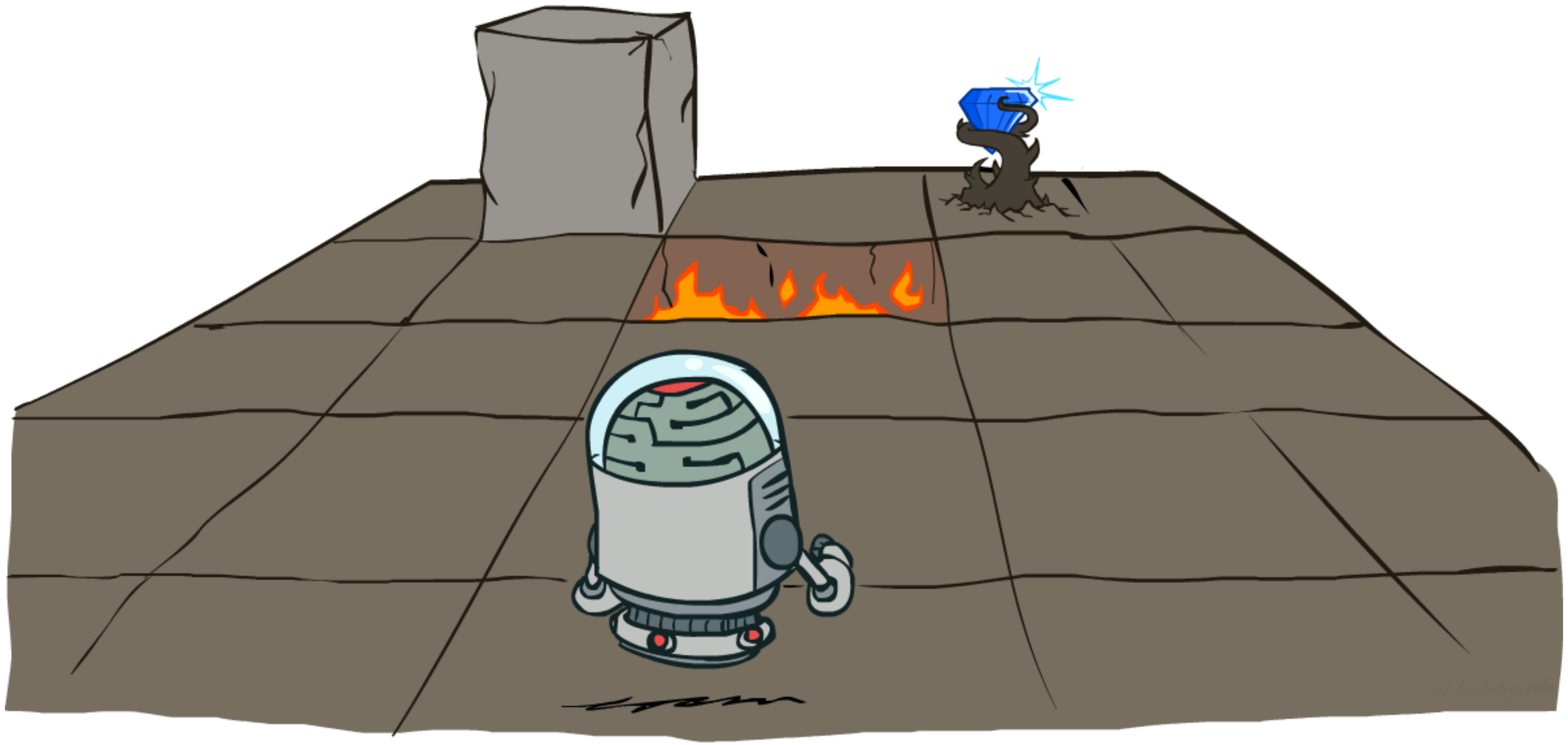
Intro to MDPs and Reinforcement Learning



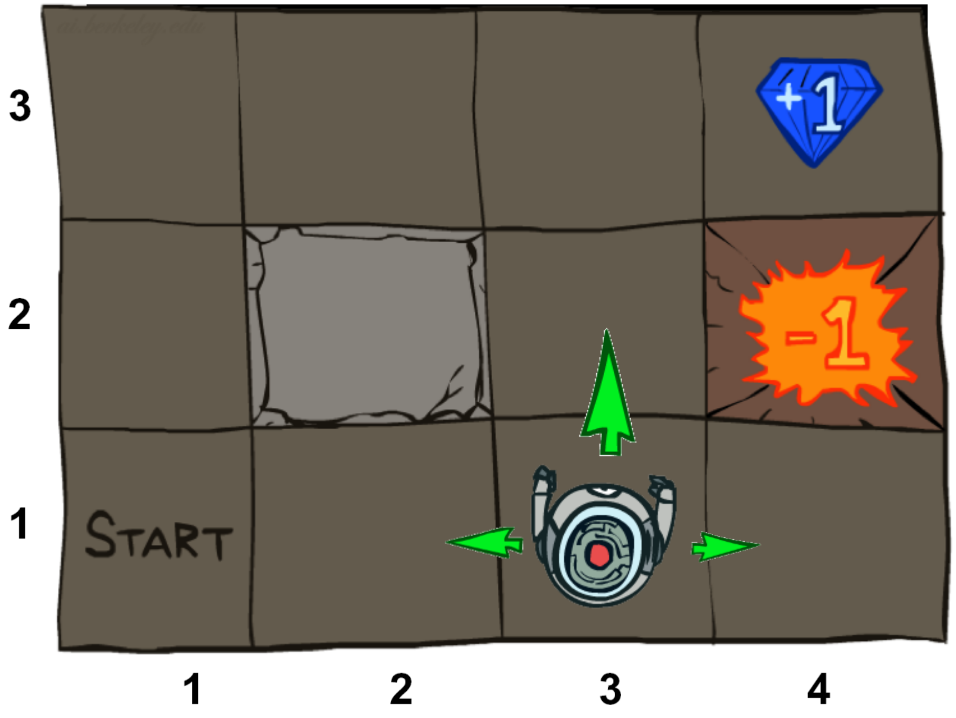
Instructor: Daniel Brown

University of Utah

[Based on slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. http://ai.berkeley.edu.]

Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
    - 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 $R(s)$, $R(s,a)$, or $R(s')$
  - A start state
  - Maybe a terminal state



- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We'll have a new tool soon

[Demo – gridworld manual intro (L8D1)]

Other examples of MDPs

- Checkers Boardgame
- Medication treatment

Other examples of MDPs

- Self-driving car
- Language Generation (LLMs)

