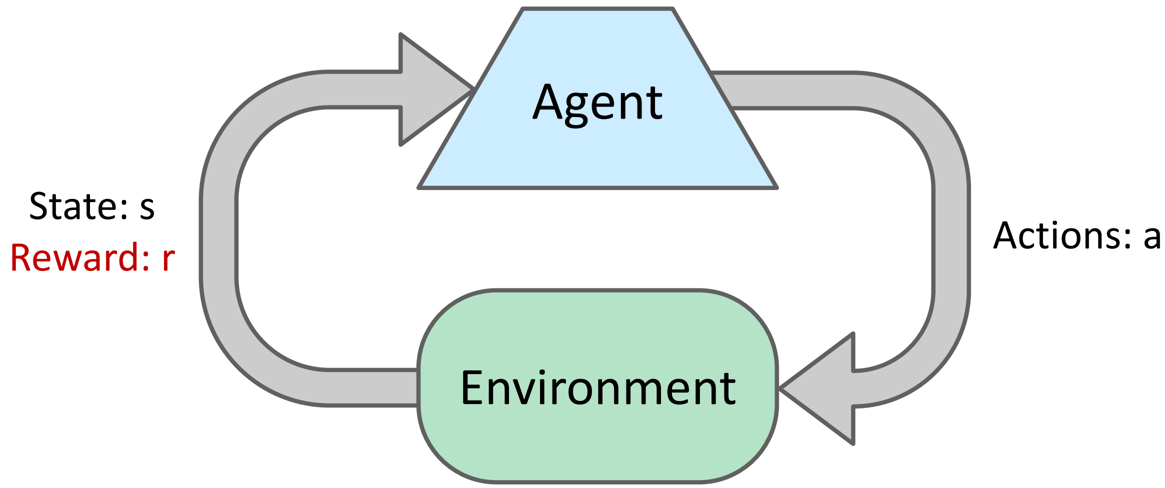
Logistics

- PS 3 due today
- PS 4 due in one week (Thurs 2/16)
- Research paper comments due on Tues
  - Paper itself will be on Web calendar after class

Reinforcement Learning



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!

Example: Animal Learning

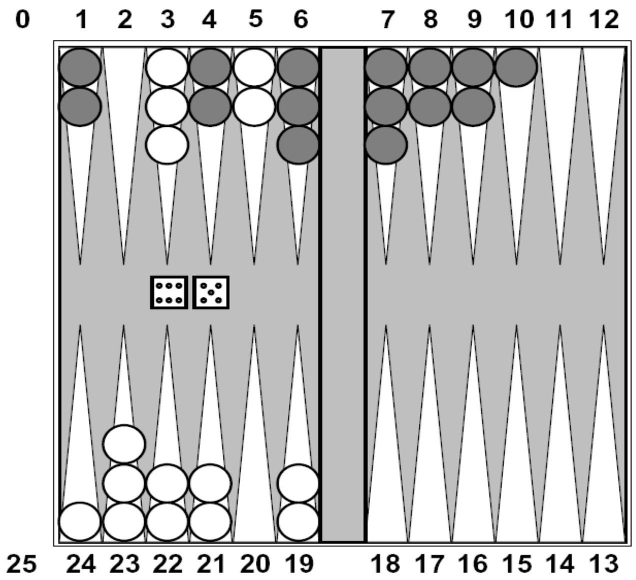
- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated
- Example: foraging

  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to V(s) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world

- You could imagine training Pacman this way…
- … but it's tricky! (It's also PS 4)



Example: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – initial]

Example: Learning to Walk

Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

Example: Sidewinding

[Andrew Ng]

[Video: SNAKE – climbStep+sidewinding]

“Few driving tasks are as intimidating as parallel parking….

https://www.youtube.com/watch?v=pB_iFY2jIdI

Parallel Parking

“Few driving tasks are as intimidating as parallel parking….

https://www.youtube.com/watch?v=pB_iFY2jIdI

Other Applications

- Go playing
- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

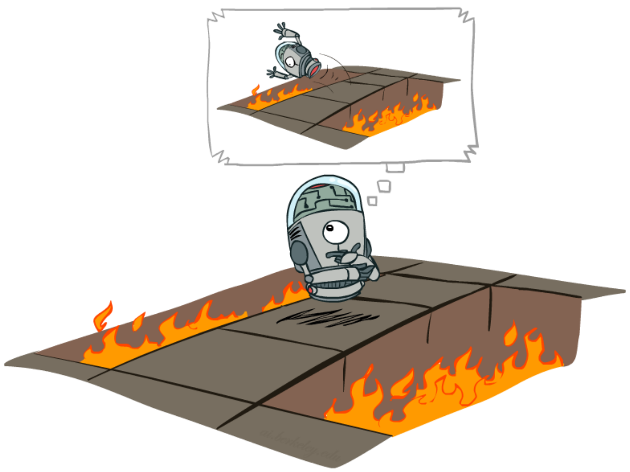
Reinforcement Learning

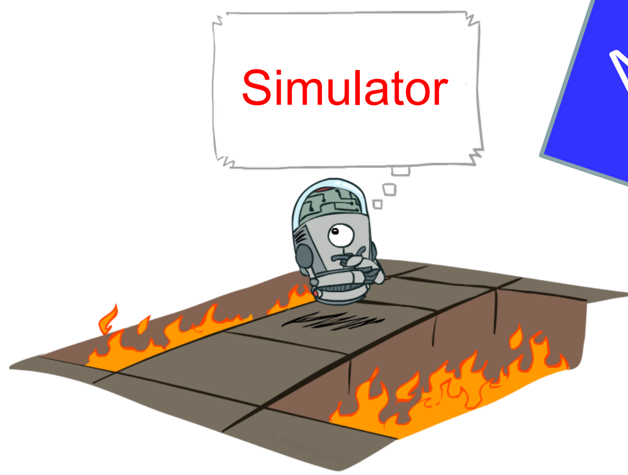
- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) A
  - A model $T(s,a,s')$
  - A reward function $R(s,a,s')$ & discount γ
- Still looking for a policy $\pi(s)$

