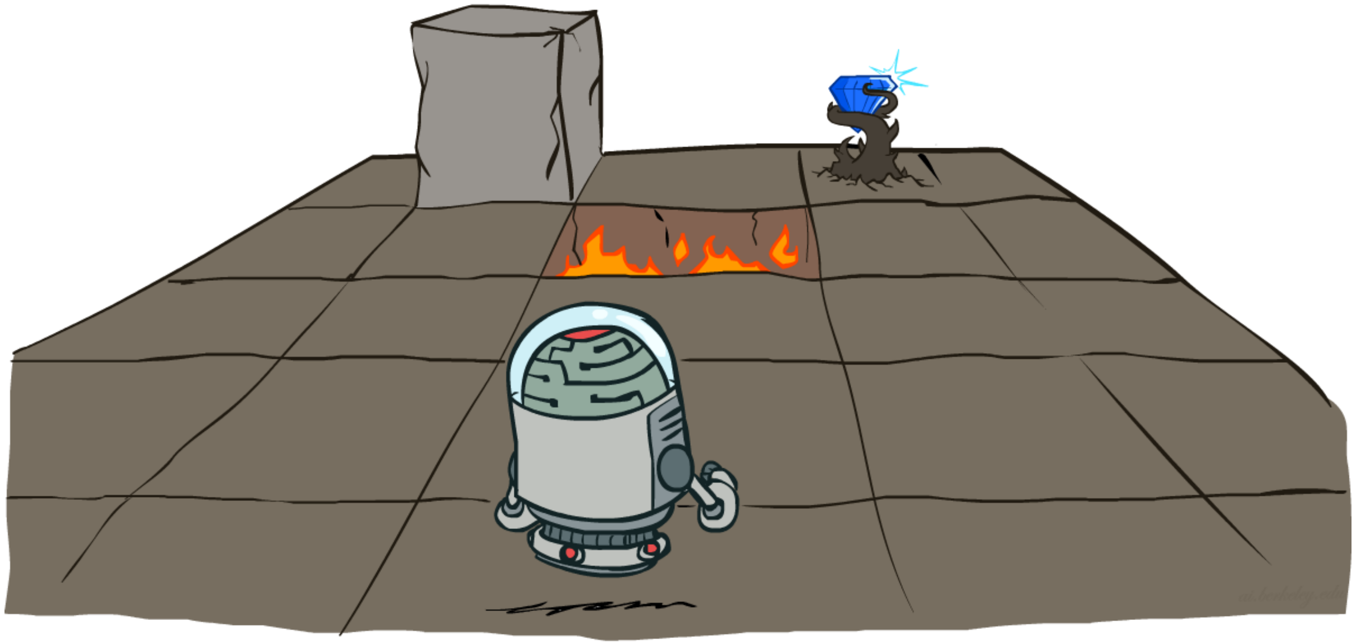
Markov Decision Processes



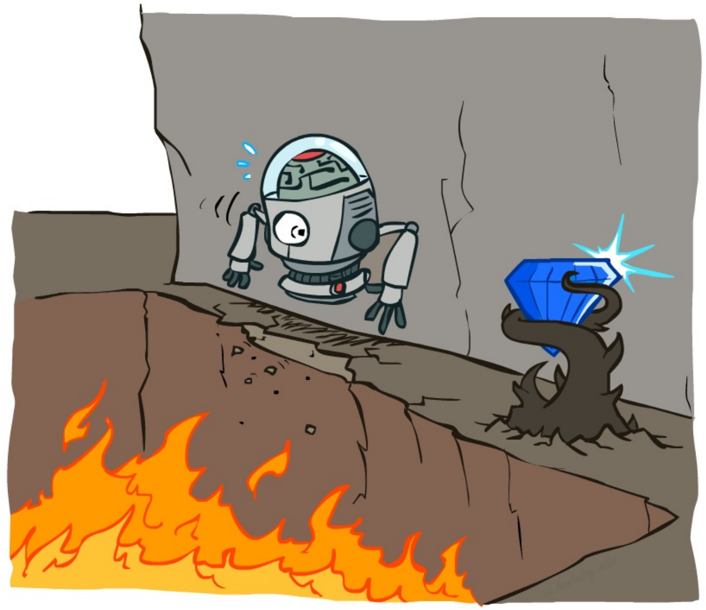
Decisions with unknown preferences

- In reality the assumption that we can write down our exact preferences for the machine to optimize is false
- A machine optimizing the wrong preferences causes problems


I’m sorry, Dave, I’m afraid I can’t do that

Sequential decisions under uncertainty

Sequential decision problem: agent's utility depends on a sequence of actions



Markov Decision Process (MDP)

- Environment history: [s_0, a_0, s_1, a_1, …, s_t]
- "Markov" generally means that given the present state, the future and the past are independent

- For Markov decision processes, "Markov" means action outcomes depend only on the current state