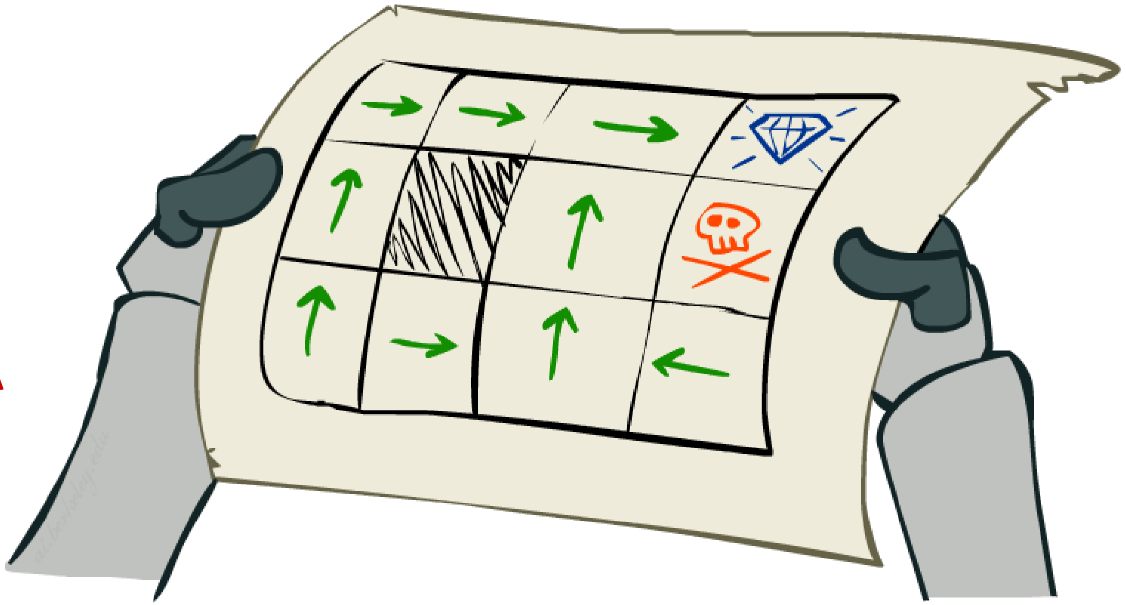
Types of Markov Models

	System state is fully observable	System state is partially observable
System is autonomous	Markov chain	Hidden Markov model (HMM)
System is controlled	Markov decision process (MDP)	Partially observable Markov decision process (POMDP)

Policies

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

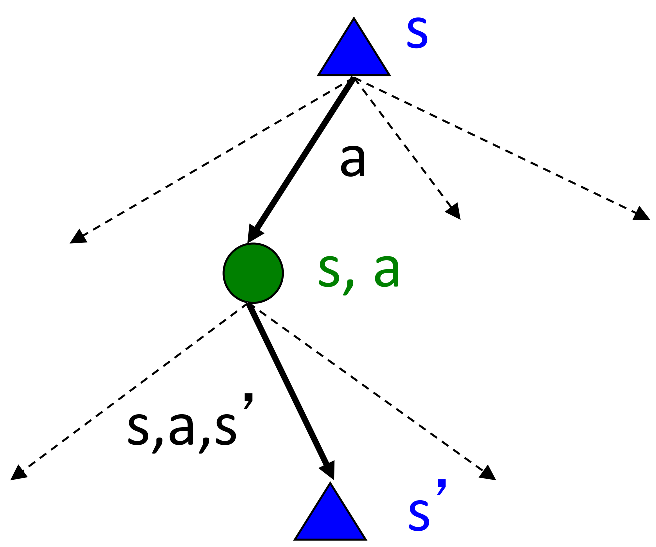
- For MDPs, we want an optimal policy $\pi^*: S \to A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent



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

Defining MDPs

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



- Important MDP quantities:
  - Policy = Choice of action for each state
  - Utility = expected sum of (discounted) rewards = "expected return"

Optimal Quantities

- The value (utility) of a state s:

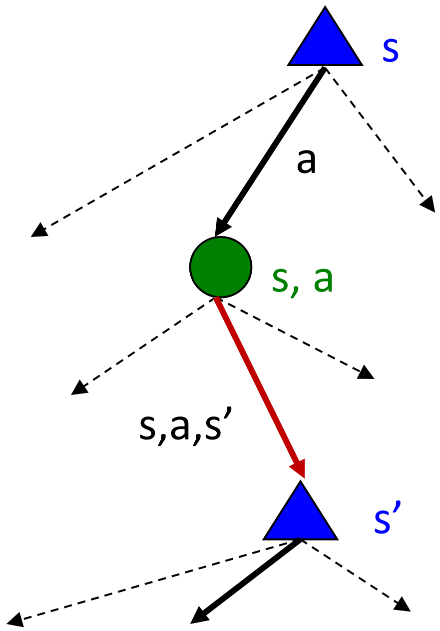
  $V^*(s)$ = expected utility starting in s and acting optimally

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

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

- The optimal policy:

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



s is a *state*

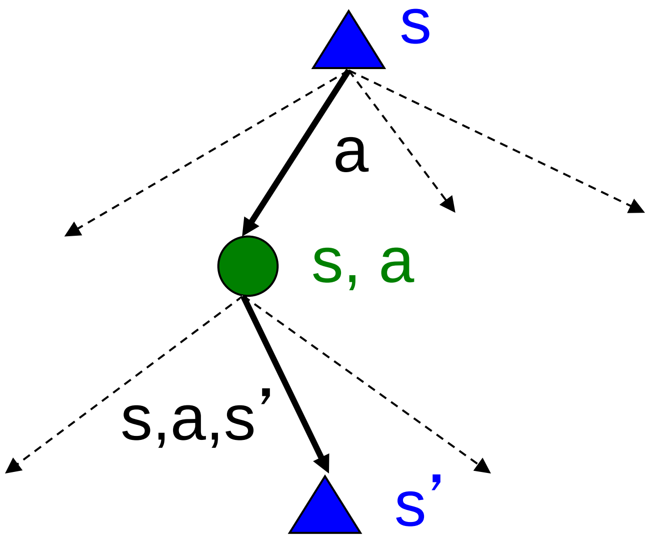
(s, a) is a *q-state*

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

[Demo – gridworld values (L8D4)]

Bellman Equations

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!



- Recursive definition of value:

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

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

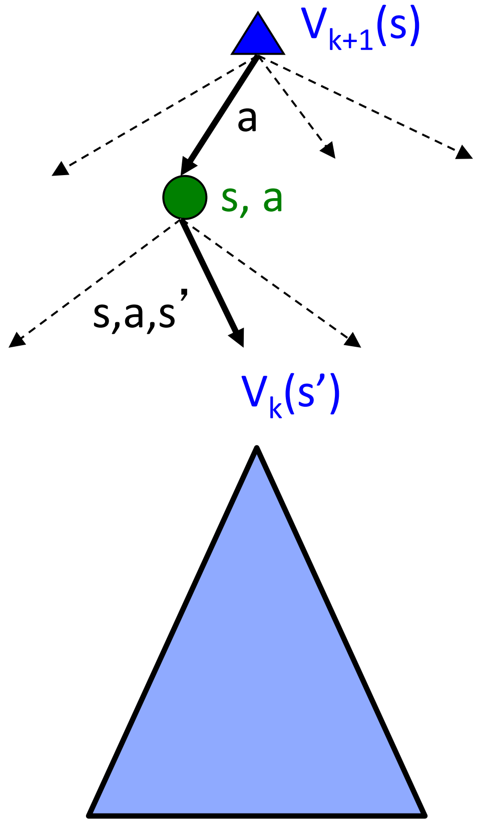
$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