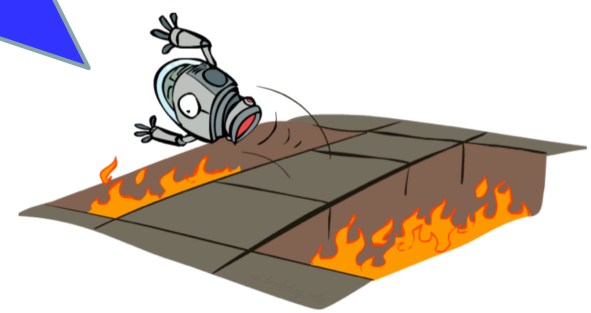
- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn

Offline (MDPs) vs. Online (RL)

Many people call this RL as well







Offline Solution (Planning)

Monte Carlo Planning

Online Learning (RL)

Diff: 1) dying ok; 2) (re)set button

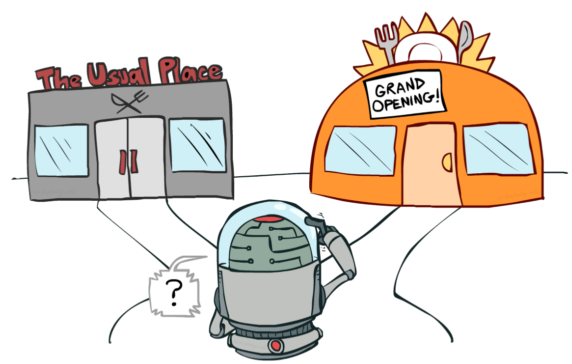
Four Key Ideas for RL

- Credit-Assignment Problem
  - What was the real cause of reward?
- Exploration-exploitation tradeoff
- Model-based *vs* model-free learning
  - What function is being learned?
- Approximating the Value Function
  - Smaller → easier to learn & better generalization

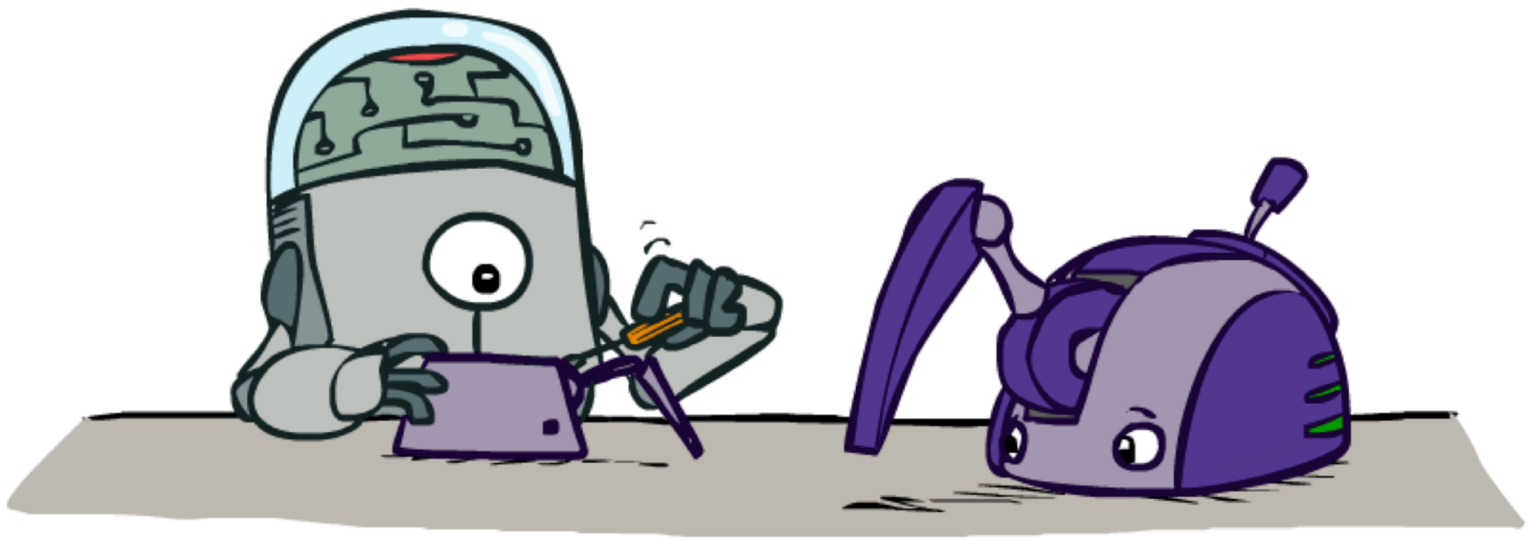
Credit Assignment Problem

Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best you can hope for???
- **Exploitation**: should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at risk of missing out on a better reward somewhere
- **Exploration**: should I look for states w/ more reward?
  - at risk of wasting time & getting some negative reward



