LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

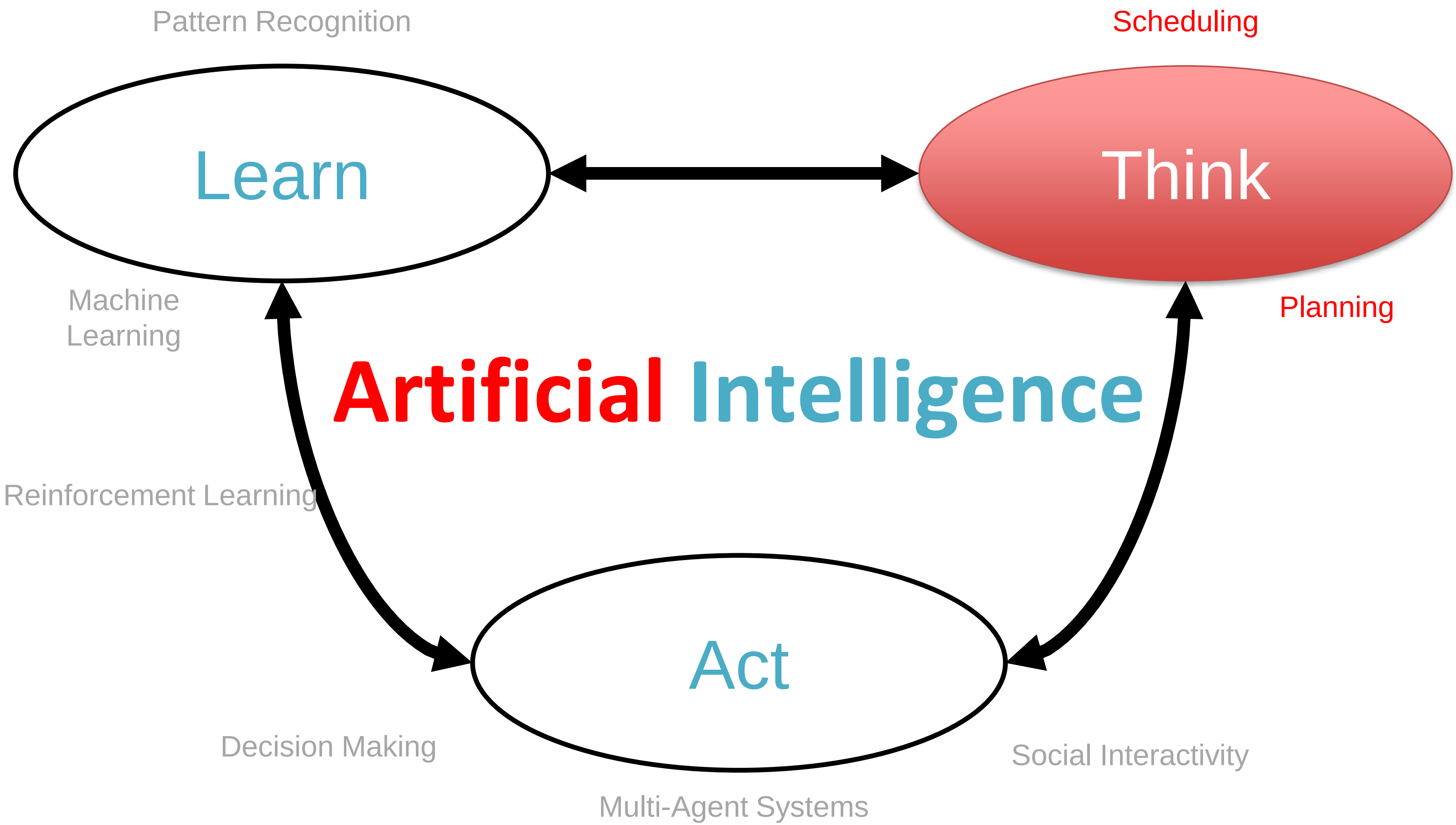
Praktikum Autonome Systeme

Reinforcement Learning

Prof. Dr. Claudia Linnhoff-Popien
Thomy Phan, Andreas Sedlmeier, Fabian Ritz
http://www.mobile.ifi.lmu.de

