Technische Hochschule Rosenheim

# Introduction to AI - Part II

## State-based Models - Markov Decision Processes

## Last semester: Search Problems

- Given a start state, actions and a goal state.
- Find a path to the goal state (sequence of actions).
- Deterministic: Given state and action, the successor state is determined.

## Uncertainty in the real world:

- Given start state, actions and a goal.
- Find a behaviour (policy) that achieves the goal.
- **But:** Nature interferes and determines the state we end up in.
  - Uncertainty in the environment
  - Non-deterministic: Given state and action, the successor state is created by a random process.

> How to make decisions under uncertainty?

> State-based Models are used in Reinforcement Learning

# What is Reinforcement Learning?

# Reinforcement Learning

The *science of decision making*: Find a way to make optimal decisions.

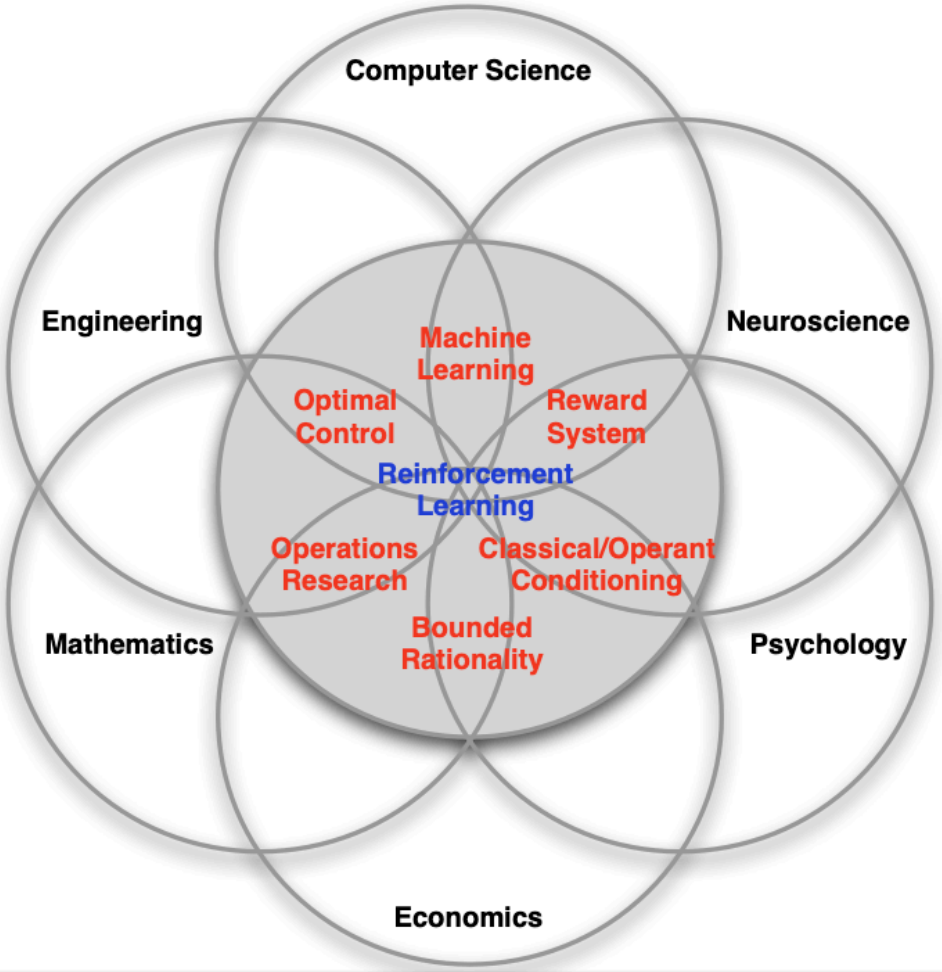



image source[https://www.davidsilver.uk/teaching/]

# Reinforcement Learning

- Going back to the 1950s, B.F. Skinner (Psychologist)
- Lab Experiments with humans and animals
- Operand Conditioning Theory

