LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND STATISTIK
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME UND DATA MINING

Lecture Notes for

# Deep Learning and Artificial Intelligence

Winter Semester 2024/2025

## Sequential Decision Problems and autonomous Agents



DBS

# AI and Decision Making

**What is behaviour?**

The reaction of a subject or an object to its environment:

- passive: a stone falling to the ground
- active: a mosquito flying into the light bulb

**What makes behaviour intelligent?**

- act like a human (imitation learning)
- pursue a certain goal (optimize expected outcome)

⇒ Decide on an action to improve on a goal.

# Prediction and Decision Making

***How to make decisions to reach a goal?***

- What do you know about the current situation?
- What are your options?
- Which option is the best?

***Problems***:

- Parts of your current situation might be unknown or not modelled
- Considering all options is often not possible
- Considering all possible impacts of choosing an option is often impossible.
- Your goal might not be reachable with one decision.

# Why is this connected to Machine learning?

**Uncertain impacts :**

- How does the environment react?
- How well do we perform the selected action?

**Uncertain costs and rewards:**

- We need to find out if and when an action pays off.
- We cannot precisely predict uncertain factors like waiting times, prices, congestion etc.

**Uncertain situation:**

- We only perceive some input about our situation, e.g., sensor data, player view, ...
- We might misinterpret a situation.

# Sequential Decision Making

Machine learning can help to reduce these uncertainties, but can it do more?

- Observe humans or other intelligent agents and imitate their behaviour
- Observe behaviour and learn the outcomes and long-term rewards of decisions

$\Rightarrow$Directly learn the best action in any situation.

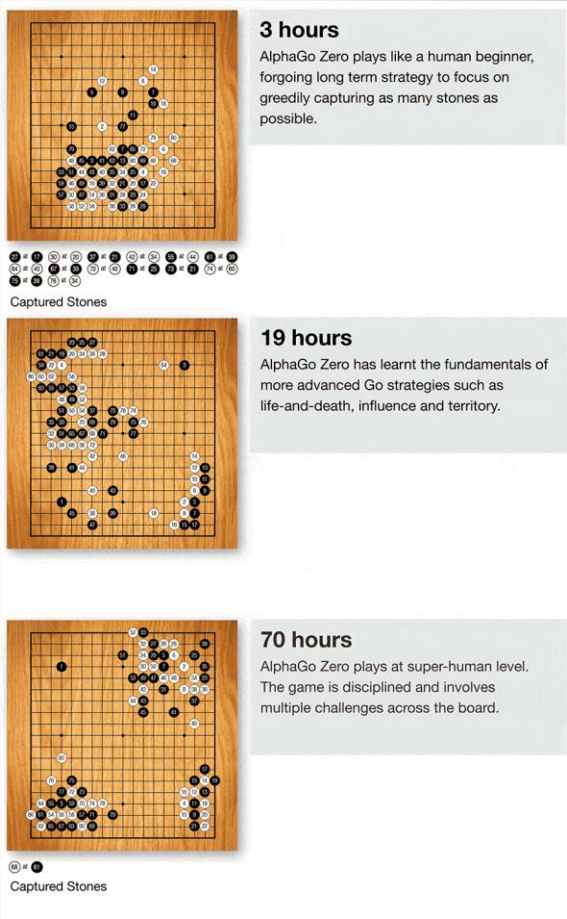
***This is the domain of Reinforcement Learning and training autonomous agents.***

# Applications Sequential Decision Making

- **AI for Games**
  - Competitive agents: Alpha Go, Atari Games, Chess,..
  - Realistic opponents: football simulations,
- **Controlling Robots and Systems**
  - Autonomous Cars
  - Drones
  - Industrial Robots
  - Power stations
- **Communicate with people**
  - Chatbots for Technical Support
  - Ordering Assistants (e.g. reservations)
  - Internet advertisement
- **Build intelligent assistant systems**
  - planning trips and routes
  - optimise processes
  - manage financial portfolio

# Reinforcement Leaning for board games

- games are a classical domain of AI because opponents behave not just nondeterministic but antagonistic
- progress in games like chess, backgammon, and Poker.
- for a long time, Go was considered one of the biggest challenges due to its enormous branching factor
- In Jan 2016: DeepMind challenged Grand Master Lee Sedol with a program called Alpha Go and won.
- Alpha Go combines tree search, reinforcement learning and Deep learning techniques and is trained on a database of Go matches.
- In 2017, DeepMind proposed Alpha Go Zero, which learns any board Game based on the rules only by playing against versions of itself. Alpha Go Zero outperforms previous versions of Alpha Go.

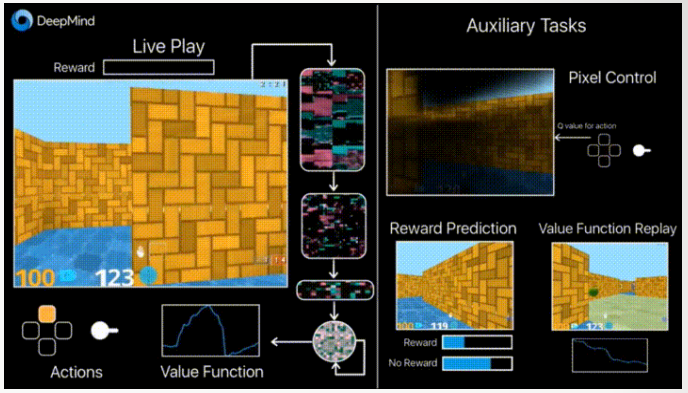



Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550, 354--.

