Fundamentals of Reinforcement Learning

Ann Nowé and Diederik M. Roijers

AI laboratory
Vrije Universiteit Brussel

Dept. of Computer Science
Vrije Universiteit Amsterdam

June 19, 2018

Schedule

- Intro Agents, Planning, and Learning
- Markov decision processes (MDPs)
- Planning methods: dynamic programming
- Reinforcement Learning
  - Actor-critic
  - Q-Learning
  - SARSA
  - Model-based learning
- Advanced topics

Note

- These lectures are based on:
  - Sutton and Barto
  - Papers
  - Some of our own work

Images from `http://www.irasutoya.com`, Sutton and Barto's book, and Diederik's PhD thesis.

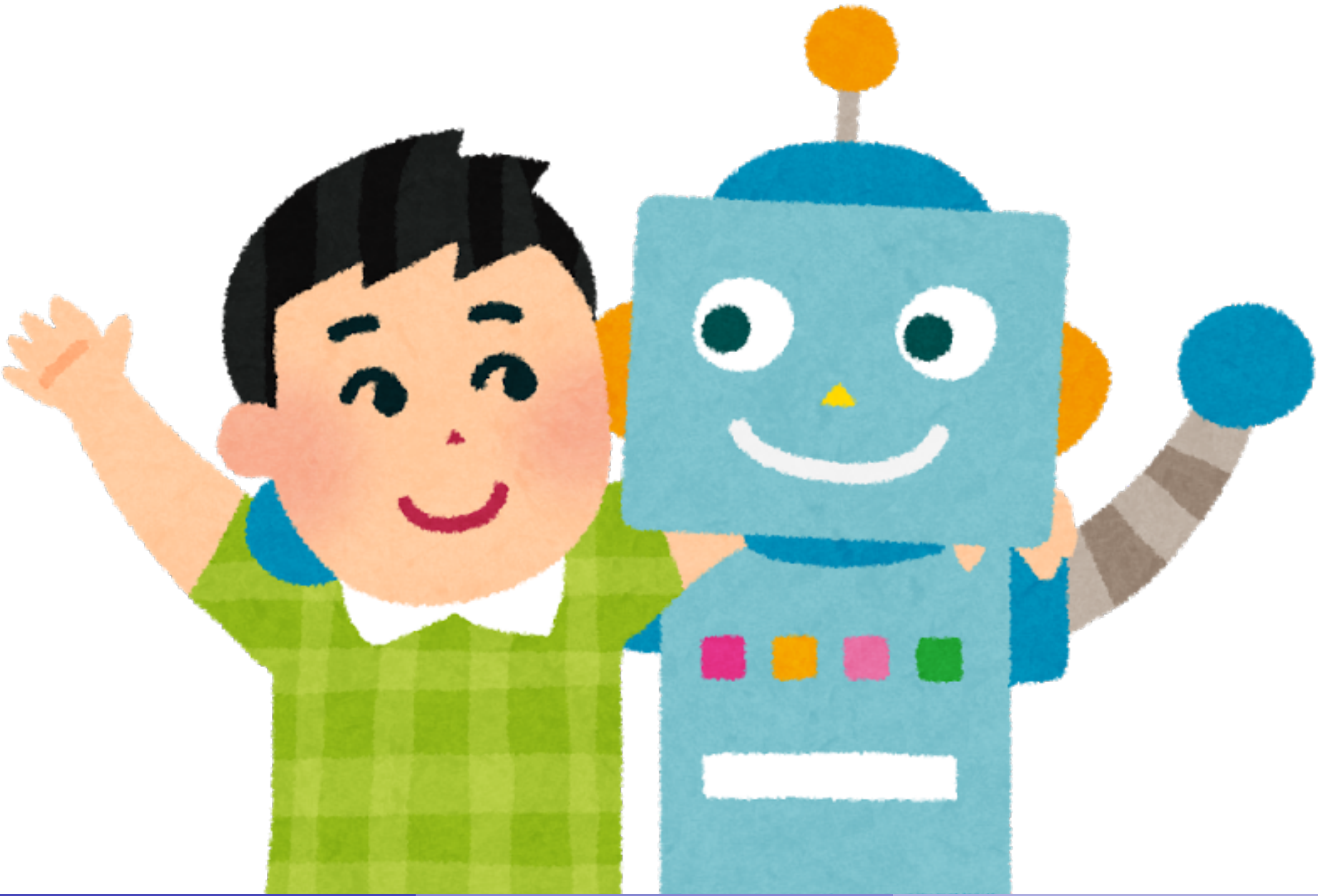
Planning and learning

- Agents
- How should a single *rational* agent interact with a *sequential decision process* to maximise its expected long-term *cumulative reward* with/without an a priori model of its environment?
- With a model: planning (part 1)
- Without a model: reinforcement learning (part 2)

Why planning and learning?

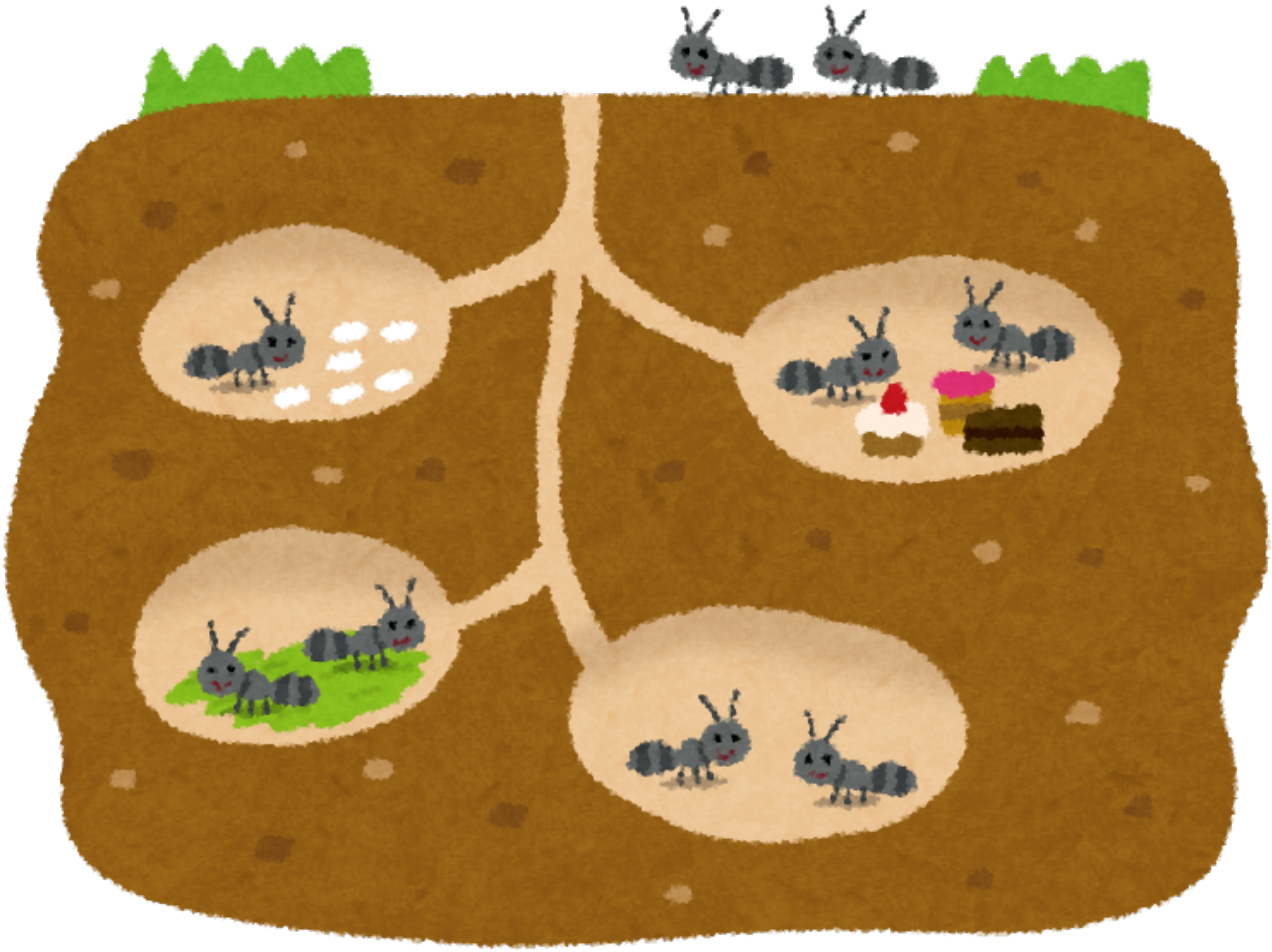
- It's hot!
- It's cool!
- It's really really useful!

What is an agent?



What is an agent?

What is an agent?



What is an agent?

- An agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors" (Russel and Norvig)

- An artificial agent typically is a computer program
  - possibly embedded in specific hardware
  - *takes actions* in an environment that changes as a result of these actions.

- An autonomous agent (Franklin and Graesser, 1996)
  - can act autonomously,
  - on a user's behalf

Intelligent Autonomous Agents

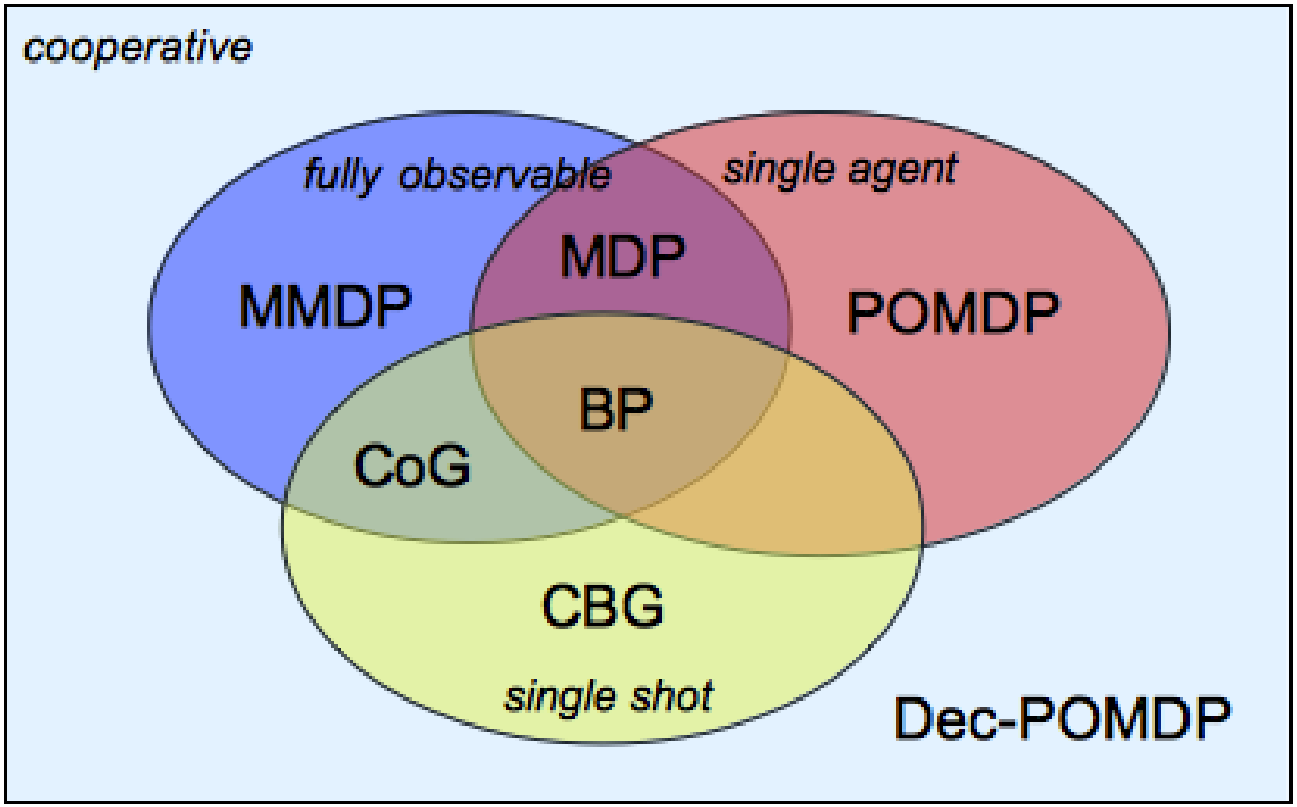
- Intelligent Autonomous Agents that
  - reason about their environment
  - reason about consequences of actions (desirability)
- Decision theory

Environments

- States
- Actions
- State transitions
- Rewards

- Sequential
- Single-agent
- Fully observable

Other models



Schedule

- Intro Agents, Planning, and Learning
- *Markov decision processes (MDPs)*
- Planning methods: dynamic programming
- Reinforcement Learning
  - Actor-critic
  - Q-Learning
  - SARSA
  - Model-based learning
- Advanced topics

Markov decision processes

- Formalisation of a single-agent, sequential, discrete-time, stationary environment, Markovian-observation-signal, decision problem.
- Sequential: multiple decisions over time.
- Discrete time steps: one action and state-transition per timestep
- Stationary environment: the state of the environment may change, but the dynamics do not.