Model-Based Learning



Model-Based Learning

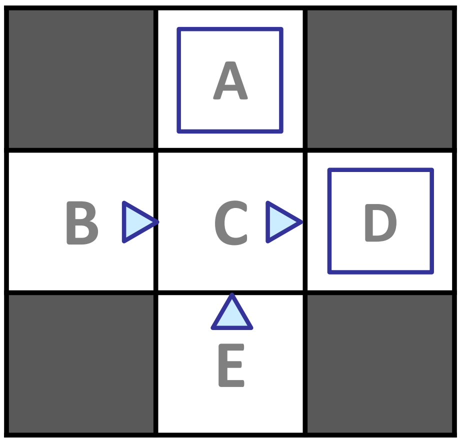
- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Explore (e.g., move randomly)
  - Count outcomes s' for each s, a $\widehat{T}(s, a, s')$
  - Normalize to g$\widehat{R}(s, a, s')$nate of
  - Discover each when we experience (s, a, s')

- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Example: Model-Based Learning

Random π



Assume: $\gamma = 1$

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
…

Convergence

- If policy explores "enough" – doesn't starve any state
- Then T & R converge

- So, VI, PI, Lao* *etc.* will find optimal policy
  - Using Bellman Equations

- When can agent start exploiting??
  - (We'll answer this question later)

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment & learn model, T=P(**s**'|**s**,**a**) and R(**s**,**a**), (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - often works well when state-space is manageable
- Model-free approach:
  - don't learn a model of T&R; instead, learn Q-function (or policy) directly
  - weaker theoretical results
  - often works better when state space is large

Two main reinforcement learning approaches

- Model-based approaches:

  Learn T + R

  $|S|^2|A| + |S||A|$ parameters (40,400)

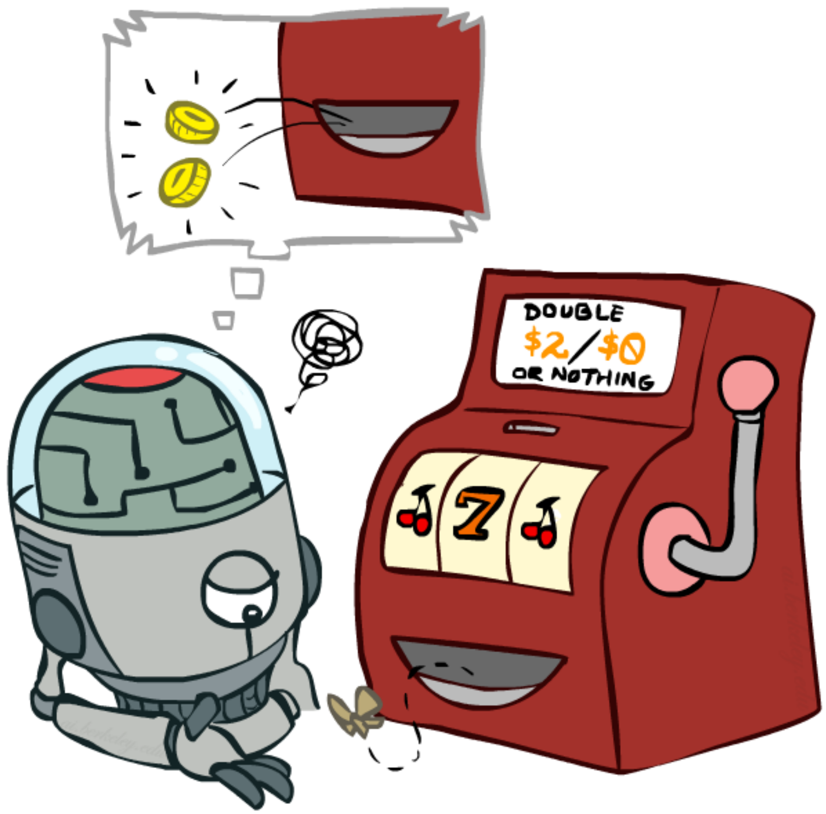
- Model-free approach:

  Learn Q

  $|S||A|$ parameters (400)

Suppose 100 states, 4 actions

Model-Free Learning



Nothing is Free in Life!

- What exactly is Free???
  - No model of T
  - No model of R
  - (Instead, just model Q)

Reminder: **Q-Value Iteration**

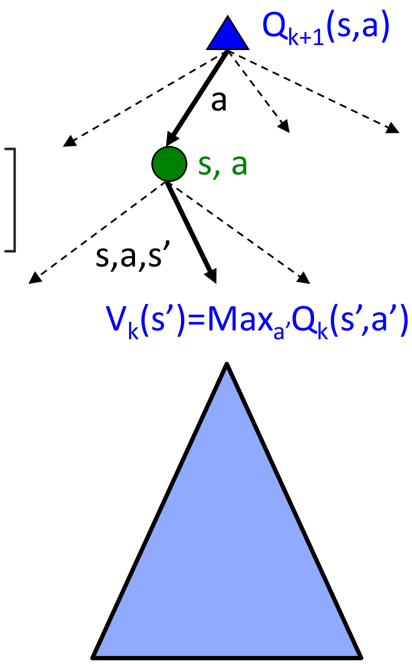
- **Forall s, a**
  - **Initialize $Q_0(s, a) = 0$** *no time steps left means an expected reward of zero*
- **K = 0**
- **Repeat** *do Bellman backups*

  For every (s,a) pair:

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

  K += 1
- **Until convergence** *I.e., Q values*

This is easy….