WiSe 2020/21

Recap: Automated Planning


Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems

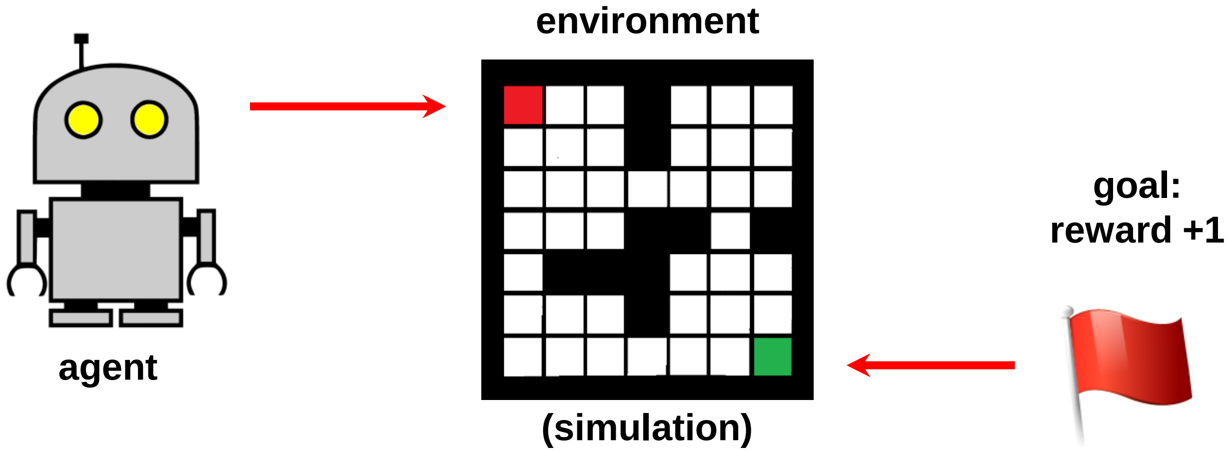
Overview

Multi-Armed Bandit

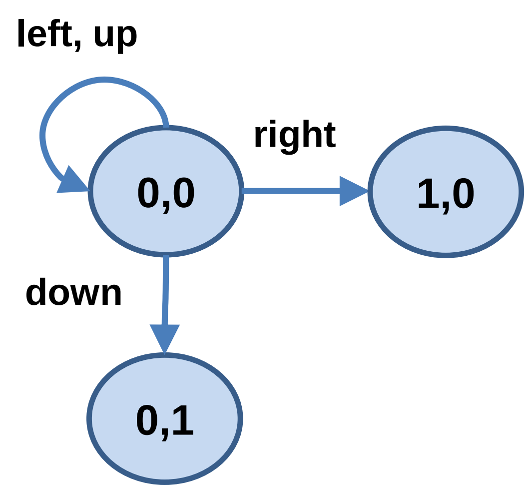
Black Jack

How many distinct states do you need to model these games?

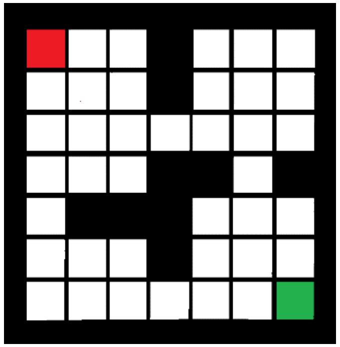
Sequential Decision Making

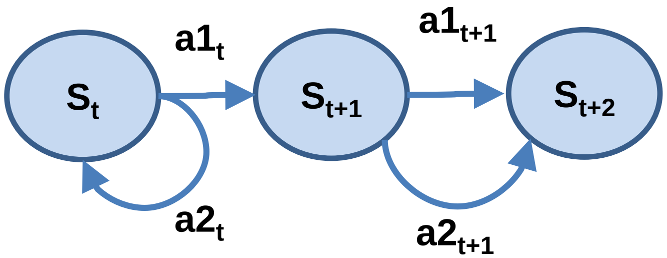


- given an an environment with:
  - multiple states
  - multiple actions with
    - different observations
    - different rewards
    - (long term) consequences



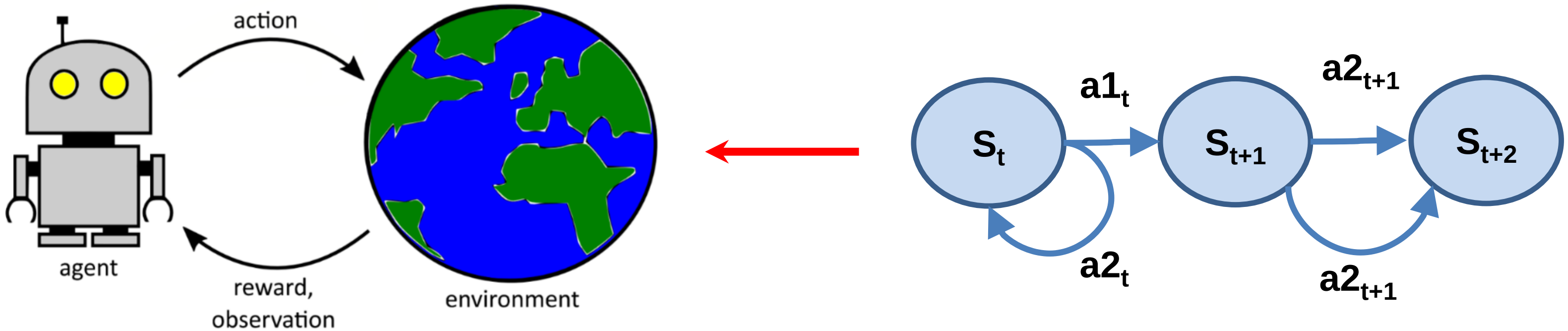
Sequential Decision Making





- we want to:
  - autonomously select actions to solve a (complex) task
  - maximize the **expected cumulative reward** for each state
- **policy** $\pi: \mathcal{S} \rightarrow \mathcal{A}$ represents the behavioral strategy of an agent:
  - policies may also be stochastic $\pi(a_t|s_t) \in [0,1]$