## Positive Reinforcement

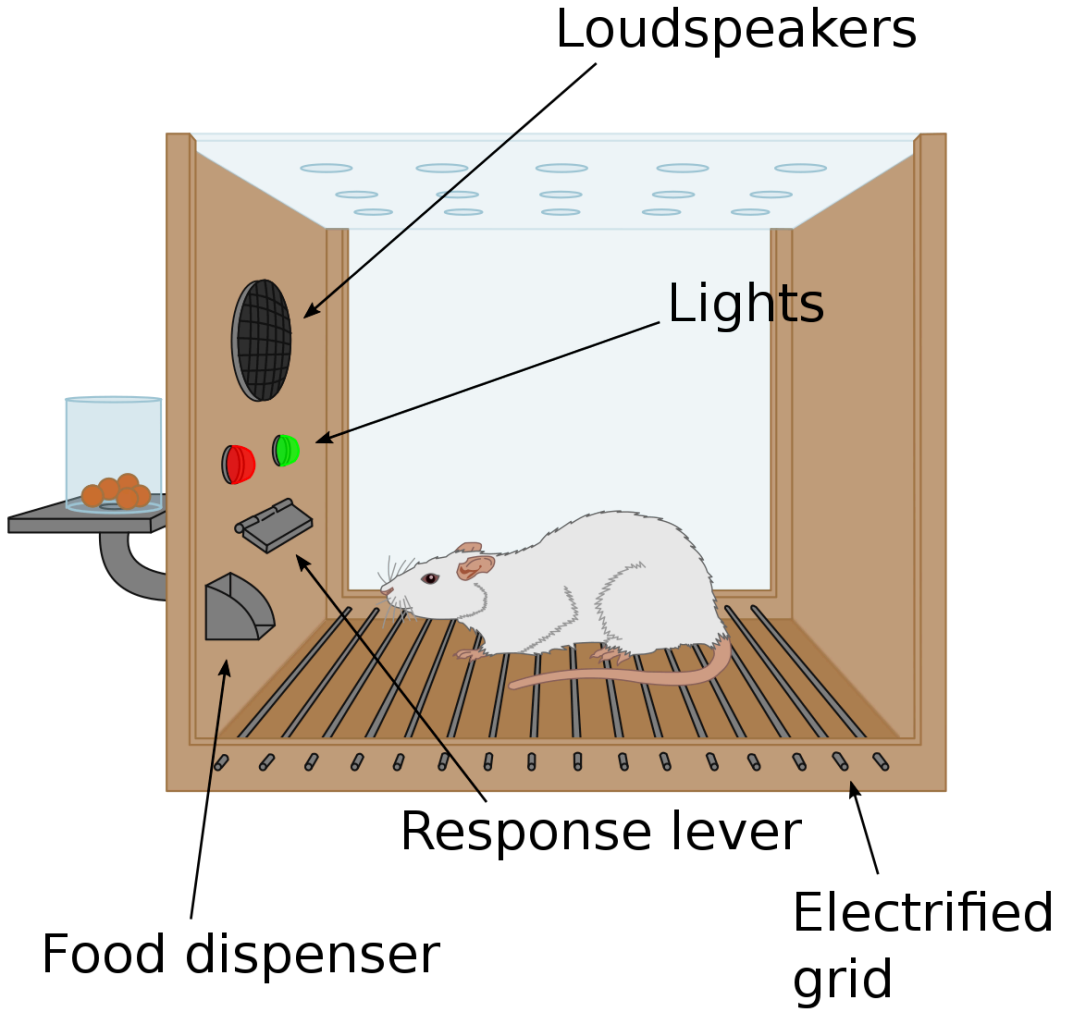
- rewards (e.g., praise)
- only works, if the reward is something the subject wants

## Negative Reinforcement

- remove negative consequences
- encourages desired behavior

**Rewards are most effective when they are immediate and come right after the desired behavior.**

- **Ever thought about gamification in apps?**



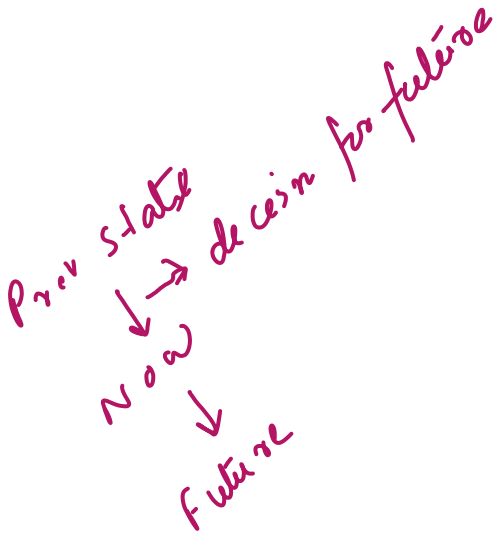
# Reinforcement Learning

## Paradigm

### Perspectives

A paradigm studied under different names in several areas

- Optimal Control (Engineering)
- Machine Learning (Computer Science)
- Operant Conditioning (Psychology)




### Differences to other Machine Learning paradigms:

- There is no supervisor - only a *reward* signal.
- Feedback is delayed, not instantaneous
- Time matters (sequential, non i.i.d. data)
- Agent's actions affect the subsequent data it receives