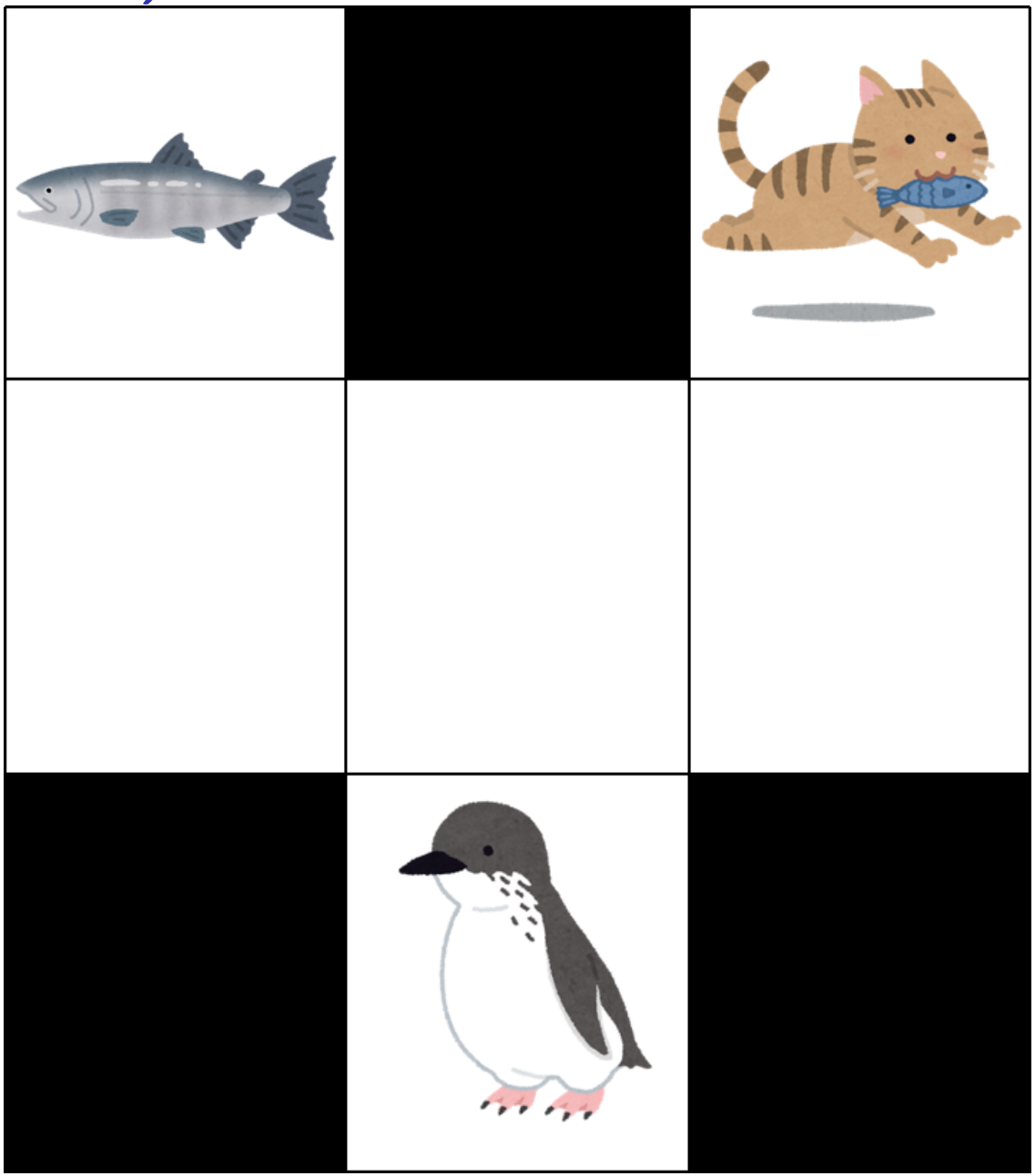
Markov decision processes

- Named after Andrey Markov (1856–1922).
- A finite MDP consists of:
  - Discrete time $t = 0, 1, 2, \ldots$
  - A discrete set of states $s \in S$
  - A discrete set of actions $a \in A(s)$ for each s
  - A transition function $P^a_{ss'} = p(s'|s, a)$: probability of transitioning to state s' when taking action a at state s
  - A reward function $R^a_{ss'} = E[r|s, a, s']$: expected reward when taking action a at state s and transitioning to s'
  - A planning horizon h or discount factor γ

Markov decision processes

- Named after Andrey Markov (1856–1922).
- A finite MDP consists of:
  - Discrete time $t = 0, 1, 2, \ldots$
  - A discrete set of states $s \in S$
  - A discrete set of actions $a \in A(s)$ for each s
  - A transition function $T(s, a, s') = p(s'|s, a)$: probability of transitioning to state s' when taking action a at state s
  - A reward function $R(s, a, s') = E[r|s, a, s']$: expected reward when taking action a at state s and transitioning to s'
  - A planning horizon h or discount factor γ

A tiny (6-state) MDP



The Markov property

$$p(s_{t+1}, r_{t+1}|s_t, a_t) = p(s_{t+1}, r_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0)$$

- Therefore, the history does not yield more information about subsequent states and rewards than the current state.
- Current state is a sufficient statistic for the history.
- A Markovian/Markov state/observation signal
- Important: only need to condition on the latest observation (s_t)

Markov?

Markov?

The credit-assignment problem

- Sequential aspect $\rightarrow$ credit assignment problem
- Suppose an agent takes a long sequence of actions, at the end of which it receives a single positive reward?
- How can it determine to what degree each action in that sequence deserves the *credit* for the reward?

Return

- The agent's goal is to maximise the expected *return*, the sum over the rewards received.
- Three settings:
  - Finite-horizon
  - Infinite-horizon continuing
  - Infinite-horizon episodic

Return: finite-horizon

- In a finite-horizon task, the return is defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{h} \gamma^k r_{t+k+1}$$

- $0 \leq \gamma \leq 1$; can be 1 only in a finite-horizon setting!

Return: infinite-horizon continuing

- In an infinite-horizon task, the return is defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- $0 \leq \gamma < 1$

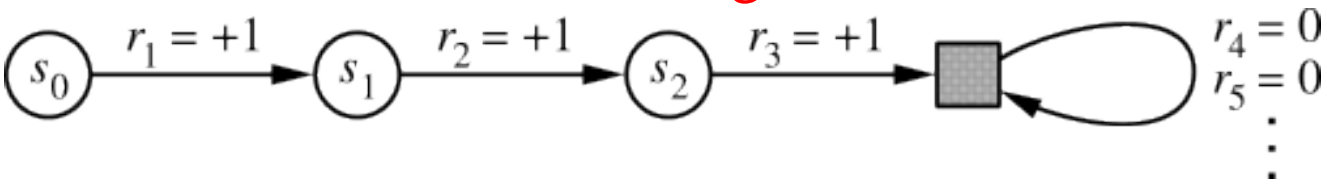
- It never stops

Return: infinite-horizon episodic

- In an infinite-horizon task, the return is defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- $0 \leq \gamma < 1$
- When the agent reaches a terminal state, it stops
- Return after reaching a terminal state (i.e., all rewards) is 0 by definition.
- Terminal state is an *absorbing state*:



Value functions

- The *state-value function* of a policy π is:

$$V^{\pi}(s) = E_{\pi}\Big[R_t|s_t = s\Big] = E_{\pi}\Big[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\Big]$$

- The *stateless value function* of a policy π is:

$$V^{\pi} = E_{\pi}\Big[R_t|\mu_0\Big] = E_{\pi}\Big[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|\mu_0\Big]$$

where μ_0 is a distribution over initial states.

- The *state-action-value* of a policy π is:

$$Q^{\pi}(s,a) = E_{\pi}\Big[R_t|s_t = s, a_t = a\Big] = E_{\pi}\Big[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\Big]$$

Bellman equation

- The definition of V^π can be rewritten recursively by making use of the transition model, yielding the *Bellman equation*:

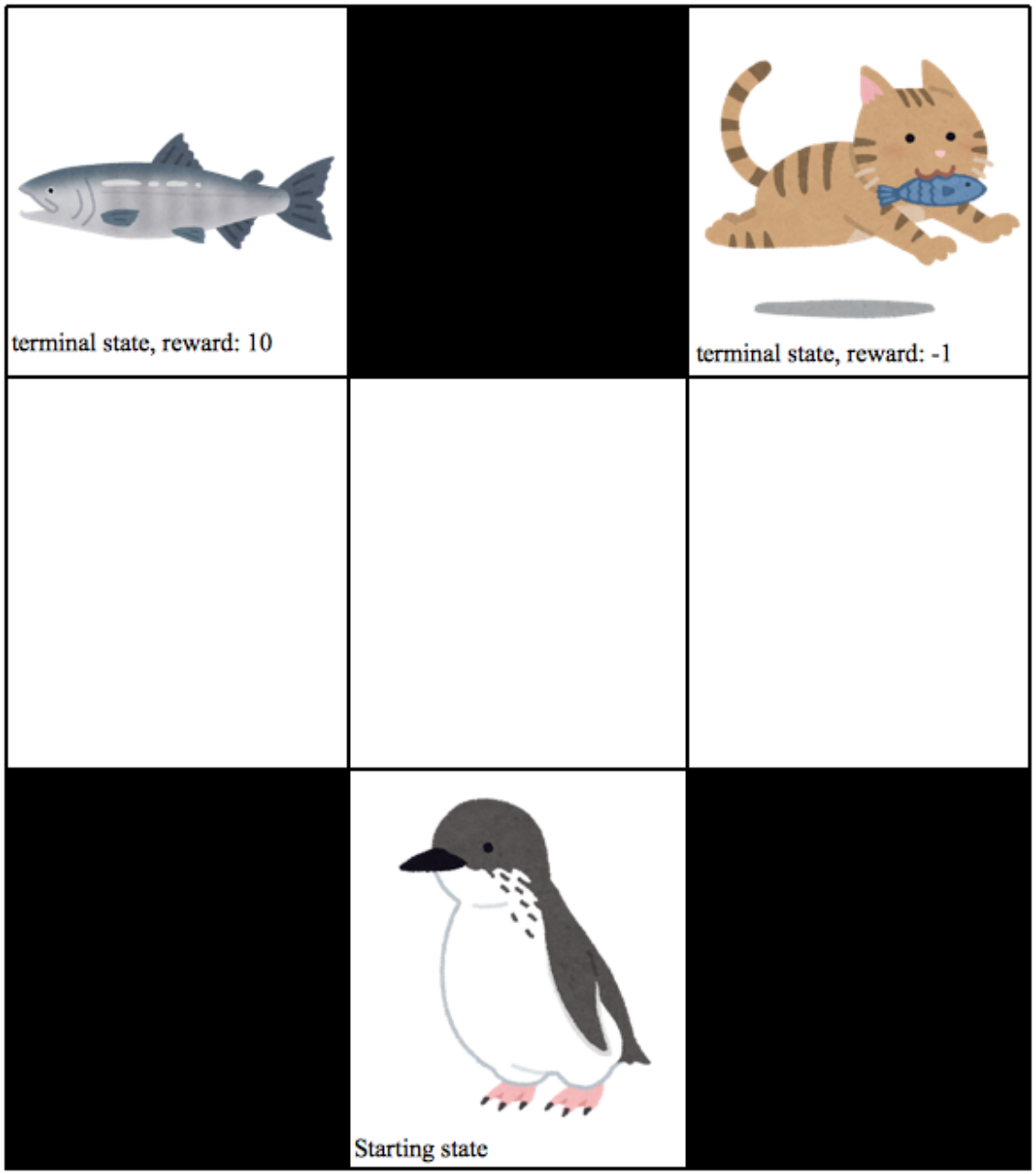
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

- This is a set of linear equations, one for each state, the solution of which defines the value of π
- A similar recursive definition holds for Q-values:

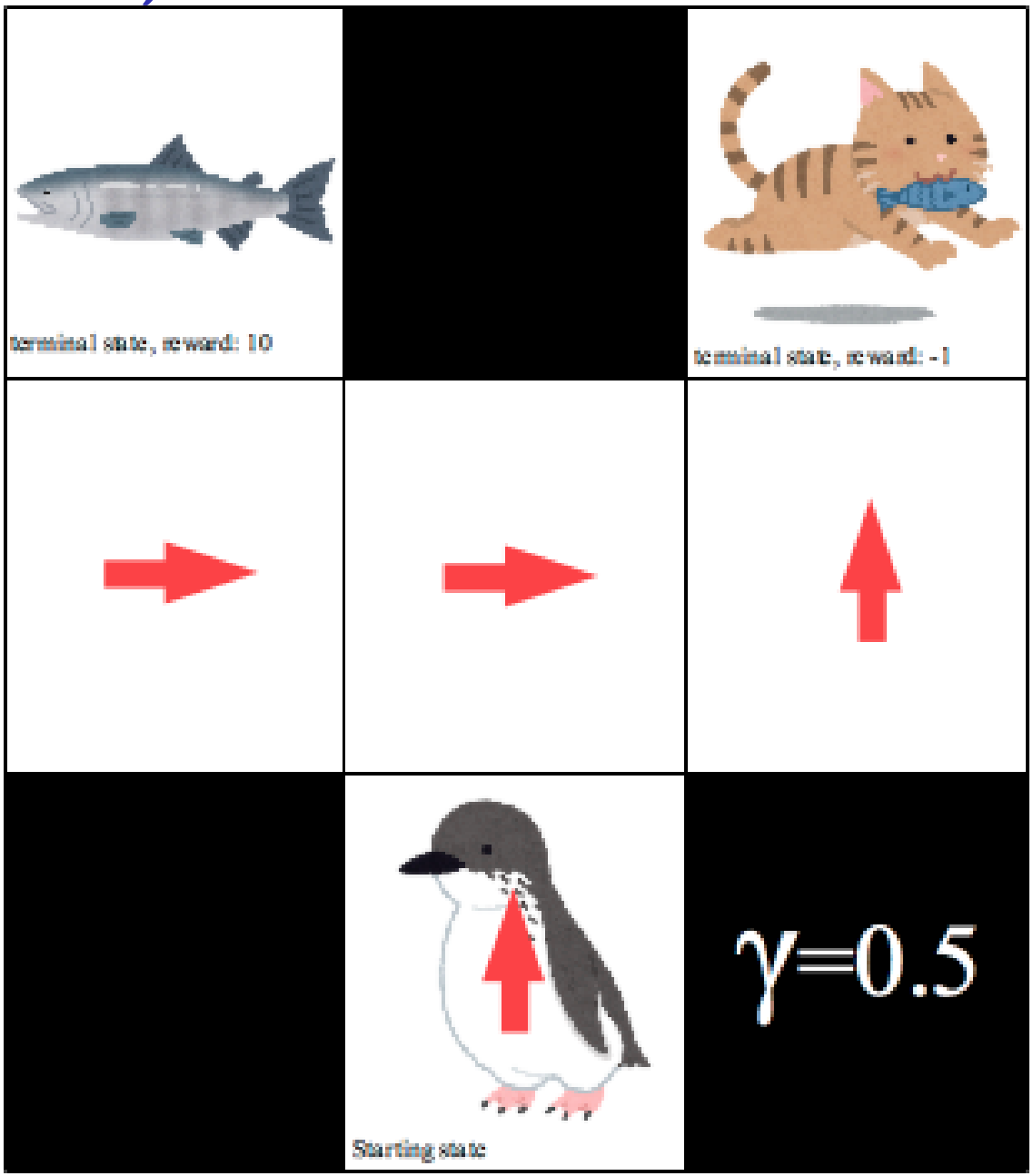
$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

- Equations named after Richard E. Bellman (1920–1984).

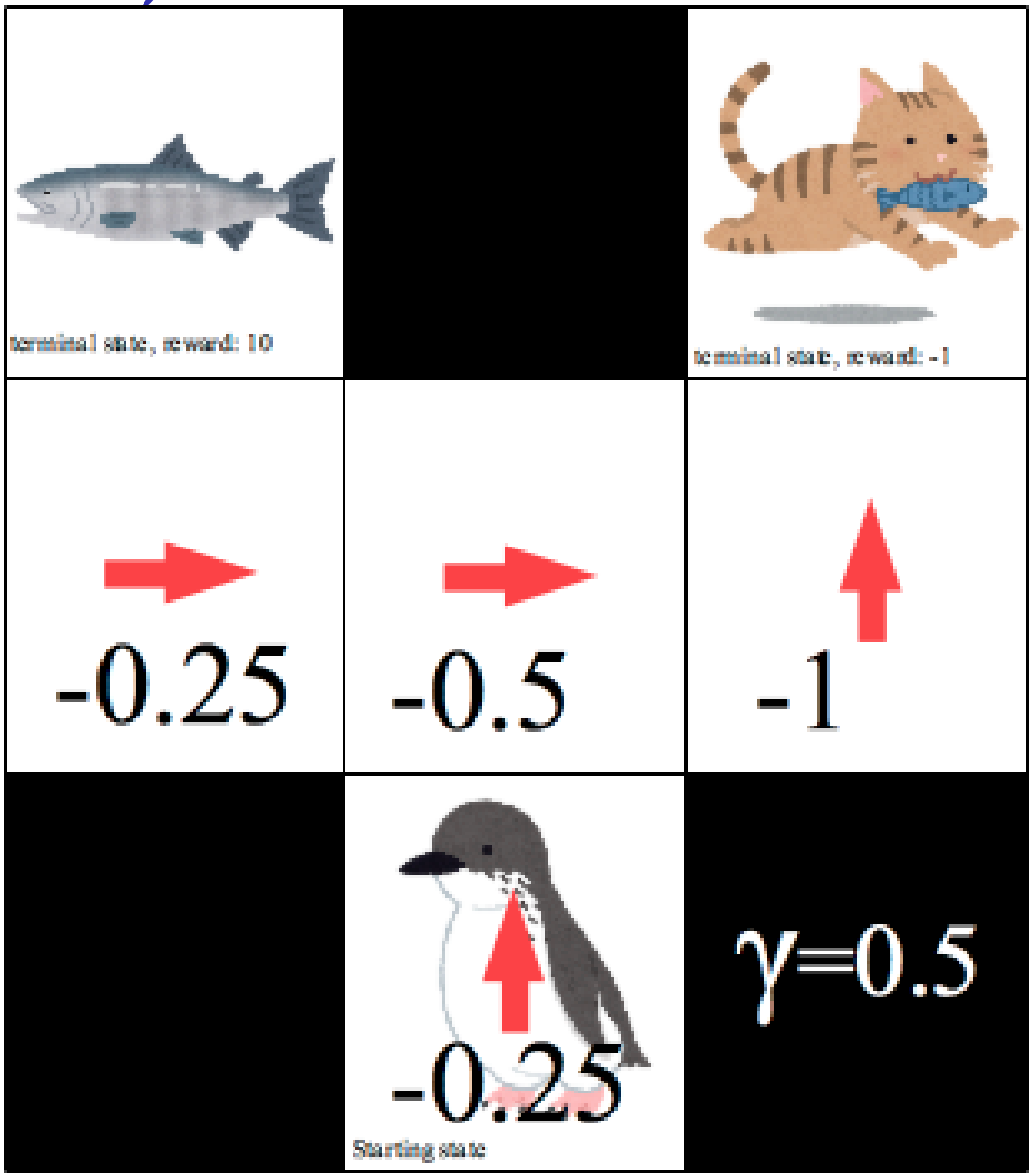
A tiny (6-state) MDP



A tiny (6-state) MDP



A tiny (6-state) MDP



Bellman optimality equations

$$V^*(s) = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right]$$

$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a' \in A} Q^*(s',a') \right]$$

Why optimal value functions are useful

An optimal policy is *greedy* with respect to V^* or Q^*:

$$\pi^*(s) \in \arg\max_a Q^*(s,a) = \arg\max_a \left[R^a_{ss'} + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right]$$

Schedule

- Intro Agents, Planning, and Learning
- Markov decision processes (MDPs)
- *Planning methods: dynamic programming*
- Reinforcement Learning
  - Actor-critic
  - Q-Learning
  - SARSA
  - Model-based learning
- Advanced topics

Planning

- Given an MDP, find $V^*(s)/Q^*(s,a)$ and π^*

$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a \in A} Q^*(s',a') \right]$$

$$\pi^*(s) \in \arg\max_a Q^*(s,a) = \arg\max_a \left[R^a_{ss'} + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right]$$

- Howard (1960): for any additive infinite-horizon MDP, there exists at least one deterministic stationary policy that is optimal.

- $\pi : S \to A$

Planning

- Given an MDP, find $V^*(s)/Q^*(s,a)$ and π^*

$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a \in A} Q^*(s',a') \right]$$

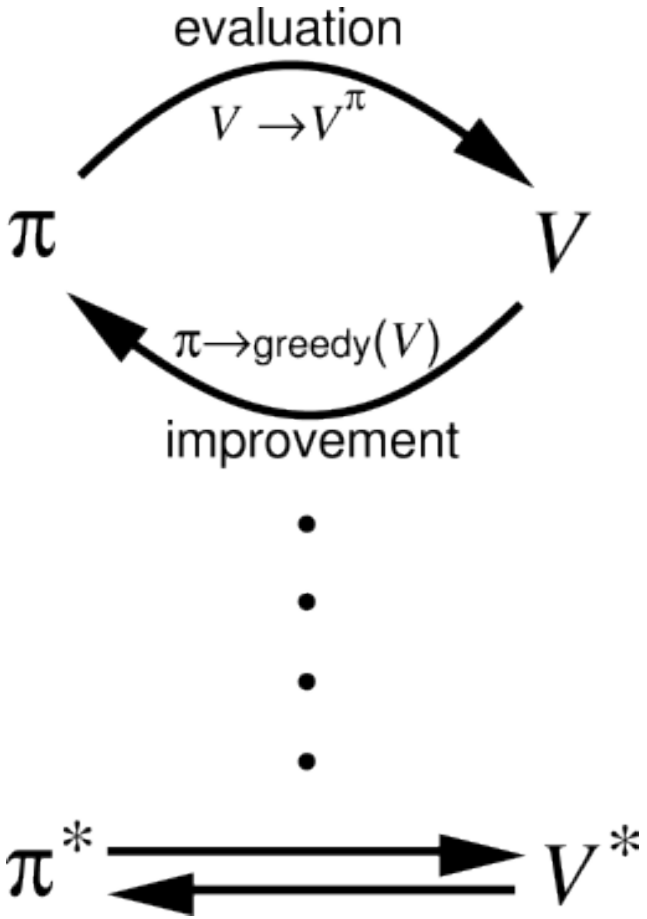
$$\pi^*(s) \in \arg\max_a Q^*(s,a) = \arg\max_a \left[R^a_{ss'} + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right]$$

- Howard (1960): for any additive infinite-horizon MDP, there exists at least one deterministic stationary policy that is optimal.

- $\pi : S \rightarrow A$

- Why does Howard's theorem not hold for finite-horizon problems?

How? A dynamic programming approach!



How? A dynamic programming approach!

- Iteratively improve
  - the value estimates
  - the (deterministic stationary) policy
- Until *convergence*

Policy evaluation

- Exploit the recursive nature of the Bellman equation
- Initial value function $V_0(s)$ is chosen arbitrarily (e.g., 0 for every s)
- Turn Bellman equation into Policy evaluation update update rule:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

- Apply to every state in each iteration
- Iterate until fixed point $\lim_{k\to\infty} \forall s : V_k(s) = V_{k+1}(s)$

Policy evaluation

- Proven to converge
- $\lim_{k \to \infty} \forall s : V_k(s) = V_{k+1}(s) = V^{\pi}(s)$
- Upper bound on complexity $O(|S|^3)^{\star}$

*M.L. Littman, T.L. Dean, L.P. Kaelbling — On the complexity of solving Markov decision problems, UAI, 1995

Policy improvement

- Find *improvable states*: s where is a better action $a \neq \pi(s)$

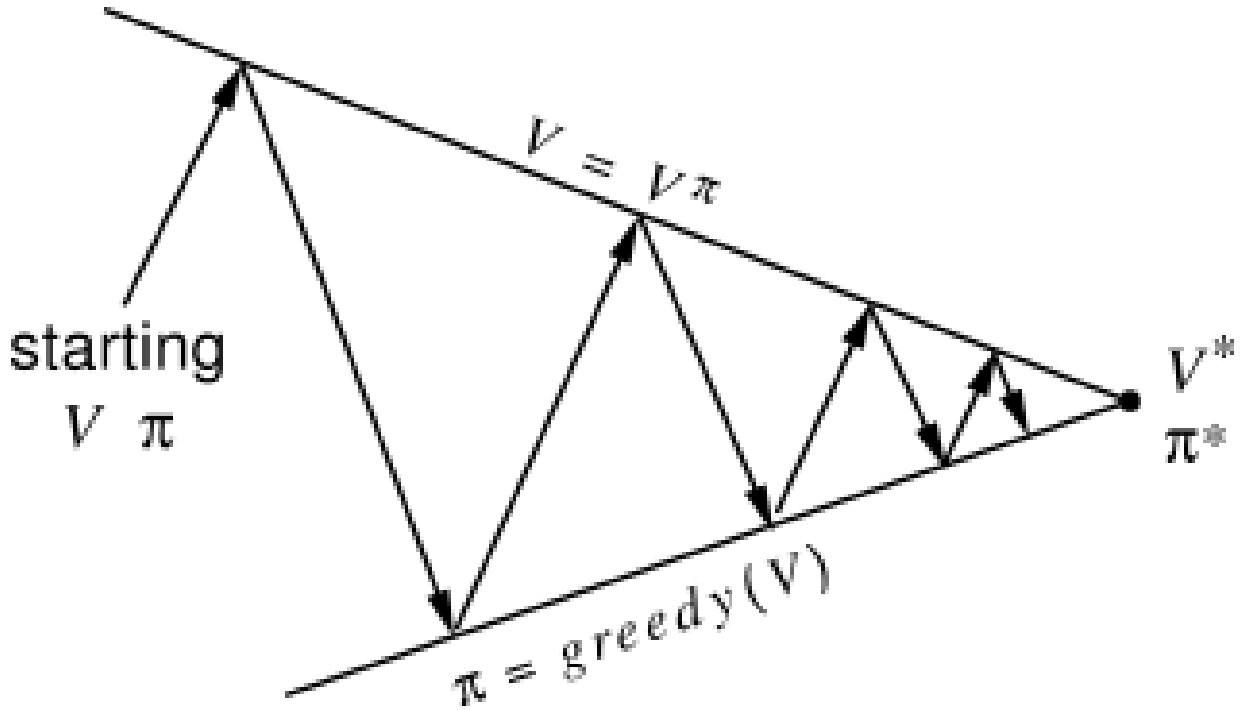
$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right] > V^{\pi}(s)?$$

- *Policy improvement theorem*: changing π to take a better action (according to above equation) in one or more improvable states will increase its value:

$$\forall s \in S : Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \Rightarrow \forall s \in S, V^{\pi'}(s) \geq V^{\pi}(s)$$

- Why does this always converge?

