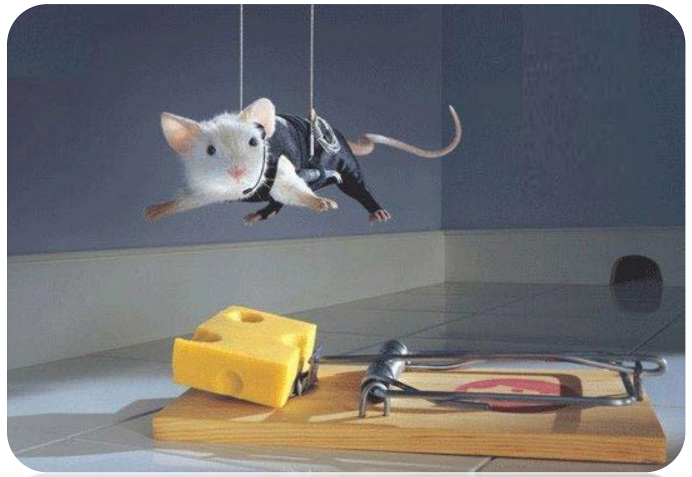


Reinforcement learning

Tony Wauters

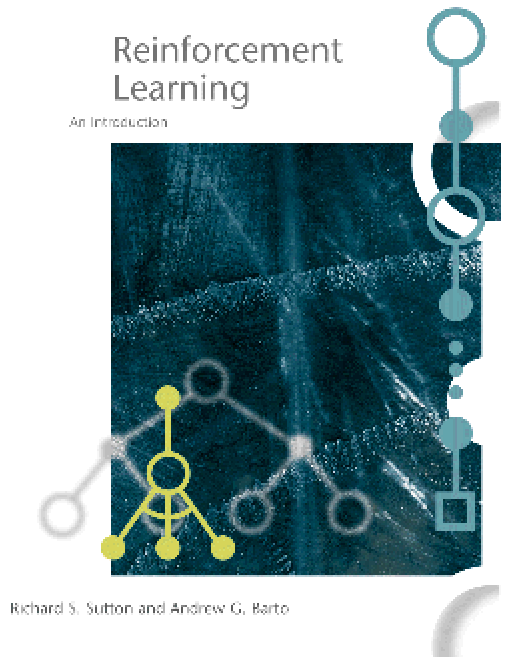
(original slides: Katja Verbeeck)

Outline

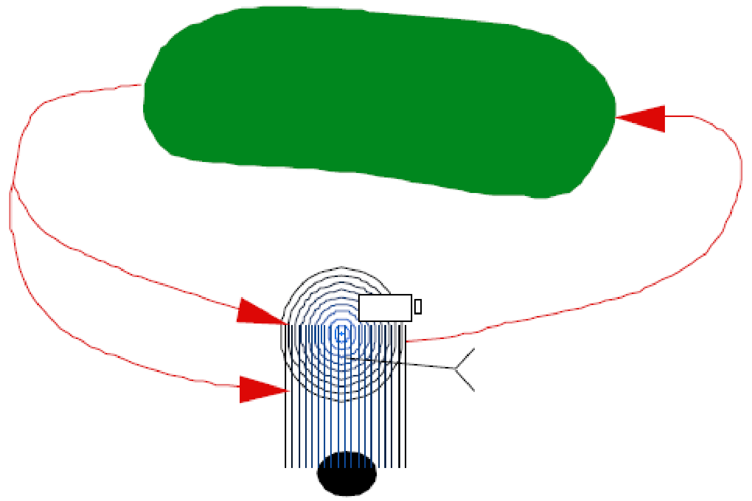
- Reinforcement Learning: an introduction from a computational perspective
- Multi-agent Reinforcement Learning: how to reach collective intelligence
- Learning automata

Reinforcement Learning: an introduction from a computational perspective

(Sutton & Barto: Reinforcement Learning: an introduction, 1998)



Reinforcement learning?



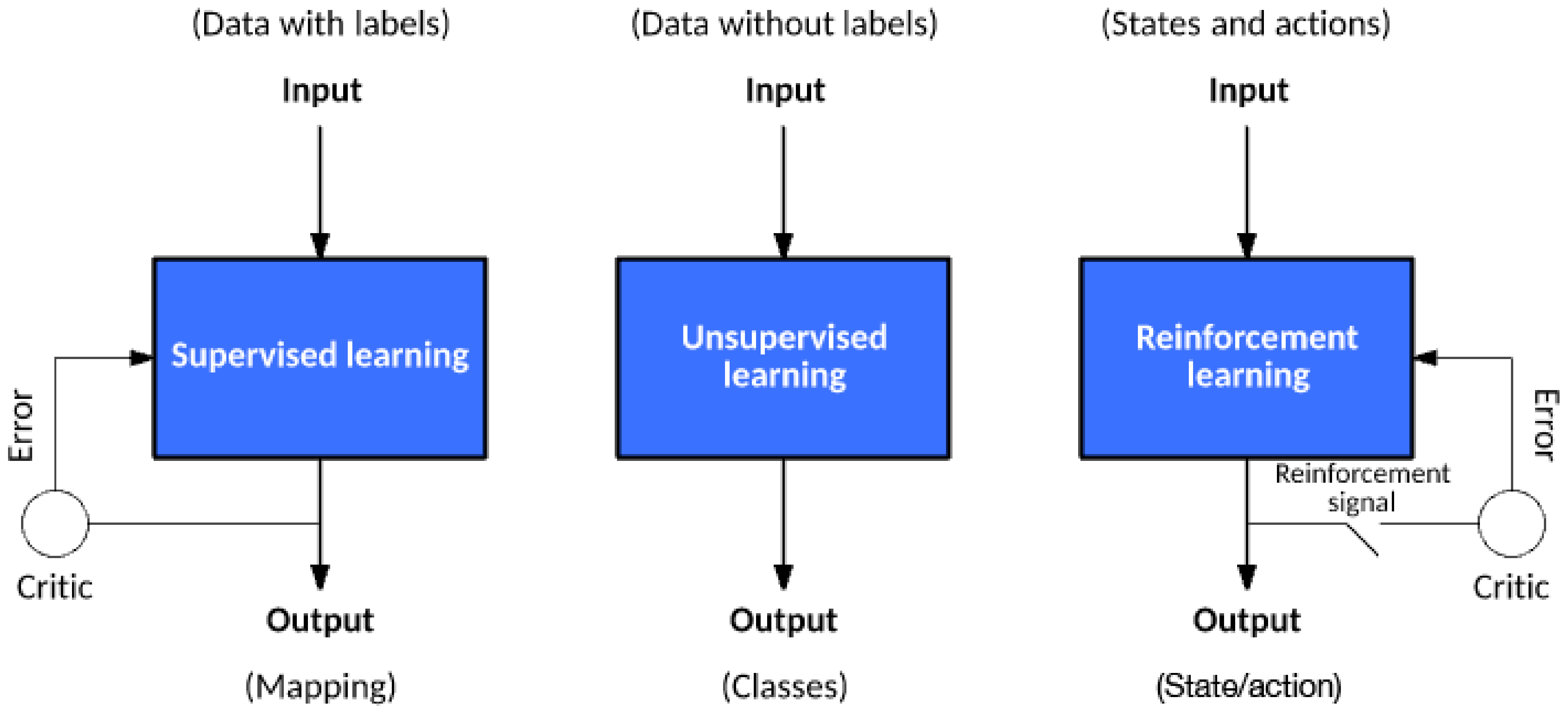
- Learning
  - from interaction
- Learning
  - about, from, and while interacting with an **external environment**
- Learning
  - what to do—how to map situations to actions—so as to maximize a numerical **reward signal**