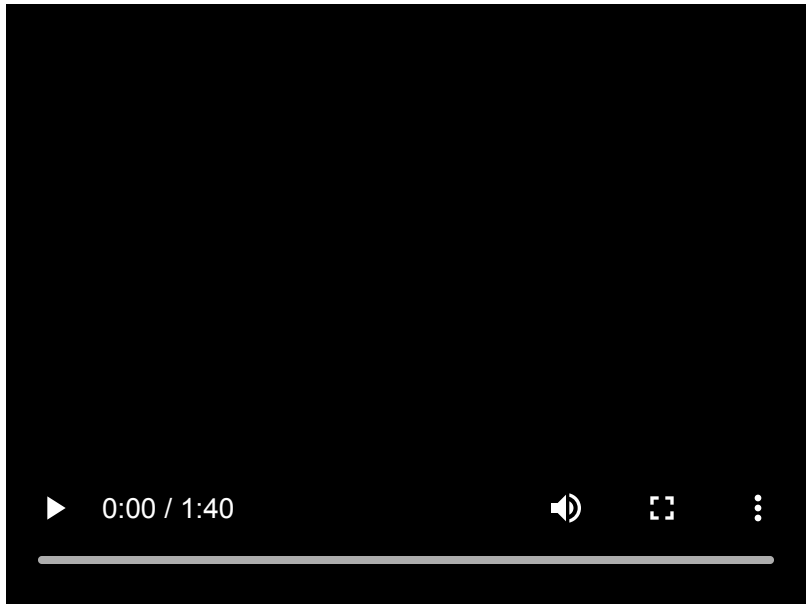
# Reinforcement Learning

## Examples

Manage an investment portfolio

Make a humanoid robot walk

Play Mario Cart



# Reinforcement Learning

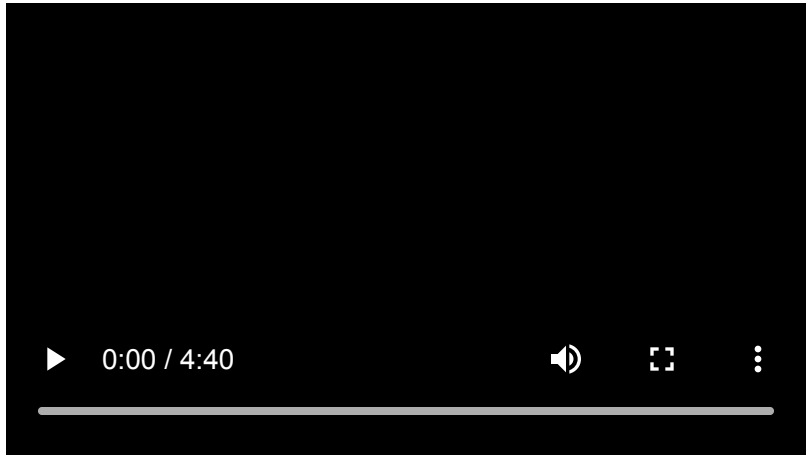
a machine learning (ML) technique that trains software to make decisions to achieve the most optimal results

## Examples

Manage an investment portfolio

Make a humanoid robot walk

Play Mario Cart



# The Reinforcement Learning Problem

## Reward

A **reward** is a scalar feedback signal.

It indicates how well the agent is doing in a particular situation.

The agent's task is to maximize the cumulative future reward.

Reinforcement Learning is based on the **reward hypothesis**.

Reward Hypothesis example: To keep the agent staying as close to the centre line as possible, the reward function could return a reward of 1.0 if the vehicle is within 3cm from the centre, a reward of 0.5 if the agent is within 10cm and a reward of 0.001 (stands for zero for practical purposes) otherwise.

**⊙ Reward Hypothesis**

All goals can be described by the maximization of cumulative future reward.

> The structure of the reward signal must truly indicate what we want to accomplish. Does maximization of rewards also make the agent achieve our goal?
> The reward signal is our way to describe **what** we want the agent to achieve - not **how** to achieve it.

# The Reinforcement Learning Problem

## Reward: Examples

### Stunt manoeuvres in a helicopter:

- POS-reward for following desired trajectory
- NEG-reward for crashing

### Backgammon

- POS/NEG reward for winning/losing a game

### Robot walk

- POS-reward for movement progress (e.g. in cm)
- NEG-reward for falling over

### Atari games

- POS/NEG reward for increase/decrease in score

### Escape from a maze

- POS-reward of +1 when the exit is found
- NEG-reward of -1 at each step (encourage quick escape)

# The Reinforcement Learning Problem

## Framework

### Goal:

Select actions to maximize "cumulative future reward".

### However:

> Actions may have long term consequences.
>
> Reward may be delayed.
>
> It may be better to sacrifice immediate reward to gain more long term reward.

# The Reinforcement Learning Problem

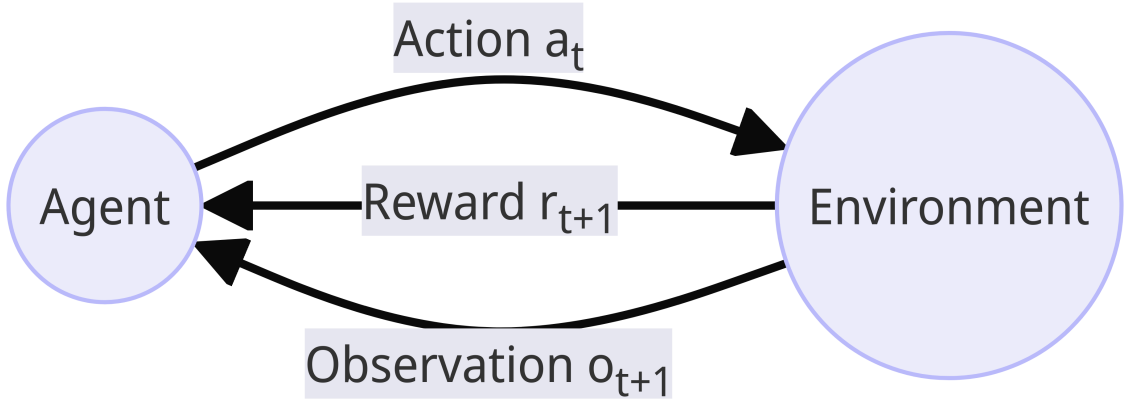
## Agent and Environment

**At each step $t$ the agent:**

- Executes action $a_t$
- Receives observation $o_{t+1}$
- Receives scalar reward $r_{r+1}$

**The environment:**

- Receives action $a_t$
- Emits observation $o_{t+1}$
- Emits scalar reward $r_{t+1}$



# The Reinforcement Learning Problem

Technische Hochschule Rosenheim

## History and State

### History

The **history** is a sequence of actions, observations and rewards:

$$h_t = a_1, o_2, r_2, \ldots, a_{t-1}, o_t, r_t$$

- All observable variables up to time $t$
- The stream of sensor data of the agent
- What happens next depends on this history:
  - Agent's perspective: The algorithm we build is a mapping from this history to an action
  - Environment's perspective: Uses history to determine which observation/reward it'll emit

### State

A **state** is the information used to determine what happens next.

$$s_t = f(h_t)$$

Ideally, a state summarizes past sensations compactly, yet in such a way that all information relevant for the future is retained.