Value Iteration Refresher

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

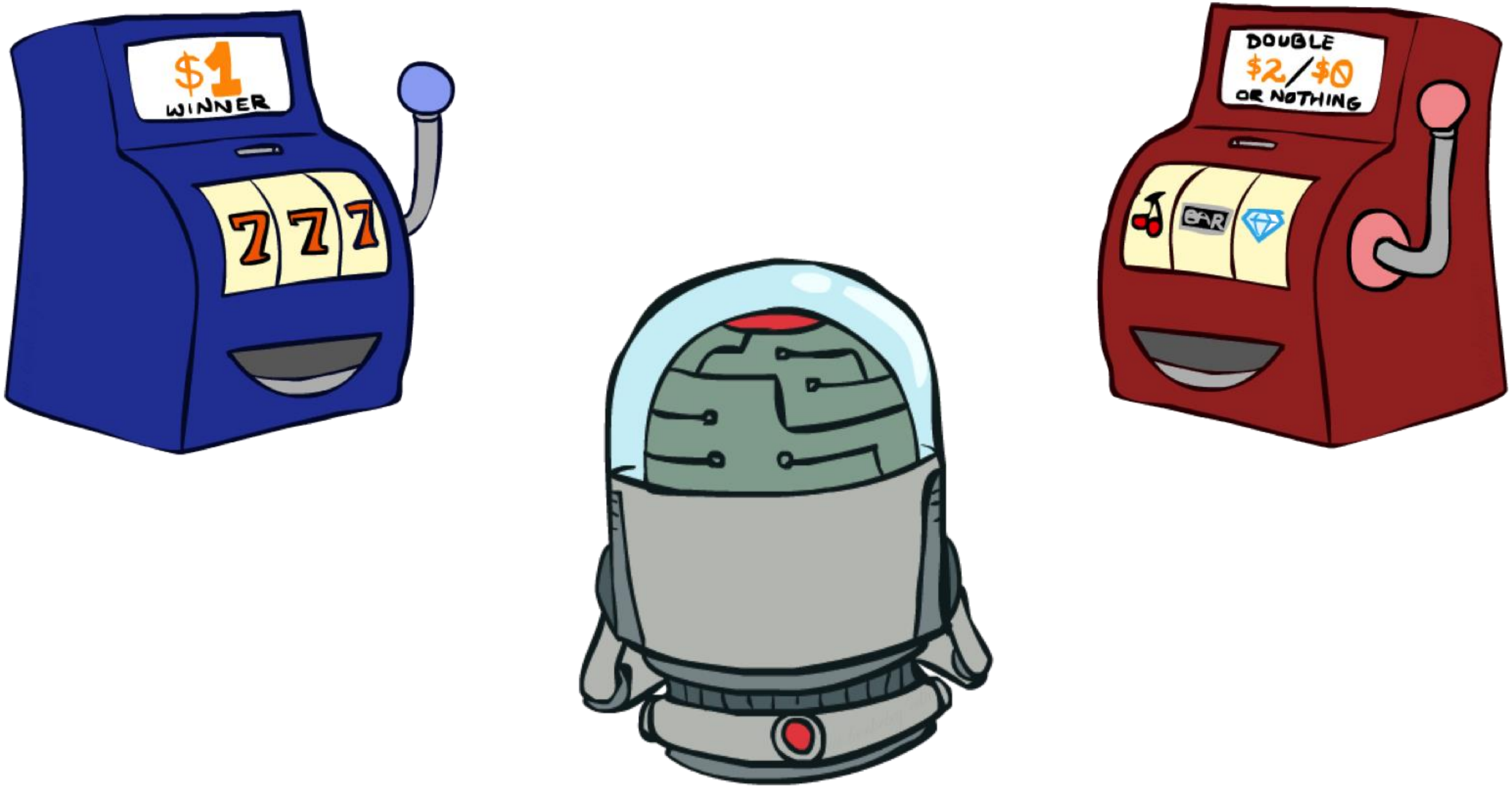
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

Bellman Update Equation



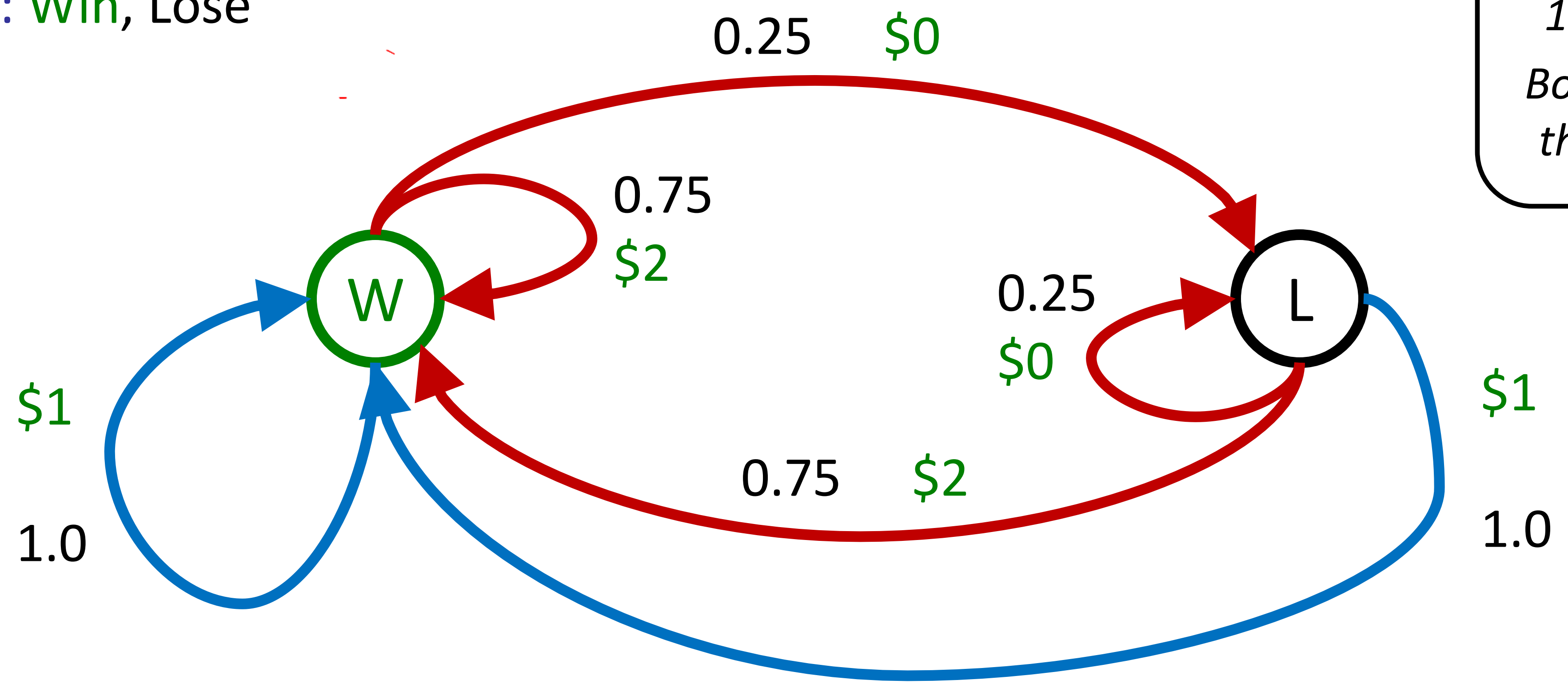
- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Double Bandits



Double-Bandit MDP

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



No discount

100 time steps

Both states have the same value

Offline Planning

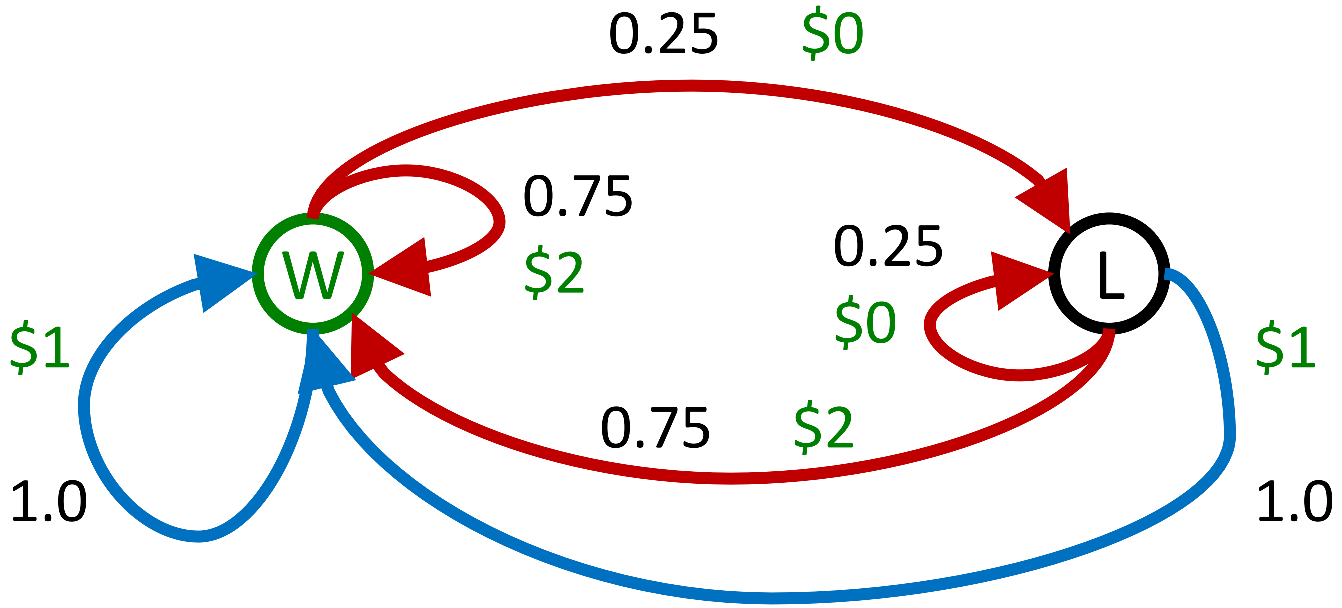
- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