Reinforcement learning – key features

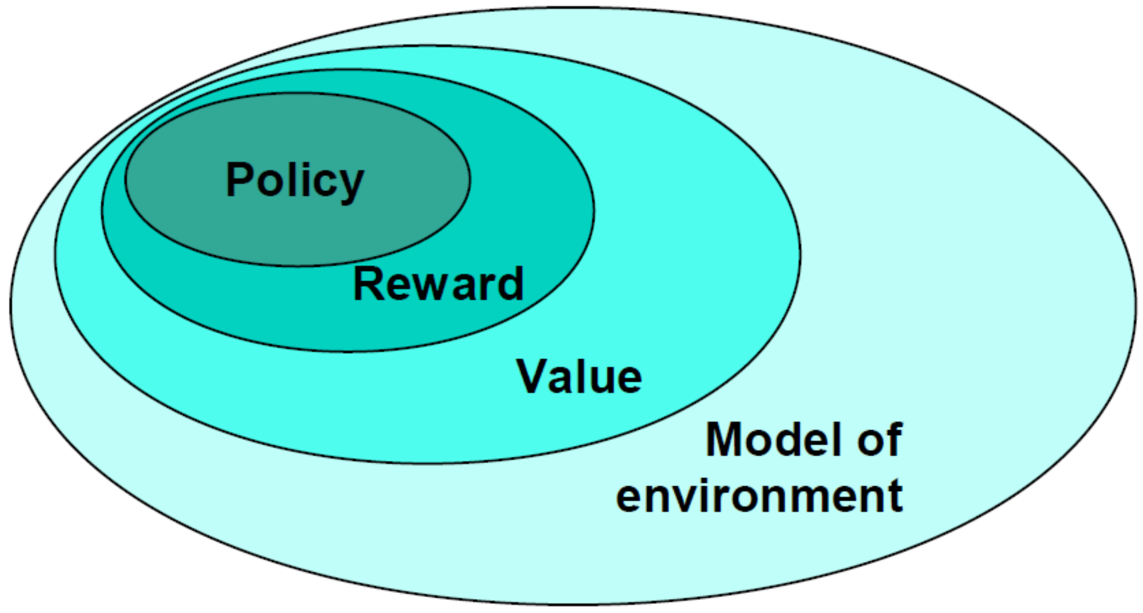
- Learner is not told which actions to take
- Trial-and-Error search
- Possibility of delayed reward
- Sacrifice short-term gains for greater long-term gains
- The need to *explore* and *exploit*
- Considers the whole problem of a goal-directed agent interacting with an uncertain environment