# The Reinforcement Learning Problem

## State(s)

### Environment State

- set of numbers that describe the environment
- typically invisible to the agent - only sees what is infront of it
- set of numbers influencing the agents decision

### Agent's State

- Agents internal representation (of what it's observed so far)
- decision at them $t$w
- information to keep and discard
- any function of history
- what the designer decides on is important

# Interlude: Markov Chains

## What is a Markov Chain?

- A Markov chain is a stochastic model ...
- that describes a sequence of possible events ...
- in which the probability of each event depends only on the state attained in the previous event.

image source: de.wikipedia.org/

# Interlude: Markov Chains

- **State**: A situation or condition at a particular point in time.
- **Transition Probability**: Probability of moving from one state to another.
- **State Space**: Set of all possible states in a system. The state space is discrete.

## Properties of Markov Chains

1. **Memorylessness**: Future behavior depends only on the present state, not on the sequence of events that preceded it.
2. **Stationarity**: Transition probabilities remain constant over time.

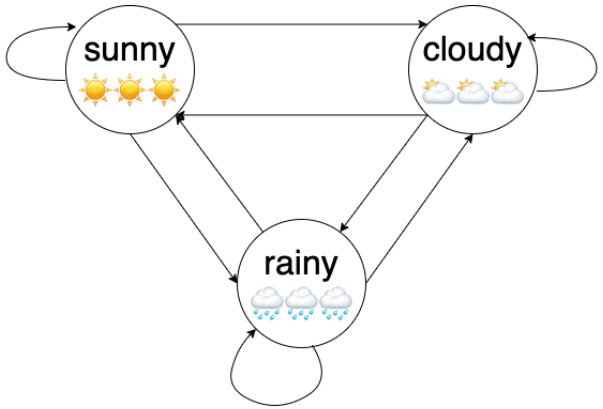
# Interlude: Markov Chains

## Example: Weather Forecast

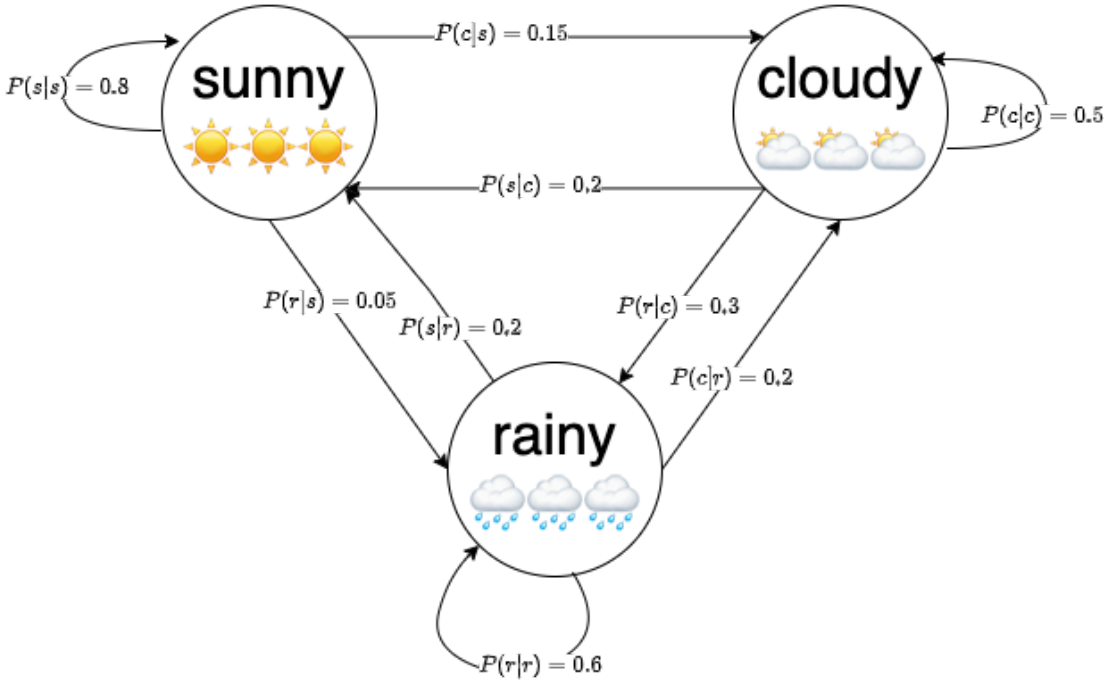
- States: Sunny, Cloudy, Rainy
- Transition Probabilities:
  - Sunny → Sunny: 0.8, $P(Sunny|Sunny) = 0.8$
  - Sunny → Cloudy: 0.15, $P(Cloudy|Sunny) = 0.15$
  - Sunny → Rainy: 0.05, $P(Rainy|Sunny) = 0.05$
  - Cloudy → Sunny: 0.2, $P(Sunny|Cloudy) = 0.2$
  - Cloudy → Cloudy: 0.5, $P(Cloudy|Cloudy) = 0.5$
  - Cloudy → Rainy: 0.3, $P(Rainy|Cloudy) = 0.3$
  - Rainy → Sunny: 0.2, $P(Sunny|Rainy) = 0.2$
  - Rainy → Cloudy: 0.2, $P(Cloudy|Rainy) = 0.2$
  - Rainy → Rainy: 0.6, $P(Rainy|Rainy) = 0.6$

## Transition Matrix

| | Sunny | Cloudy | Rainy |
|---|---|---|---|
| Sunny | 0.8 | 0.2 | 0.2 |
| Cloudy | 0.15 | 0.5 | 0.2 |
| Rainy | 0.05 | 0.3 | 0.6 |



# Markov Example: Weather Forecast




## Transition Matrix

- Transition probabilities
- Sum of all transition probabilities from a state is 1

## Variation: Higher order Markov chains

- More complex modelling using prior knowledge about the processes to be modeled:
  - Higher order Markov chains: Transition probabilities depend on multiple previous states
  - Markov Chains have 'memory' of $m$ previous states

# The Reinforcement Learning Problem

## Markov State

A general definition of a state - regardless of the point of view.