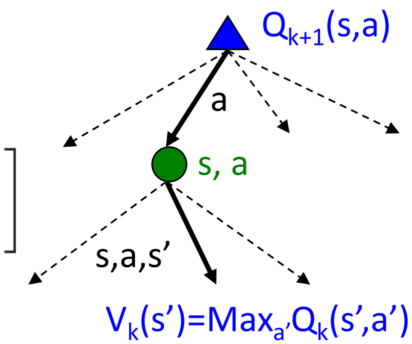
Puzzle: Q-Learning

- **Forall s, a**
  - **Initialize $Q_0(s, a) = 0$** *no time steps left means an expected reward of zero*
- **K = 0**
- **Repeat** *do Bellman backups*

  For every (s,a) pair:

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

  K += 1
- **Until convergence**



Q: How can we compute without R, T ?!?

A: Compute averages using sampled outcomes

Simple Example: Expected Age

Goal: Compute expected age of CSE students

Known P(A)

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

Note: never know **P(age=22)**

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.

Anytime Model-Free Expected Age

Goal: Compute expected age of CSE students

Let A=0
Loop for i = 1 to ∞
    a_i ← ask "what is your age?"
    A ← (1-α)*A + α*a_i

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

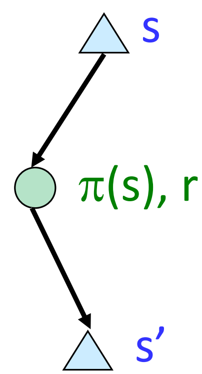
Let A=0
Loop for i = 1 to ∞
    a_i ← ask "what is your age?"
    A ← (i-1)/i * A + (1/i) * a_i

Unknown P(A): "Model Free"

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

Sampling Q-Values

- Big idea: learn from every experience!
  - Follow exploration policy $a \leftarrow \pi(s)$
  - Update Q(s,a) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often



- Update towards running average:

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

Update to Q(s,a): $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)sample$

Same update: $Q(s,a) \leftarrow Q(s,a) + \alpha(sample - Q(s,a))$

Rearranging: $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{difference})$
Where $\text{difference} = (R(s,a,s') + \gamma \max_{a'} Q(s', a')) - Q(s,a)$

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s.
  Choose some action a
  Execute it in real world: *(s, a, r, s')*
  Do update:

$$\text{difference} \leftarrow [R(s,a,s') + \gamma \, \text{Max}_{a'} \, Q(s', a')] - Q(s,a)$$

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

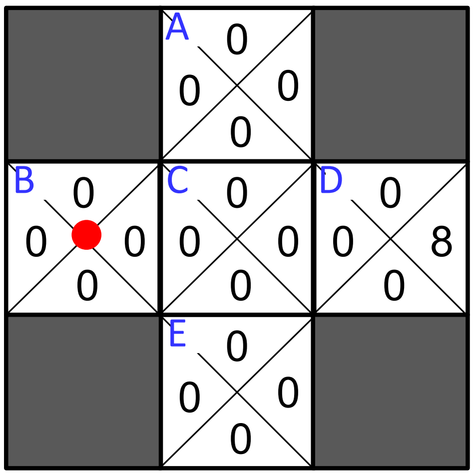
Trial

Note parallel to RTDP
- Both have trials
- What is difference?

Example

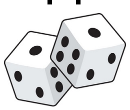
Assume: $\gamma = 1$, $\alpha = 1/2$

Observed Transition: B, east, C, -2



In state B. What should you do?

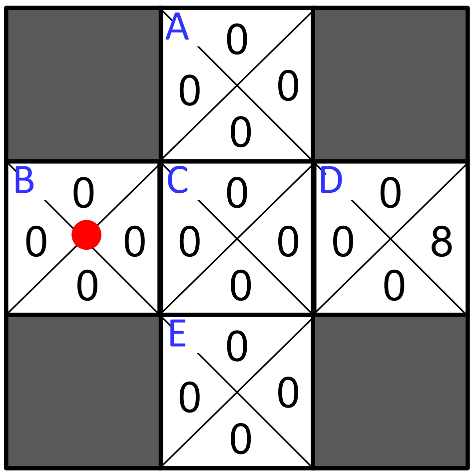
Suppose (for now) we follow a random exploration policy

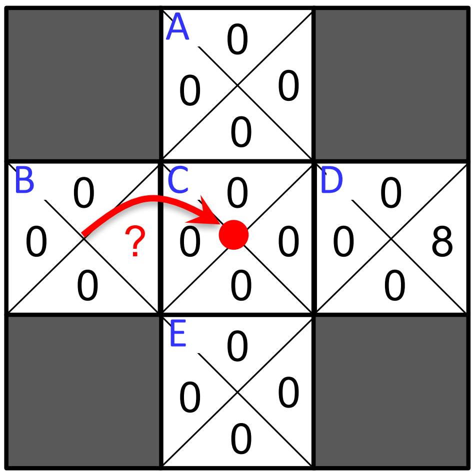
 → "Go east"