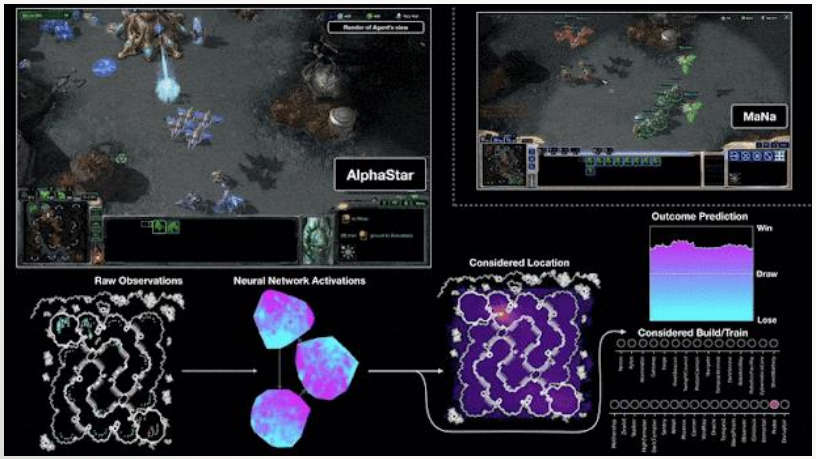
# Games as Testbed for AGIs

*AI is not built for a single purpose but receives images and general controls, just like humans.*

- DQN on Atari Games
  https://deepmind.com/research/publications/playing-atari-deep-reinforcement-learning/
- OpenAI Five (Dota2)
  https://blog.openai.com/openai-five/
- Alpha Star:
  https://openreview.net/pdf?id=Np8Pumfoty
  https://github.com/google-deepmind/alphastar

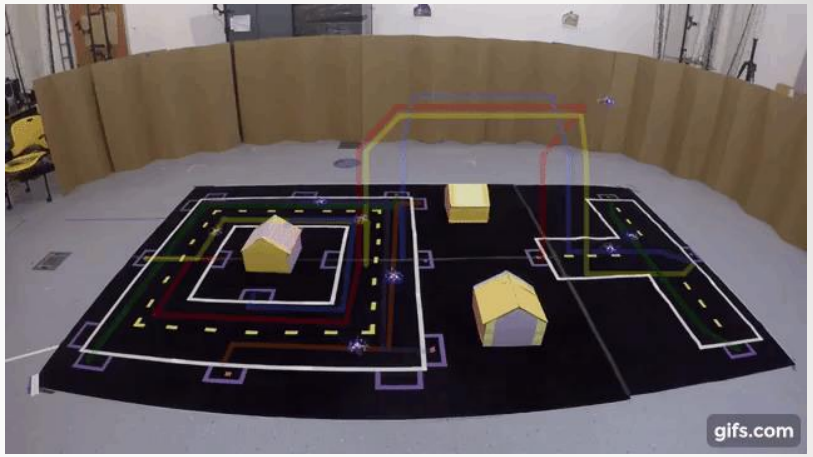



https://techxplore.com/news/2016-11-deepmind-boost-ai-unreal-agent.html




# RL in Control and Robotics

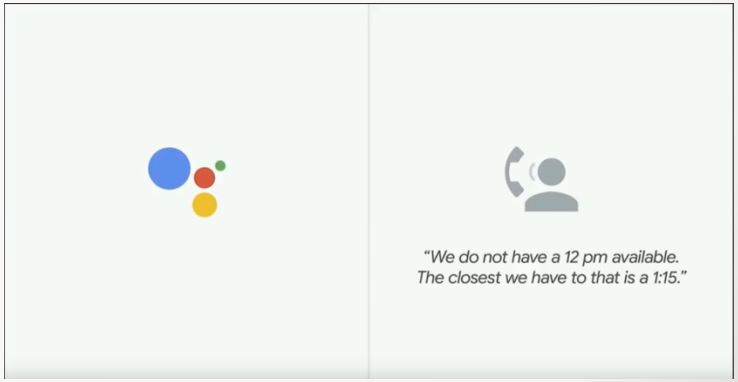
- Control robots and drones




https://robohub.org/drones-that-drive/

- Chatbots

For example Google's assistant




(https://www.youtube.com/watch?v=D5VN56jQMWM)

# Model-free and Model-based Approaches

- Model-based approaches a.k.a. planning
  - based on a complete model of the environment
  - computes rewards and actions within this models
  - optimises policy based on this model

- Model-free approaches
  - Assumes an interactive environment
  - Agent gets observations and selects actions
  - Environment process actions and sends new observation
  - The agents learn the best action based on the encountered interactions (episodes)

# Prediction and Control

- Prediction:

  Predicts how good an agent fares on a given problem.