A Markov state (i.e. information state) contains all useful information from the history.

**Markov State**

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

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \ldots, S_t]$$

**In other words:** "The future is independent of the past given the present."

- The distribution over the next state is the same, conditioned on the current state, as it is, conditioned on the whole history.
- If we had such a representation, we could ignore all previous states, only consider the current state and still get the same characterization of the future.

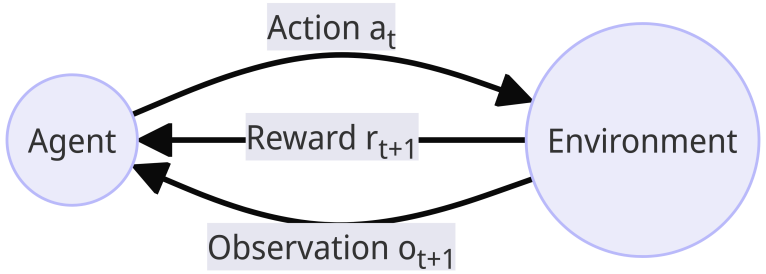
**It is on us to define such a state.** For example, consider the two state representations for a helicopter:

1. `(xyz-coordinate)`
2. `(xyz-coordinate, xyz-velocity, angular velocity, ...)`

# The Reinforcement Learning Problem



## Types of Environments




### Full observability

- Agent directly observes environment state $s_t^e$.
- Then all pieces of information collapse into one state representation - that can be used as the agent's state $s_t^a$.

$$o_t = s_t^a = s_t^e$$

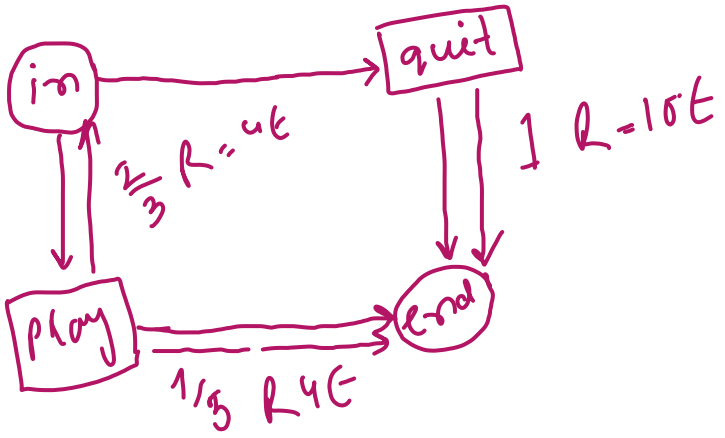
- Formalism to describe such scenario:
  **Markov Decision Process (MDP)**

### Partial observability

- Agent observes environment indirectly only through observations
  - Robot with sensors (limited vision, measurement range, etc.)
  - Trading agent only observes market prices
- Now, the states differ $s_t^a \neq s_t^e$ and $s_t^e$ is unknown to the agent.
- Formalism to describe such scenario:
  **Partially Observable Markov Decision Process (POMDP)**

# Markov Decision Processes

# Markov Decision Processes




S = Set of state
$S_{start}$ = "in"
A(S) actions to state
T(S, a, S') transition position
R(S, a, S')

## Example Simple Dice Game:

- you can choose to roll the dice or end the game
- if you roll the dice you get a reward of 4 €
  - you have a 2/3 chance (dice roll >=3) to go again and another chance of winning 4€
  - a 1/3 chance of losing, i.e., the game ends and you only get 4€.
- if you decide to end the game right away, you take 10€
  - the game ends right away