Policy iteration



Policy iteration

- Begin with arbitrary policy
- Repeat:
  - Policy evaluation (PE) (until convergence)
  - Policy improvement (PI) (on one or more states)
- PI
  - On all improvable states $\rightarrow$ Howard's (1960) policy iteration
  - On one improvable state $\rightarrow$ Simple policy iteration
  - Mansour and Singh's (1999) Randomised PI
  - (Recursive) Batch-switching PI [Kalyanakrishnan, Mall, and Goyal (2016), Gupta and Kalyanakrishnan (2017)]

Policy iteration

- Begin with arbitrary policy
- Repeat:
  - Policy evaluation (PE) (until convergence)
  - Policy improvement (PI) (on one or more states)
- PI
  - On all improvable states $\rightarrow$ Howard's (1960) policy iteration
  - On one improvable state $\rightarrow$ Simple policy iteration
  - Mansour and Singh's (1999) Randomised PI
  - (Recursive) Batch-switching PI [Kalyanakrishnan, Mall, and Goyal (2016), Gupta and Kalyanakrishnan (2017)]
  - In practice, Howard's PI seems to be most effective

Value iteration

- We do not have to wait for policy evaluation to complete improving the policy
- Value iteration (VI) integrates evaluation and improvement in one update rule:

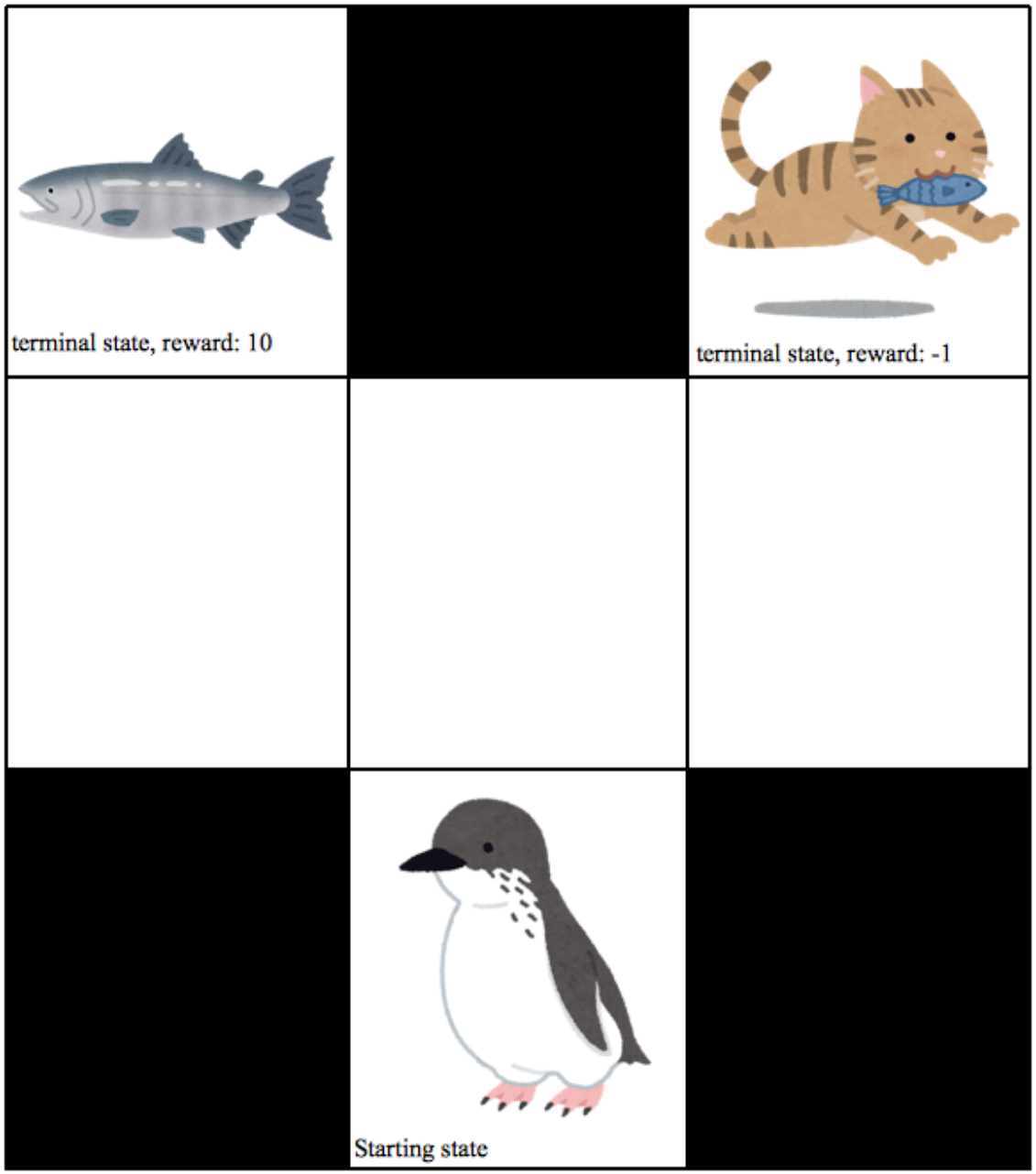
$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

- This can also be written:

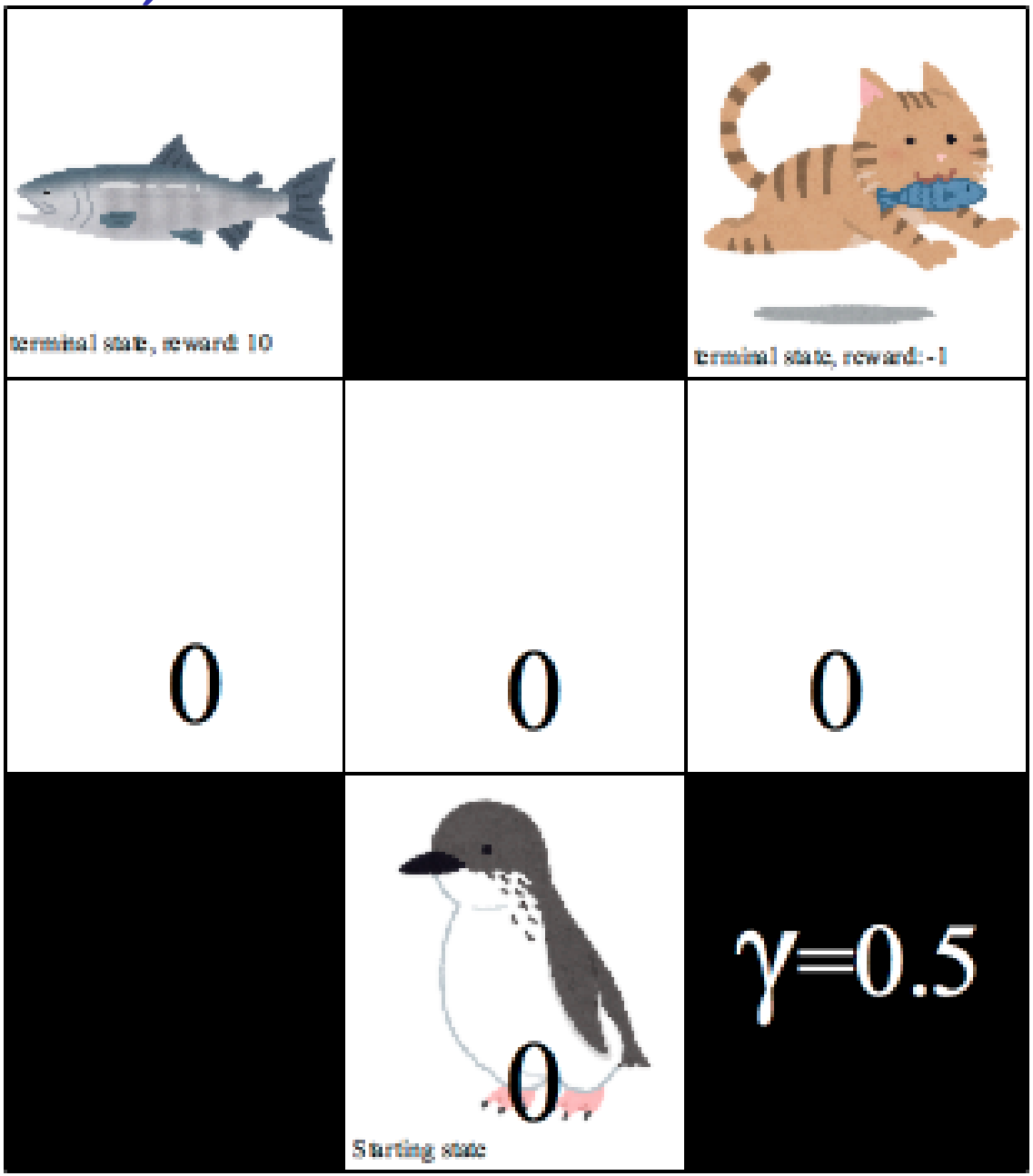
$$V_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a),$$
$$Q_{k+1}(s, a) \leftarrow \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