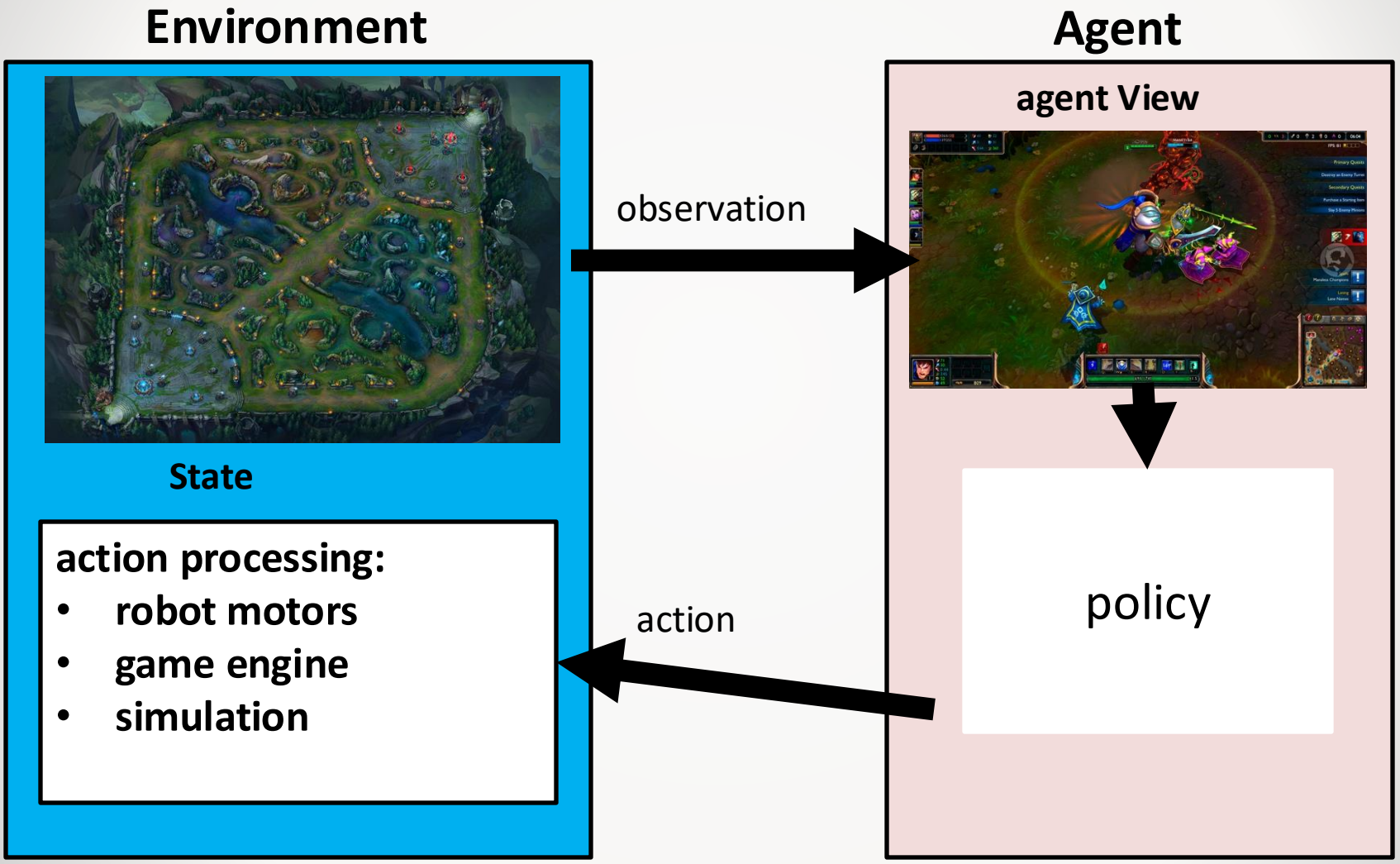
  $\Rightarrow$ Evaluate future behaviour

- Control:

  Optimises the action policy of an agent.

  $\Rightarrow$ Optimize future behaviour

# Agent and Environment



# Environment

**Represents the world in which the agent is acting.**
**(for example, a game, a simulation or a robot)**

- provides information about the state
  (for example, images, sounds, and game states)
- receives actions and applies them to alter its state

**Properties of Environments**

- partially/fully observable
- with known model/model-free
- deterministic/non-deterministic
- single vs. multi-agent
- competitive vs collaborative
- static/dynamic / semi-dynamic
- discrete/continuous (states and/or actions)

# Agents

An autonomous entity within the environment.

types of agents:

- simple reflex agent

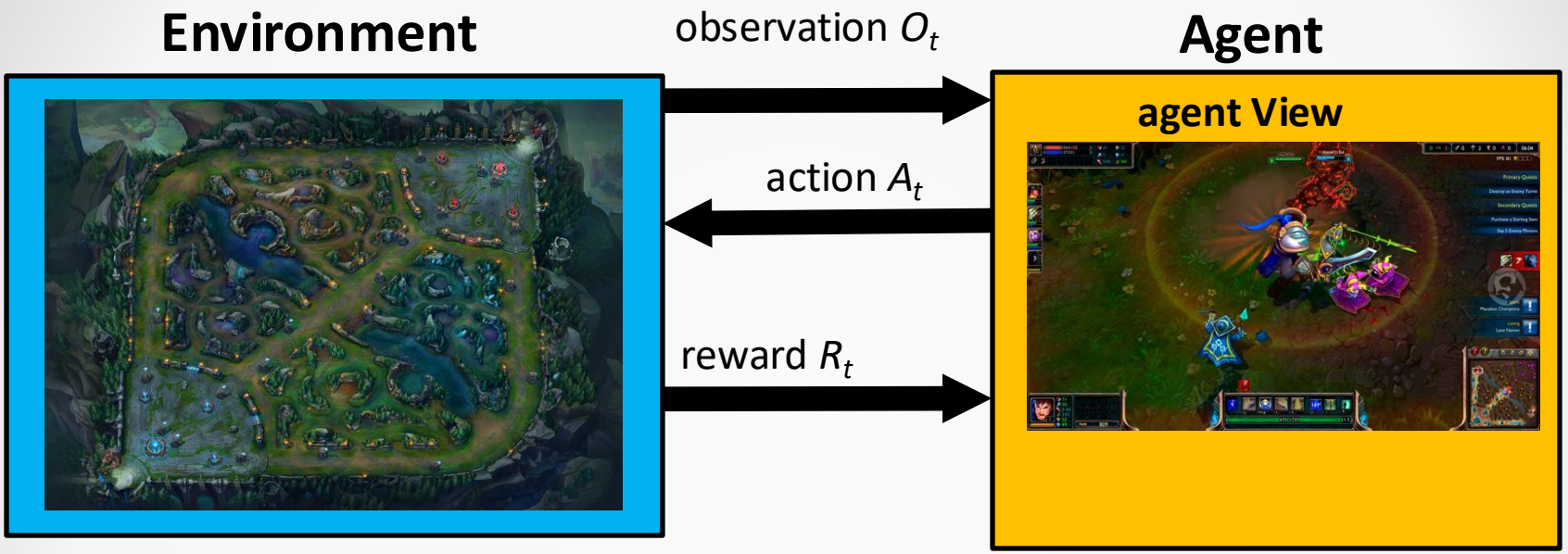
  ⇒ condition-action-rule
  (example: If car-in-front-is-braking, then initiate-braking.)
- model-based reflex agents (add internal state from history)
- goal-based agents (work towards a goal)
- **utility-based agents** (optimises rewards/minimises costs)
- **learning agents** (learns how to optimise rewards/costs)