Example

Assume: γ = 1, α = 1/2

Observed Transition: B, east, C, -2





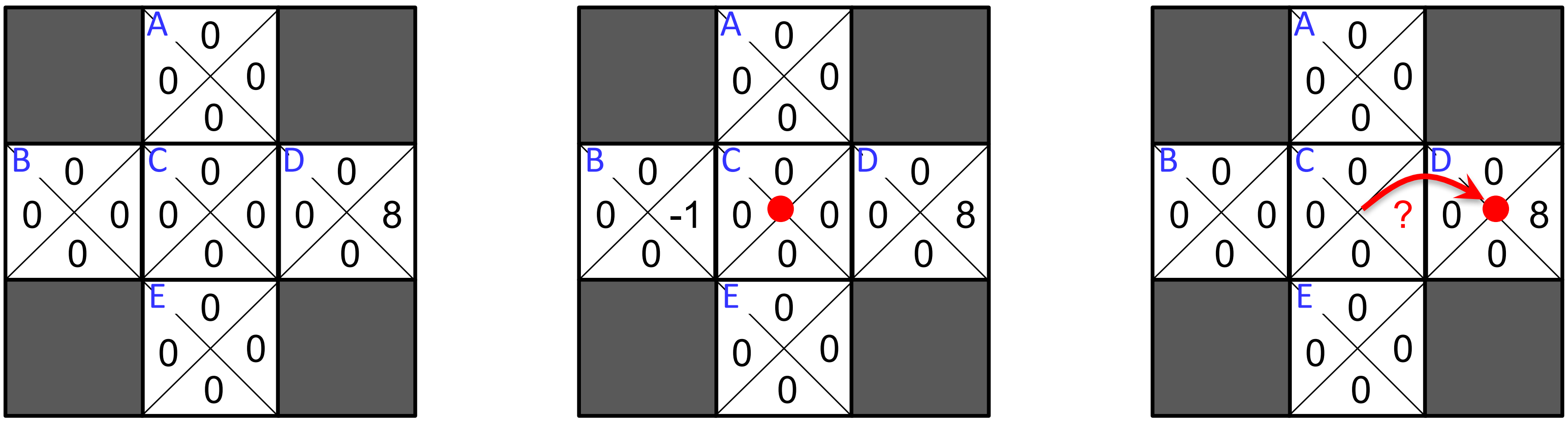
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

-1 ½ 0 ½ -2 0

Example

Assume: γ = 1, α = 1/2

Observed Transition: B, east, C, -2 C, east, D, -2



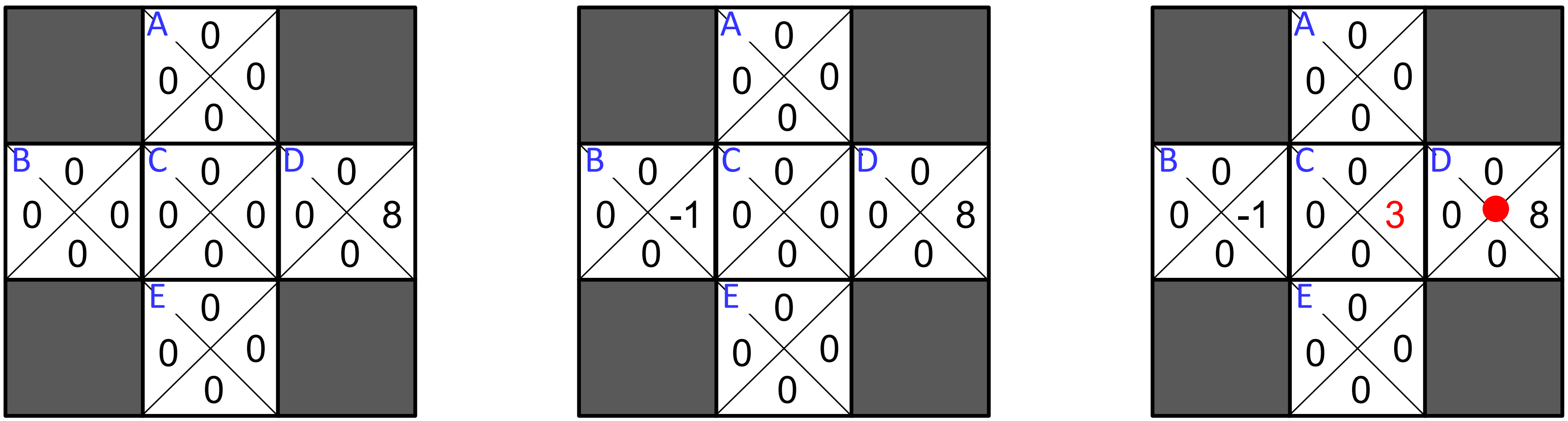
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

3 ½ 0 ½ -2 8

Example

Assume: γ = 1, α = 1/2

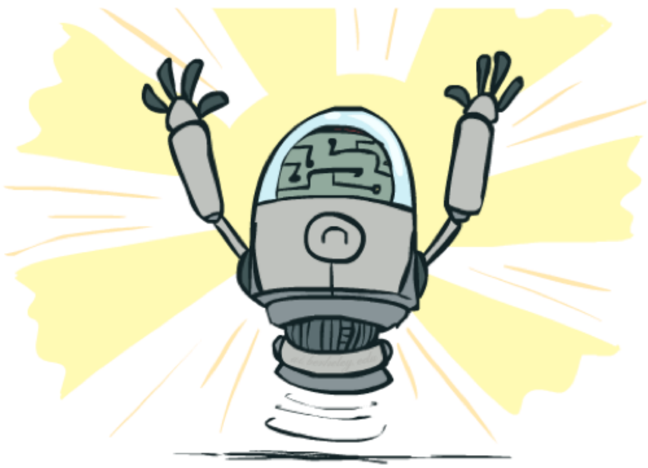
Observed Transition: B, east, C, -2 C, east, D, -2



$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Q-Learning Properties

- Q-learning converges to optimal Q function (and hence ***learns*** optimal policy)
  - even if you're acting suboptimally!
  - This is called off-policy learning
- Caveats:
  - *You have to explore enough*
  - *You have to eventually shrink the learning rate, α*
  - *… but not decrease it too quickly*
- And… if you want to ***act*** optimally
  - You have to switch from explore to exploit



[Demo: Q-learning – auto – cliff grid (L11D1)]

Video of Demo Q-Learning Auto Cliff Grid

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s.