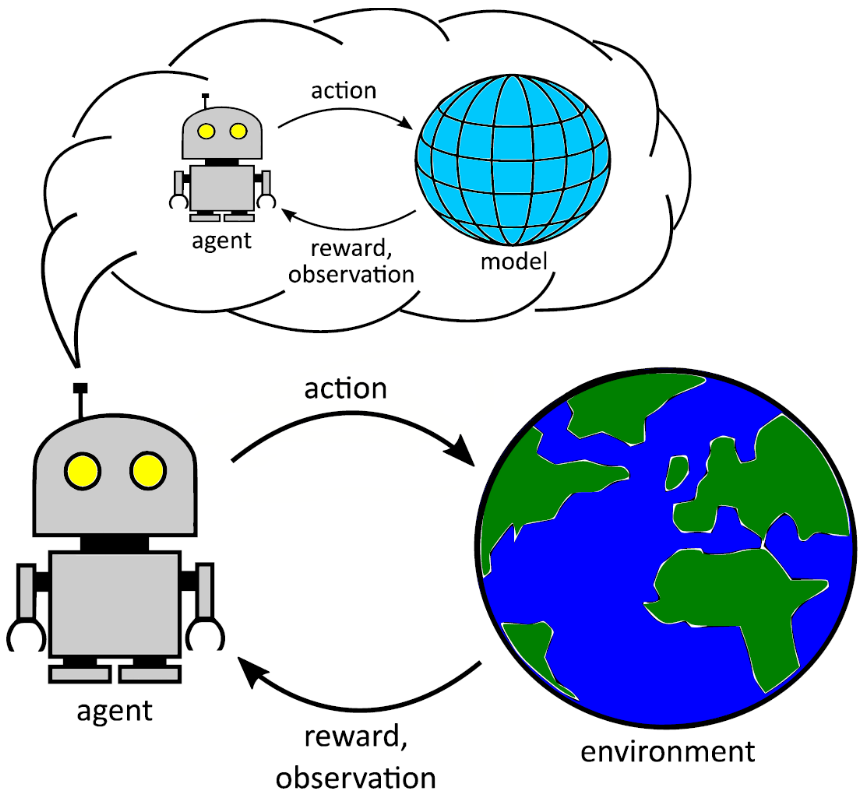
Sequential Decision Making



- a **Markov Decision Process** (MDP) is defined as $\mathrm{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$
  - $\mathcal{S}$ is a (finite) set of states
  - $\mathcal{A}$ is a (finite) set of actions
  - $\mathcal{P}(s_{t+1}|s_t, a_t) \in [0, 1]$ is the probability for reaching $s_{t+1} \in \mathcal{S}$ when executing $a_t \in \mathcal{A}$ in $s_t \in \mathcal{S}$
  - $\mathcal{R}(s_t, a_t) \in \mathbb{R}$ is a reward function

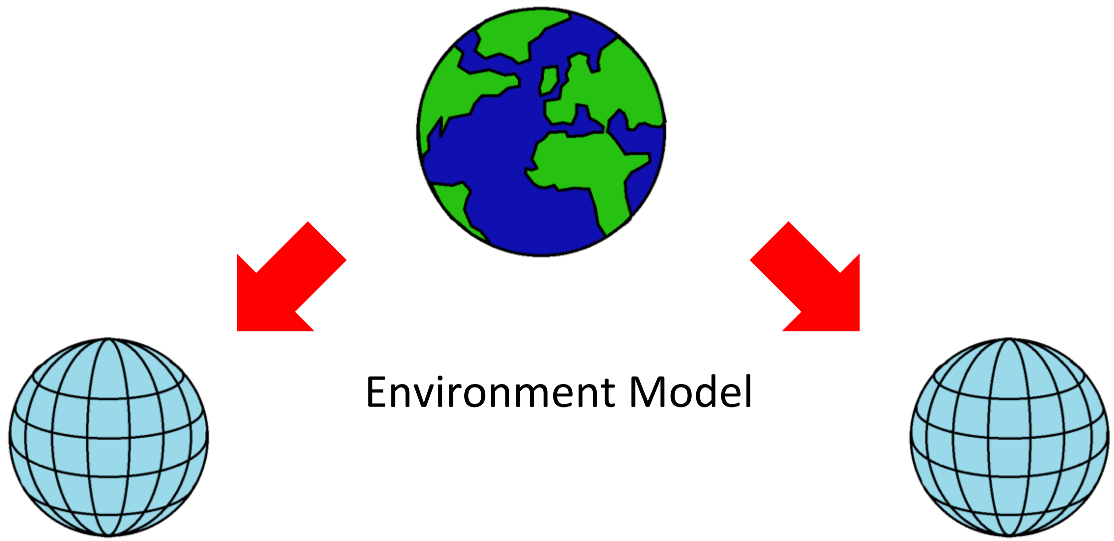
Automated Planning

- **Goal:** Find (near-)**optimal policies** π^* to solve complex problems
- Use (heuristic) lookahead search on a **given model** $\widehat{M} \approx M$ of the problem