**What is the optimal strategy to maximize your expected winnings.**

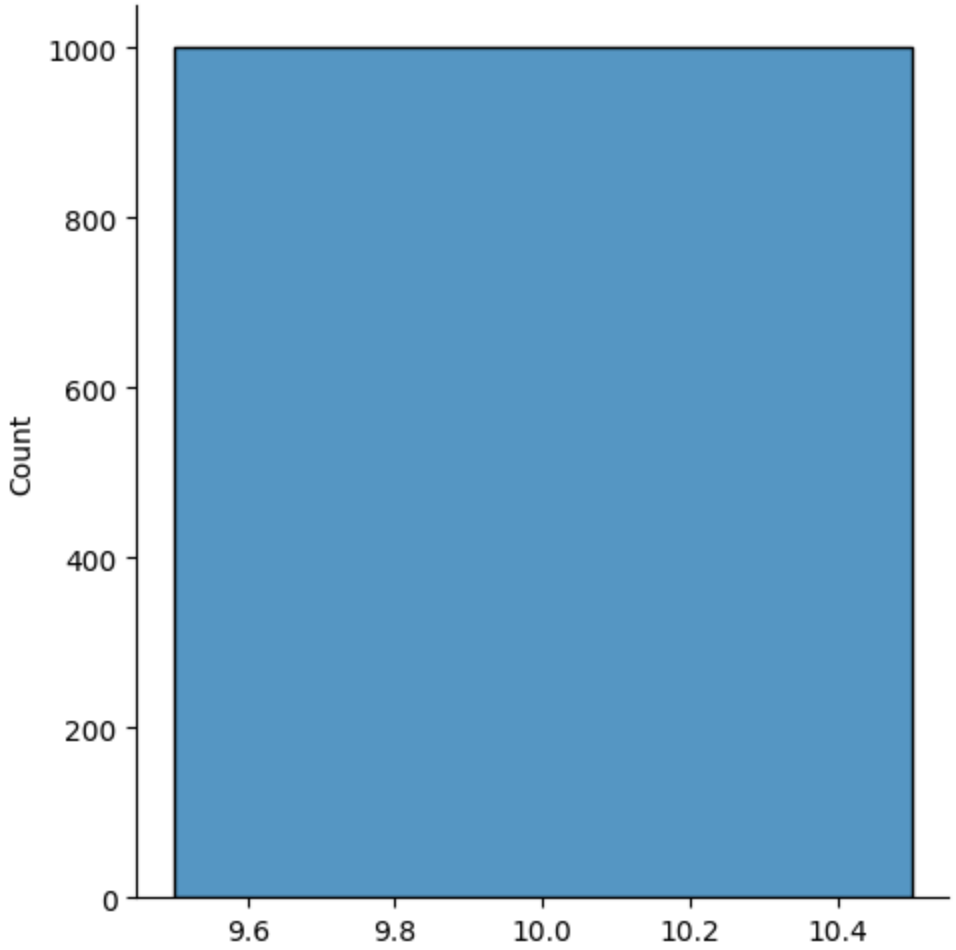
# Markov Decision Processes

- We can measure the value of a state by the expected return
- How would be do this?
- Well there is some probability of ending up in a state, and some reward for ending up in that state
  $\rightarrow$ calculate the expected value of the return

# Markov Decision Processes

Technische Hochschule Rosenheim

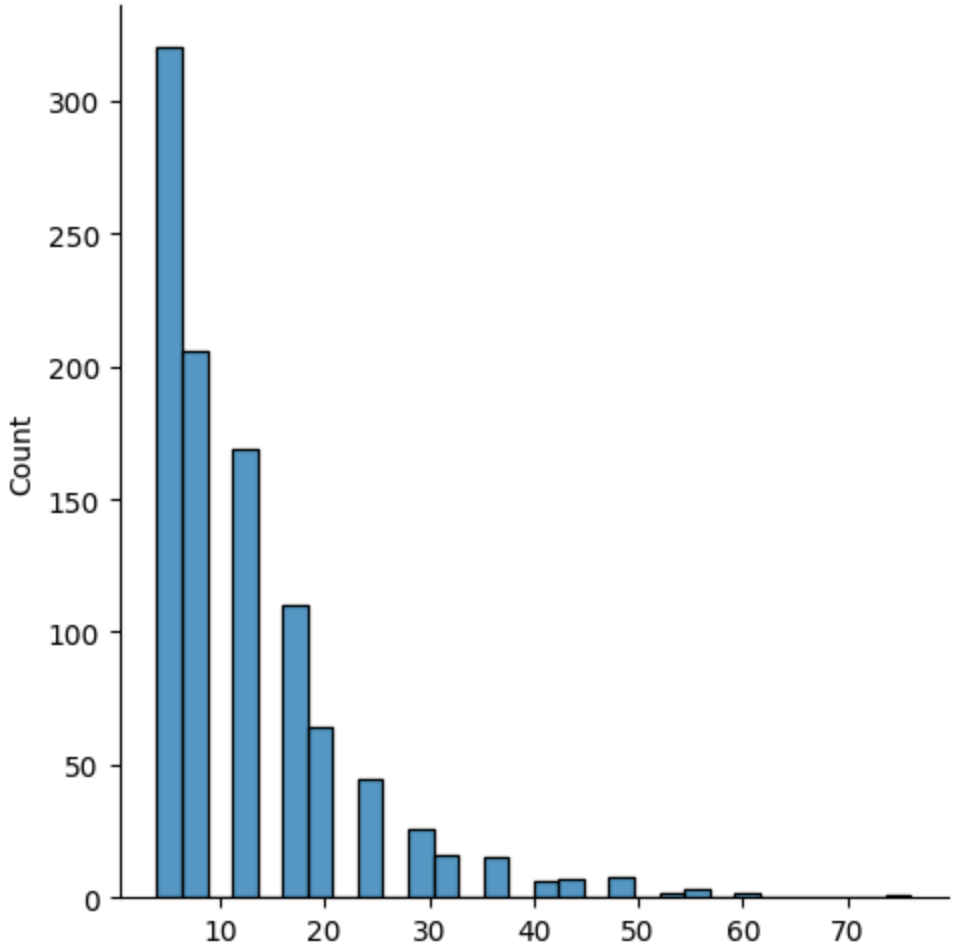
## Expected Utility: leave



# Markov Decision Processes

Technische Hochschule Rosenheim

## Expected Utility: stay + play



# Markov Decision Processes

## Example: MDP

A Markov Decision Process (MDP) is a graph depicting **states**, **actions**, **chance nodes** and **rewards** for an environment in which all states are Markov.

**Markov Decision Process**

- A set of states $\mathcal{S}$.
- A start state $s_{\text{start}} \in \mathcal{S}$.
- A terminal state $s_{\text{end}} \in \mathcal{S}$.
- A set of actions $\mathcal{A}(s)$: Available actions at each state $s$.
- Transition probabilities $\mathcal{T}(s, a, s')$: Probability of ending up in state $s'$ as successor state if we are in state $s$ and take action $a$.
- Reward function $\mathcal{R}(s, a, s')$: Reward $r$ for the transition $(s, a, s')$.

# Markov Decision Processes

## Transition probabilities

**Transition probabilities**

The **transition** probables $\mathcal{T}(s, a, s')$ specify the probability of ending up in state $s'$ if we are in state $s$ and take action $a$.

## Mapping the Example of the Dice game:

| $s$ | $a$ | $s'$ | $\mathcal{T}(s, a, s')$ |
|---|---|---|---|
| in | quit | end | 1 |
| in | stay | in | $\frac{2}{3}$ |
| in | stay | end | $\frac{1}{3}$ |

# Markov Decision Processes



## Policy

**⊙ Policy (deterministic)**

A **policy** $\pi(s)$ is a mapping from states $\mathcal{S}$ to actions $\mathcal{A}(s)$.
For any state, the policy specifies which action to take.