# Sequential Decision Making

- behavior is a sequence of actions
- sometimes immediate rewards must be sacrificed to acquire rewards in the future
  ***for example***: invest today to receive a pension in old age
- short-term rewards might be very unlikely in many situations
  ***for example***: score a goal from your own half in football

⇒ intelligent behavior needs to plan ahead

# Agent and Environment




At each step t the environment:

- emits observation $O_t$
- emits scalar reward $R_t$
- receives and processes action $A_t$

At each step t the agent:

- selects action $A_t$
- receives observation $O_t$
- receives scalar reward $R_t$

# History and State

- What is the information we can base the decision on?
- History/Episode/Trajectory:
  Sequence of observation, action, reward

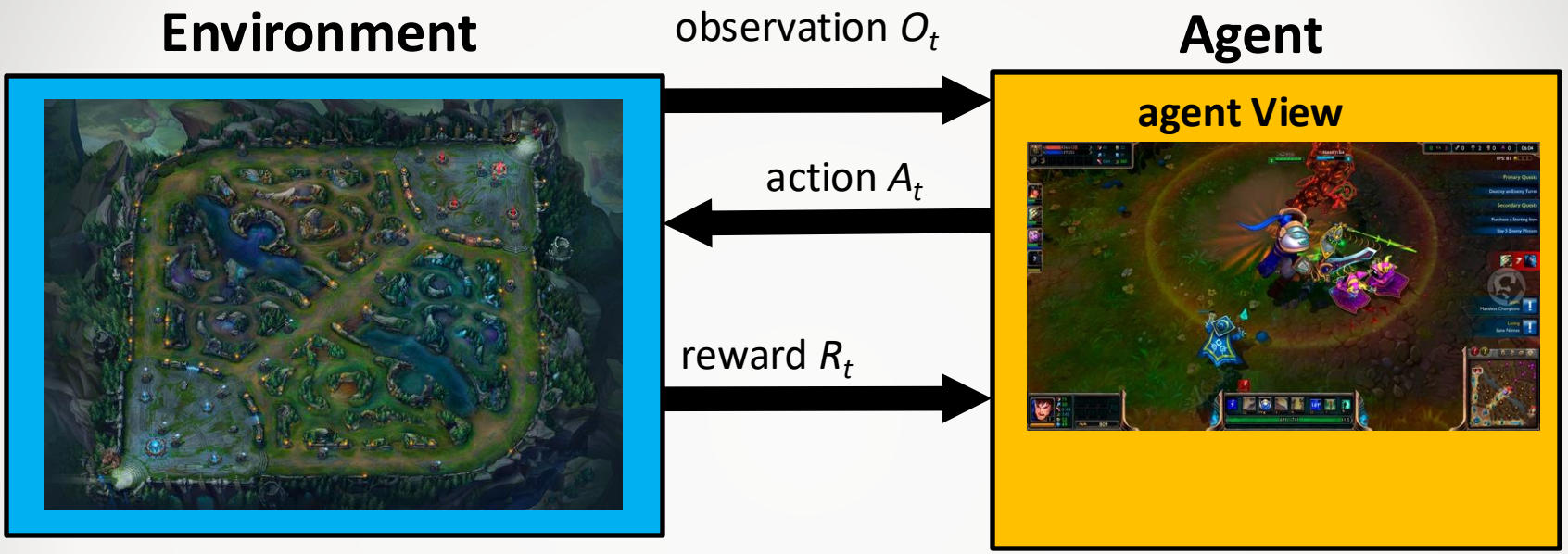
$$H_t = O_1, A_1, R_1, O_2, A_2, R_2, \ldots, A_{t-1}, R_{t-1}, O_t$$

- all observable variables until time *t*
- The history documents:
  - How the environment developed so far?
  - How the environment reacted to the last action?
  - Which action the agent selected at time *t* given the history until *t*?
- **State:** Used to decide about the next decision.

  formally: $S_t = f(H_t)$

  (everything relevant to estimate the impact of a decision)

# Environment and Agent State




Environment state $S_t^e$:

- Complete status of e
- *Observation/reward are derived from* $S_t^e$
- $S_t^e$ *is usually invisible to the agent*
- $S_t^e$ *might contain irrelevant information*

Agent state $S_t^a$:

- Internal representation within agent *a*
- used for making the next decision
- $S_t^a$ might be a function f($H_t$)

# Information State

A ***Markov state*** or ***information state*** contains all useful information from history.