Explicit Model vs. Generative Model

Real Environment



Environment Model

Explicit
(Full MDP)

Generative
(Generate Samples)

Dynamic Programming

- Uses Transition Probabilities $P(s_{t+1}|s_t, a_t)$
- Policy Iteration / Value Iteration

Monte Carlo Planning

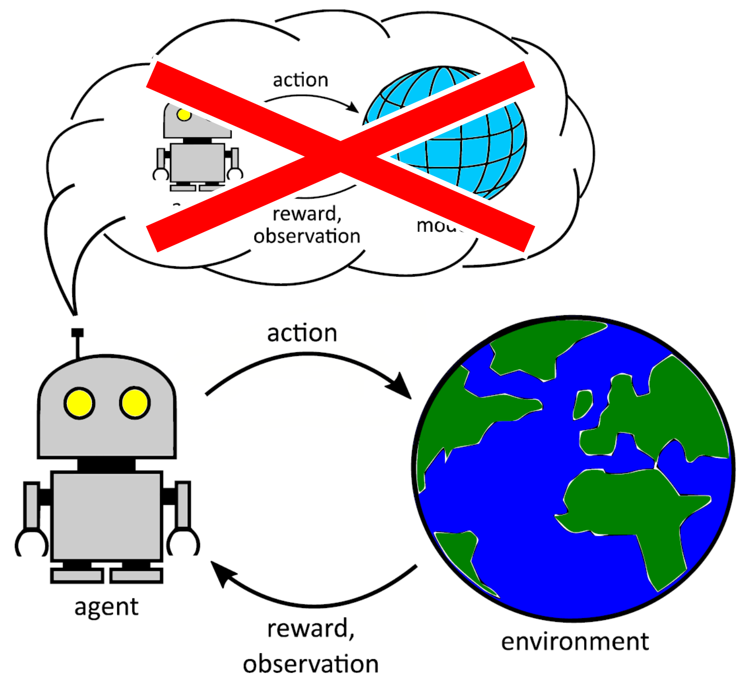
- Uses Samples from the Model to approximate V* / Q*
- MC Rollout / MCTS