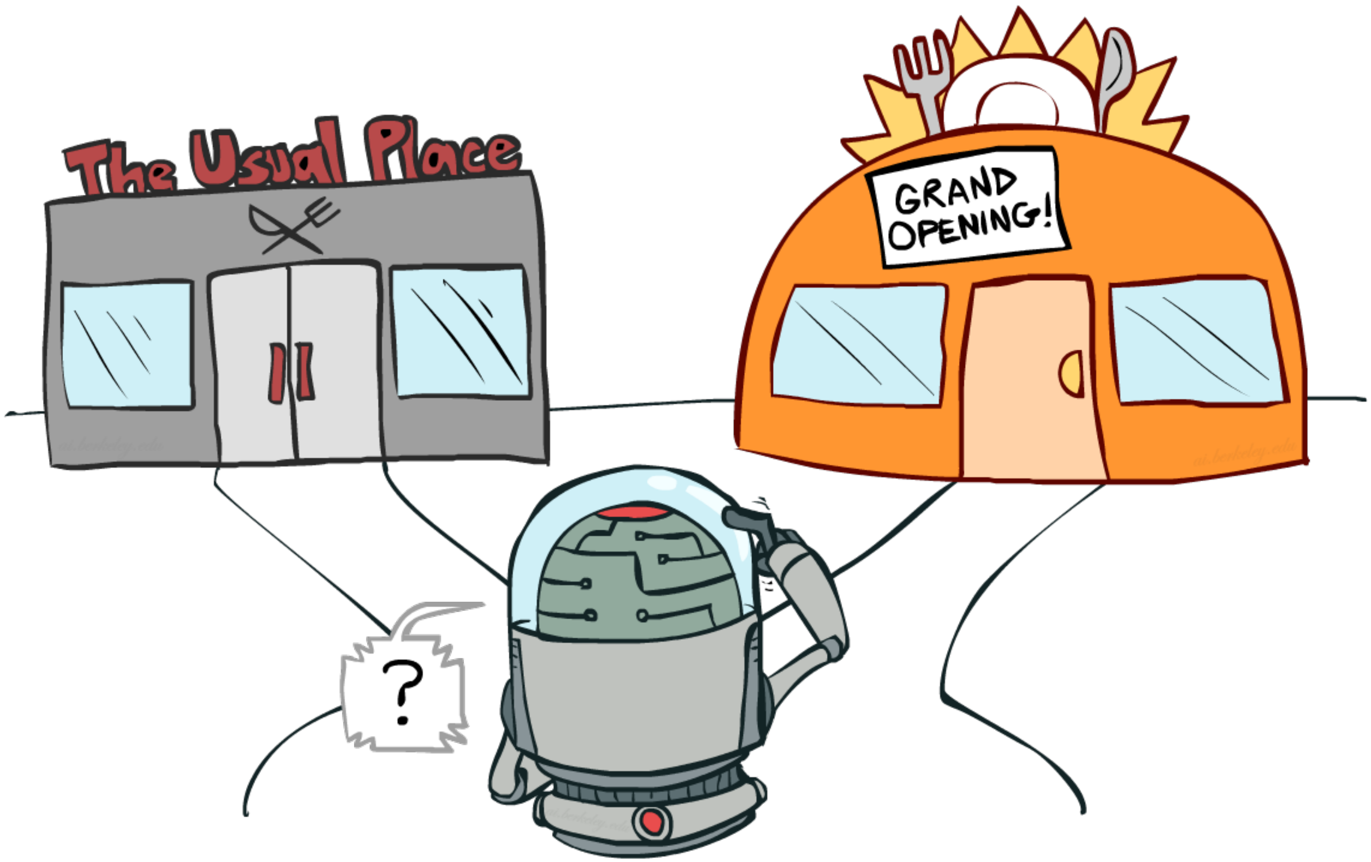
  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

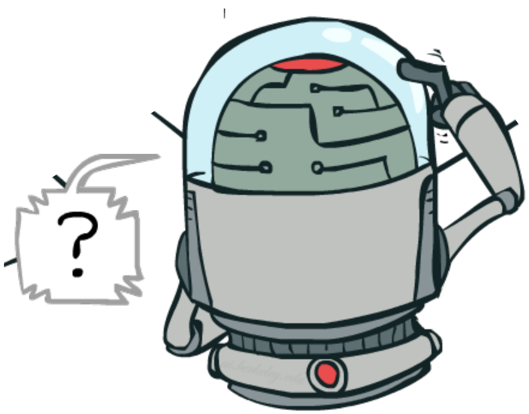
$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Exploration vs. Exploitation



Questions

- How to explore?

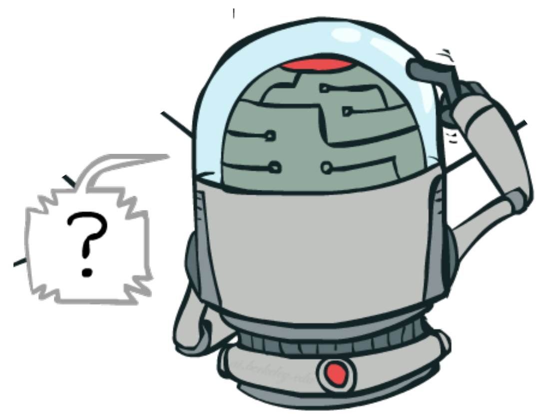


- When to exploit?

- How to even think about this tradeoff?