**Definition**: A state $S_t$ is Markov if and only if

$$P(S_{t+1}|S_t) = P(S_{t+1}|H_t)$$

- everything to decide the future is contained in $S_t$

  $\Rightarrow$ $S_t$ is a sufficient statistic of the future
- the environment state $S_t^e$ is Markov
- the history $H_t$ is Markov

# Observability of the Environment

- Full observability: $O_t = S_t^e = S_t^a$
  the agent observes the complete environment state $S_t^e$
  = (fully observable) Markov Decision Process (MDP)

- Partially observability: $O_t \neq S_t^e$
  = partially observable Markov decision process (POMDP)
  $\Rightarrow$ agent must construct $S_t^a$ from $H_t$
  $\Rightarrow$ *Beliefs of environment state:*

$$S_t^a = \left(P(S_t^e = s_1), .., P(S_t^e = s_n)\right)$$

***example**: robot with a camera (not exact location), poker player (only sees public cards), chat bot (does not know what the costumer wants from it)*

# Deterministic Sequential Planning

- Set of states $\boldsymbol{S} = \{s_1,..,s_n\}$
- Set of actions ***A(s)*** for each state $s \in S$
- Reward function **R: R(s)** (if negative = cost function)
- Transition function **T: S×A => S: t(s,a) = s'**

  **(deterministic case !!)**
- this implies:
  - episode = $s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, s_3 ..., s_l, a_l, r_l, s_{l+1}$
  - *reward of the episode:* $\sum_{i=1}^{l} \gamma^i r_i$ *with* $0 < \gamma \leq 1$

    *($\gamma$=1: all rewards count the same, $\gamma$ <1: early rewards count more )*
  - ***sometimes****: process terminates when reaching a terminal state* $s^T$ *or process end after k moves.*
  - *a strategy can be described by the actions in the episode which imply fixes states and rewards*

# Deterministic Sequential Planning

- Static, discrete, deterministic, known and fully observable environments:
  - S, A are discrete sets and known
  - t(s,a) has a deterministic state s'
  - Agent knows the current state

- **goal**: find a sequence of actions (path) that maximize the cumulated future rewards.

**examples**:

- routing from A to B on the map or find an exit
- riddles like the goat, wolf, cabbage transport problem

# Example: Goat, Wolf, Cabbage

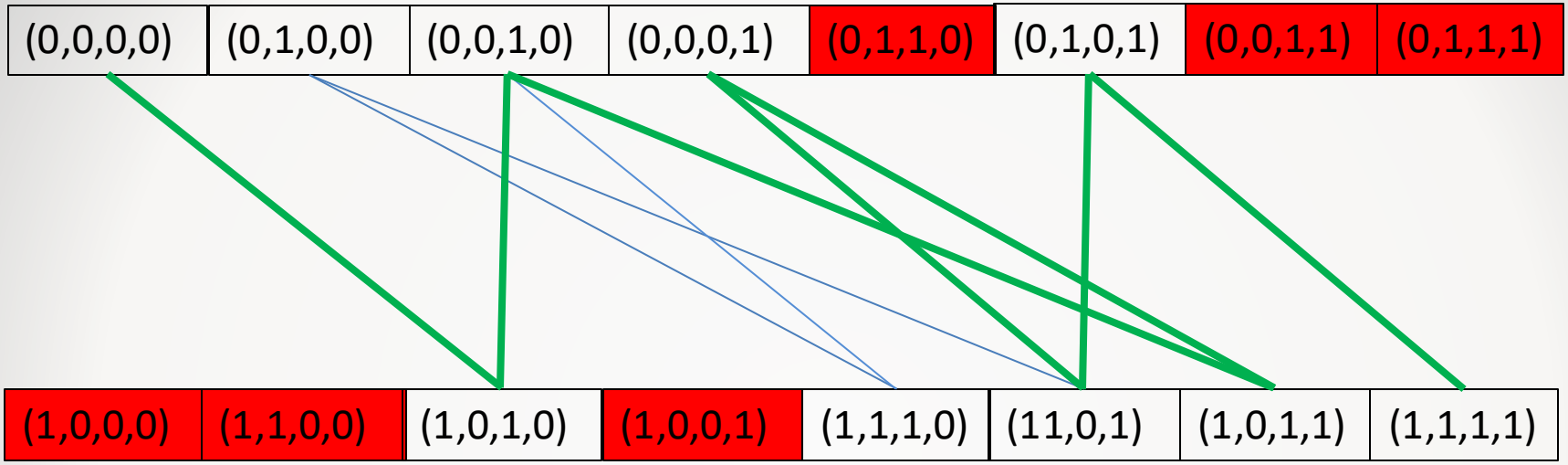
**Task**:

- You have a goat, a wolf and a cabbage you want to transport over a river on a small boat.
- The boat can only carry you an one item.
- If the wolf is alone with the goat, the wolf eats the goat.
- If the goat is alone with the cabbage the goat will eat the cabbage.

**Which strategy can transport all three items over the river without losing one with a minimum amount of crossings?**

**Approach**: A state models the boat, wolf, cabbage, goat as binary variable. (not all combinations are viable)
Each crossing costs 1 time unit. The task is fulfilled when reaching the state (1,1,1,1) = all items are on the other side.