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

$$=$$

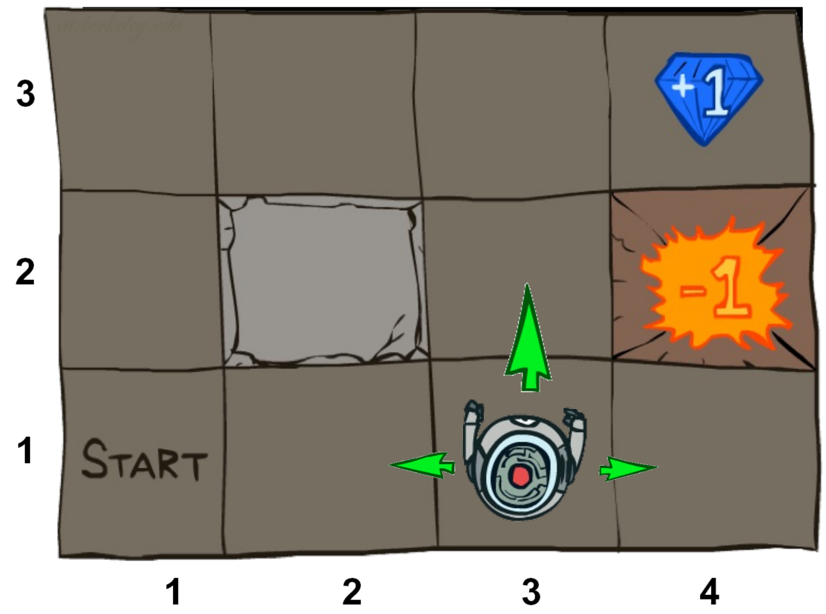
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

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

Andrey Markov
(1856-1922)

Markov Decision Process (MDP)

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition model $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s' | s, a)$
  - A reward function $R(s, a, s')$ for each transition
  - A start state
  - Possibly a terminal state (or absorbing state)
  - Utility function which is additive (discounted) rewards

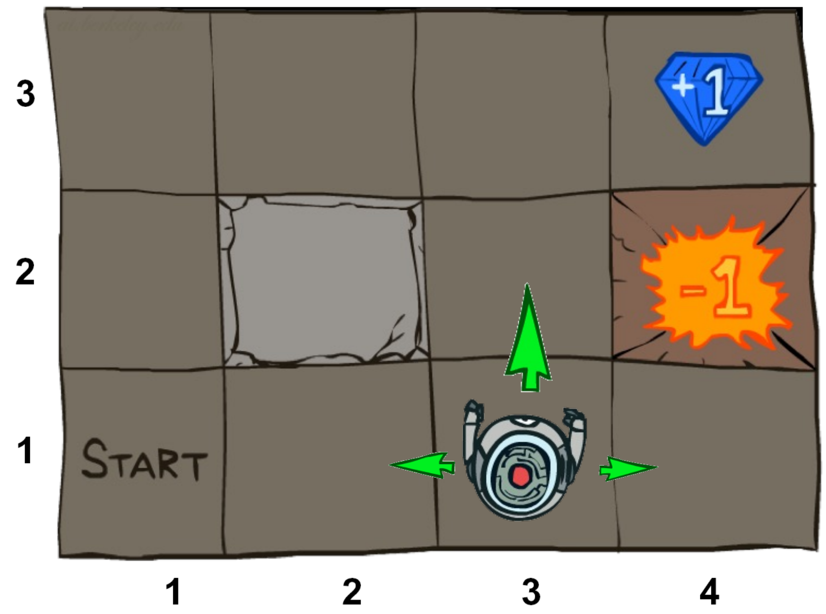


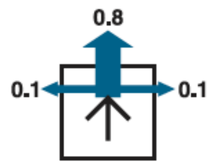
- MDPs are fully observable but probabilistic search problems

[Demo – gridworld manual intro (L8D1)]