Trial-and-error

Machine learning paradigms

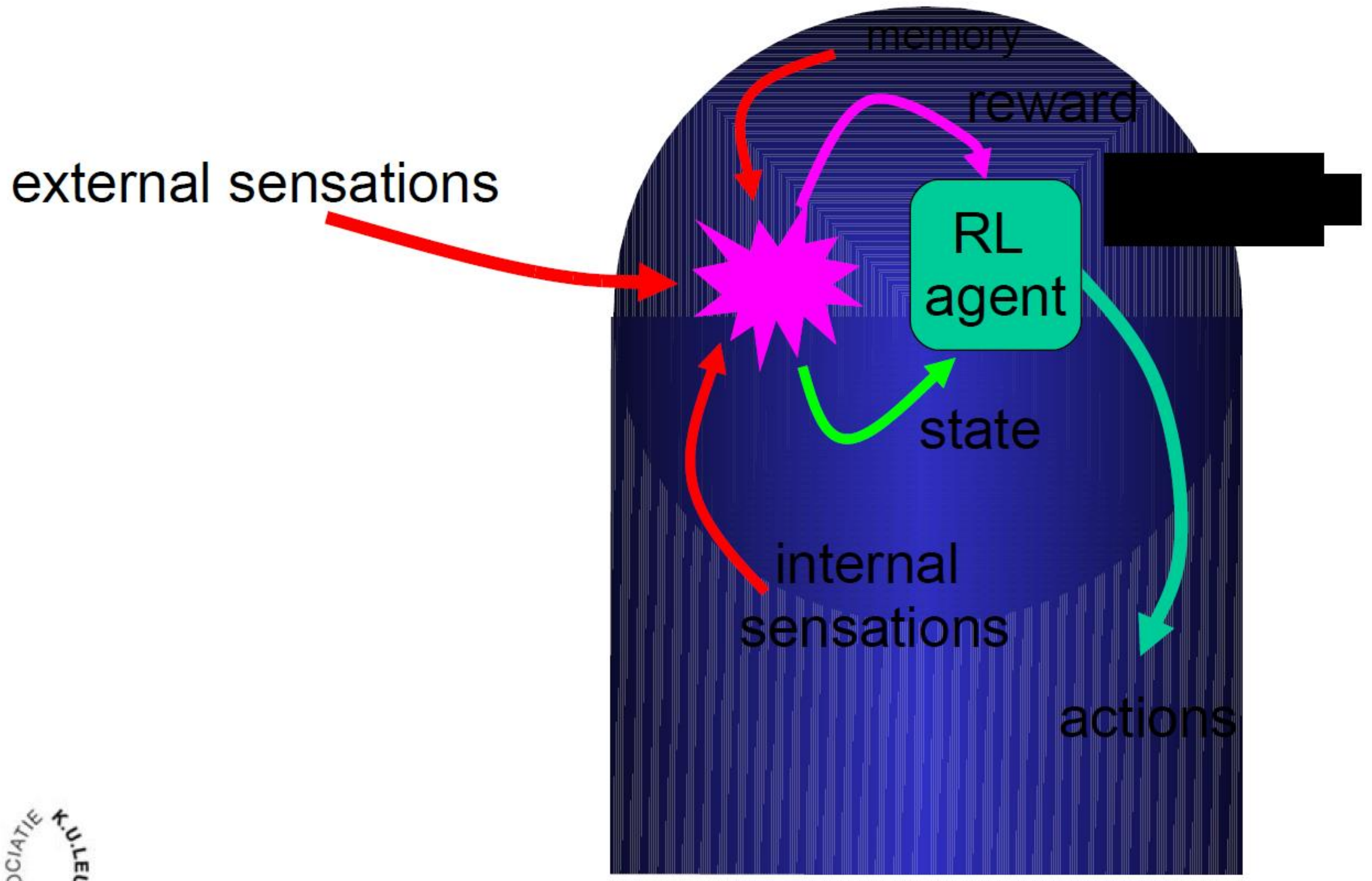


RL elements

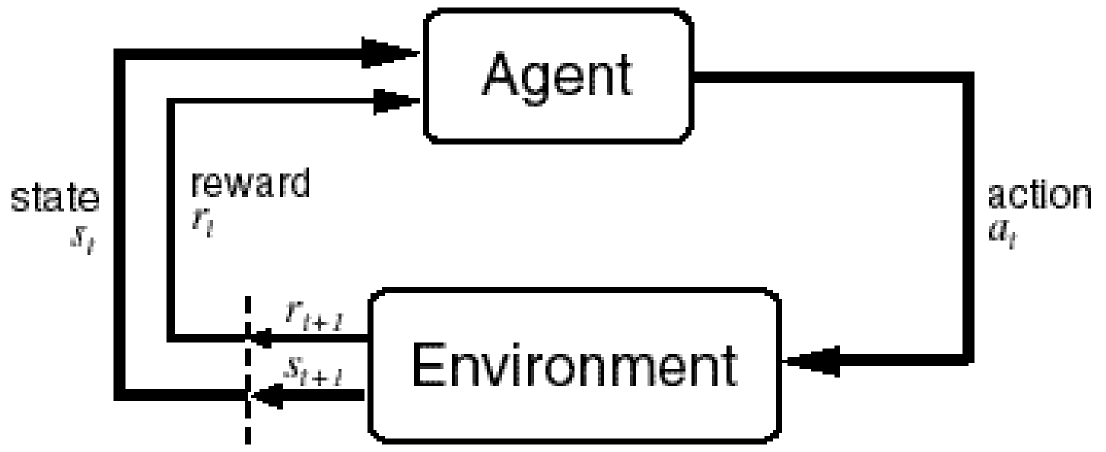


- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what

A somewhat less misleading view …



The Agent-Environment Interface



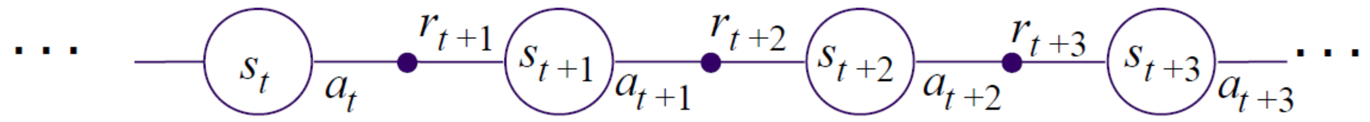
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \Re$

and resulting next state: s_{t+1}



Learning how to behave

Policy at step t, π_t:

a mapping from states to action probabilities

$\pi_t(s,a) =$ probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent’s goal is to get as much reward as it can over the long run.

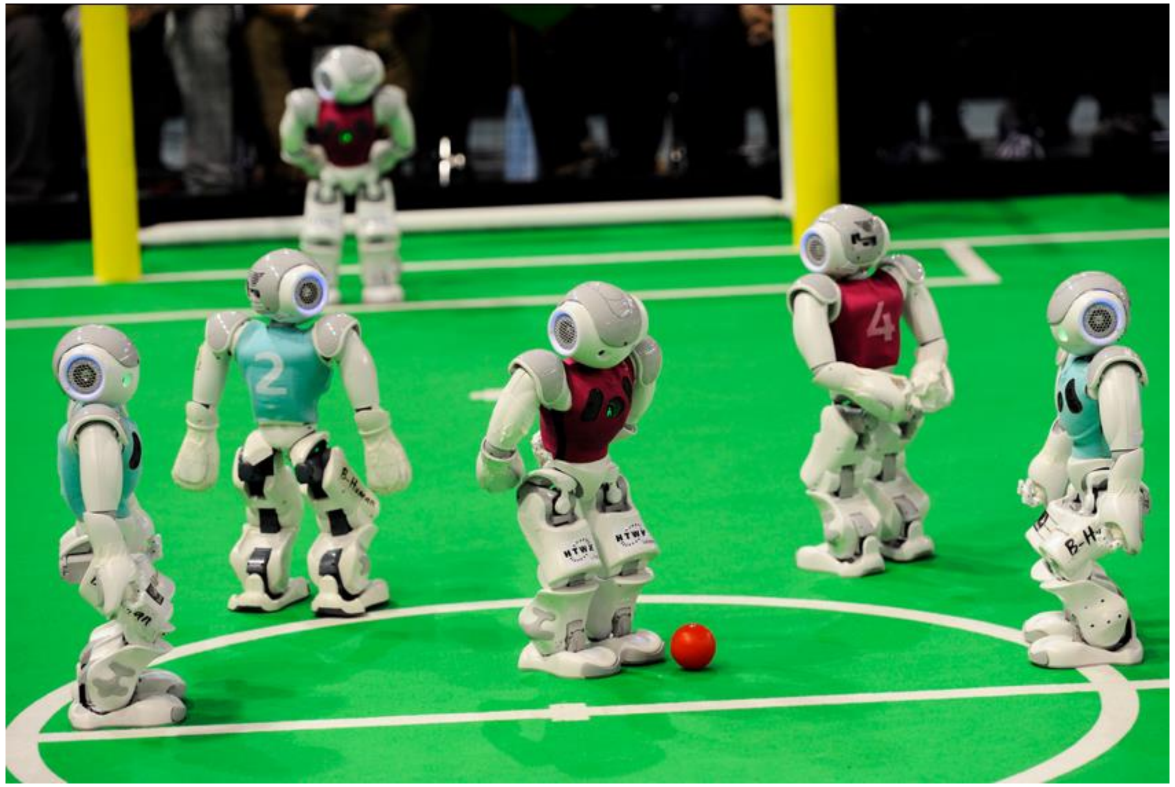
Examples of reinforcement learning

- Robocup Soccer Teams Stone & Veloso, Reidmiller et al.
  - World’s best player of simulated soccer, 1999; Runner-up 2000
- Inventory Management Van Roy, Bertsekas, Lee & Tsitsiklis
  - 10-15% improvement over industry standard methods
- Dynamic Channel Assignment Singh & Bertsekas, Nie & Haykin
  - World's best assigner of radio channels to mobile telephone calls
- Elevator Control Crites & Barto
  - (Probably) world's best down-peak elevator controller
- Many Robots

(https://www.youtube.com/watch?v=fRj34o4hN4I must watch!!!)
  - navigation, bi-pedal walking, grasping, switching between skills...
- TD-Gammon and Jellyfish Tesauro, Dahl
  - World's best backgammon player

Robocup

- https://www.youtube.com/watch?v=iNLcGqbhGcc



Other examples

- Drifting car ☺: https://www.youtube.com/watch?v=opsmd5yuBF0
- Many more examples: **https://www.youtube.com/playlist?list=PL5nBAYUyJTrM48dViibyi68urttMlUv7e**

Robotic Control , DARPA (URBAN) Challenge

2004 : the Mojave Desert for 150 miles (240 km) None of the robot vehicles finished the route. CMU's Red Team traveled the farthest distance, completing 11.78 km

2005 212 km (132 mi) off-road course. All but one of the 23 finalists in the 2005 race surpassed the 11.78 km (7.32 mi) distance completed by the best vehicle in the 2004 race. Five vehicles successfully completed the course.

2007 The course involved a 96 kilometres (60 mi) urban area course, to be completed in less than 6 hours. Rules included obeying all traffic regulations while negotiating with other traffic and obstacles and merging into traffic.