# Example: Goat, Wolf, Cabbage




- (b,w,g,c)=(boat, wolf, goat, cabbage)
- invalid states: w=g≠b, g=c ≠b
- find the shortest path between (0,0,0,0) and (1,1,1,1)

**Problem**: When selecting an action, we do not know if it is on the shortest path.

**Solution**: Breadth-First search=first path arriving at (1,1,1,1) is optimal. (if costs per action vary, use Dijkstra's algorithm)

# Nondeterministic Planning Problems

**Problem: We might not know the outcome of an executed action**

- the goat might not like the cabbage
- the wolf seems not to be hungry and leaves the goat be
- the boat can sink on each trip and I might drown

$\Rightarrow$ transitions lead to uncertain future states

$\Rightarrow$ **t(s,a) is a stochastic function T**: $P(s' | s,a)$ for all $s,s' \in S$, $a \in A$

**implications**:

- we do not know the future state after performing action *a*.
- the reward R(s)/cost C(s) gets stochastic
- when searching a terminal state, there might be no secure path leading us to the target (the boat can always sink).

$\Rightarrow$ We need an action for any situation, we might encounter and not just the ones on a single path to the goal.

# Policies and Utilities

- in **dynamic**, discrete, **non-deterministic**, known and fully observable environments
- a **policy $\boldsymbol{\pi}$** is a mapping defining for every state $s \in S$ an action $\pi(s) \in A(s)$ (agent knows what to do in any situation)
- **stochastic policies**: Sometimes it is beneficial to vary the action then $\pi(s)$ is a distribution function over A(s).
  - think of a game where strictly following a strategy makes you predictable.

    *What is the best policy for **Rock-Paper-Scissor**?*
  - sometimes best policy involves trying out different things

# Markov Process

Markov processes are memoryless random processes generating sequences of states:

A **Markov Process** (or Markov Chain) is a tuple $\langle S, P \rangle$ :

- S is a finite set of states
- P is a state transition probability matrix

$$P_{s,s'} = \Pr[S_{t+1} = s' | S_t = s]$$

$\Rightarrow$ shortened definition: start and terminal states can be modelled as ordinary states, but are often mentioned separately.

$\Rightarrow$ Markov chains describe state sequence under a particular policy

# Markov Reward Process

**Markov Reward Process** $\langle S, P, R, \gamma \rangle$:

- S is a finite set of states
- P is a state transition probability matrix

  $P_{s,s'} = \Pr[S_{t+1} = s' | S_t = s]$
- R is a reward function $R_s = E[R_t | S_t = s]$
- $\gamma$ discount factor, $\gamma \in [0,1]$

$\Rightarrow$ MDP under a fixed policy $\pi$ $(model\ prediction)$

$\Rightarrow$ allows to evaluate sequences of states:

$\sum_{i=1}^{l} \gamma^i r_i$ *with* $0 < \gamma \leq 1$ (discounted reward)

# Markov Decision Process

A **Markov Decision Process** $\langle S, A, P, R, \gamma \rangle$:

- S is a finite set of states
- A is a finite set of actions
- P is a state transition probability matrix

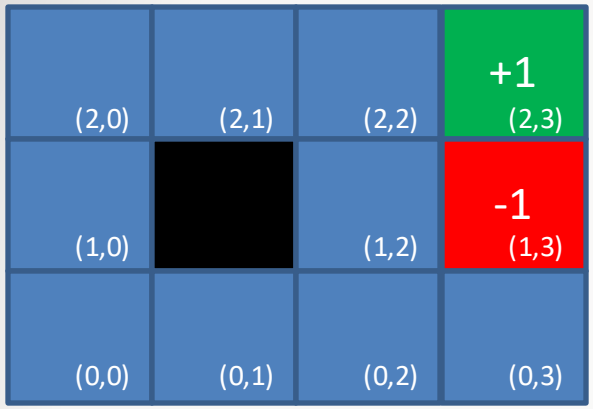
$$P_{s,s'} = \Pr[S_{t+1} = s' | S_t = s, a]$$

- R is a reward function $R_s = E[R_t | S_t = s]$
- $\gamma$ discount factor, $\gamma \in [0,1]$

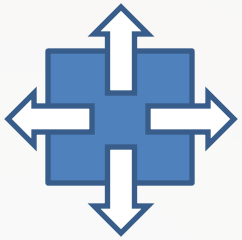
$\Rightarrow$ environment where all states are Markov

$\Rightarrow$ allows to evaluate the quality of all possible policy

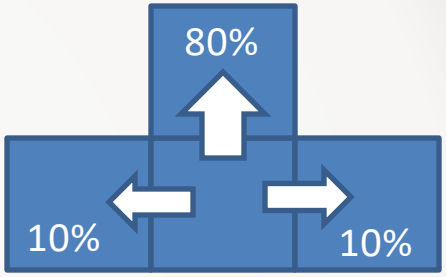
# simple example: grid world



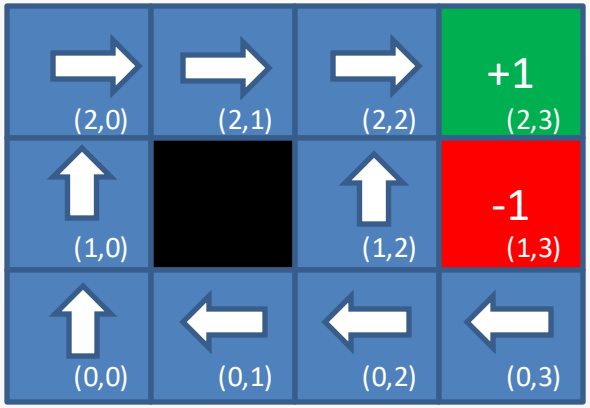

states S and rewards R(S)



actions




transition function P(S'|S,a)




policy

# Finite and Infinite Horizon MDP

When does the process stop?