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



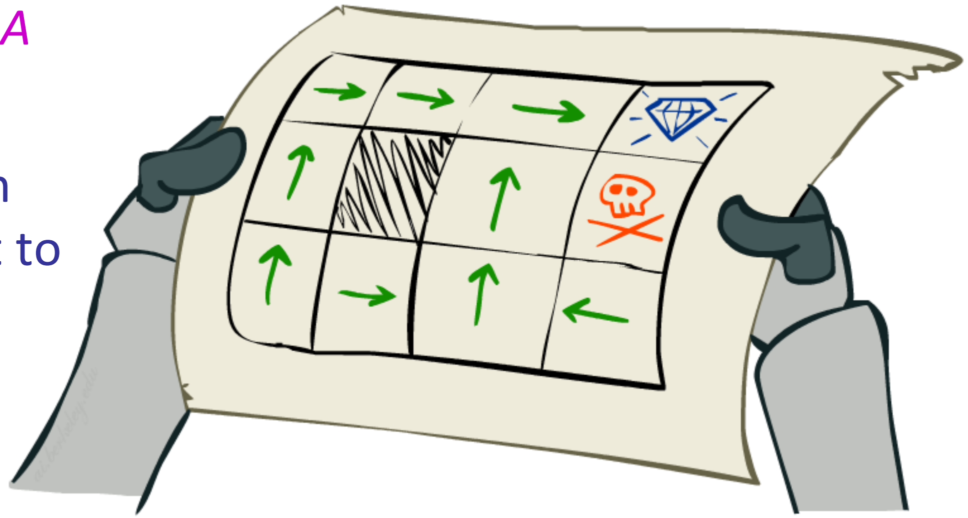


Policies

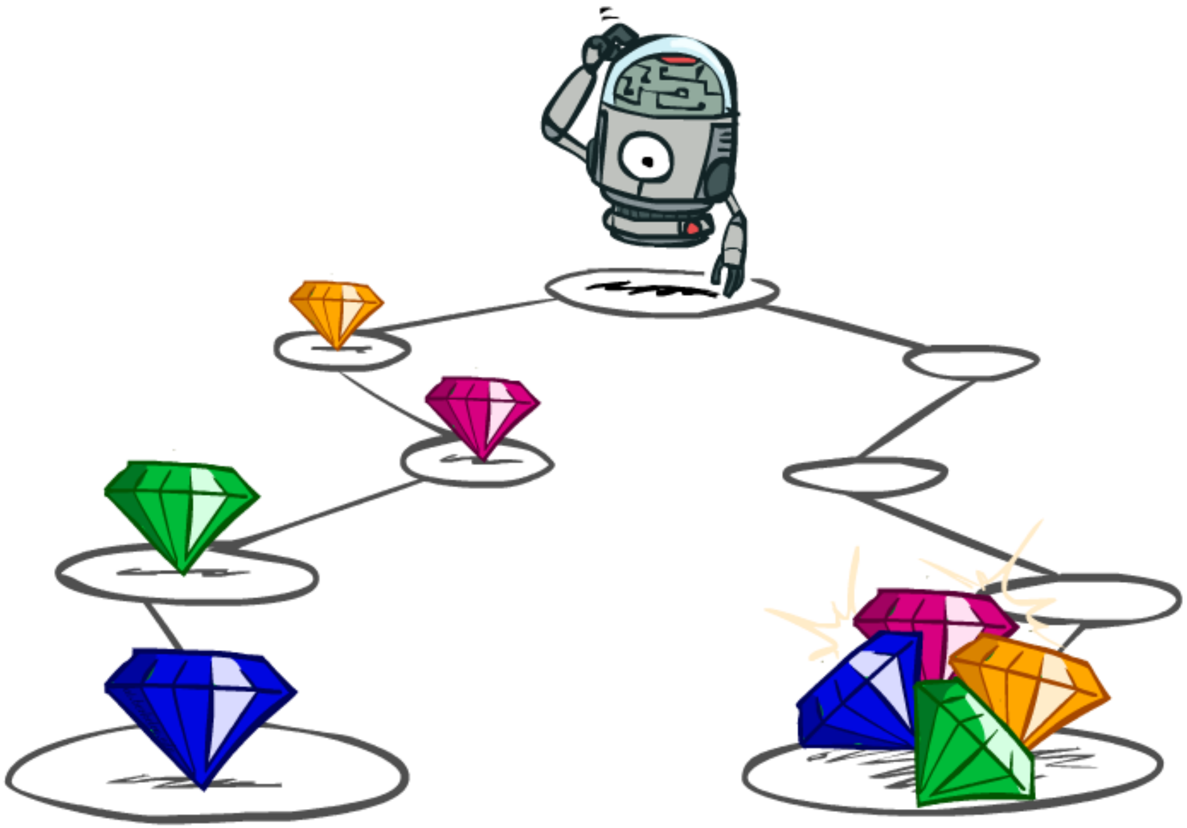
- A policy π gives an action for each state, π: *S* → *A*

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

- For MDPs, we want an optimal ***policy*** π^*: *S* → *A*
  - An optimal policy maximizes expected utility
  - An explicit policy defines a reflex agent



Utilities of Sequences

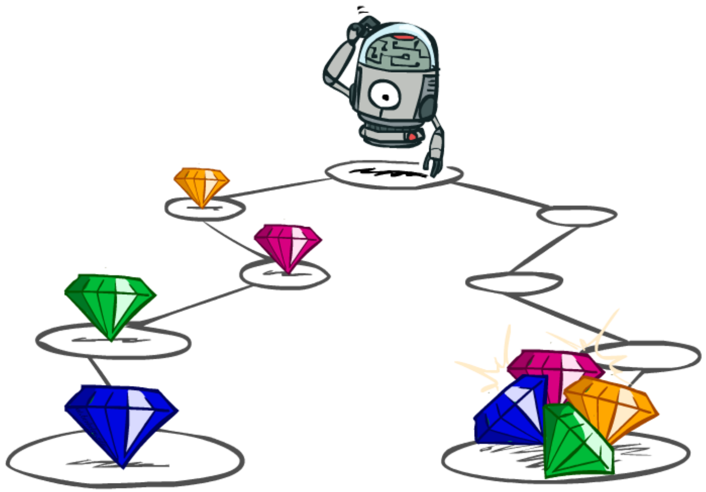


Utilities of Sequences

- What preferences should an agent have over reward sequences?

- More or less? [1, 2, 2] or [2, 3, 4]

- Now or later? [0, 0, 1] or [1, 0, 0]



Stationary Preferences

- Theorem: if we assume ***stationary preferences***:

  $[s_0, a_0, s_1, a_1, s_2, \ldots] \succ [s'_0, a'_0, s'_1, a'_1, s'_2, \ldots]$, $s_0 = s'_0$, $a_0 = a'_0$, and $s_1 = s'_1$

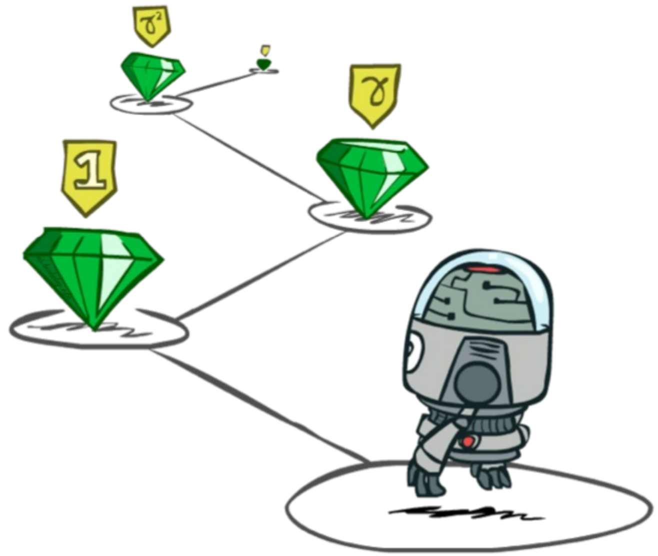
  $\Leftrightarrow \quad [s_1, a_1, s_2, \ldots] \succ [s'_1, a'_1, s'_2, \ldots]$

  then there is only one way to define utilities:

  - ***Additive discounted utility***:

  $$U_h([s_0, a_0, s_1, a_1, s_2, \ldots]) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \cdots$$

  where $\gamma \in [0,1]$ is the ***discount factor***

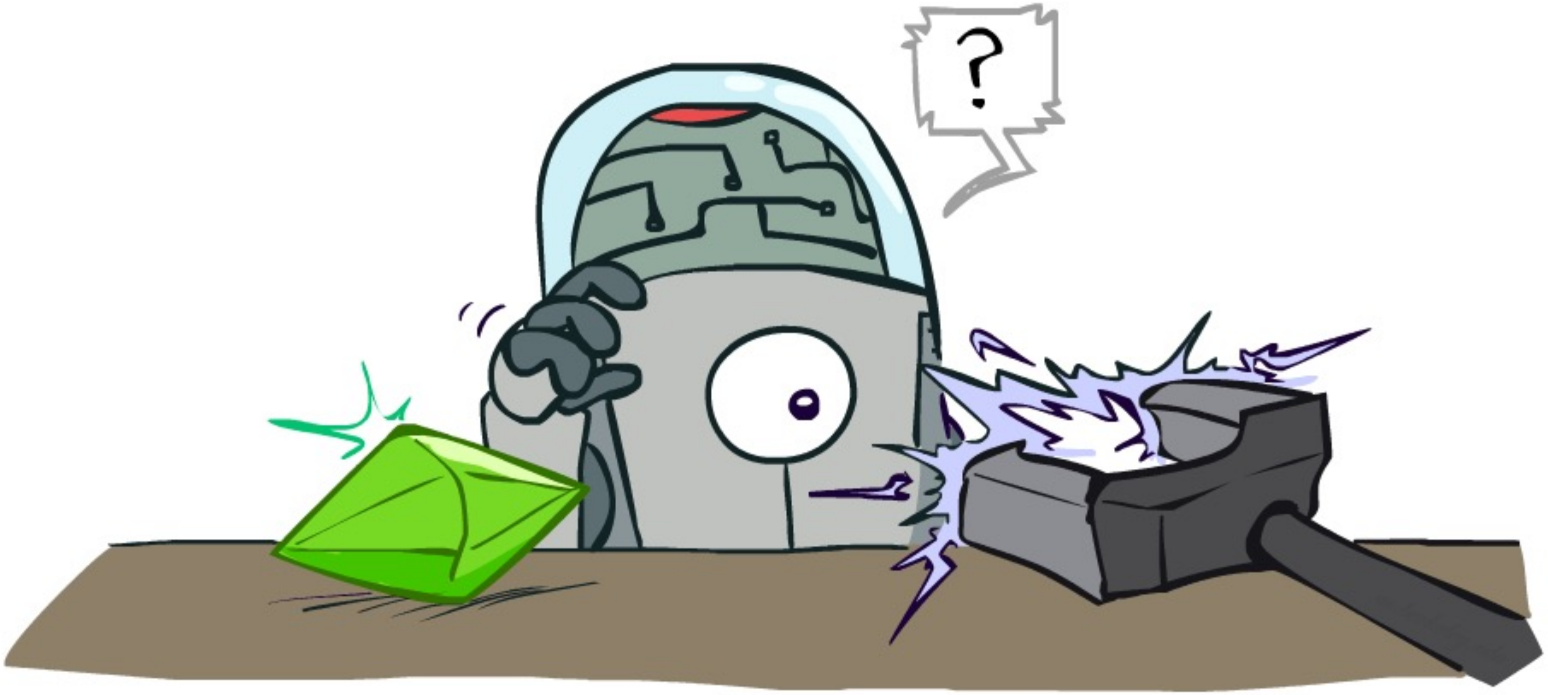


Solving MDPs

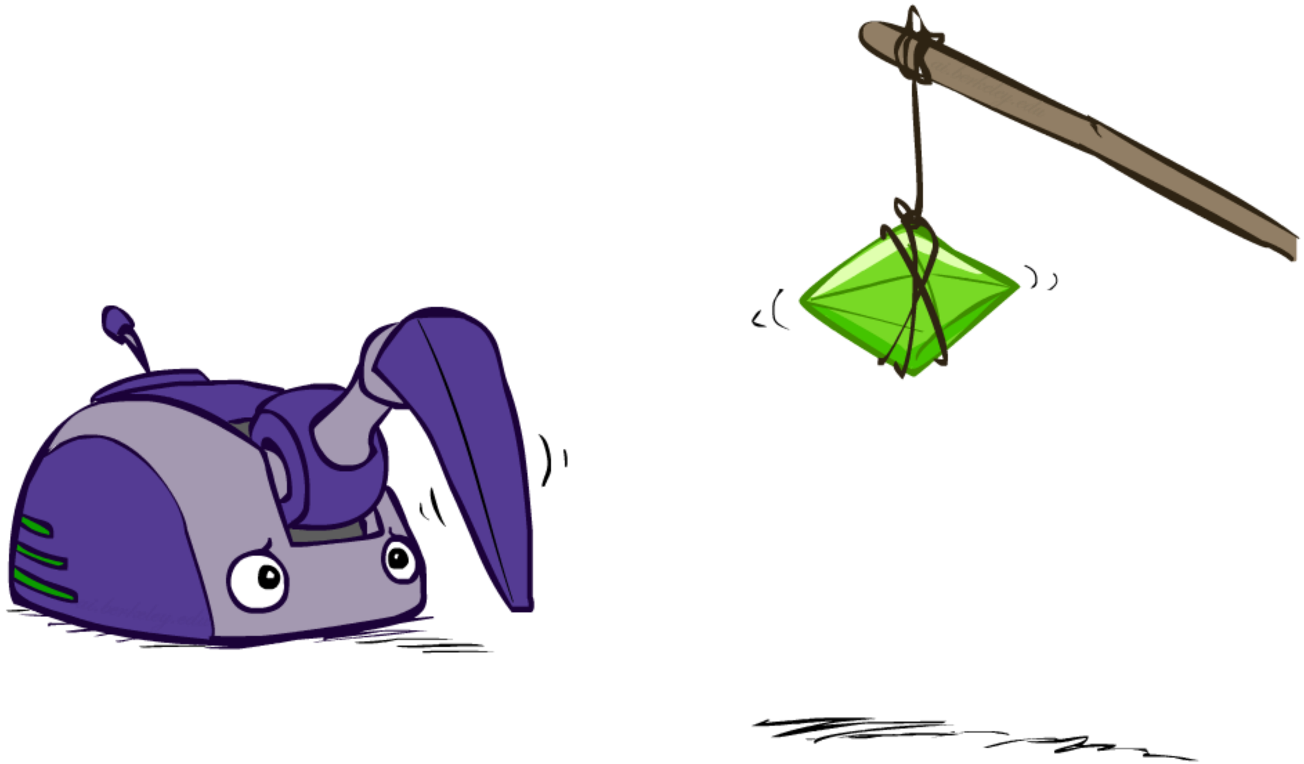
- Value iteration

- Policy iteration

Reinforcement Learning

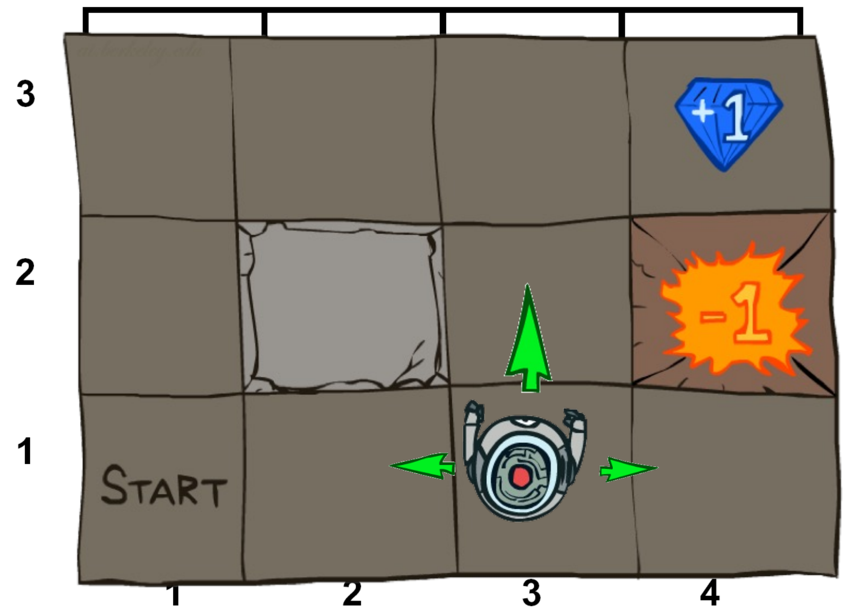


Reinforcement Learning



Markov Decision Process (MDP)