AUTONOMOUS CARS TODAY

- adaptive cruise control , lane assist, parking assist
- → up to full autonomous driving

A closer look on the objective

Suppose the sequence of rewards after step t is:

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

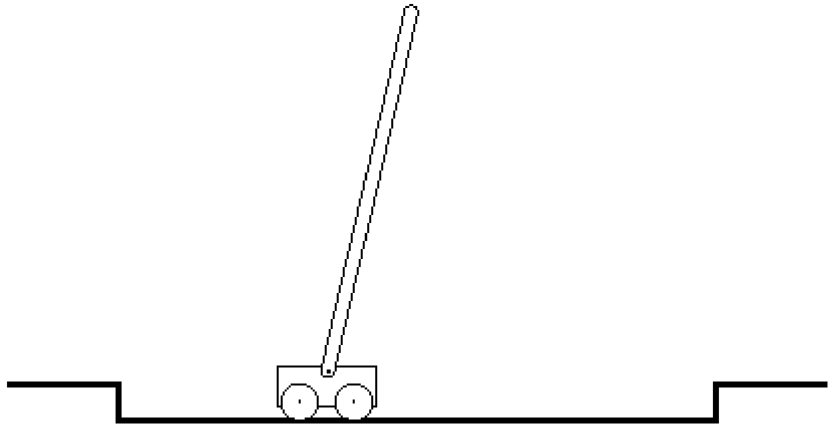
What do we want to maximize?

- In general,we want to maximize the *expected return*, for each step t.
- **Episodic tasks**: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = \underbrace{r_{t+1}}_{\text{Immediate reward}} + \underbrace{r_{t+2} + \ldots + r_T}_{\text{Long term reward}}$$

Cart-pole balancing

- As an **episodic task** where episode ends upon failure:
  - Reward +1 for each step before failure
  - return number of steps before failure
- As a **continuing task** with discounted return:
  - Reward -1 upon failure; 0 otherwise
  - Return - g ^{k} , for k steps before failure
- In either case, return is maximized by avoiding failure for as long as possible.

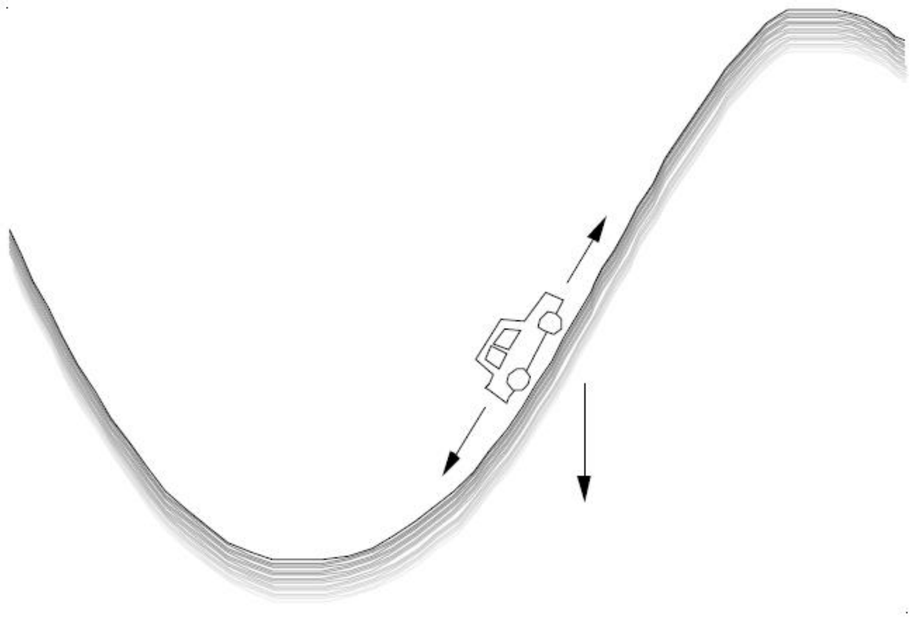


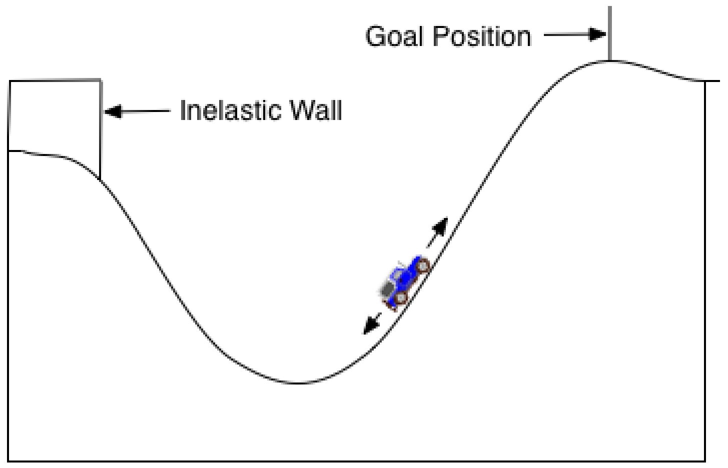
Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track.

Movies:
https://www.youtube.com/watch?v=Lt-KLtkDlh8
https://www.youtube.com/watch?v=cyN-CRNrb3E

Mountain – car problem

- Get to the top of the hill as quickly as possible.
- Reward -1 for each step where NOT at top of hill
- Return = - number of steps before reaching top of hill
- Return is maximized by minimizing number of steps reach the top of the hill.





Getting the Degree of Abstraction Right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (voltages to motors), or high level (accept job offer), “mental” (shift focus of attention), etc.
- States can be low-level “sensations”, or abstract, symbolic, based on memory, or subjective (“surprised” or “lost”).
- The environment is not necessarily unknown to the agent, only incompletely controllable.
- A RL agent is not like a whole animal or robot

The 2-armed bandit problem

- just one state
- 2 actions, each has a scalar reward (coins or not)which is chosen with a probability distribution
- Aim: maximize the total reward over time T

→ which lever is best?

The Exploration/Exploitation Dilemma

$$E\{r_t \mid a_t\} = Q^*(a_t)$$ **Action value estimates**

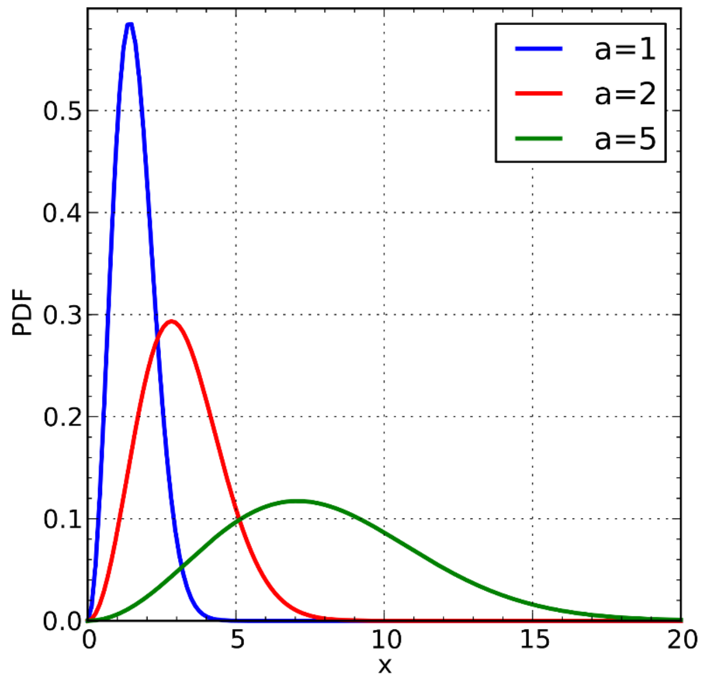
- Suppose you form estimates : $Q_t(a) \sim Q^*(a)$
- The greedy action at *t* is $a_t^* = \operatorname{argmax}_a Qt(a)$
- If $a_t = a_t^*$ => exploitation
- If $a_t \neq a_t^*$ => exploration
- You can’t exploit all the time; you can’t explore all the time
- You can never stop exploring; but you should always reduce exploring.