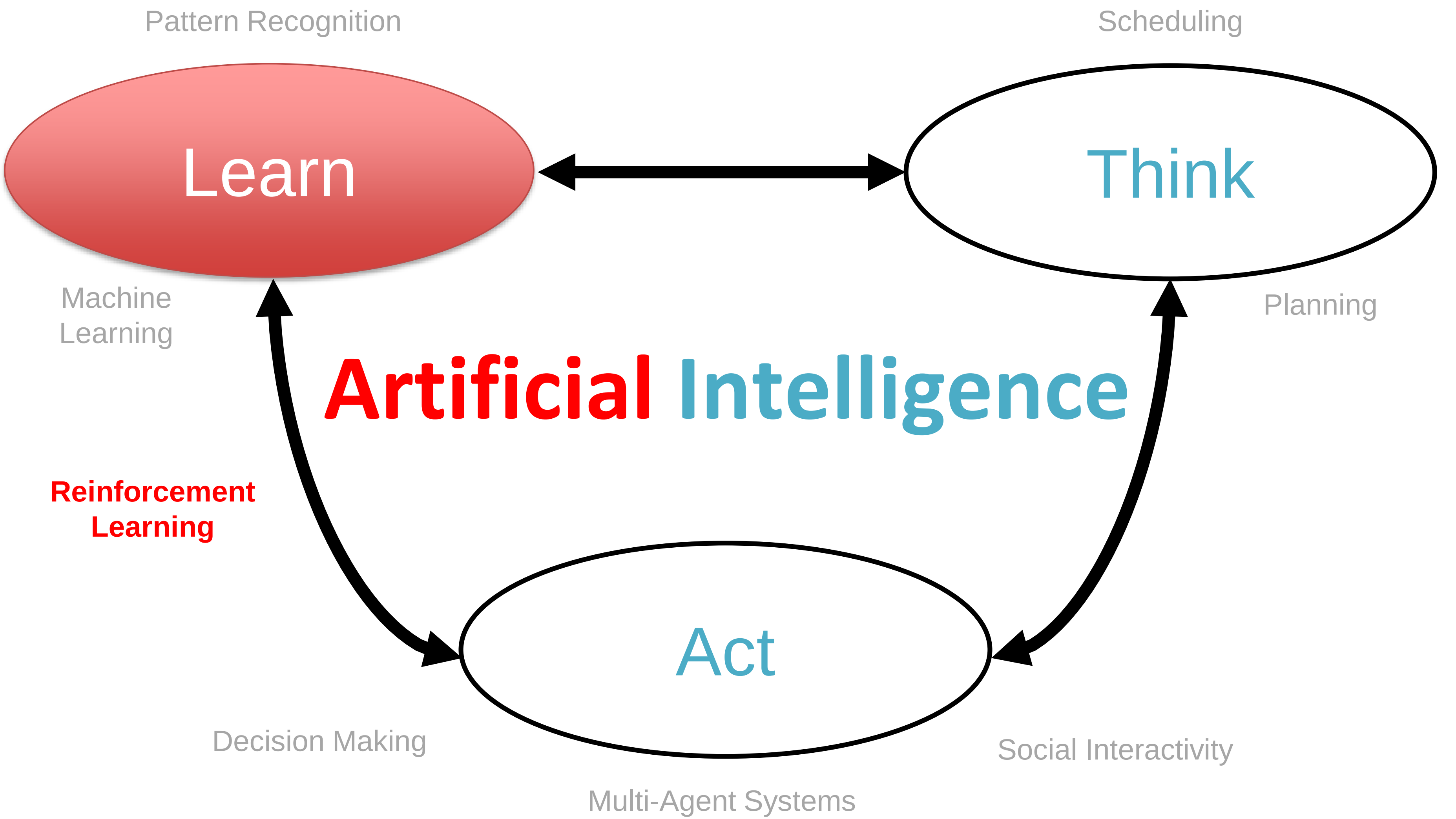
Model Free Reinforcement Learning

…and if we don't have a model?

Step1: Model Free Prediction

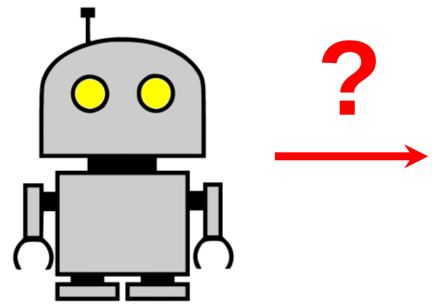
Step2: Model Free Control

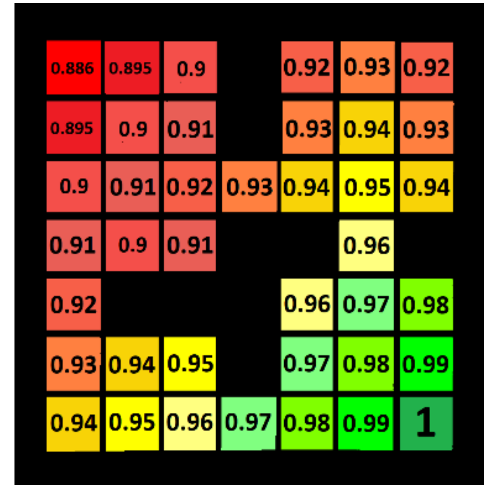



Pattern Recognition
Scheduling
Learn
Think
Machine Learning
Planning
Artificial Intelligence
Reinforcement Learning
Act
Decision Making
Social Interactivity
Multi-Agent Systems

Model Free Reinforcement Learning

…and if we don't have a model?





Step1: Model Free Prediction

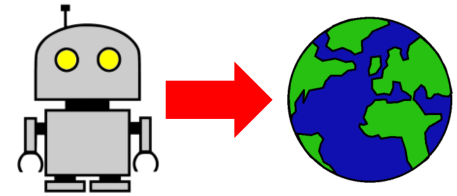
"How good is the policy π?"

"What is the value of state s (when using π)"?

- We have a policy π and want to evaluate it, i.e.:
- Estimate the Value Function V_π of an unknown MDP
- **We don't try to improve the policy yet!**
  This will be done later in "model free control"

Model Free Reinforcement Learning

Model Free Prediction



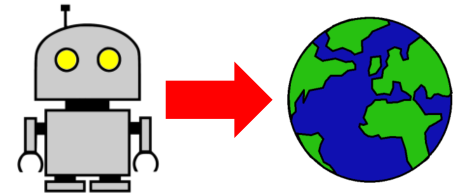
Evaluation Method 1: Monte Carlo (MC)

- **Perform MC rollouts directly in the real world**
  - Complete some episodes (i.e. use policy π)
  - Calculate the value as the mean of the returns

- **First-Visit Monte-Carlo Policy Evaluation**
  Goal: evaluate a state s
  - The **first** time a state is visiten in an episode
  - Increment a counter N(s) ← N(s) + 1
  - Increment the total return $S(s) \leftarrow S(s) + G_t$
    Recall: G_t is the total discounted reward:
    $$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$
  - Value is then estimated as $V(s) = \frac{S(s)}{N(s)}$

Model Free Reinforcement Learning

Model Free Prediction



- **First-Visit Monte-Carlo Policy Evaluation**
  - Value is then estimated as $V(s) = \frac{S(s)}{N(s)}$

This is not iterative!