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition model $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s' | s, a)$
  - A reward function $R(s, a, s')$ for each transition
  - A start state
  - Possibly a terminal state (or absorbing state)
  - Utility function which is additive (discounted) rewards



- MDPs are fully observable but probabilistic search problems

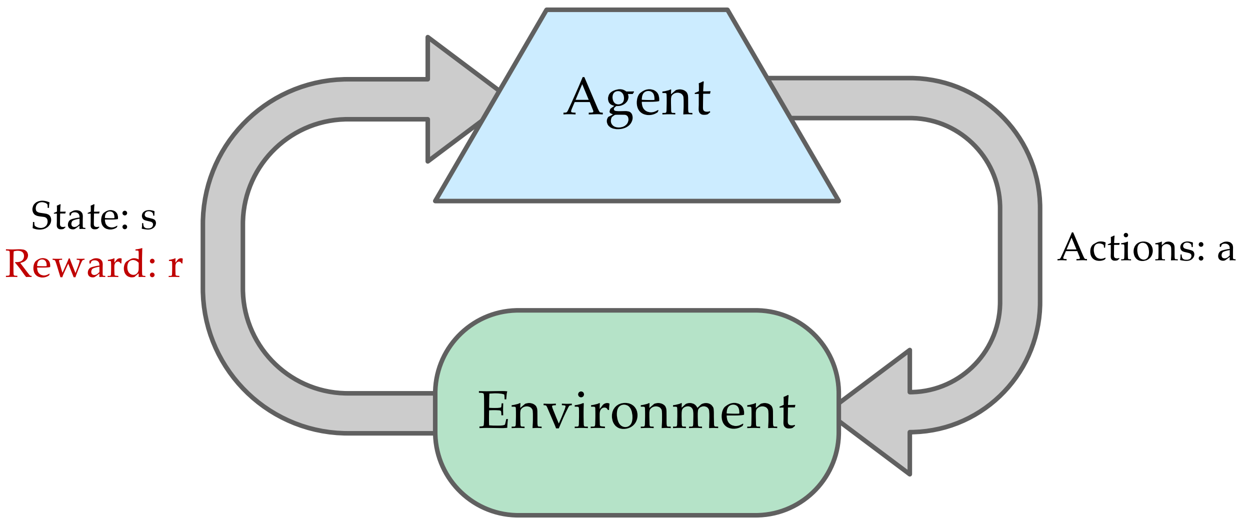
[Demo – gridworld manual intro (L8D1)]