### Example:

| state $s$ | policy $\pi(s)$ |
|---|---|
| (1,1) | UP |
| (1,0) | RIGHT |
| (2,0) | RIGHT |

> Note: Just because we move `RIGHT` in state `(1,0)` does not mean we definitely end up in `(2,0)`. The transition is probabilistic and not within the agent's control. But it may still be the best action to take in that state.

**Goal: Find an (optimal) policy.**

# Markov Decision Processes

## Policy Evaluation: Return / Utility

### How do we evaluate a given policy?

- Following a policy yields a random path: $s_0; a_1, r_1, s_1; a_2, r_2, s_2, \dots$
- For each path, we can calculate the **Return** / **Utility** that we are going to get.

**Return / Utility**

The **Return** / **Utility** is the sum of (discounted) rewards along a random path starting at time step $t$.

$$R_t = \underbrace{r_{t+1}}_{\text{immediate reward}} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \dots$$

with the **discount factor** $\gamma \in [0, 1]$ weighting immediate reward versus rewards further into the future.

Since episodes are generated randomly by following the policy, the Return is a random quantity (it's not a fixed value but varies with each evaluated path).

# Markov Decision Processes

## Policy Evaluation: Value Function

**Value (expected utility / return)**

The **value** of a policy is the **expected** utility / return

**State-Value Function**

The **state-value function** $V_\pi(s)$ of an MDP is the expected return starting from state $s$, and then following policy $\pi$.

$$V_\pi(s) = \mathbb{E}[R_t|S_t = s]$$

**Action-Value Function**

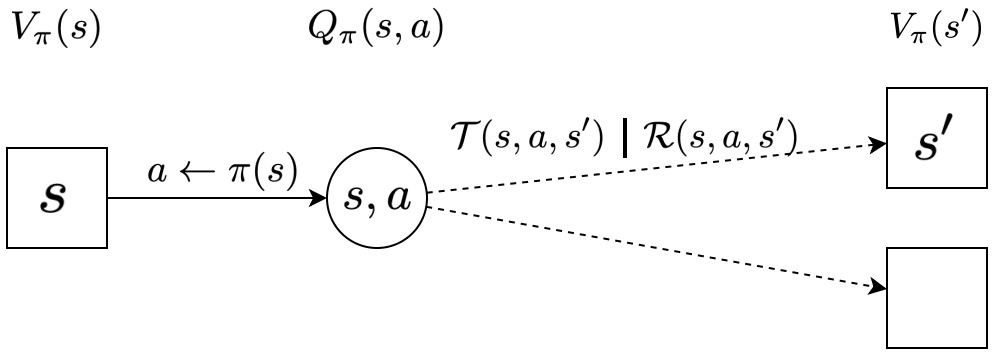
The **action-value function** $Q_\pi(s, a)$ of an MDP is the expected **return/ utility** starting from state $s$, taking action $a$, and then following policy $\pi$.

$$Q_\pi(s, a) = \mathbb{E}[R_t|S_t = s, A_t = a]$$

# Markov Decision Processes

## Policy Evaluation: Recurrence Equation

The value of being in some state $s$ can be expressed recursively via an expectation over the value of the successor states $s'$.




### How good is it to be in state $s$?

It's the immediate reward $\mathcal{R}(s, \pi(s), s')$ we are going to get when taking action $\pi(s)$, plus the value from $s'$ onwards $V_\pi(s')$ - weighted by the probabilities $\mathcal{T}(s, \pi(s), s')$ of transitioning to any possible successor state $s'$.

$$V_\pi(s) = \sum_{s'} \mathcal{T}(s, \pi(s), s') \cdot [\mathcal{R}(s, \pi(s), s') \cdot \gamma V_\pi(s')]$$

# Markov Decision Processes

## Back to the Dice Game

- We can evaluate the expected utility of the two policies we discussed earlier.
- Let $\pi$ be "play" policy: $\pi(in) = \text{play}$
- $V_\pi(in) = \frac{1}{3} * (4 + V_\pi(end)) + \frac{2}{3}(4 + V_\pi(in))$
- $V_\pi(end) = 0$

### Closed form solution:

- $V_\pi(in) = \frac{1}{3} \times 4 + \frac{2}{3} * (4 + V_\pi(in))$
- $V_\pi(in) = \frac{1}{3} \times 4 + \frac{2}{3} * (4 + V_\pi(in))$
- $V_\pi(in) = \frac{12}{3} + \frac{2}{3} \times V_\pi(in)$
- $V_\pi(in) = 12$

# Markov Decision Processes

## Policy Evaluation: Iterative Algorithm

The recurrence equation can be solved iteratively by repeatedly updating the value function.

### Algorithm

- Initialize $V_\pi^{(0)}(s) = 0$ for all states $s$
- For $n = 1, \ldots, N$:
  - For each state $s$:
    - $V_\pi^{(n)}(s) \leftarrow \sum_{s'} \mathcal{T}(s, \pi(s), s') \cdot [\mathcal{R}(s, \pi(s), s') \cdot \gamma V_\pi^{(n-1)}(s')]$

# Markov Decision Processes

## Recap

- **MDP**: graph with states, chance nodes, transition probabilities, rewards
- **Policy**: mapping from states to actions (solution to MDP)
- **Value of policy**: expected utility over random paths
- **Policy evaluation**: iterative algorithm to compute the value of a policy

## Optimal Value Function

So far, a particular policy was given, and we computed the value function (expected return at each state) if we followed that policy.