Questions

- How to explore?
  - Random Exploration
  - Uniform exploration
  - Epsilon Greedy
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on ***current policy***
  - Exploration Functions (such as UCB)
  - Thompson Sampling
- When to exploit?
- How to even think about this tradeoff?



Exploration Functions

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

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

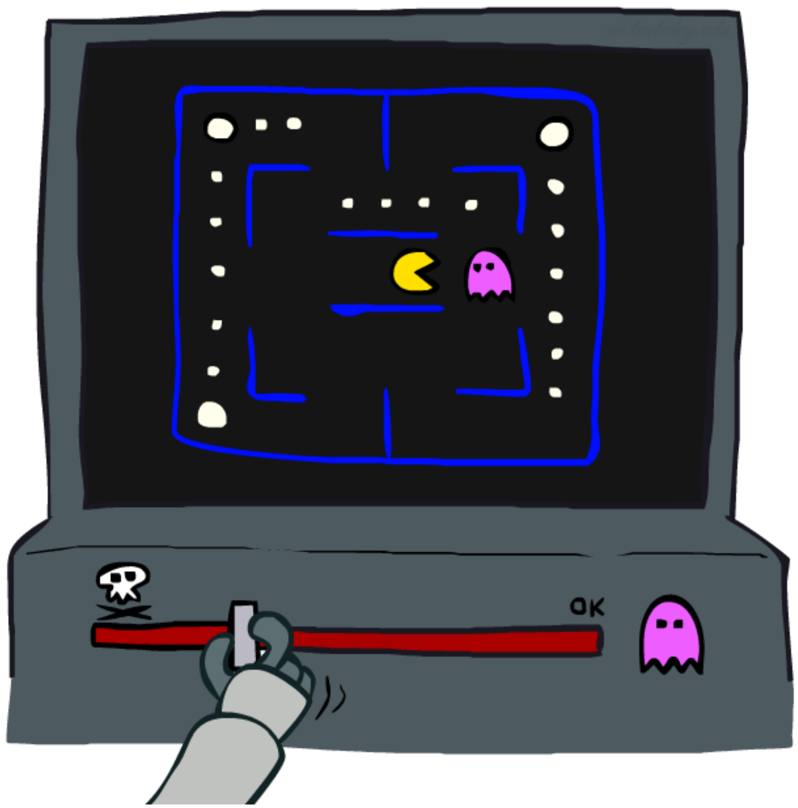
Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

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

Video of Demo Crawler Bot

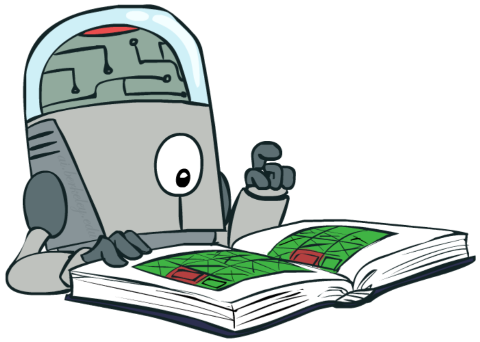
More demos at: http://inst.eecs.berkeley.edu/~ee128/fa11/videos.html

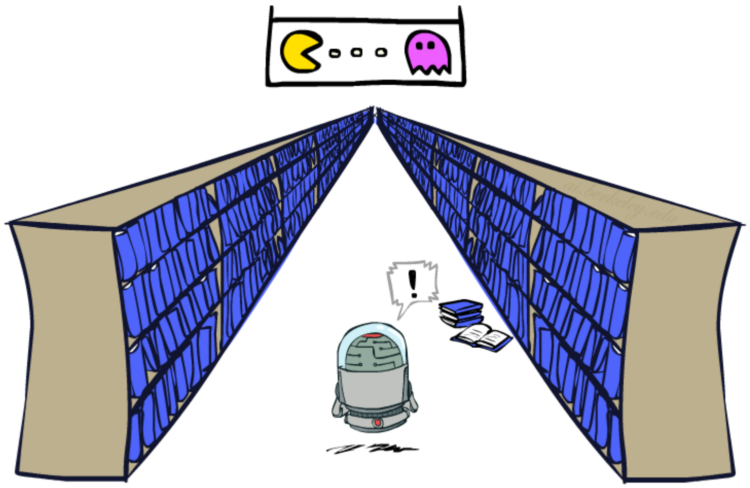
Approximate Q-Learning



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

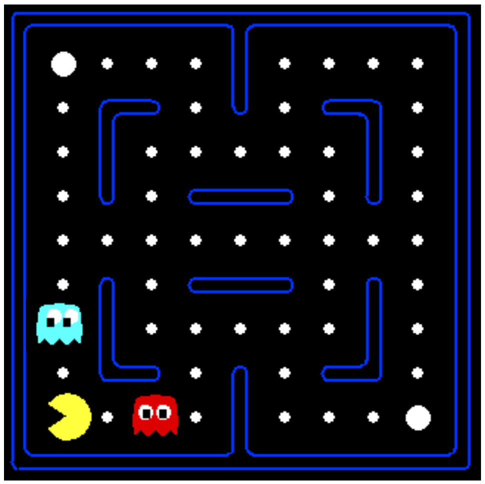




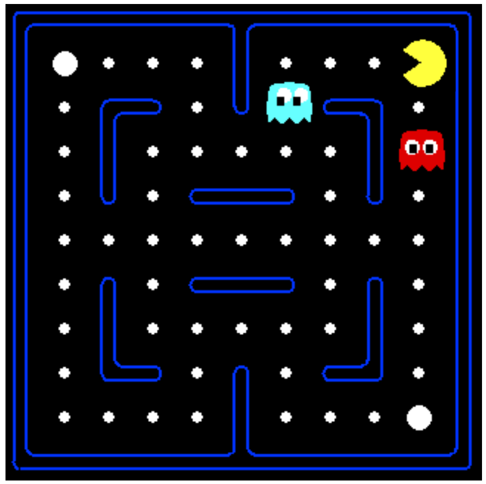
[demo – RL pacman]