If we want to update V(s) after every episode, the mean can also be calculated incrementally:

- use the **incremental mean**

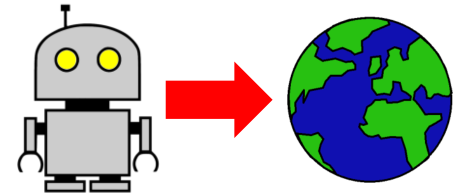
$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

- or the **running mean** (forgets old episodes)

$$V(s_t) \leftarrow V(s_t) + \propto (G_t - V(S_t))$$

Model Free Reinforcement Learning

Model Free Prediction



Evaluation Method 2: Temporal difference learning (TD)

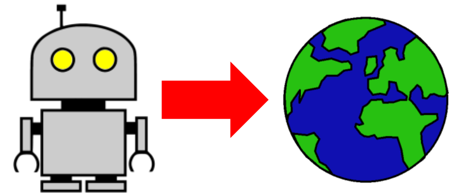
"Why only learn after the end of episodes?"

- TD learns from incomplete episodes (i.e. it *bootstraps*)
- Simplest TD: TD(0)
- Every step: Update the Value $V(S_t)$ toward the estimated return:
  - $R_{t+1} + \gamma V(S_{t+1})$
  - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$

- $V(s_t) \leftarrow V(s_t) + \propto (G_t - V(S_t))$ [Monte-Carlo]

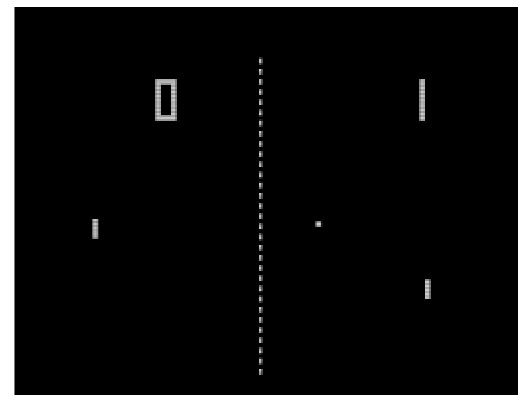
Model Free Reinforcement Learning

Model Free Prediction



What is better - MC or TD learning?

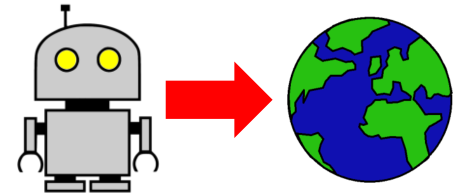
- No clear winner
- TD can learn before the end of episodes
  … or if there never are final ends
  How well do you play pong? =)

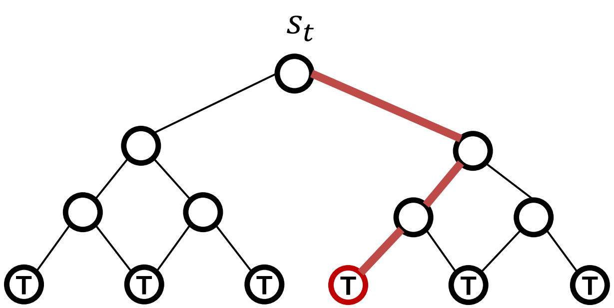


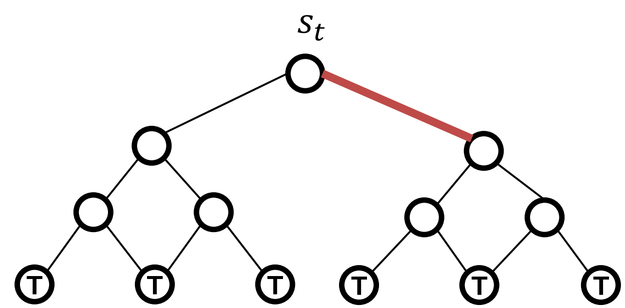
- MC has high variance, but NO bias
  – good convergence
- TD has low variance, but is biased
  – converges, but not always when using function approximation