**Now:** How should we act (policy) to get the maximum value out of the MDP?

**Optimal Value Function**

The **optimal state-value function** $V_{\text{opt}}(s)$ is the maximum state-value function over all policies.

$$V_{\text{opt}}(s) = \max_{\pi} V_{\pi}(s)$$

- The optimal value function specifies the best possible performance in the MDP. Knowing the optimal value function means we **solved** the MDP.
- Knowing the optimal value function, at any state, we can choose the action that takes us to the best successor state.

**Optimal Action-Value Function**

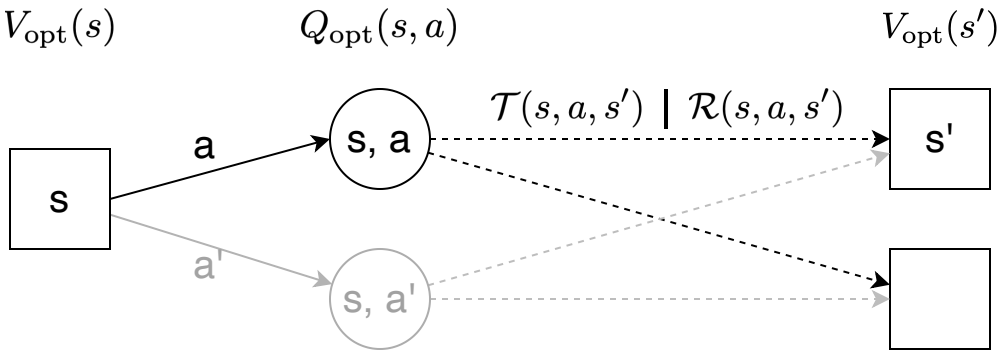
The **optimal action-value function** $Q_{\text{opt}}(s, a)$ is the maximum action-value function over all policies.

$$Q_{\text{opt}}(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

# Markov Decision Processes

## Value Iteration: Recurrence Equation

**Goal:** Find the optimal policy by computing the optimal value function.




**Optimal value if we take action $a$ in state $s$**

$$Q_{opt}(s,a) = \sum_{s'} \mathcal{T}(s,a,s') \cdot [\mathcal{R}(s,a,s') \cdot \gamma V_{opt}(s')]$$

**Optimal value in state $s$**

$$V_{opt}(s) = \underbrace{\max_{a \in \mathcal{A}(s)}}_{\text{evaluate all actions}} Q_{opt}(s,a)$$

**Optimal policy $\pi_{opt}(s)$**

$$\pi_{opt}(s) = \arg\max_{a \in \mathcal{A}(s)} Q_{opt}(s,a)$$

## Value Iteration [Bellman 1957]: Algorithm

The **Value Iteration** algorithm is very similar to policy evaluation.

### Algorithm

- Initialize $V_{opt}^{(0)}(s) = 0$ for all states $s$
- For $n = 1, \ldots, N$:
  - For each state $s$:
    - $V_{opt}^{(n)}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{T}(s, a, s') \cdot [\mathcal{R}(s, a, s') \cdot \gamma V_{opt}^{(n-1)}(s')]$

### Stopping criterion

In practice, we can check convergence by comparing the change in the value vector $V_{opt}$ from one iteration $n$ to the next $n + 1$ against a small constant.

$$\sum_{s} |V_{opt}^{(n)}(s) - V_{opt}^{(n+1)}(s)| < \epsilon$$

# Reinforcement Learning

## Reality - Learning from Data

**Unfortunately, in reality the MDP is unknown:**

- Unknown transition probabilities
- Unknown rewards
- Infinite number of states

$\rightarrow$ Estimate value function from data (experience).

$\rightarrow$ Experience: Sample sequences of states, actions and rewards from online or simulated interaction with the environment.

**Reinforcement Learning methods specify how an agent changes its policy as a result of its experience.**

> Dedicated course on Reinforcement Learning later in the study program.

# State-based Models

## Summary

In Reinforcement Learning, a fully observable environment is modeled as a *Markov Decision Process*.

A *Markov Decision Process (MDP)* is tool to model an environment that exhibits probabilistic state transitions.

An *MDP* is a graph consisting of states, actions, chance nodes, transition probabilities and rewards.

We can evaluate a *policy* by computing the *value function* under that policy.

*Value Iteration* is an algorithm to compute the optimal policy on an MDP.

The *optimal policy* is a policy that maximizes the value function over all states.

In Reinforcement Learning, the full MDP is almost never known. Instead, it has to be estimated from sample sequences.

**No lecture on the 24th of April (next week)**

**The Lab Exercises will still take place!**

# References


- Richard Sutton and Andrew Barto. Reinforcement Learning - An Introduction. MIT Press. 1998.
- Material based on course "Introduction to Reinforcement Learning" from UC London by David Silver: Course and "Introduction to Artificial Intelligence: Principles and Techniques" from Stanford by Percy Liang and Dorsa Sadigh: Course.

# Markov Decision Processes

## Example

You are in a new city, and you want to go to a concert.

For getting to the concert, you have two options:

- Either you **walk** or
- take the **bus**.

If you decide to walk to the concert, you will get to the concert in **5 hours**. But there is a small chance that you get lost in the city and end up back at the hotel after **1 hour**.

If you decide to take the bus, you'll notice that the bus is very unreliable. It's very likely that the bus breaks down, it has to call the mechanic, but the mechanic fails to fix it, and finally you end up back at your hotel after **4 hours**. However, if you're lucky the bus works and takes you to the concert in **1 hour**.

# Markov Decision Processes

## Example: MDP