A Softmax Action Selection

- Softmax action selection methods **grade action probabilities by estimated values.**
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

~ "computational temperature"

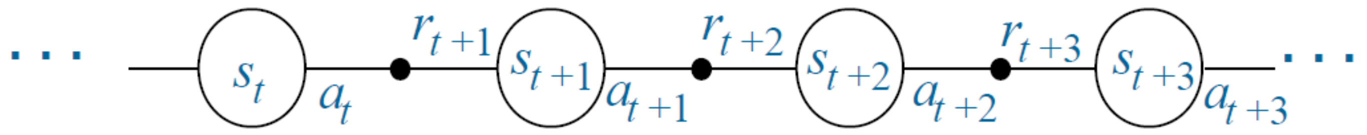


Non associative versus associative learning

- N-Armed bandit problem is single stage or state less i.e. Non associative
- An RL problem usually involves a sequence of decisions i.e. Associative

The Agent-Environment Interface

- World: You are in state 34.
  - Your immediate reward is 3. You have 2 actions.
- Robot: I’ll take action 2.
- World: You are in state 77.
  - Your immediate reward is 0. You have 3 actions.
- Robot:I’ll take action 1.
- World: You are in state 34. (again)
  - Your immediate reward is 1. You have 2 actions. …



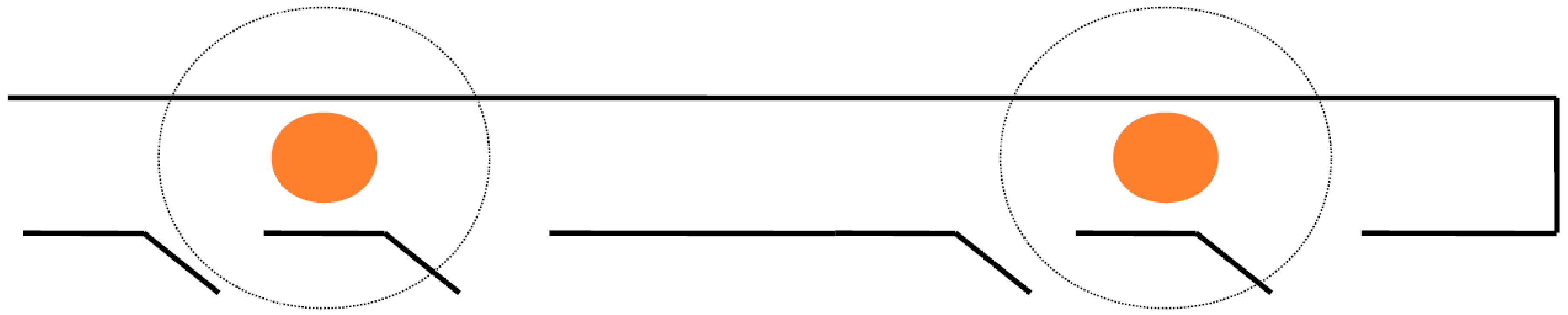
The Markov Property

- A state should summarize past sensations so as to retain all "essential" information, which is the information needed in order to behave optimally, i.e., it should have the Markov
- Property:

$$P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, X_{n-2} = i_{n-2}, \ldots)$$
$$= P(X_{n+1} = j | X_n = i)$$

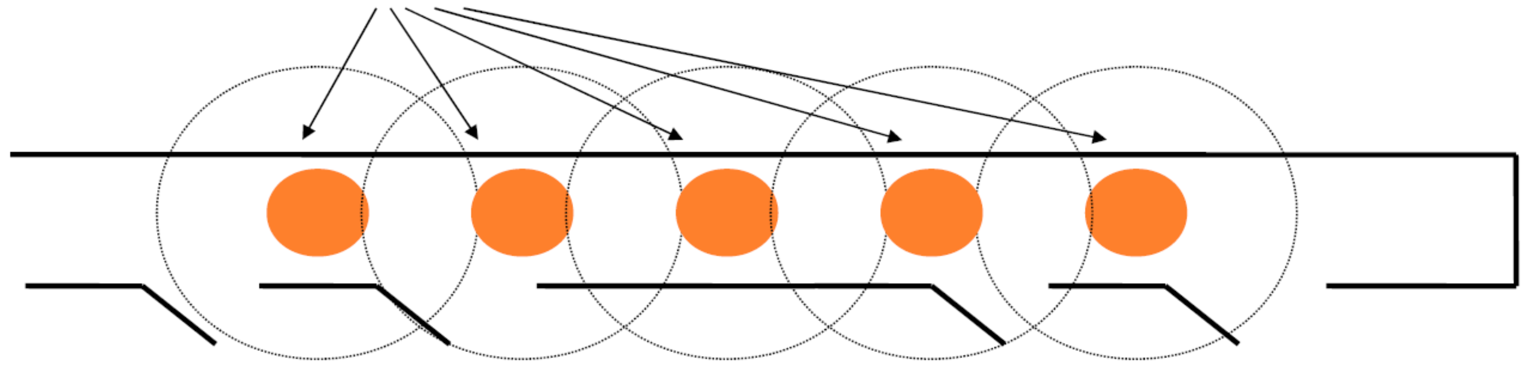
- Knowing how you got in this state is not giving you extra information