Model Free Reinforcement Learning

Model Free Prediction







Monte-Carlo

$$V(s_t) \leftarrow V(s_t) + \propto (G_t - V(S_t))$$

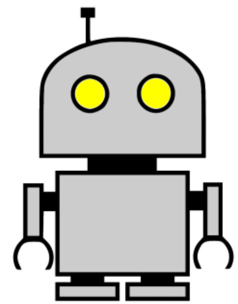
Temporal-Difference

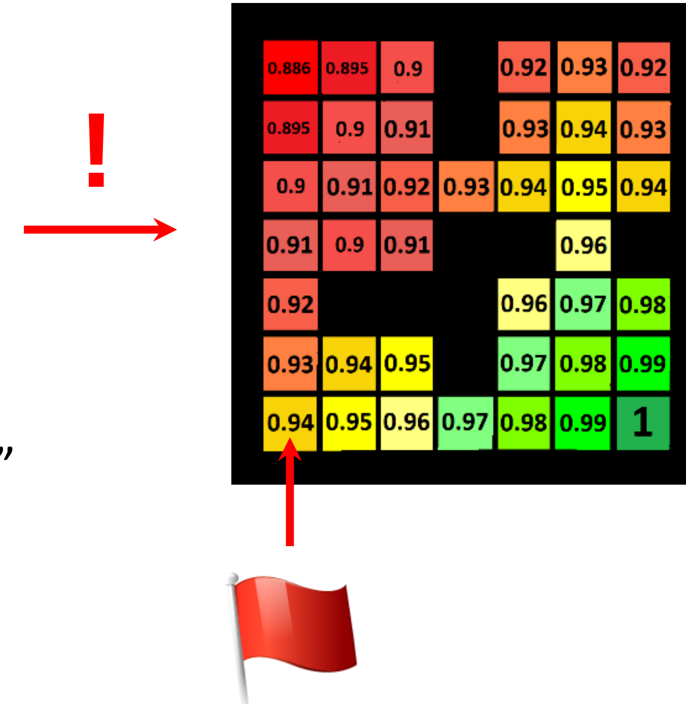
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

Model Free Reinforcement Learning

…and if we don’t have a model?

Step2: Model Free Control





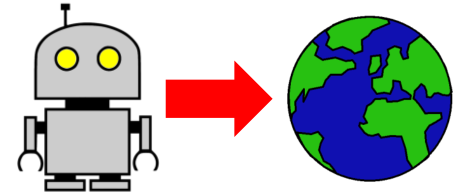
„Can we improve the policy without a model?”

We want to:

- improve a policy π
- optimize the value function of an unknown MDP

Model Free Reinforcement Learning

Model Free Control



Recap: Model-Free VS Model-Based Policy Iteration

- Computing improvements using V(s) requires the transition probabilities of the MDP:

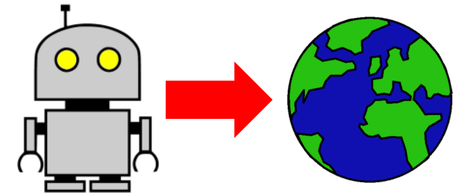
$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, R_s^a + \gamma P_{ss'}^a V(s')$$

- No model needed when using the Action-Value function Q:

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, Q(s, a)$$

Model Free Reinforcement Learning

Model Free Control

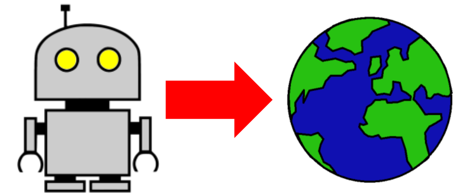


Pull Arm 1: Reward 0 | V(A1) = 0
Pull Arm 2: Reward +1 | V(A2) = +1
Pull Arm 2: Reward +3 | V(A2) = +2
Pull Arm 2: Reward +2 | V(A2) = +2

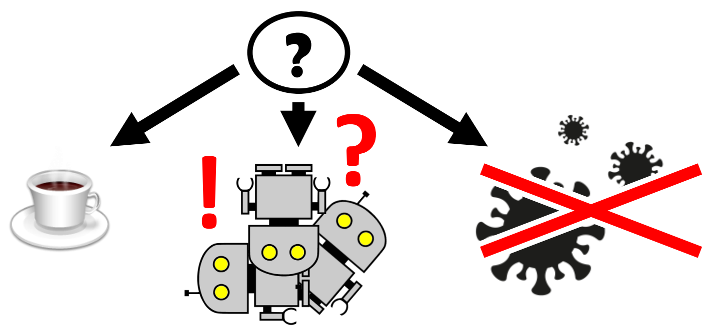
„Certain that Arm 2 (A2) is the best arm?“

Model Free Reinforcement Learning

Model Free Control



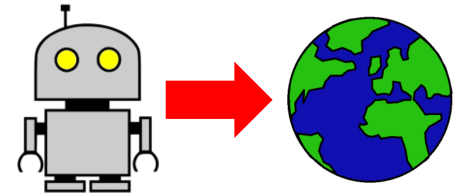
- to approximate $Q^*(s_t, a_t)$, we need to explore sufficiently
  - otherwise overfitting on „well-known“ states
  - unexpected / undesirable behaviour on „new“ states
  - detect / adapt to changes in the environment



- multi-armed bandit based exploration
  - example: **ϵ-greedy**, $\epsilon > 0$

$$\text{with probability} \begin{cases} \epsilon, \text{ select randomly} \\ 1 - \epsilon, \text{ select action } a_t \text{ with highest } Q(s_t, a_t) \end{cases}$$

Model Free Reinforcement Learning
Model Free Control



ε-greedy Monte-Carlo Control

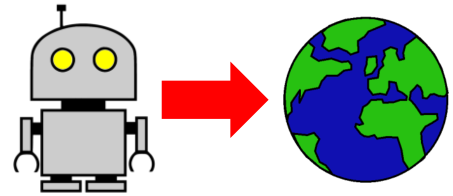
- Same idea as before perform some rollouts
- **Evaluate**, i.e. calculate $Q(S_t, A_t)$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

- **Improve** using the new Q:

$$\pi \leftarrow \varepsilon{-}greedy(Q)$$

Model Free Reinforcement Learning
Model Free Control



"Why not use TD instead and update every step?"