***Finite horizon:***

- process terminates after n steps
- $\gamma$=1 allowed ⇒ all future rewards/costs are weighted equally
- **but**: policies become non-stationary $\pi(s) \rightarrow \pi^t(s)$

***Infinite horizon:***

- process potentially goes on forever or until a terminal state is reached
- termination might depend on the policy
  (proper policy: eventually reaches goal at some future time)
- $\gamma$<1 usually required:
  - future rewards/costs are weighted less
  - rewards of infinite random walks converge to a fixed value
  - human behaviour favours immediate rewards
  - immediate rewards earn more interest in financial applications
- Ergodic Markov Processes optimize the average reward for $\gamma$=1

# Partially Observable Markov Decision Process

A **Partially Observable Markov Decision Process (POMDP)** is a MDP with hidden states (c.f. Hidden Markov Model).

A **POMDP** is a tuple $\langle S, A, O, P, R, Z, \gamma \rangle$:

- S is a finite set of states
- A is a finite set of actions
- O is a finite set of observations
- P is a state transition probability matrix

  $P_{s,s'} = \Pr[S_{t+1} = s' | S_t = s]$
- R is a reward function $R_s = E[R_{t+1} | S_t = s]$
- Z is an observation function, $Z^a_{s'o} = P[O_{t+1} = o | S_{t+1} = s', A_t = a]$
- $\gamma$ discount factor, $\gamma \in [0,1]$

$\Rightarrow$ we only observe o but not s

$\Rightarrow$ Z describes the likelihood of observation o when taking action a in state s'

# Belief States

Given history $h_t = O_1, R_1, A_1, O_2, R_2, A_2, \ldots, A_{t-1}, O_t, R_t$.
A belief state b(h) is a probability distribution over states, conditioned on the history h:

$$b(h) = (P[S_t = s^1 | H_t = h_t], .., P[S_t = s^n | H_t = h_t])$$

$\Rightarrow$ b(h) described the likelihood that we are in state $s^i$ at time t given the history $h_t$.

$\Rightarrow$ $H_t$ satisfies the Markov property and thus, b(h) satisfies it as well.

$\Rightarrow$ POMDP can be described by history or belief state trees

$\Rightarrow$ adds uncertainty about the state to the expectation

# Further Variants of MDPs

- Countably infinite state and/or action spaces

  ⇒ can be handled similarly as ordinary MDPs

- Continuous state and/or action spaces

  ⇒ closed form for linear quadratic model (LQR)

- Continuous time
  - Requires partial differential equations
  - Hamilton-Jacobi-Bellman (HJB) equation
  - Limiting case of Bellman equation ( t->0)
- …

# Evaluating Policies

We can compute the (discounted) reward of an episode, but how can we compute the reward of following a policy?

⇒ in a nondeterministic environment, we might observe a variety of different histories following the policy $\pi$ ⇒ reward uncertain

Estimate the expected reward following $\pi$ :

- We define the utility/state-value function of state s following policy $\pi$ as:
  $U^{\pi}(s_i) = E[\sum_{t=i}^{\infty} \gamma^t R(s_t) \,|s_i = s, \pi]$

- Correspondingly, we define the action-value function:
  $$q_{\pi}(s, a) = E[\sum_{t=i}^{\infty} \gamma^t R(s_t) \,|s_i = s, A_i = a, \pi]$$

⇒ How to compute $U^{\pi}(s_i)$ and $q_{\pi}(s, a)$?

# Bellman's Equations

- What is the reward of following $\pi$? (prediction)

  Utility $U^{\pi}(s)$: expected reward when following $\pi$ in state s w.r.t. some $0 \leq \gamma \leq 1$:

  - $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,|s_0 = s, \pi]$
  - $U^{\pi}(s) = \gamma^0 R(s) + E[\sum_{t=1}^{\infty} \gamma^t R(s_t) \,|s_0 = s, \pi]$
  - $U^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi) U^{\pi}(s')$

    (Bellmann Equation)

- What is the optimal policy $\pi^*$? (control)

  Bellman Optimality Equation:

$$U^{\pi^*}(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s, a)\, U^{\pi^*}(s')$$

# Policy Evaluation

- Policy evaluation is much simpler than solving the bellman equation (c.f. Markov Reward Process)

$$U^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi)\, U^{\pi}(s')$$

- Note that the non-linear function “max” is not present
- We can compute this with standard solvers for linear equations systems.
- Solving linear equation systems is in $O(n^3)$. (infeasible for large n!)
- In large state spaces, a simplified Bellman update for k iterations can be more performant:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi_i)\, U_i(s')$$

# Finding optimal Policies: Policy Iteration

- We are looking for the optimal policy.
- The exact utility values are not relevant if one action is clearly the optimal.

**Idea**:

Alternate between:

- **Policy evaluation**: given a policy, calculate the corresponding utility values
- **Policy improvement**: Calculate the policy given the utility values $\pi(s) = \underset{a \in A}{arg\max} \sum_{s' \in S} P(s'|s,a)\, U^*(s')$

(greedy policy selection)

# Policy Iteration

**repeat**

$U \leftarrow \text{PolicyEvaluation}(\pi, U, mdp)$

$unchanged? \leftarrow true$

**for each** state s in S **do**

**if** $\max_{\boldsymbol{a} \in \boldsymbol{A}} \sum_{\boldsymbol{s'}} \boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s}, \boldsymbol{a})\, \boldsymbol{U}[\boldsymbol{s'}] > \sum_{\boldsymbol{s'}} \boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s}, \boldsymbol{\pi})\, \boldsymbol{U}[\boldsymbol{s'}]$ **then**