No discount

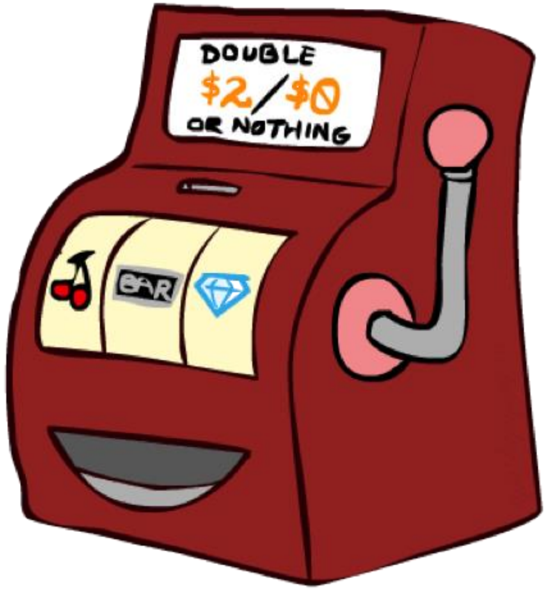
100 time steps

Both states have the same value

	Value
Play Red	150
Play Blue	100



Let's Play!

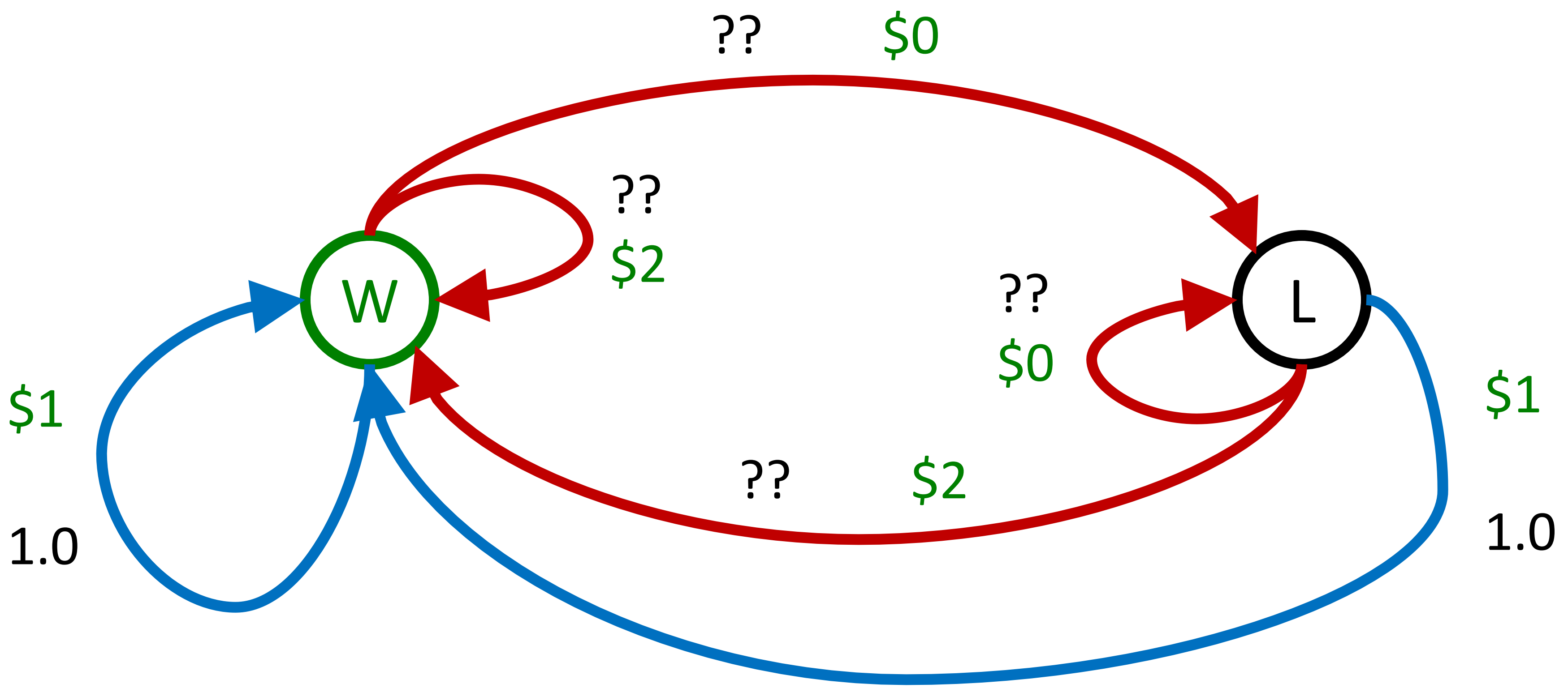


$2 $2 $0 $2 $2

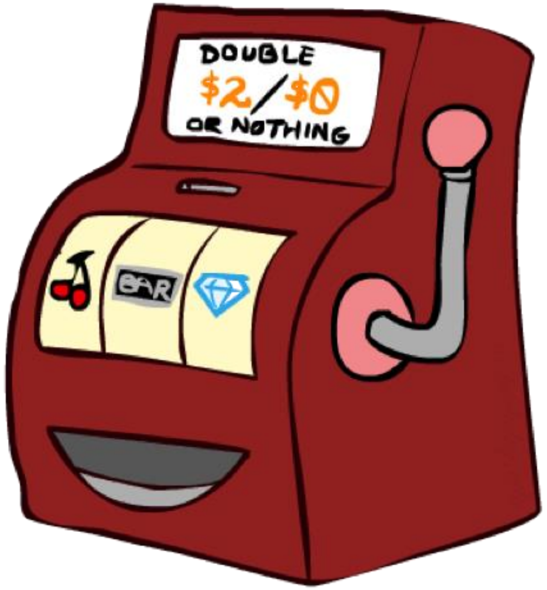
$2 $2 $0 $0 $0

Online Planning

- Rules changed! Red's win chance is different.



Let’s Play!