- Guaranteed to converge to $V^*(s)$ and π^*

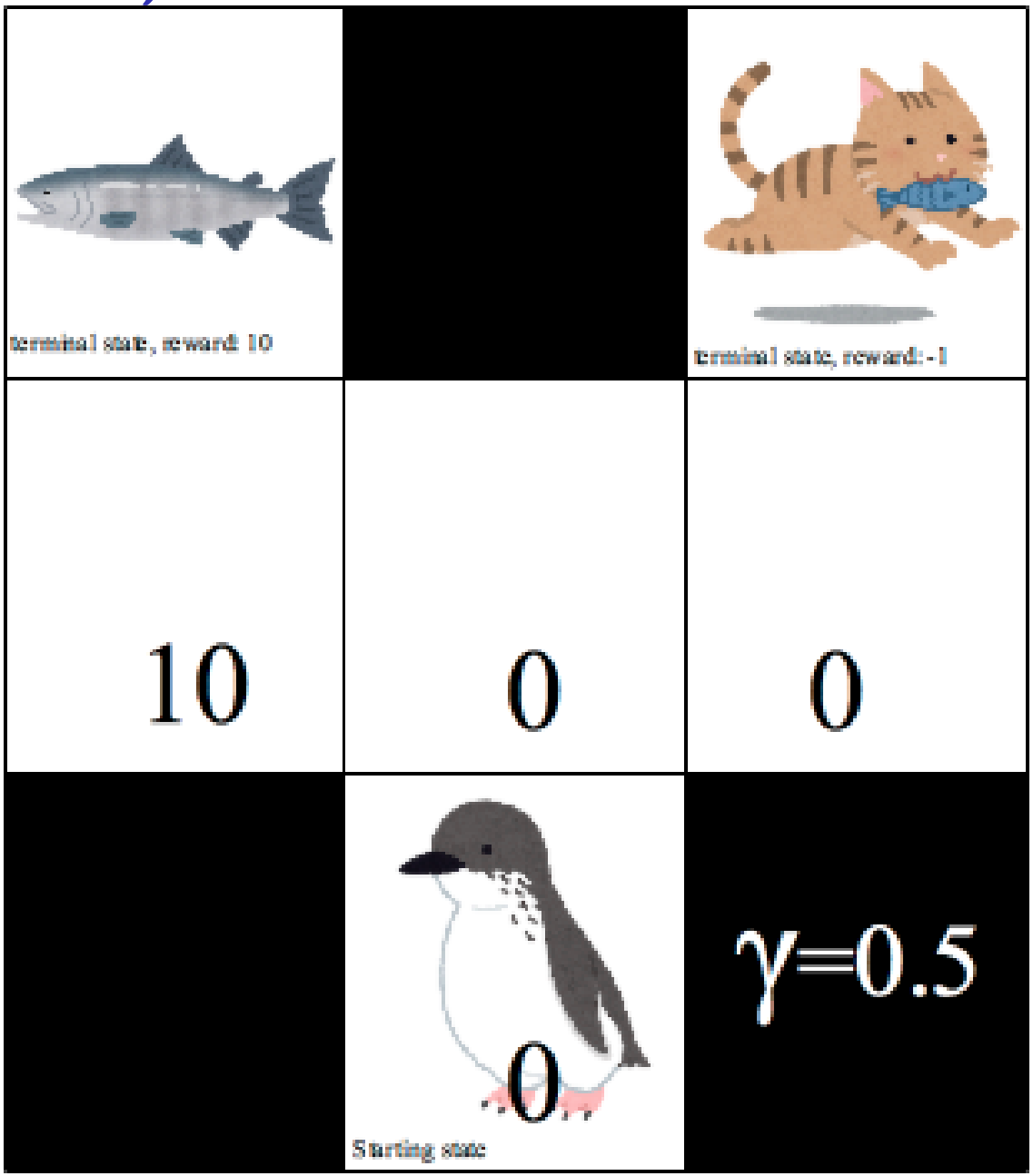
A tiny (6-state) MDP



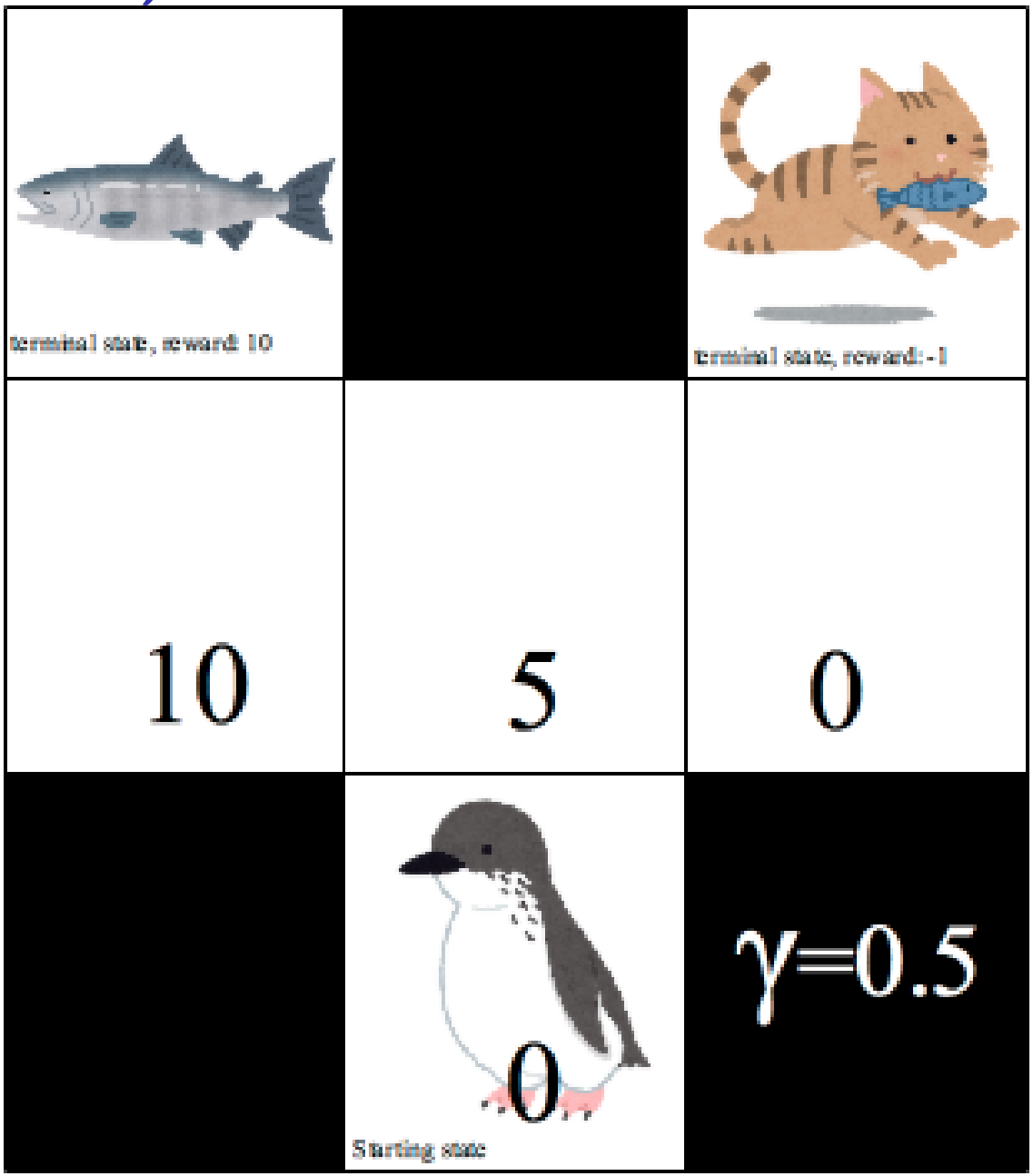
A tiny (6-state) MDP



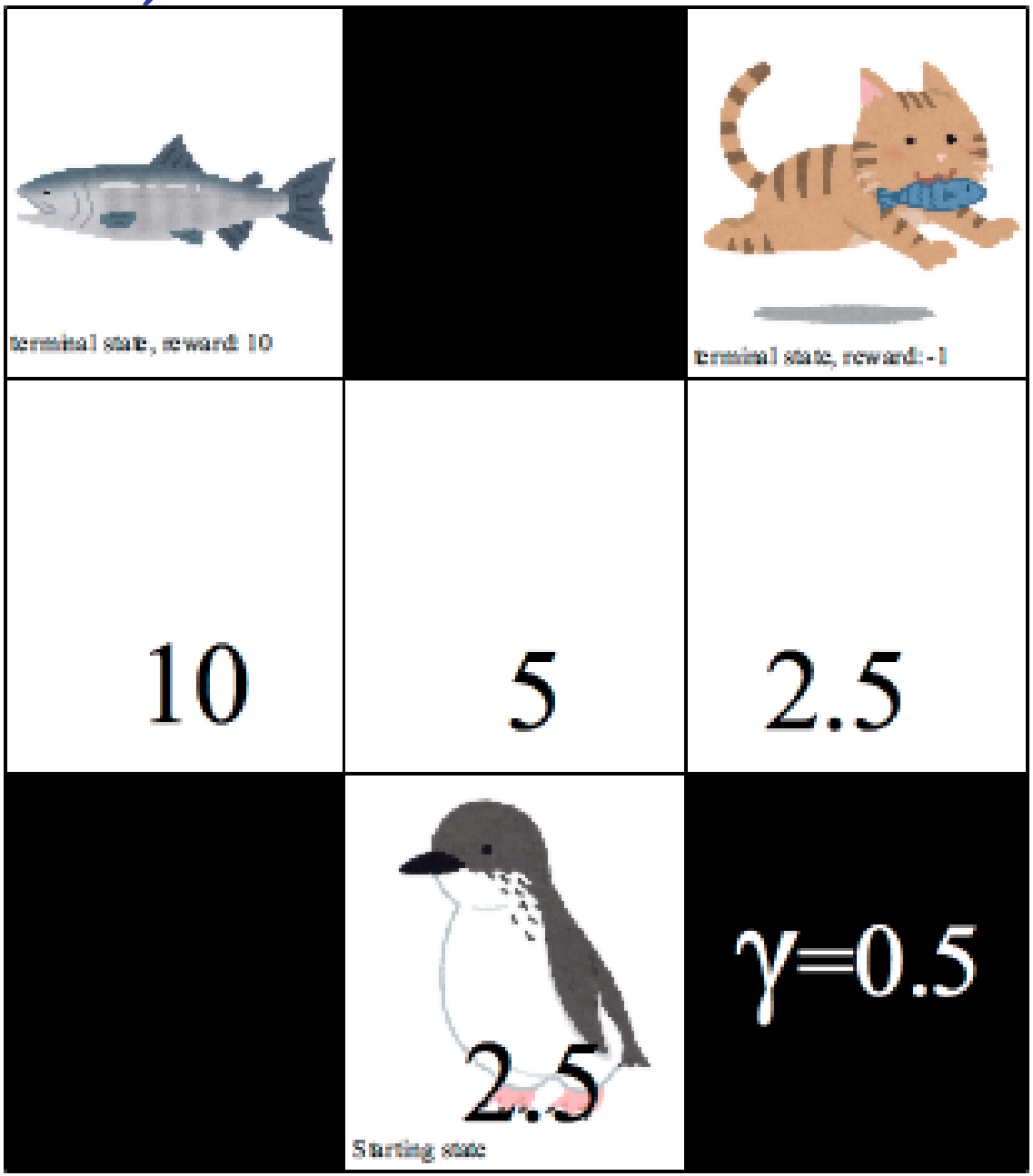
A tiny (6-state) MDP



A tiny (6-state) MDP



A tiny (6-state) MDP



Efficiency of dynamic programming

- An MDP has $|A|^{|S|}$ deterministic stationary policies
- Worst-case computational complexity of DP is polynomial in $|S|$, $|A|$, and $\frac{1}{1-\gamma} \log \left(\frac{1}{1-\gamma} \right)$*
- MDP planning can also be done with *linear programming*
  - Better runtime guarantees, but impractical for large MDPs

*M.L. Littman, T.L. Dean, L.P. Kaelbling — On the complexity of solving Markov decision problems, UAI, 1995,

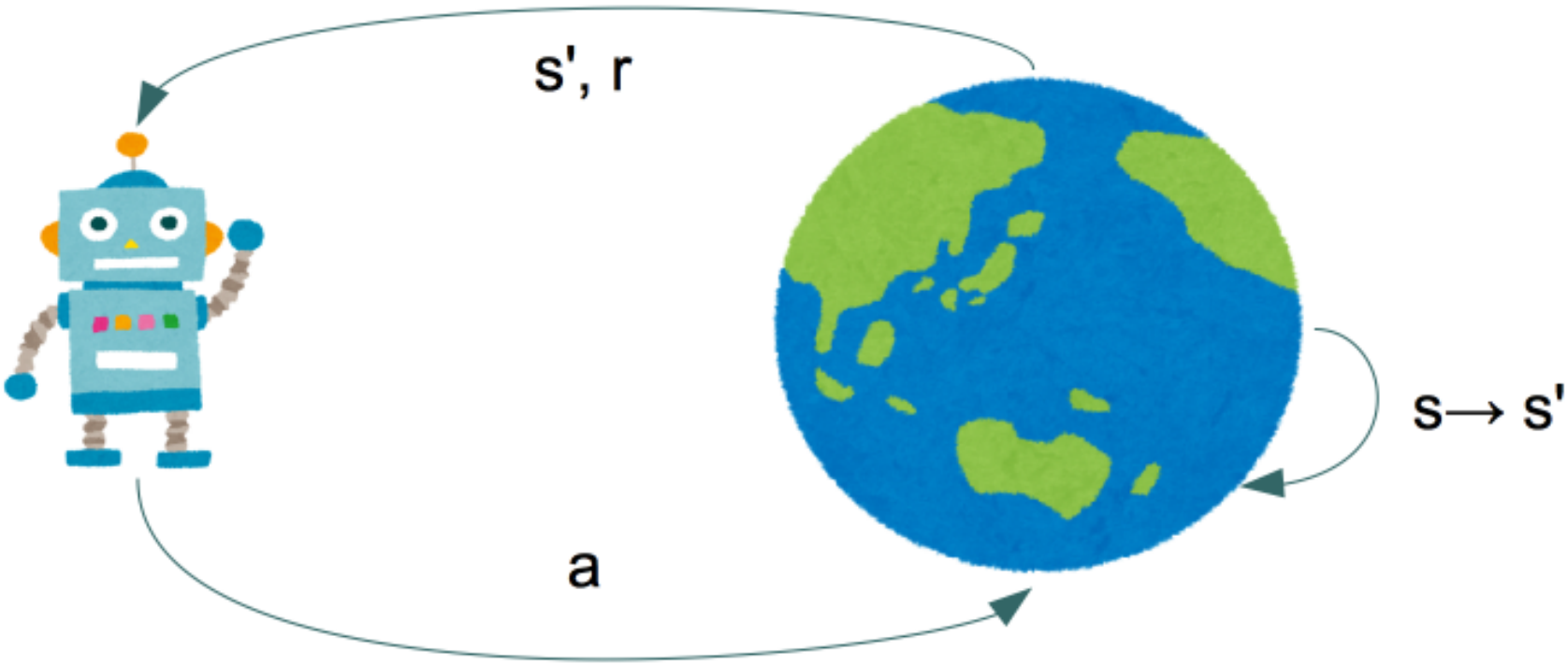
Schedule

- Intro Agents, Planning, and Learning
- Markov decision processes (MDPs)
- Planning methods: dynamic programming
- *Reinforcement Learning*
  - Actor-critic
  - Q-Learning
  - SARSA
  - Model-based learning
- Advanced topics