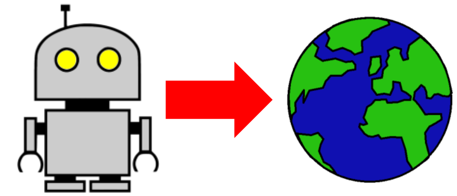
- **Sarsa** (**S**tate, **A**ction, **R**eward, **S**tate', **A**ction')

$$Q(S,A) \leftarrow Q(S,A) + \propto (\underbrace{\boldsymbol{R + \gamma Q(S',A')}}_{}) - Q(S,A))$$

MC VS Sarsa: Empirical G_t is replaced by current reward R + discount * estimated return from the next step (taking action a')

Model Free Reinforcement Learning

Model Free Control



“Why not consider more than the last reward?”

- **N-Step Sarsa**

n=1 (**Sarsa**) $q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1})$

n=2 $q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2})$

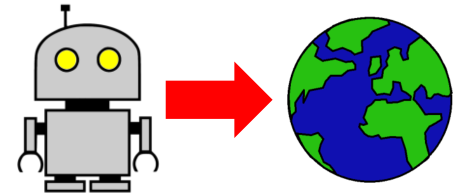
…

n=∞ (**MC**) $q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n Q(S_{t+n})$

$$Q(S, A) \leftarrow Q(S, A) + \propto (q_t^{(n)} - Q(S, A))$$

Model Free Reinforcement Learning

Model Free Control

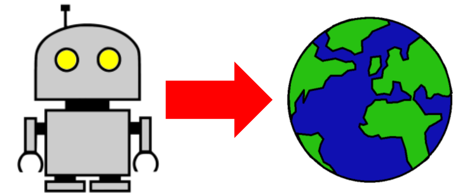


Sarsa: Pseudo Code (Tabular)

$$Q(S,A) \leftarrow Q(S,A) + \propto (R + \gamma Q(S',A') - Q(S,A))$$

```
Repeat n_episode times:
        s = reset environment
        a = policy(s)
        For t in max_step_times:
                s‘,r,done = environment.step(a)
                a‘ = policy(s‘)
                td_target = r + discount*Q(s‘,a‘)
                td_error = td_target – Q(s,a)
                Q[s,a] = Q(s,a) + ∝ * td_error
                s = s‘;
                a = a‘
                if done or t == max_step_times:
                        break
```

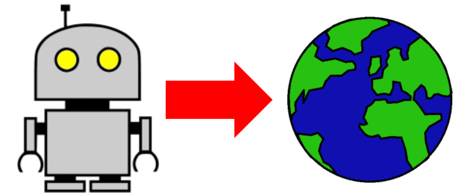
Model Free Reinforcement Learning
Model Free Control



On-Policy VS Off-Policy Learning

- Notice how both MC Control and Sarsa used the current policy to choose the next action
- We can also learn off-policy:
  - Choose the next action using one policy $\mu(a|s)$
  - But evaluate using a different policy π and it‘s $q_\pi(s, a)$
- This way, we can learn from observing others
- Or re-use experience collected using an old policy

Model Free Reinforcement Learning
Model Free Control



Sarsa:

$$Q(S,A) \leftarrow Q(S,A) + \propto (R + \gamma Q(S',A') - Q(S,A))$$

Off-Policy learning of the action-value function:

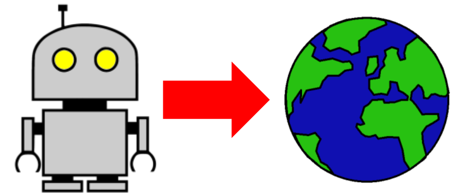
Q-Learning

$$Q(S,A) \leftarrow Q(S,A) + \propto (R + \gamma \max_{a'} Q(S',a') - Q(S,A))$$

Sarsa VS Q-Learning: The successor action a‘ is not taken based on the behaviour policy but greedy

Model Free Reinforcement Learning

Model Free Control



Q-Learning: Pseudo Code (Tabular)

$$Q(S,A) \leftarrow Q(S,A) + \propto (R + \gamma \max_{a'} Q(S',a') - Q(S,A))$$

```
Repeat n_episode times:
        s = reset environment
        For t in max_step_times:
                a = policy(s)
                s‘,r,done = environment.step(a)
                td_target = r + discount*max(Q(s‘))
                td_error = td_target – Q(s,a)
                Q[s,a] = Q(s,a) + ∝ * td_error
                s = s‘
                if done or t == max_step_times:
                        break
```

Questions?