\$0 \$0 \$0 \$2 \$0

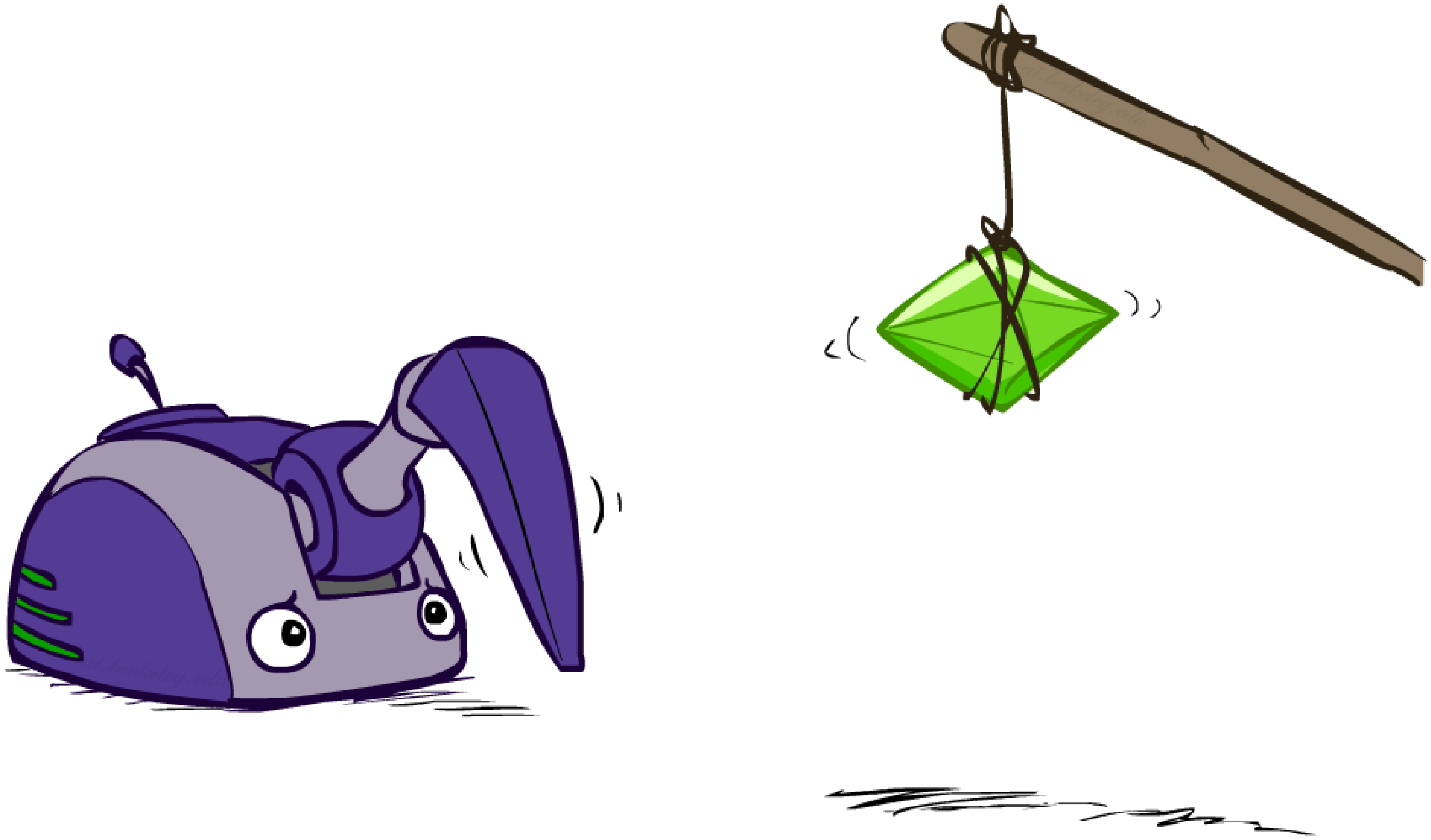
\$2 \$0 \$0 \$0 \$0

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



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

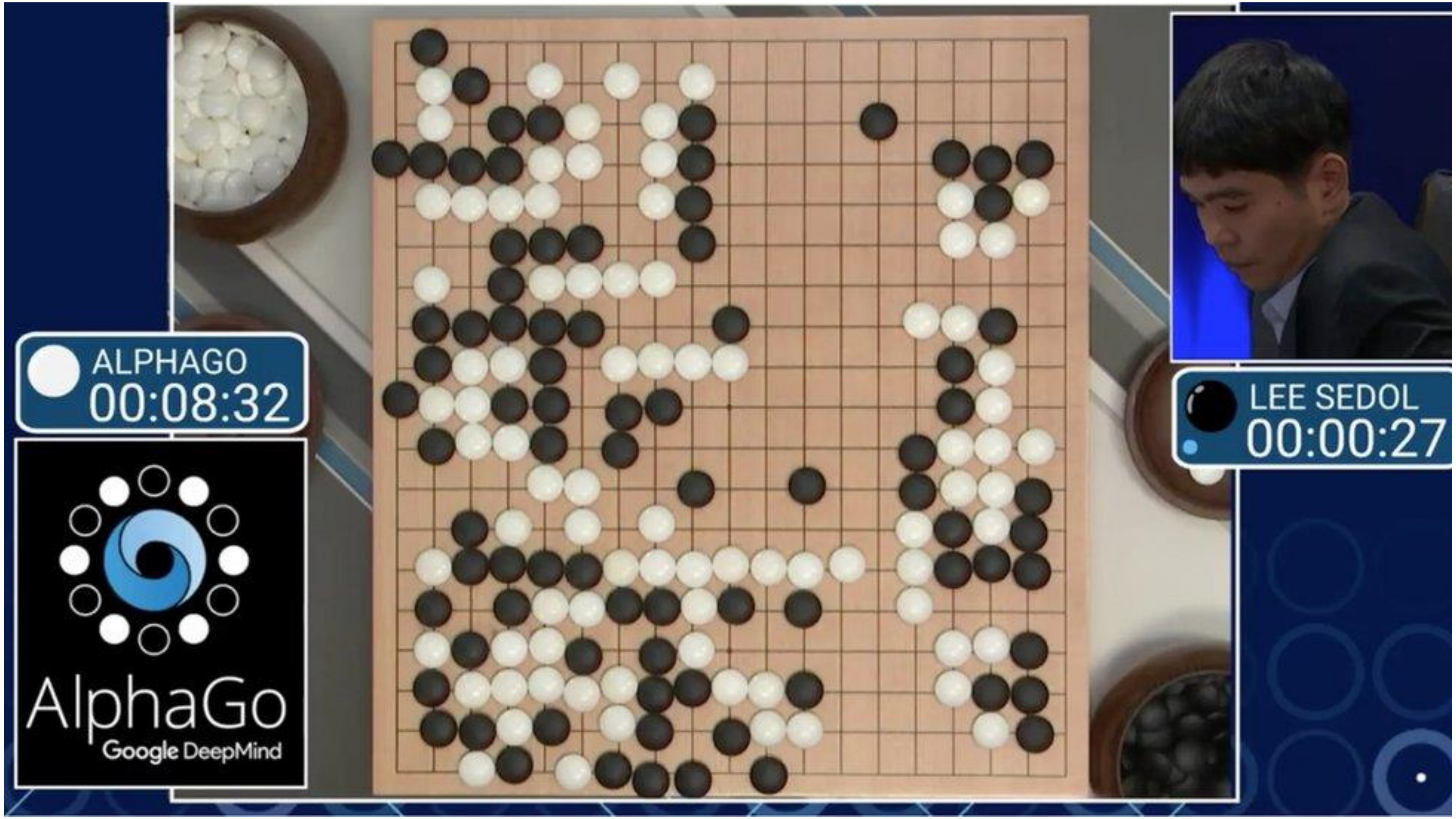
Finished

[Kohl and Stone, ICRA 2004]

[Video: AIBO WALK – finished]