A non-example

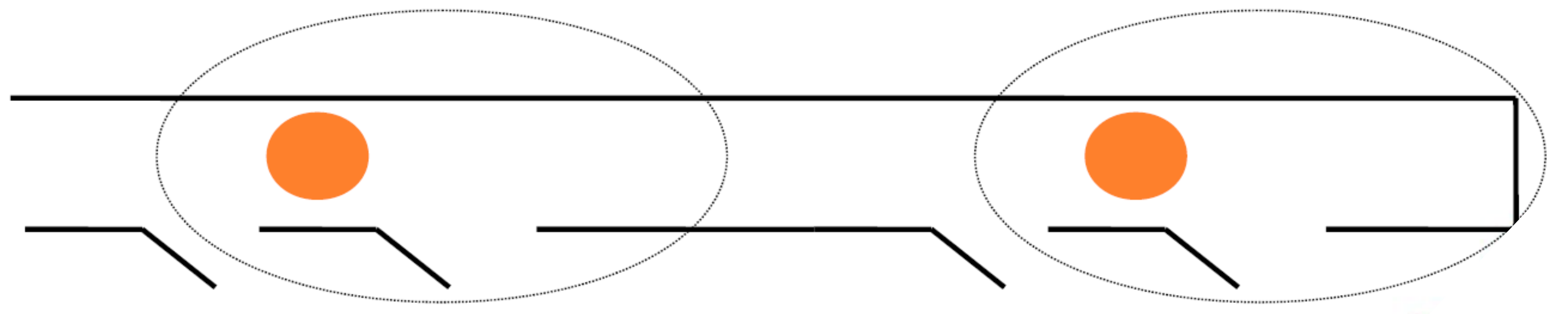


Make it Markovian

Augment state with history



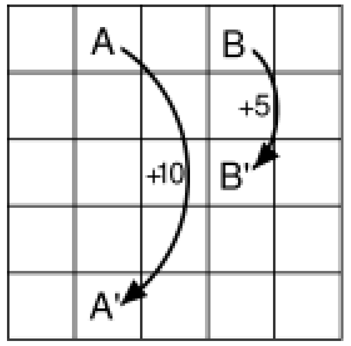
Augment sensor input



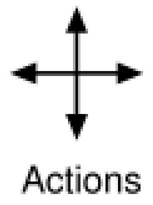
Markov Decision Processes

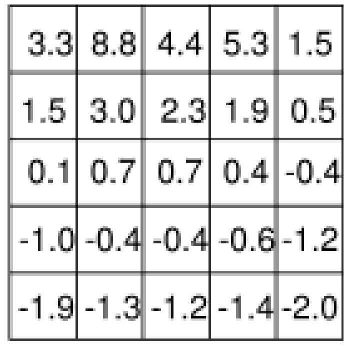
- If a reinforcement learning task has the Markov Property, it is basically a Markov Decision Process (MDP).
- If state and action sets are finite, it is a finite MDP.
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step "dynamics" defined by **transition probabilities**
  - **reward probabilities**

Value functions



(a)





3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

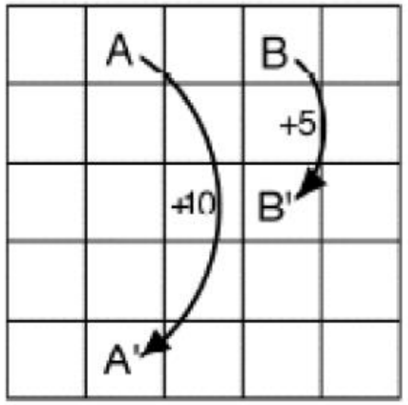
- Almost all reinforcement learning algorithms are based on estimating **value functions**—functions of states (or of state-action pairs) that estimate how *good* it is for the agent to be in a given state (or how good it is to perform a given action in a given state).
- The notion of “how good” here is defined in terms of future rewards that can be expected.
- Rewards the agent can expect to receive in the future depend on what actions it will take → value functions are defined with respect to particular **policies**.
- Calculated using policy and value iteration methods

Gridworld example

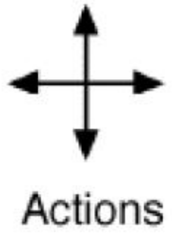
Actions: `north`, `south`, `east`, `west`; deterministic.

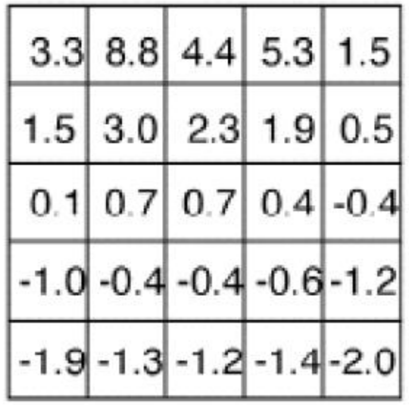
If would take agent off the grid: no move but reward = –1

Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



(a)





3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

State-value function for equiprobable random policy; $\gamma = 0.9$

Q-learning and action-value functions

- **Model-free** RL technique
- **Action value function (Q):** ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter

$$Q : S \times A \to \mathbb{R}$$

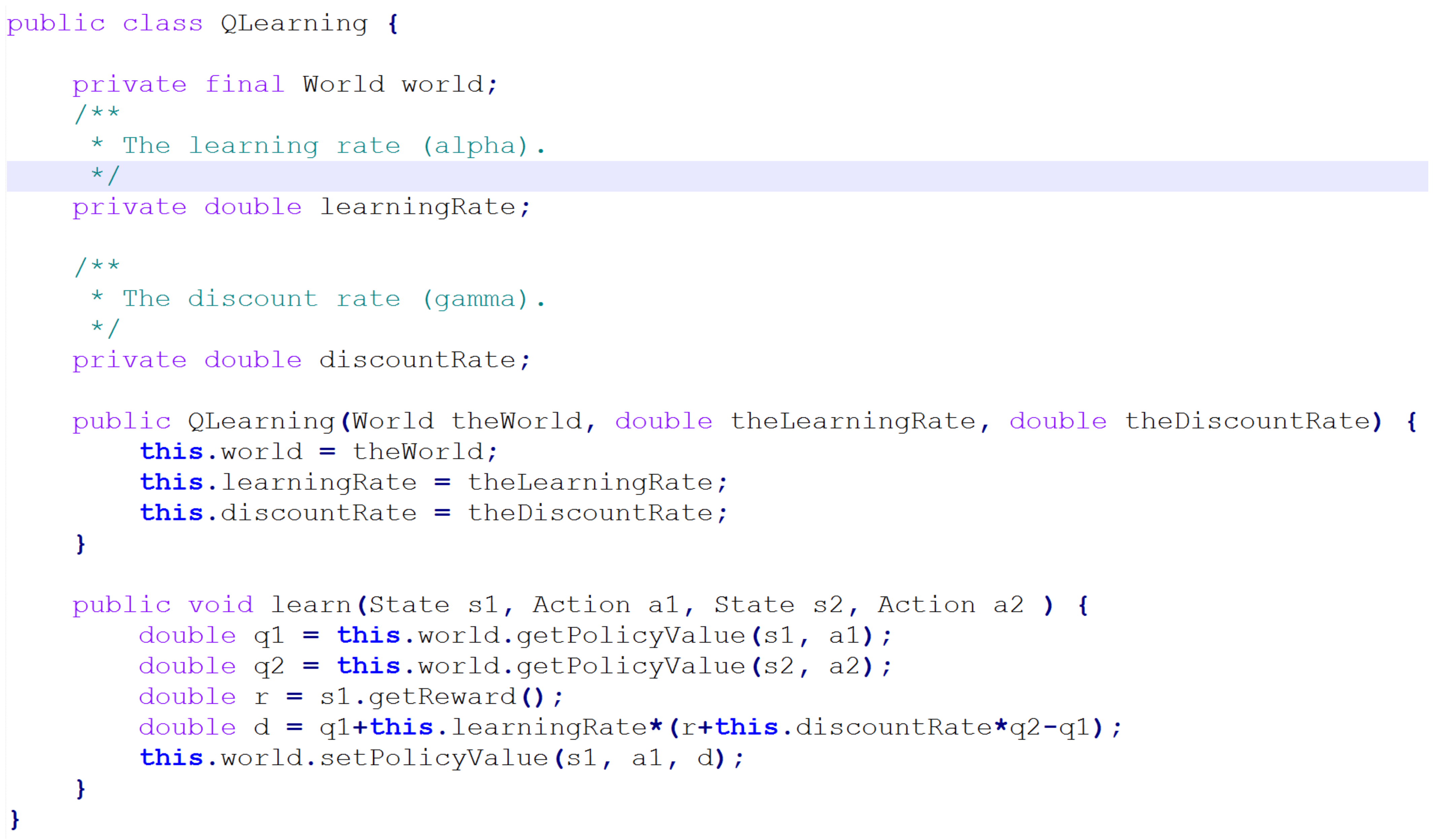
- Shown to learn the optimal action-selection policy for any given (finite) Markov decision process (MDP).