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 $\pi(s)$

- 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

Reinforcement Learning



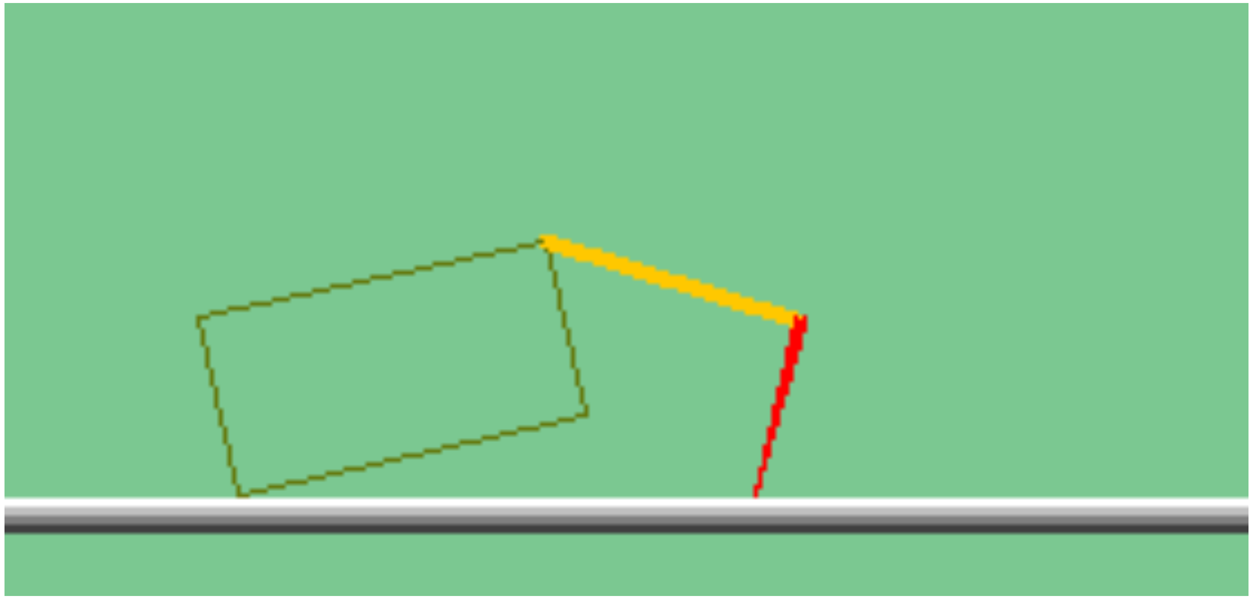
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards
  - All learning is based on observed samples of outcomes!

Reinforcement learning

Basic ideas:

- ***Exploration***: you have to ***try unknown actions*** to get information
- ***Exploitation***: eventually, you have to use what you know
- ***Sampling***: you may need to repeat many times to get good estimates
- ***Generalization***: what you learn in one state may apply to others too

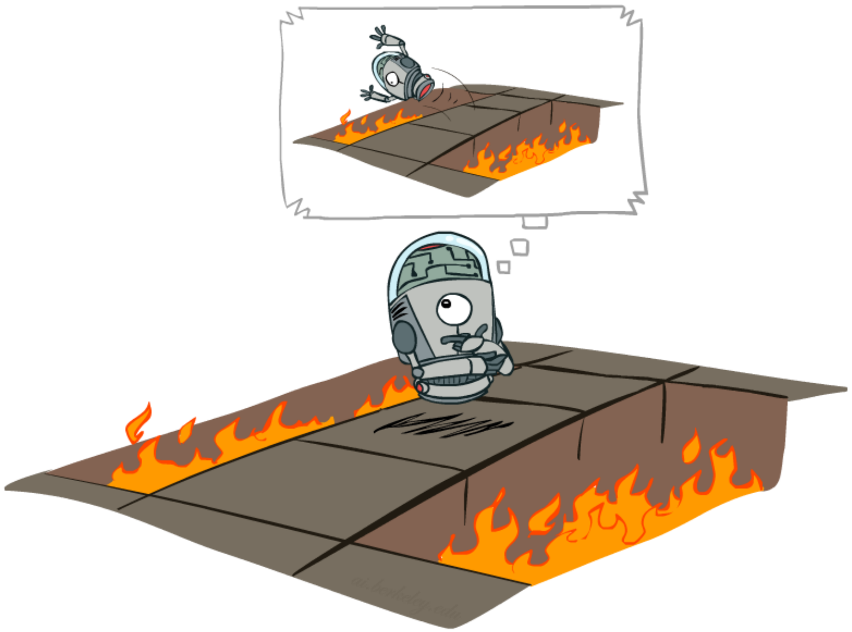
The Crawler!



Video of Demo Crawler Bot

DeepMind Atari (©Two Minute Lectures)

Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler

Video of Demo Q-learning – Manual Exploration – Bridge Grid

How to Explore?