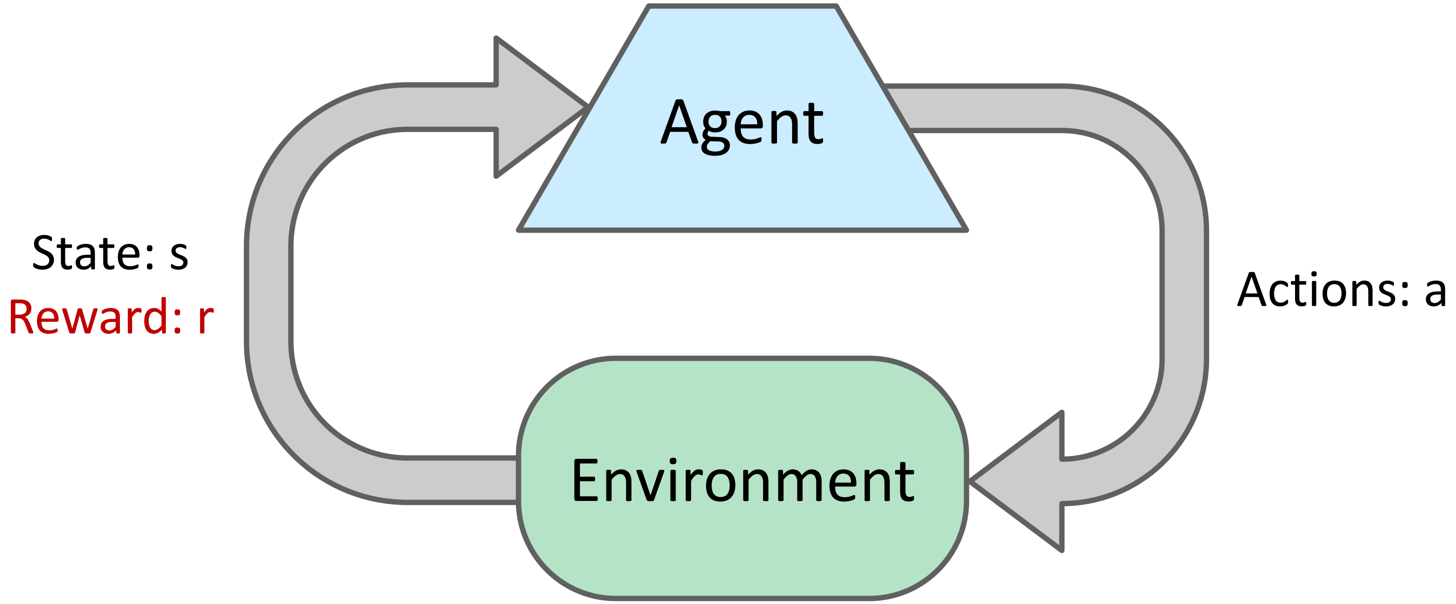
https://vision-locomotion.github.io/


021 3 1


ALPHAGO
00:08:32
LEE SEDOL
00:00:27
AlphaGo
Google DeepMind

ChatGPT

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!

Why Reinforcement Learning?

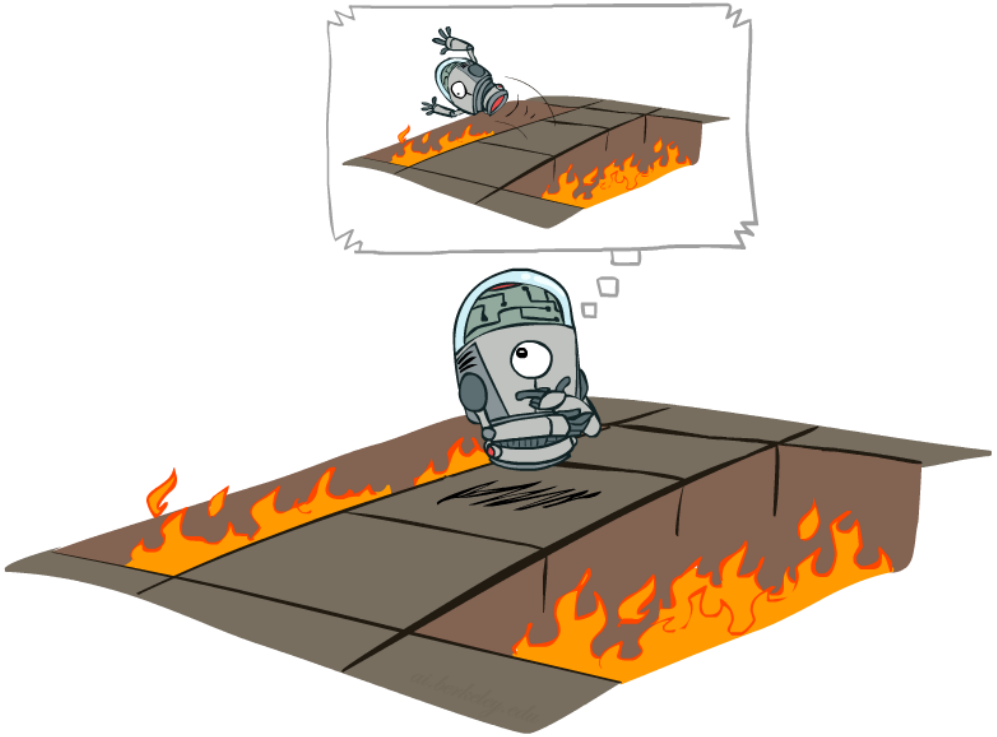
- Takes inspiration from nature
- Often easier to encode a task as a sparse reward (e.g. recognize if goal is achieved) but hard to hand-code how to act so reward is maximized (e.g. Go)
- General purpose AI framework

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

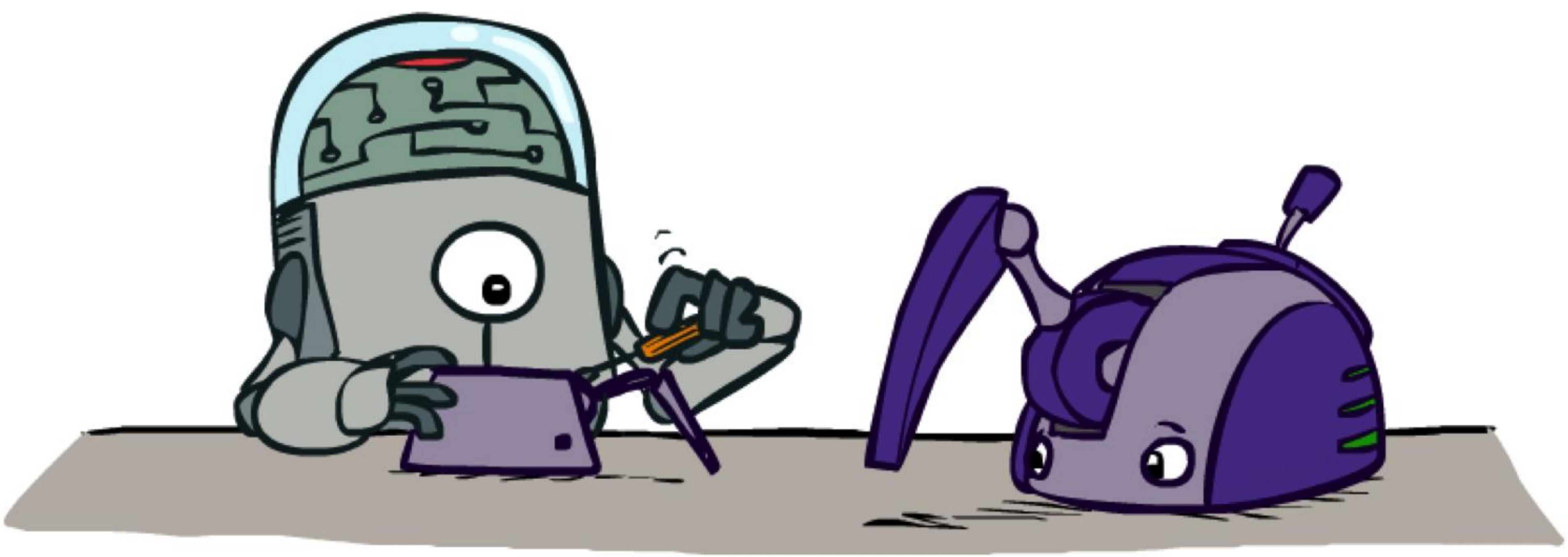
Offline (MDPs) vs. Online (RL)