Reinforcement Learning

- Agent does not have a (complete) model of the environment
- Agent learns about environment through interaction
- How to act to optimise expected (discounted) sum of rewards
  - i.e., how to attain the optimal *value* through an optimal *policy*
- We assume infinite-horizon setting in this presentation

Reinforcement Learning: MDPs



Reinforcement Learning versus Planning

- Up until now: agent is given a model of environment
  - i.e., knows how the world works
  - and therefore only needs to *plan*
- From now on: agent is not given a model
  - i.e., needs to learn how to act through interaction
  - *reinforcement learning*

TD(0): Policy Evaluation without a Model

- Evaluate a policy π, by executing the policy
- Update state-value-estimates by exploiting Bellman equation
- Single-step updates
- TD(0) uses as update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- TD(0) *bootstraps*; it bases updates on existing estimates (like DP)

TD(0)

- Evaluate a policy π, by executing the policy
- Update state-value-estimates by exploiting Bellman equation
- Single-step updates
- TD(0) uses as update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

temporal difference (TD) error

- TD(0) *bootstraps*; it bases updates on existing estimates (like DP)

TD(0) algorithm

- Input: π, initial value estimate $V(s)$ for every state, α.
- Repeat (for each episode):
  - Initialise s (initial state)
  - For each time-step, t, during episode:
    - $a \leftarrow \pi(s)$ (if deterministic, else sample a from $\pi(s)$)
    - take action a, and observe reward, r, and next state, s'
    - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    - $s \leftarrow s'$

Reinforcement Learning Choices

- Agent needs to learn a good policy
- Choices
  - explicitly learn a model of the environment?
  - explicitly represent the policy the agent is learning?
  - learn a value function?

Model-free Reinforcement Learning

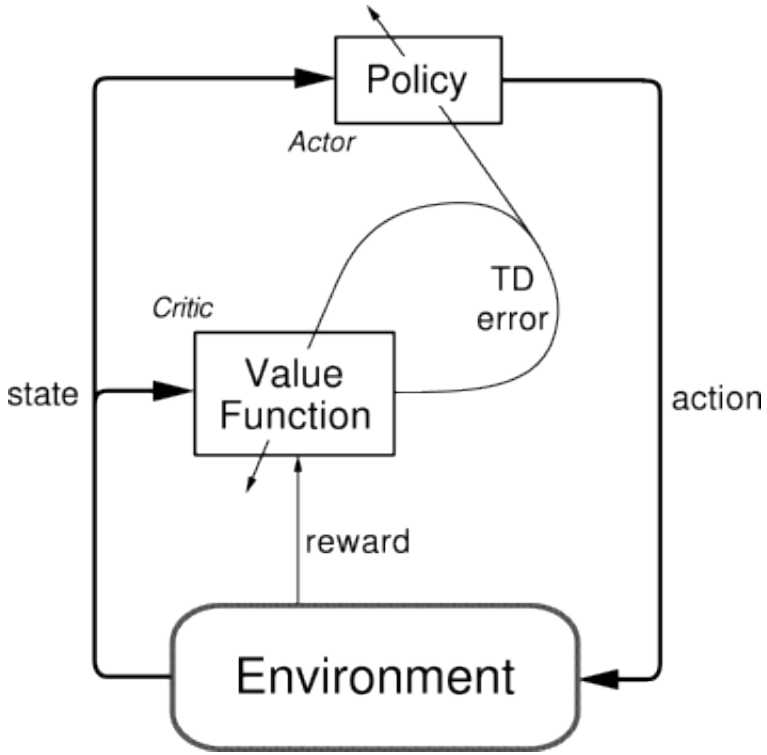
- Agent directly learns a policy and/or value function directly
- A is given
- S is often given as well
- Policy search
- Actor-Critic
- Value-function-(only)-based methods

Reinforcement Learning Choices

- Agent needs to learn a good policy
- Choices
  - explicitly learn a model of the environment? *no*
  - explicitly represent the policy the agent is learning? *yes*
  - learn a value function? *yes*

Actor-critic

Represent both policy (actor) and critic (value function) explicitly.



Use TD-error to change policy.

Similar scheme to Policy Iteration

Actor-critic

- TD-error: $\delta \leftarrow r + \gamma V(s') - V(s)$
- For example:
  - $\pi(s, a) = \frac{e^{\theta(s,a)}}{\sum_{a'} e^{\theta(s,a')}}$ (parameters θ)
  - update: $\theta(s, a) \leftarrow \theta(s, a) + \beta\delta$
- Explicitly represent policy and value
- Important when action space is huge
- Can learn explicitly stochastic policies (important for multi-agent)
- Important for Deep RL

Reinforcement Learning Choices

- Agent needs to learn a good policy
- Choices
  - explicitly learn a model of the environment? *no*
  - explicitly represent the policy the agent is learning? *no*
  - learn a value function? *yes*

Temporal Difference RL Methods

- Model-free Reinforcement Learning
- Value-function-(only)-based method
- Exploits Bellman equation (bootstrapping)
- On-policy RL: SARSA
- Off-policy RL: Q-learning

From policy evaluation to learning

- SARSA and Q-Learning optimise the policy, and learn value function and policy simultaneously (like VI)
- Update state-action-value-estimates, $Q(s, a)$, by exploiting Bellman equation
- Single-step updates (*bootstrapping*)
- Action selection based on $Q(s, a)$ estimates (exploration versus exploitation)

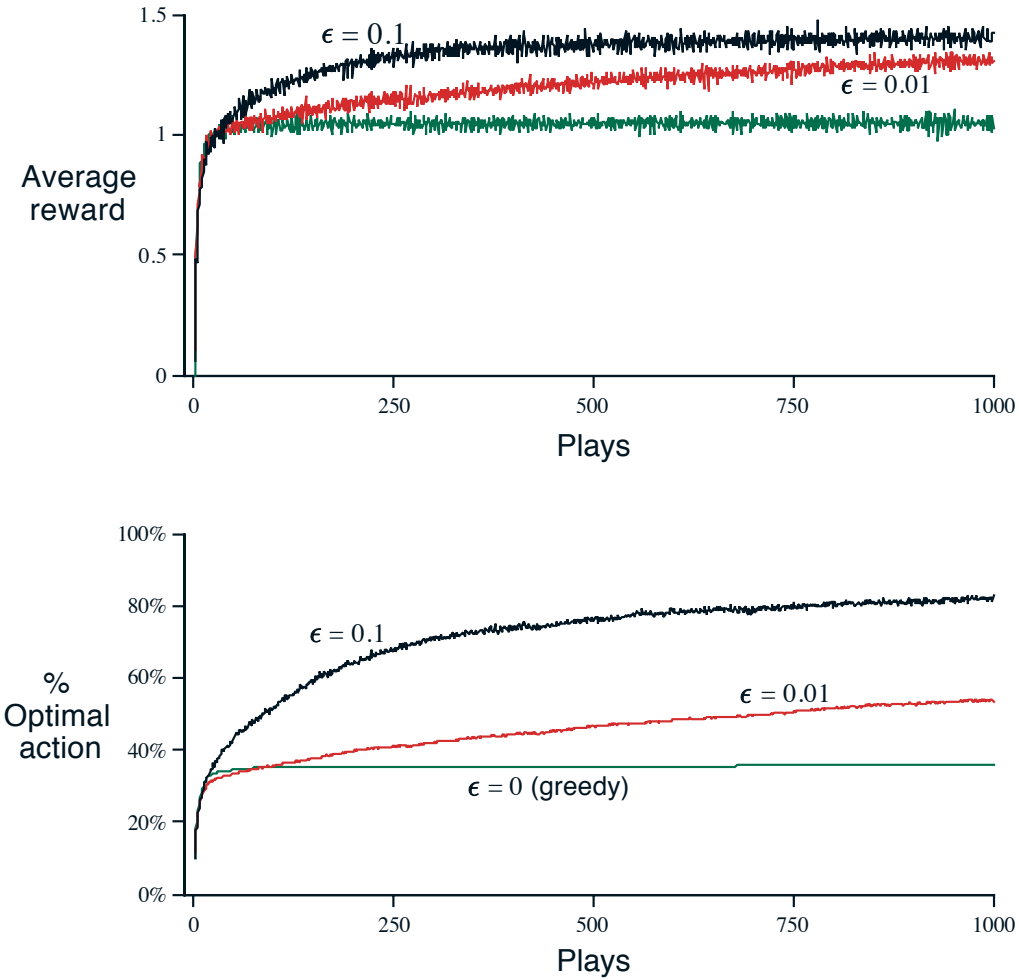
Action selection

- $\tilde{a}^* \leftarrow \arg\max_a \tilde{Q}(s, a)$ (greedy action)
- ε-greedy: with $P = 1 - \varepsilon$ take greedy action, with $P = \varepsilon$ take random action.
- Softmax: probability of selecting a in state s higher if $Q(s, a)$ is high (also for non-greedy actions)

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q(s,a')}{\tau}}}$$

- See e.g., Sutton and Barto book chapter 2 (`http://incompleteideas.net/sutton/book/ebook/node14.html`)

ϵ-greedy



SARSA algorithm

- Input: π, initial value estimate $Q(s, a)$ for every state-action pair, α.
- Repeat (for each episode):
  - Initialise s (initial state)
  - $a \leftarrow$ select action based on Q and an exploration mechanism
  - For each time-step, t, during episode:
    - ★ take action a, and observe reward, r, and next state, s'
    - ★ $a' \leftarrow$ select action based on Q and an exploration mechanism
    - ★ $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    - ★ $s \leftarrow s'$; $a \leftarrow a'$

SARSA

- SARSA optimises policy, as it derives which action to take from Q-values
- On-policy: it learns the Q-values of the executed policy
- Executed policy includes (!) exploration mechanism
- Exploration mechanism restricts policy space
- Also possible to (approximately) learn optimal policy and associated Q-values

Q-learning

- Input: π, initial value estimate $Q(s, a)$ for every state-action pair, α.
- Repeat (for each episode):
  - Initialise s (initial state)
  - For each time-step, t, during episode:
    - $a \leftarrow$ select action based on Q and an exploration mechanism
    - take action a, and observe reward, r, and next state, s'
    - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    - $s \leftarrow s'$

Q-learning

- Q-learning optimises policy, as it derives which action to take from Q-values, and maximises next action.
- Off-policy: it approximates the Q-values of the optimal policy
- Exploration mechanism is not accounted for in value estimates
- The basis algorithm of a highly popular class of algorithms (including many Deep RL methods)

Theory

Model-free temporal difference methods such as Q-learning and SARSA are guaranteed to converge to the optimal policy in the limit under the following conditions:

- S and A are finite
- $\sum_t \alpha_t^{sa} = \infty$ and $\sum_t (\alpha_t^{sa})^2 < \infty$
- $Var\left(R_a^{ss'}\right) < \infty$
- $\gamma < 1$