Q-learning algorithm

$$Q_{t+1}(s_t,a_t) = \underbrace{Q_t(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t,a_t)}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t,a_t)}_{\text{old value}} \right)$$

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$;
    until $s$ is terminal
```

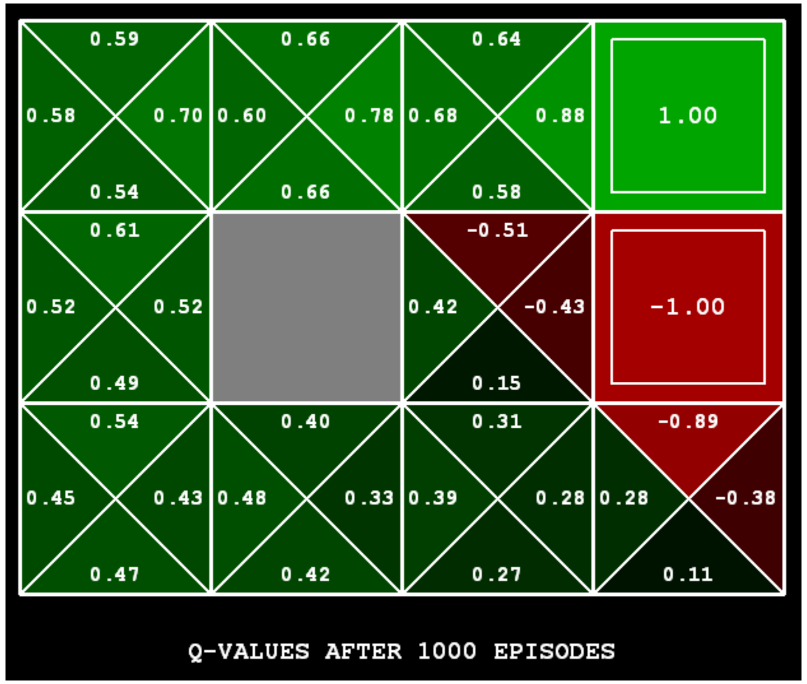
Q-learning example in Java



```java
public class QLearning {

    private final World world;
    /**
     * The learning rate (alpha).
     */
    private double learningRate;

    /**
     * The discount rate (gamma).
     */
    private double discountRate;

    public QLearning(World theWorld, double theLearningRate, double theDiscountRate) {
        this.world = theWorld;
        this.learningRate = theLearningRate;
        this.discountRate = theDiscountRate;
    }