Offline Solution

Online Learning

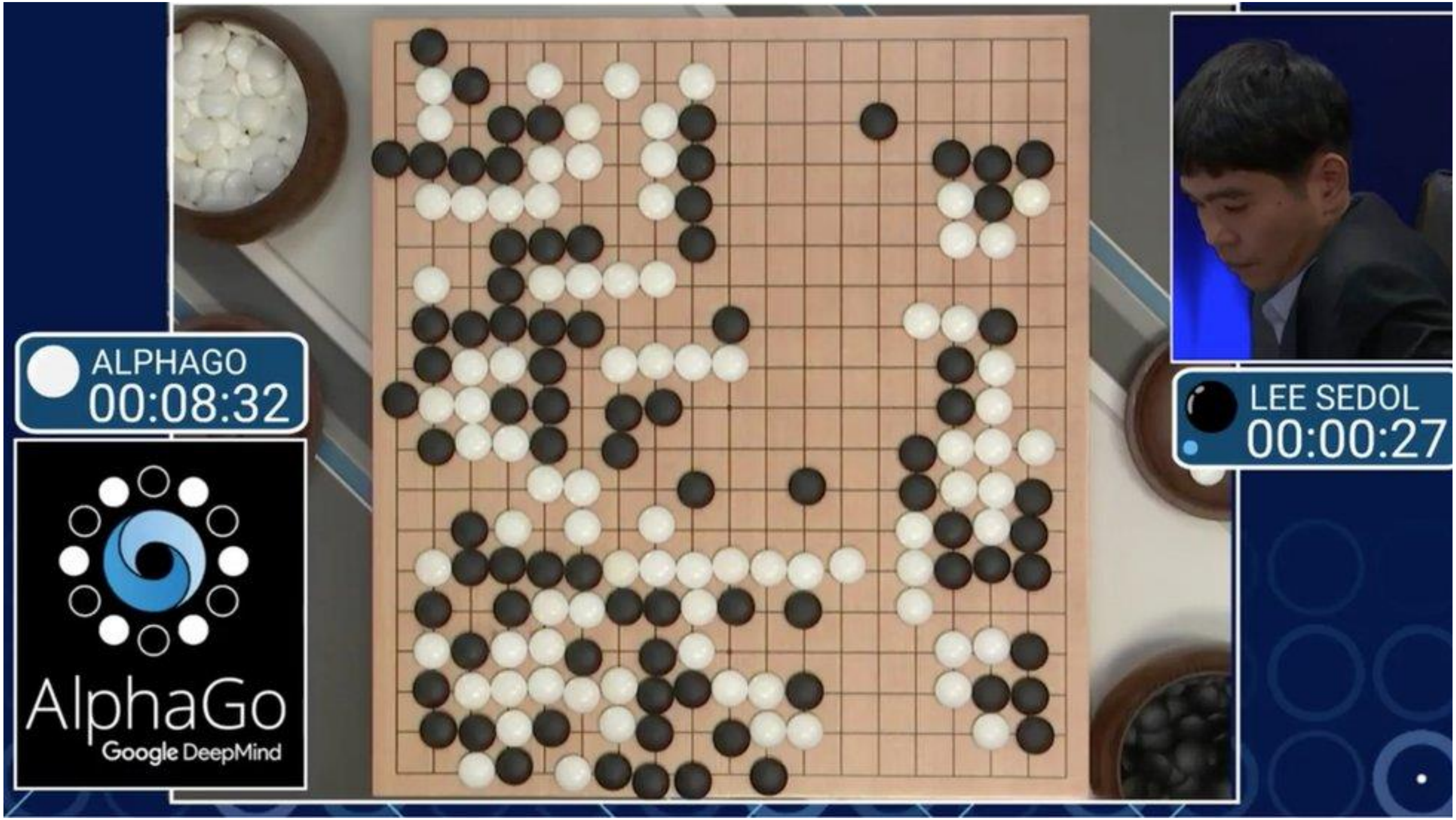
Model-Based Learning



Simple View of Model-Based RL

- 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 iteration, as before

Sometimes Model of World is Known



Deep RL Makes a Big Splash!

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature › letters › article

Published: 25 February 2015

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

TC

Login

Search

TechCrunch+
Startups
Venture
Security
AI
Crypto
Apps
Events
Startup Battlefield
More

Startups

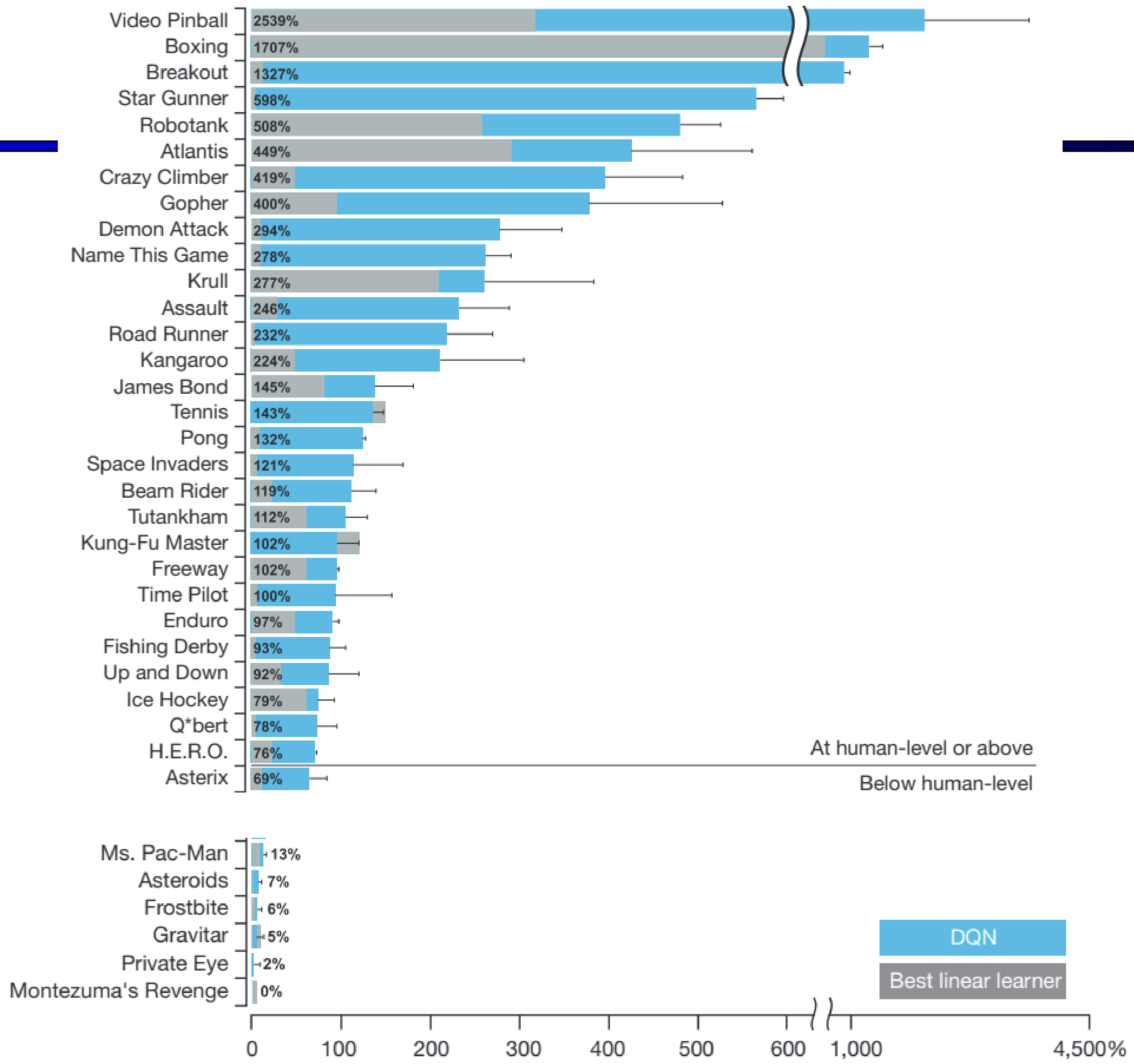
Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