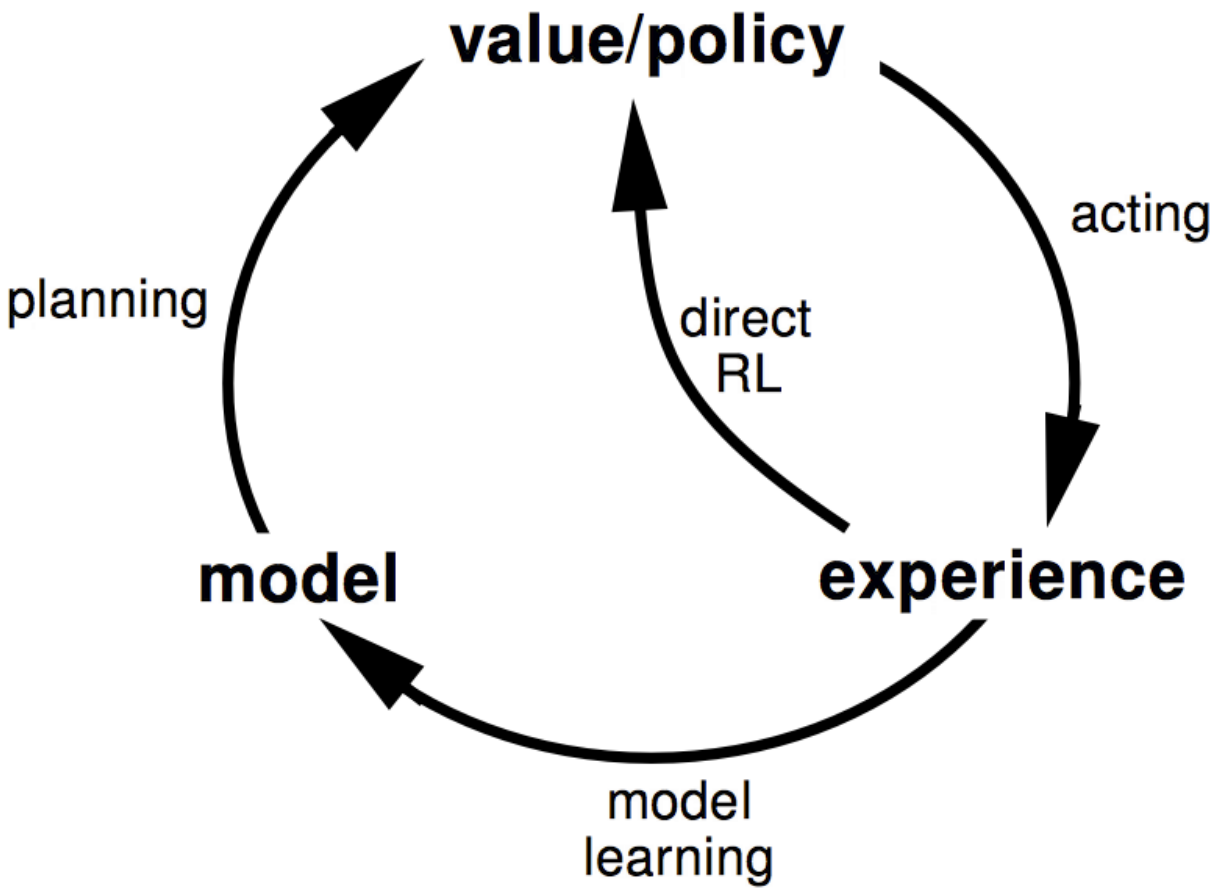
Reinforcement Learning Choices

- Agent needs to learn a good policy
- Choices
  - explicitly learn a model of the environment? *yes*
  - explicitly represent the policy the agent is learning? *possibly*
  - learn a value function? *possibly*

Model-based Reinforcement Learning

- Agent constructs an MDP model
- S and A are usually given
- $R^a_{ss'}$ or just R^a_s
- $P^a_{ss'}$

Model-based Reinforcement Learning



Schedule

- Intro Agents, Planning, and Learning
- Markov decision processes (MDPs)
- Planning methods: dynamic programming
- Reinforcement Learning
  - Actor-critic
  - Q-Learning
  - SARSA
  - Model-based learning
- *Advanced topics*

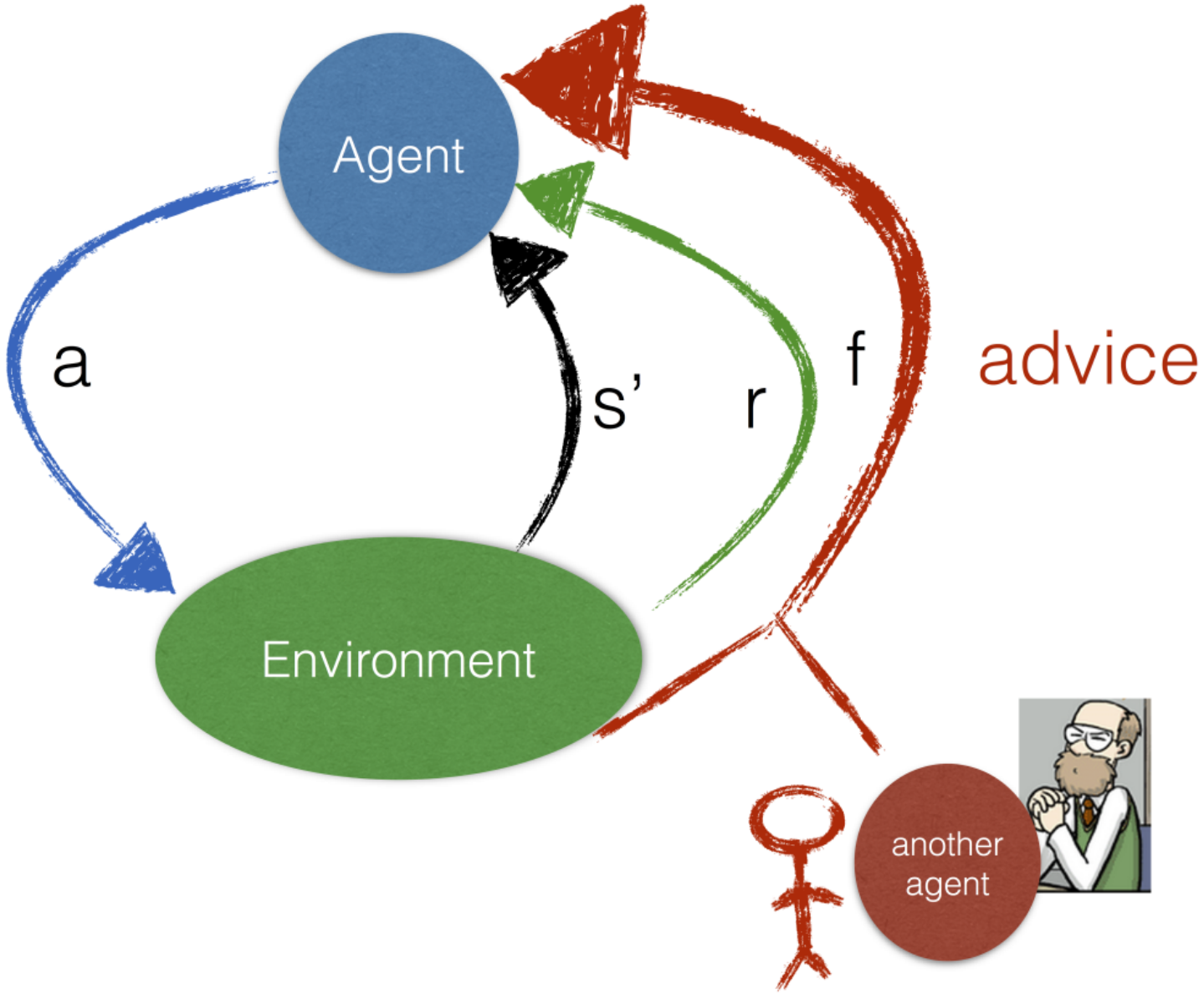
What we haven't told you...

- Prior knowledge / transfer learning
- Function approximation
- Partial observability
- Multiple Objectives
- Multi-agent settings (this afternoon)

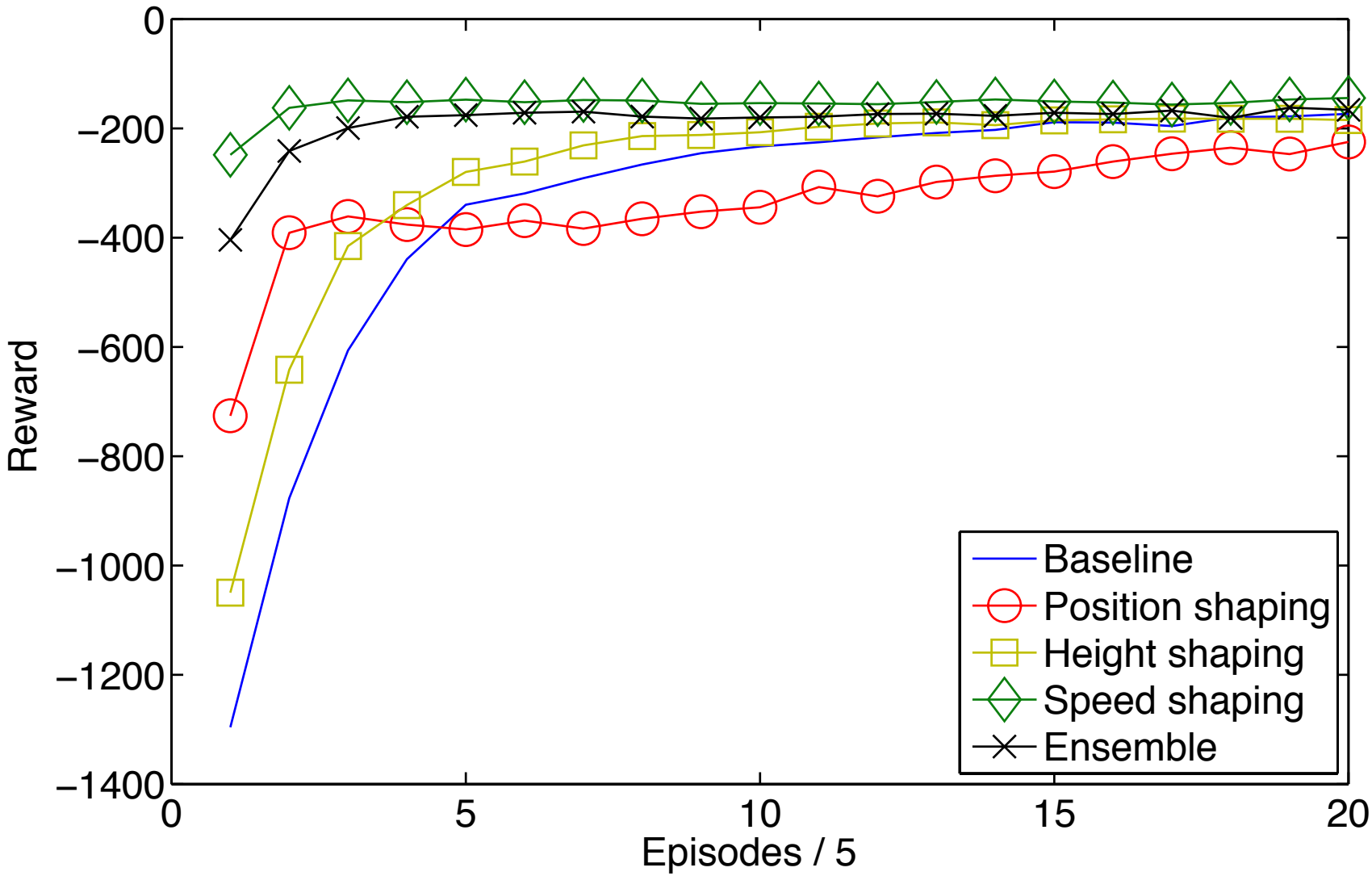
Prior knowledge / Transfer learning

- What if we already know a lot (or have good intuition)?
- What if have trained an agent on a similar task?
- Potential functions: bias the immediate rewards, but retain optimal policy (with equal value)
- Policy advice
- Human teaching (external rewards)