Example: Pacman

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

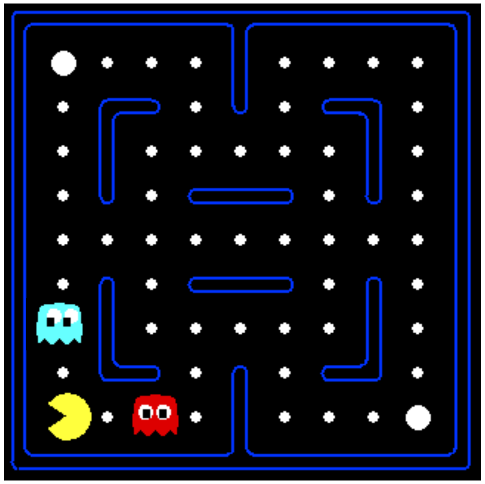


In naïve q-learning, we know nothing about this state:

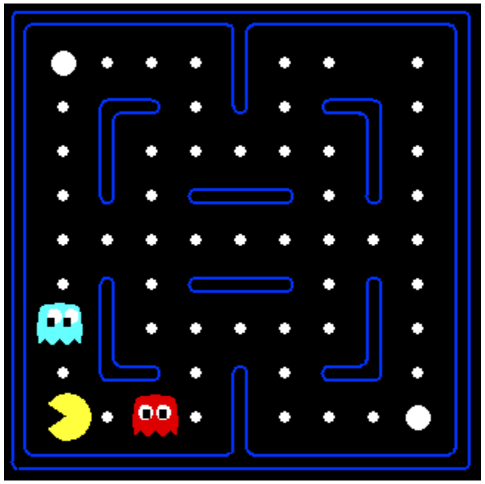


Example: Pacman

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



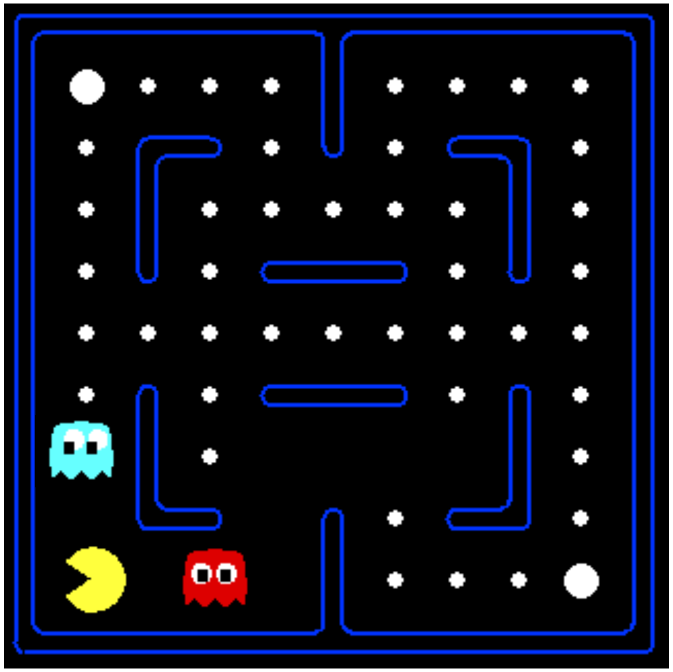
Or even this one!



Feature-Based Representations

Solution: describe a state using a **vector of features** (aka "properties")

- Features = functions from states to R (often 0/1) capturing important properties of the state
- Example features:
  - Distance to closest ghost or dot
  - Number of ghosts
  - 1 / (dist to dot)^2
  - Is Pacman in a tunnel? (0/1)
  - …… etc.
  - Is it the exact state on this slide?
- Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Combination of Features

- 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 sharing features may actually have very different values!

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:

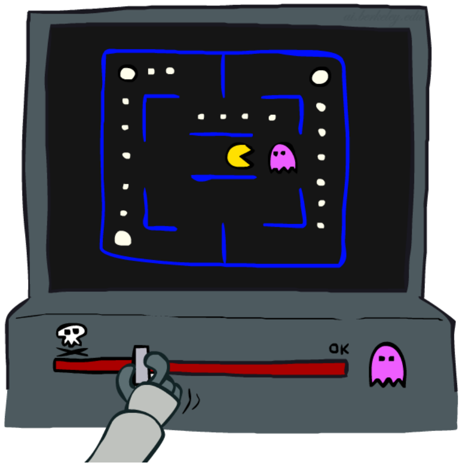
transition $= (s, a, r, s')$

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

Forall i do:

$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a)$$ 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: in a few slides!

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s.
  Choose some action a
  Execute it in real world: *(s, a, r, s')*
  Do update:

$$\text{difference} \leftarrow [R(s,a,s') + \gamma \,\text{Max}_{a'}\, Q(s', a')] - Q(s,a)$$

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

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

- **Forall *i***
  - Initialize $w_i = 0$
- **Repeat Forever**

  Where are you? s.
  Choose some action a
  Execute it in real world: *(s, a, r, s')*
  Do update:

  difference ← [R(s,a,s') + γ $Max_{a'}$ Q(s', a')] - Q(s,a)

  Q(s,a) ← Q(s,a) + α(difference)