Catherine Shu @catherineshu / 6:20 PM MST • January 26, 2014

Comment

The Arcade Learning Environment

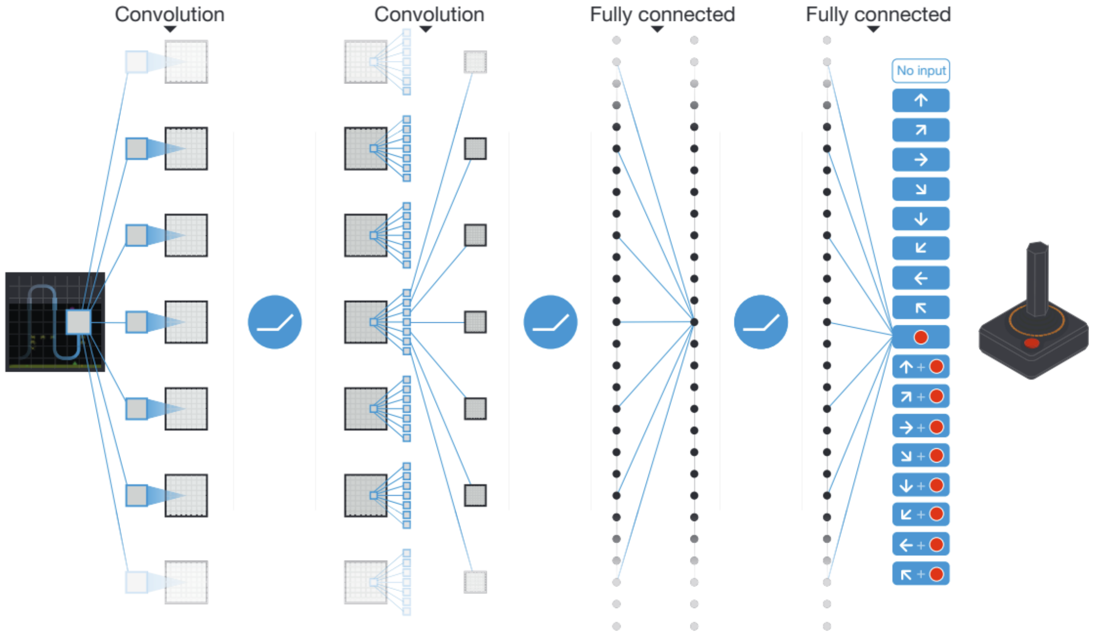

021 3 1

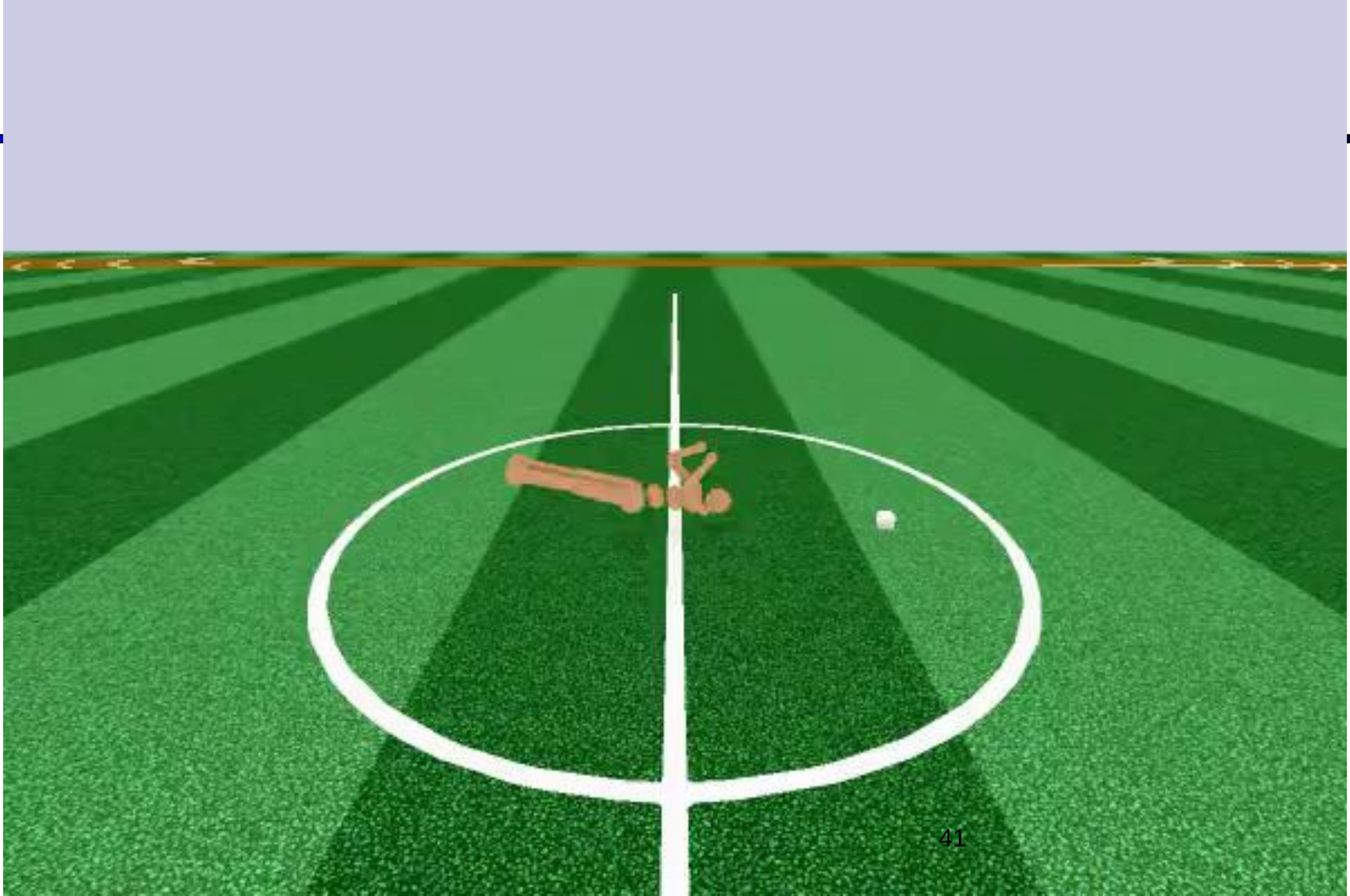


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
At human-level or above
Below human-level
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

DQN only works for discrete action spaces

- Next Time: How to deal with continuous action spaces






O
A

When might RL be a good tool for your problem?

When might RL be a good tool for your problem?

- Is your problem a sequential decision making problem?
- Are there “actions” that effect the next “state”?
- Do you know the rules of these effects?
- Can you write down a clear objective/score/reward/cost?
- Do you have a simulator?
- Lots of examples of sequences of decisions and their long-term consequences?
- Is it unclear what to do in each state? Exploration required?
- Are you looking for unique/creative/super-human solutions?

When might RL not be a good tool?

When might RL not be a good tool?

- Single step or static problem
- No clear reward signal.
- Reward signal is unavailable or very hard to write down.
- Well-known model of the environment.
- Deterministic environment
- Low-tolerance for exploration and trial and error
- No need for adaptive or novel solutions. The goal is to perform the task in a very predictable way.