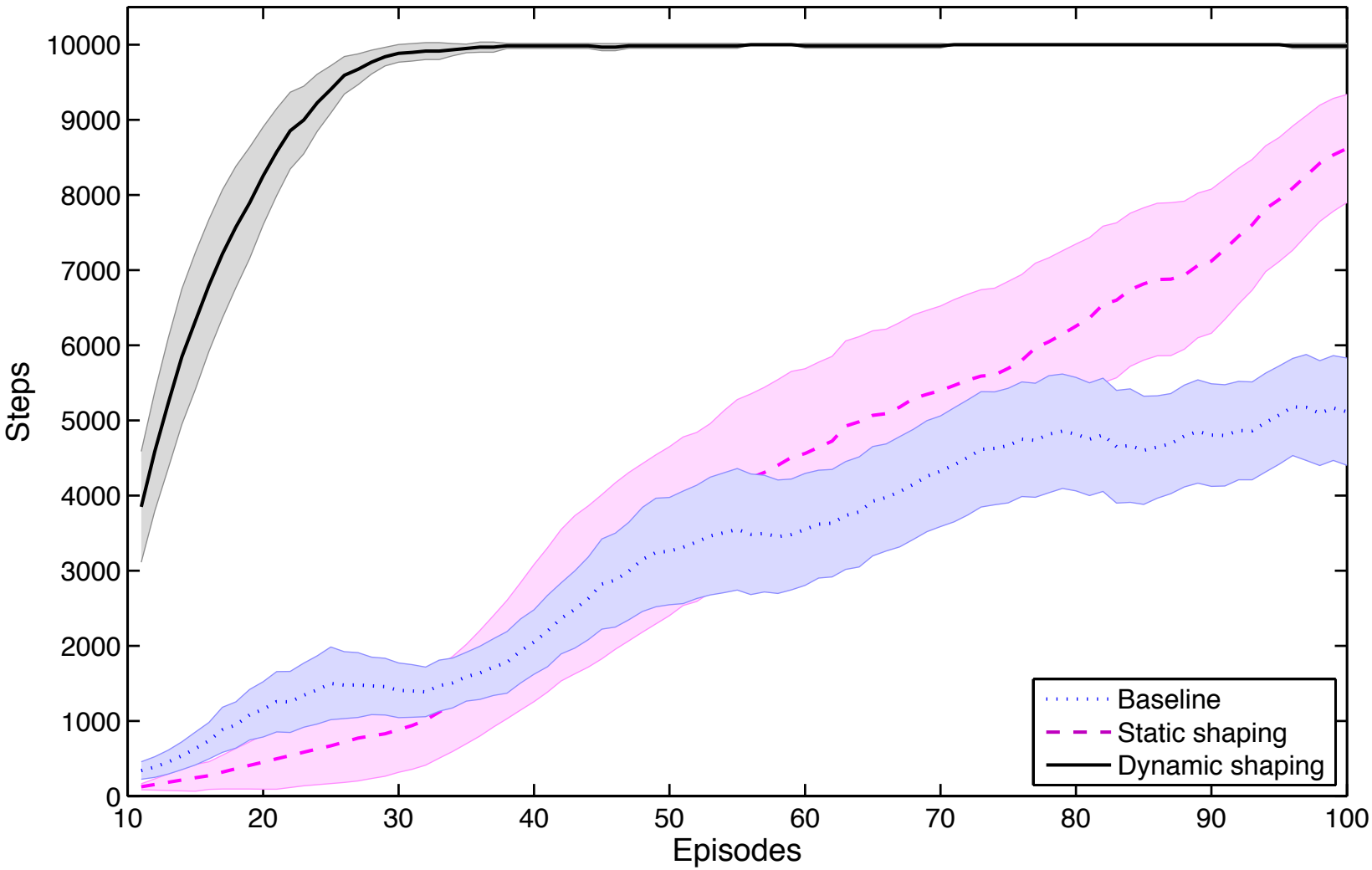
Dynamic Reward Shaping



Tuning problem of Shaping - Mountain Car

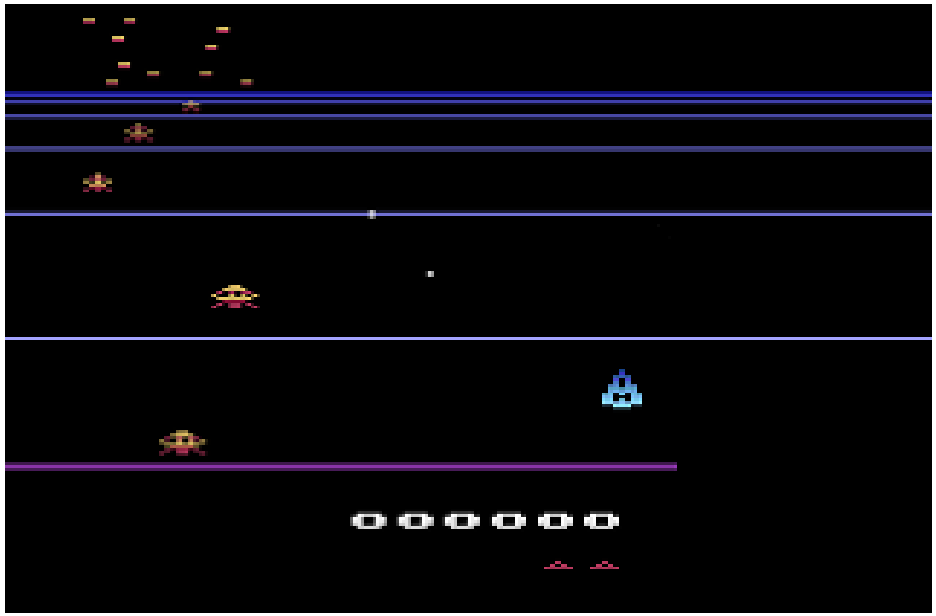


Dynamic Reward Shaping - Cart Pole



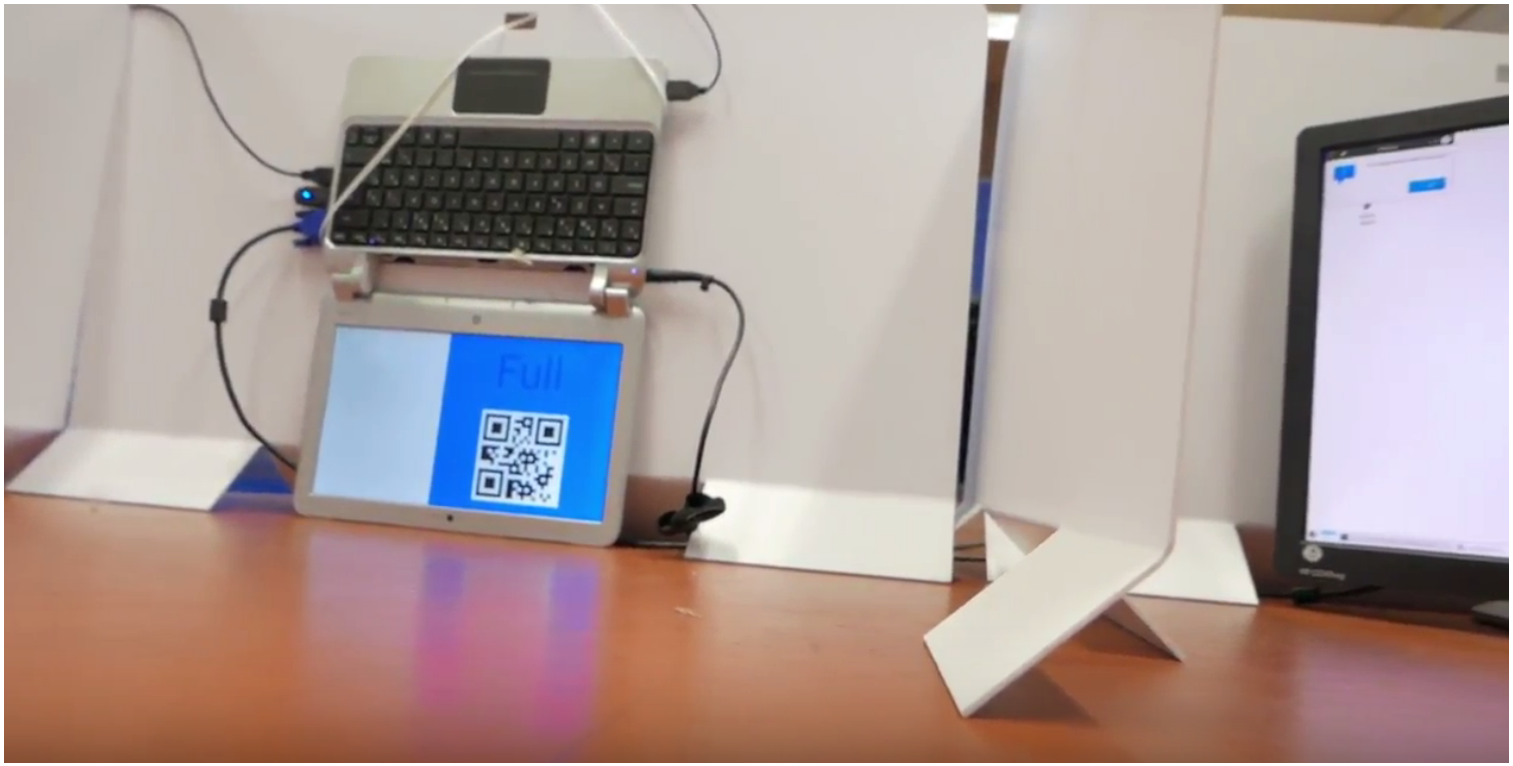
Function approximation

- State-spaces can be too big
- Agent may never visit every state
- Deep learning
- Atari games, Alpha-Go, robotics



Partial observability

- Cannot see the entire state $\rightarrow$ non-Markovian observations



From Steckelmacher et al., AAAI 2018

Engineering a decision-making system

- Model the task we want it to perform
- Multiple events can happen
- Design a reward function
- Make the system plan/learn/optimise
- It does not do what we expect/want

Designing a reward function: typical workflow

- Human designers of decision-making software (i.e., us)
  - Identify (un)desirable events
  - Assigning reward to these (un)desirable events
    - ⋆ Collapse it into one scalar function
    - ⋆ Implicitly assuming rather arbitrary preferences
    - ⋆ Output becomes uninterpretable (one number)

What we can do instead

- Model the world in clear measurable objectives (with proper units)
- Understand what people (really) want
  - Model the user (separately from the environment)
  - Interact with the user
- Make informed decisions and inspect alternatives
- Enable handling users with uncertain/changing preferences
- Explain the decisions in terms people can understand

Multi-objective modelling

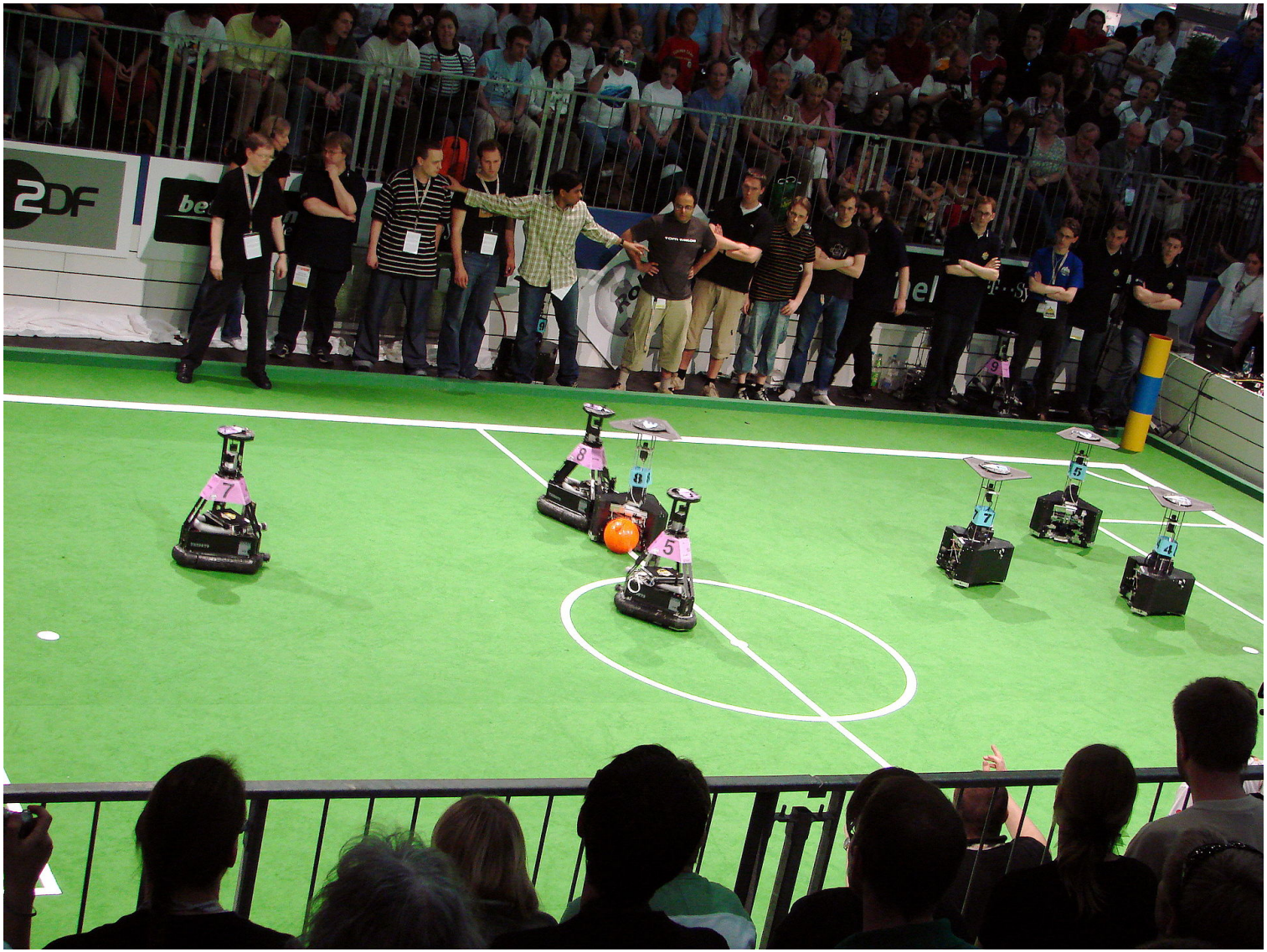
- Each objective has its own reward function
- Each representing a desirable feature of solution
- Measurable and interpretable numbers

Multi-objective modelling

- Each objective has its own reward function
- Each representing a desirable feature of solution
- Measurable and interpretable numbers
  - Easy to specify
  - Easy to communicate
  - Useful for design and deployment alike

Multiple agents

- We cannot look inside the other agents' heads



From Robocup 2006, photo by Photocapy

Closing

- Brief intro to RL
- Planning: given an MDP model find the optimal policy (Value Iteration, Policy Iteration)
- Learning: learn how to behave from interaction with the environment (without knowledge about the reward and transition functions)

Closing

- Brief intro to RL
- Planning: given an MDP model find the optimal policy (Value Iteration, Policy Iteration)
- Learning: learn how to behave from interaction with the environment (without knowledge about the reward and transition functions)
- There is so much out there!