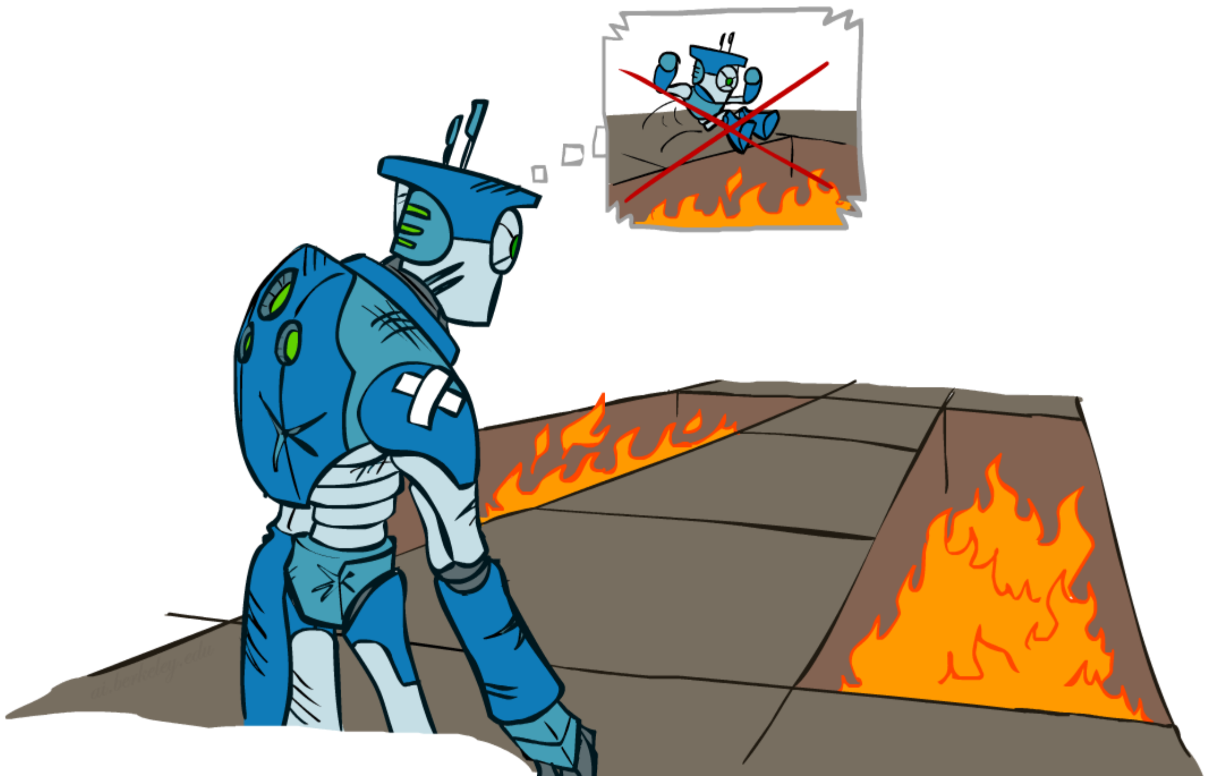
- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability $1-\varepsilon$, act on current policy
  - Problems with random actions?
    - You do eventually explore the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

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

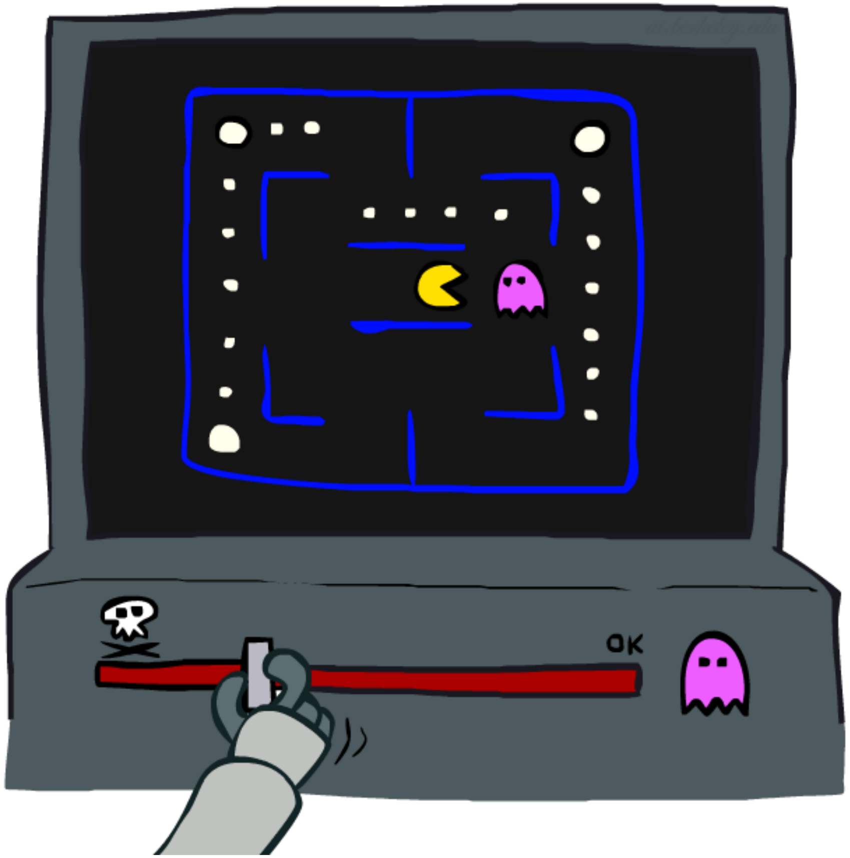
Video of Demo Q-learning – Exploration Function – Crawler