    public void learn(State s1, Action a1, State s2, Action a2 ) {
        double q1 = this.world.getPolicyValue(s1, a1);
        double q2 = this.world.getPolicyValue(s2, a2);
        double r = s1.getReward();
        double d = q1+this.learningRate*(r+this.discountRate*q2-q1);
        this.world.setPolicyValue(s1, a1, d);
    }
}
```

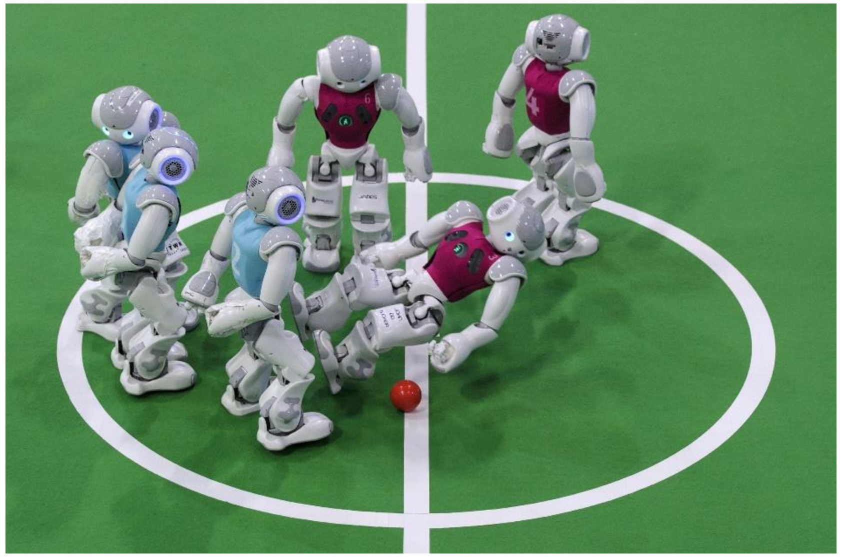
Q-learning parameters

- Initial Q-values
  - Optimistic: encourages exploration
- Learning rate α ∈ [0,1] - how fast do you learn?
  - Fixed value, e.g. 0.1
  - Varying value from 1→0
- Discount factor Υ∈ [0,1]
  - the importance of future rewards (e.g. 0.9)
- How to store the Q-values?
  - A simple table
  - If the state space is to large, Q-learning is often combined with supervised learning techniques as a function approximator (e.g. artificial neural networks)



Multi-agent reinforcement learning

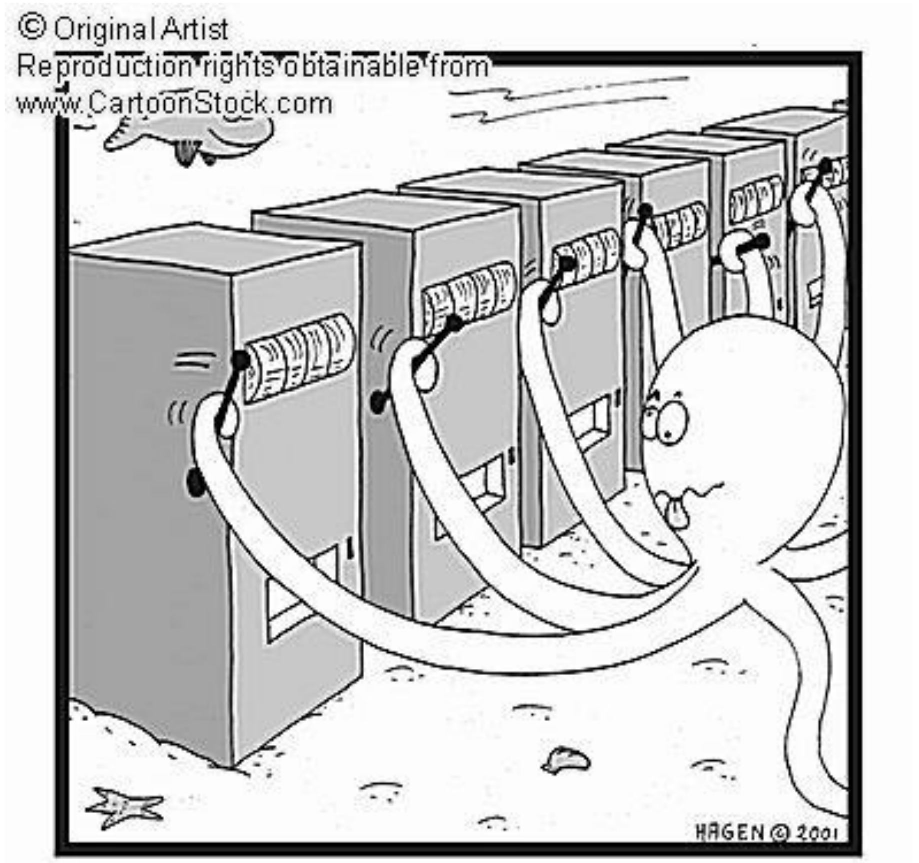
- How can multiple RL agents learn to coordinate on joint optimal/efficient/fair solutions?
- Challenges:
  - Each agent has just incomplete information
  - Each agent is restricted in its capabilities
  - System control is distributed
  - Data is decentralized
  - Computation is asynchronous
  - Communication is not for free, often unreliable and delayed



Learning automata

Learning Automata (LA)

- Simple RL method
- Policy iteration method
- Probability vector *p*
- One probability value for each action
- Update scheme
  - <u>LRI</u>, LRP, LRϵP
- Learning rate

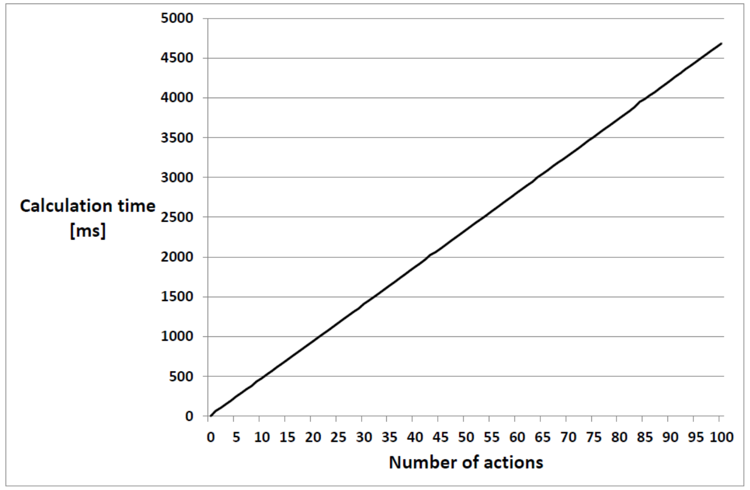


Compulsive gambling

Linear update scheme

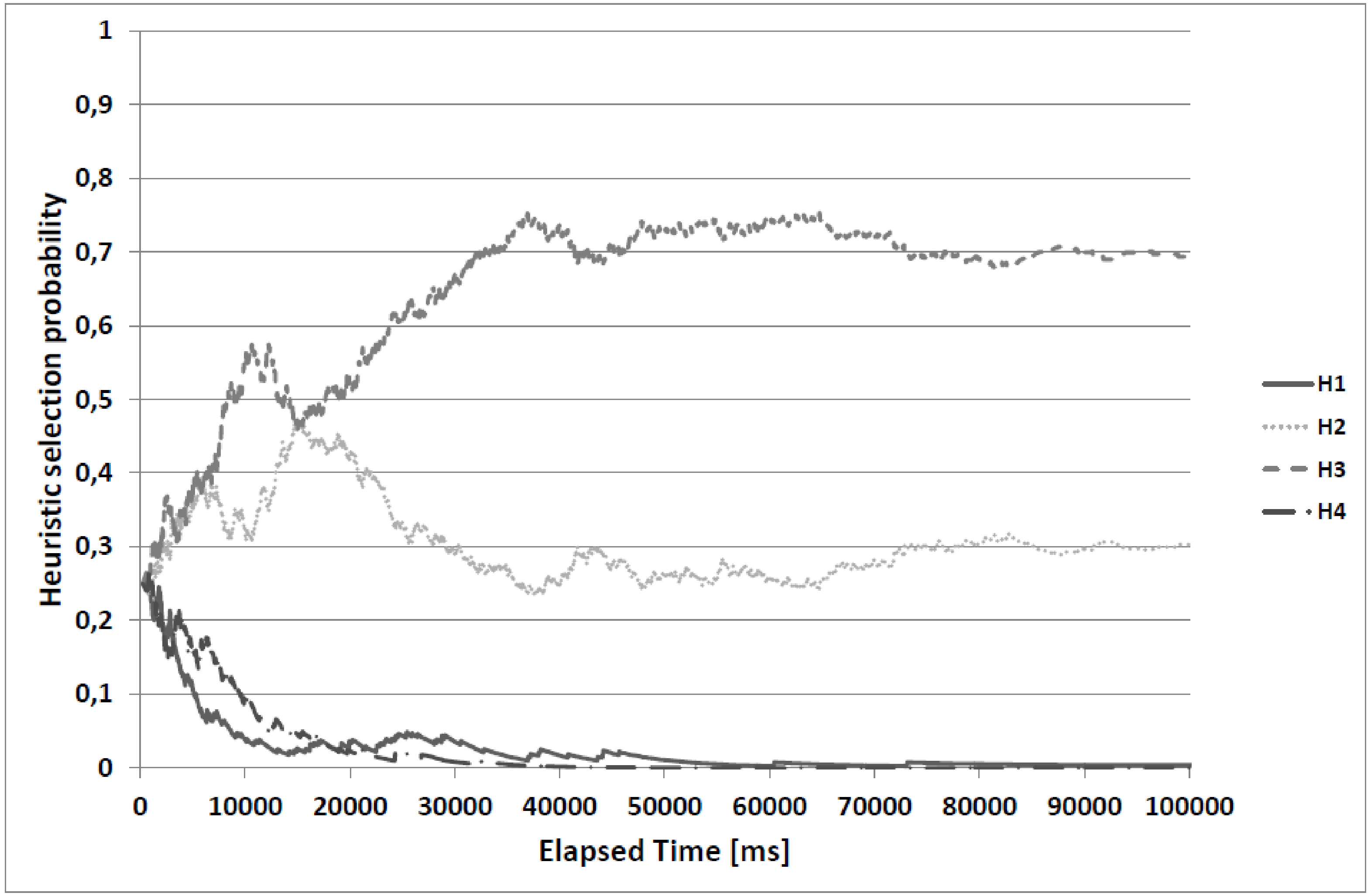
- Linear reward-inaction:

$$
\begin{aligned}
p_m(t+1) &= p_m(t) + \alpha_{reward}\beta(t)(1 - p_m(t)) \\
&\quad \text{if } a_m \text{ is the action taken at time } t \\
p_j(t+1) &= p_j(t) - \alpha_{reward}\beta(t)p_j(t) \\
&\quad \text{if } a_j \neq a_m
\end{aligned}
$$



Calculation time for
1 million selections and updates
LRI learning automaton
PC:Intel Core I7-2600 3.4Ghz CPU.

Evolution of the probabilities in time


Heuristic selection probability
Elapsed Time [ms]
H1
H2
H3
H4