$\boldsymbol{\pi}[\boldsymbol{s}] \leftarrow \underset{a \in A}{\text{argmax}} \sum_{\boldsymbol{s'}} \boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s}, \boldsymbol{a})\, \boldsymbol{U}[\boldsymbol{s'}]$

$\boldsymbol{unchanged?} \leftarrow \boldsymbol{false}$

**until** $unchanged?$

**return** $\pi$

# Value Iteration

- if we use Bellman updates  anyway, we can join both steps
- update Bellman optimality equation directly:

$$U^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\, U^*(s')$$

- Non-linear system of equations
- Use Dynamic Programming:
  - Compute utility values for each state by using the current utility estimate
  - Repeat until it converges to $U^*$
  - convergence can be proofed by contraction

(c.f.: S.Russel, P. Norwig: Artificial Intelligence: A modern Approach, Pearson(2016), page 654 for the proof)

# Value Iteration

**repeat**

$U \leftarrow U'$

$\delta \leftarrow 0$

**for each** state s in S **do**

$U'[s] \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s'} P(s'|s,a)\, U[s']$

if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$

**until** $\delta < \frac{\epsilon(1-\gamma)}{\gamma}$

**return** $U$

# Synchronous Dynamic Programming

- Algorithms are based on state-value function U(s)
- Policy Iteration (prediction & control):
  - iterates prediction (Bellman Expectation Equation) and evaluation (Greedy Policy Improvement)
- Value Iteration (control):
  - updates utility and policy based on Bellman Optimality Equation
- Complexity $O(mn^2)$ per iteration for m actions and n states (with Bellmann updates)

# Dynamic Programming with asynchronous backups

- Dynamic Programming with synchronous backups
  ⇒ all states are backed up in each iteration

- Dynamic Programming with asynchronous backups
  - select state to backup in varying order
  - for each selected state, the appropriate backup is applied
  - allows to backup some states more often than others
  - guaranteed to converge if all states continue to be selected

  Advantages:
  - some states might never be visited under the optimal policy $\pi_*$
  - utility U(s) might take different times to converge for different states

  ⇒ asynchronous upupdates can significantly speed up computation

# Directions for Async. Dyn. Programmming

- In-place dynamic programming
- Prioritised sweeping
- Real-time dynamic programming

# In-place Dynamic Programming

- Synchronous updates need 2 copies of U(s) for all s $\in$ S:

$$U_{new}[s] \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s'} P(s'|s,a)\, U_{old}\,[s']$$

$U_{new}[s]$ is computed based on $U_{old}\,[s']$

- Idea: directly update $U'_{new}$ after back up of one state.

for all s in S:

$$\mathrm{U}[s] \leftarrow R(s) + \gamma \max_{a \in A} \textstyle\sum_{s'} P(s'|s,a)\, U[s']$$

$\Rightarrow$ less memory

$\Rightarrow$ back ups are done on more recent versions of the value function

# Prioritized Sweep

- ***Bellman error***: Difference between right and left side of the Bellman equation.

$$\left| max_{a \in A} \left( R(s) + \gamma \sum_{s'} P(s'|s,a)\, U[s'] \right) - U'[s] \right|$$

***Idea***: Update the state with the largest Bellman error first and update Bellman errors of affected states (those relying on updated U(s))

$\Rightarrow$ requires to know the predecessors

$\Rightarrow$ algorithm uses priority queue on states and always updates the one with the largest Bellman error until convergence.

# Real-time dynamic programming

**idea**: Not all states are relevant to an agent.

$\Rightarrow$ some states might not even be visited under a reasonable policy

$\Rightarrow$ use the experience of the agent to select visited states

**In particular:**

- select state $s_t$
- select action $a_t$ leading to reward $R_{t+1}$
- Backup $s_t$ with $U(s_t)= \max_{a \in A}\left(R^a(s_t) + \gamma \sum_{s'} P(s'|s,a)\ U[s']\right)$
- sample over the possible follow states $s_{t+1}$ and proceed with the sampled state (different variant use different sample schemes)

**Caution**: Not applicable to all types of MDPs:

- requires a proper policy, a terminal state and assumes costs
- requires monotonic lower bounds on the utility

# Performance Bottlenecks on MDPs

- size of state space $|S|$ is often huge
  (for example exponential in descriptive variables)
- transitions are potential $|S|^{2} \cdot |A|$
- In all dynamic programming approaches so far we do full-widths backups:
  - $\Rightarrow$ We consider all potential follow-up states no matter their likelihood
  - $\Rightarrow$ Unlikely steps often do not influence the utility to a high degree due to having a small likelihood

Solution approaches:

- summarize states
- build models considering a continuous state description vector than a discrete state
- sample backups instead of computing full-width backups

$\Rightarrow$ utilities are expectations and can be estimated using sampling

# Summary MDPs

- MDPs rely on a Markov model with assumptions about: states, actions, rewards, transition probabilities, etc.
- If all the information is available, computing the optimal policy does not require any samples or learning
- Transitions probabilities are usually not defined but have to be estimated based on observations.
- usually observations ≠ states
  - partially observable MDP, estimate belief states (c.f. HMM)
  - set of possible states and transitions is unknown
- If the agent does not know the model, we require methods learning from experience
  ⇒ model free approaches and reinforcement learning

# Literature

- Lecture notes D. Silver: Introduction to Reinforcement Learning (https://www.davidsilver.uk/teaching/)

- S. Russel, P. Norvig: Artificial Intelligence: A modern Approach, Pearson, 3rd edition, 2016