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

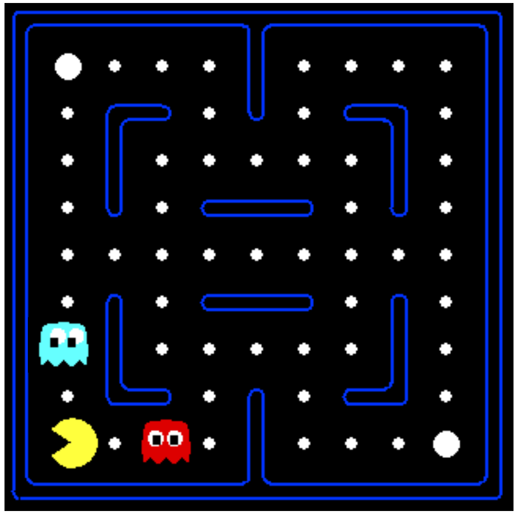


Approximate Q-Learning

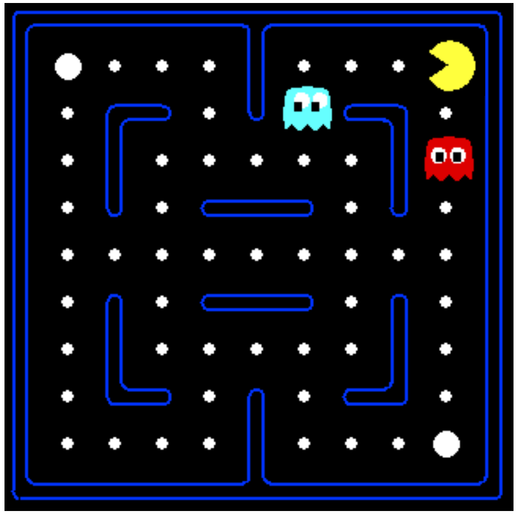


Example: Pacman

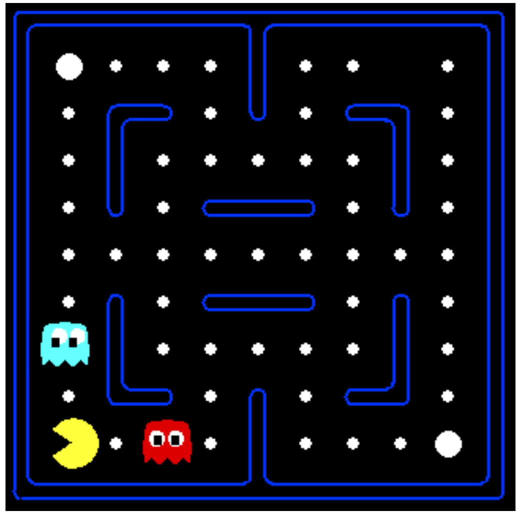
Let's say we discover through experience that this state is bad:



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



Or even this one!



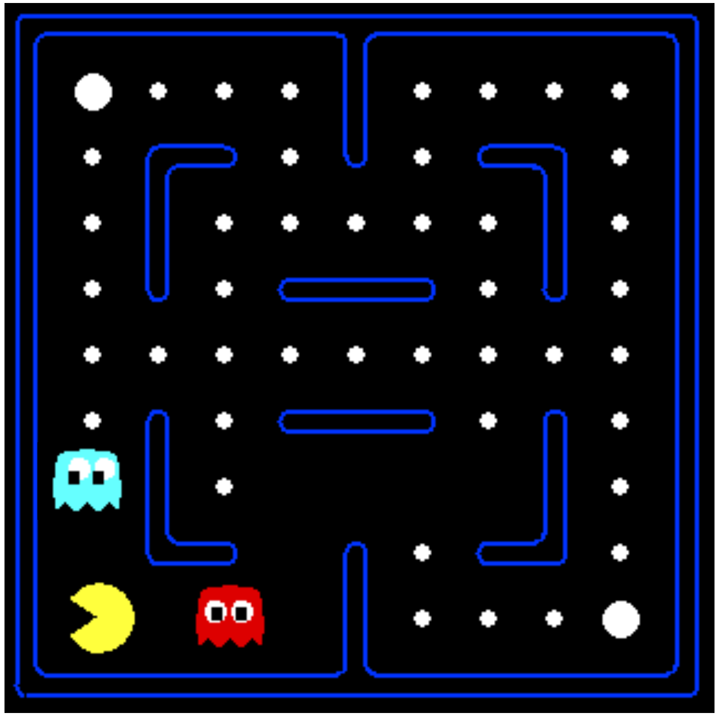
Video of Demo Q-Learning Pacman – Tiny – Watch All

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

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

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.
    - 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 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

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

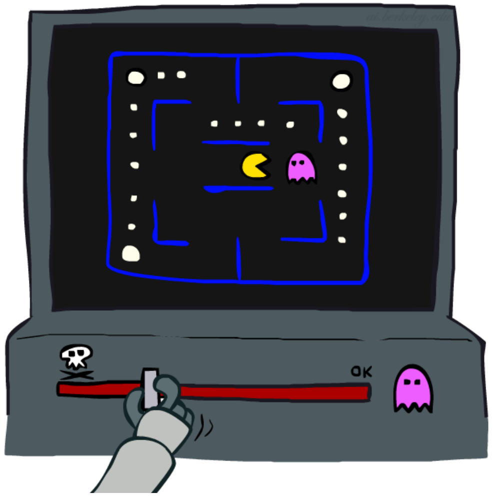
- Q-learning with linear Q-functions:

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

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

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

- Formal justification: online least squares

Video of Demo Approximate Q-Learning -- Pacman