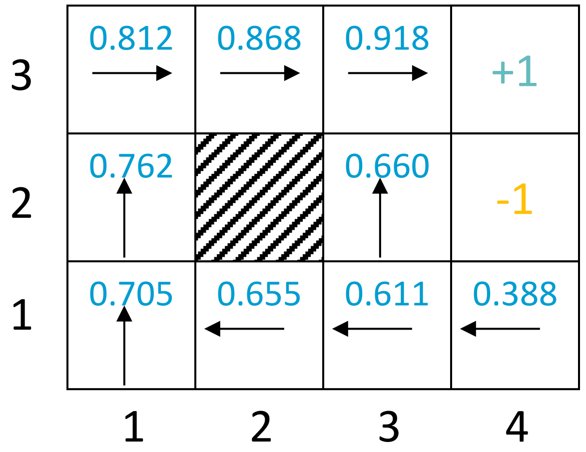
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

# Intelligent Agents : Automated Planning and Acting

## Decision Making: Foundations




Marcel Gehrke

IM FOCUS DAS LEBEN

# Content: Planning and Acting

1. With **Deterministic** Models
2. With **Temporal** Models
3. With **Nondeterministic** Models
4. With **Probabilistic** Models
5. By **Decision Making**
   A. Foundations
      - Utility theory
      - Markov decision processes
      - Reinforcement learning

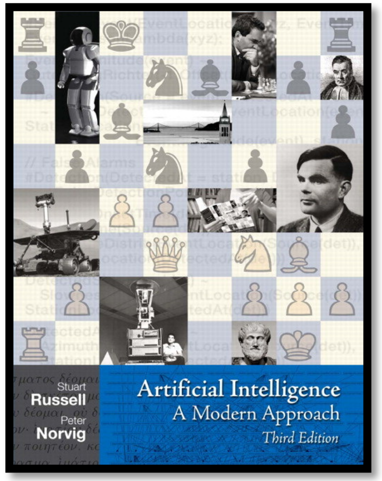
   B. Extensions

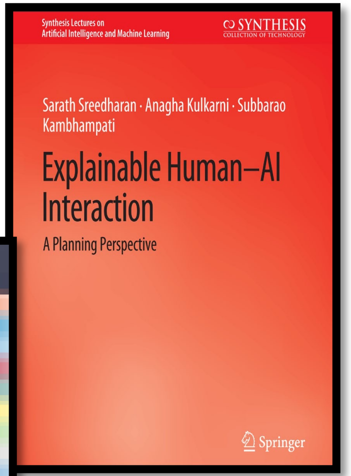
   C. Structure
6. With **Human-awareness**

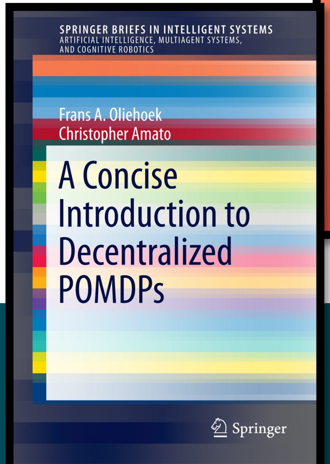
# Literature

- Second half presents different directions research has taken
- Content based on
  - Artificial Intelligence: A Modern Approach (3rd ed.; abbreviation: *AIMA*)
    - Stuart Russell, Peter Norvig
    - Decision making (Chs. 16 + 17), reinforcement learning (Ch. 21)
  - A Concise Introduction to Decentralized POMDPs (*DecPOMDP*)
    - Frans A. Oliehoek, Christopher Amato
  - Explainable Human-AI Interaction: A Planning Perspective (*HA-AI*)
    - Sarath Sreedharan, Anagha Kulkarni, Subbarao Kambhampati
  - Further research papers announced in lectures

- I do not expect you to read all the books!




http://aima.cs.berkeley.edu

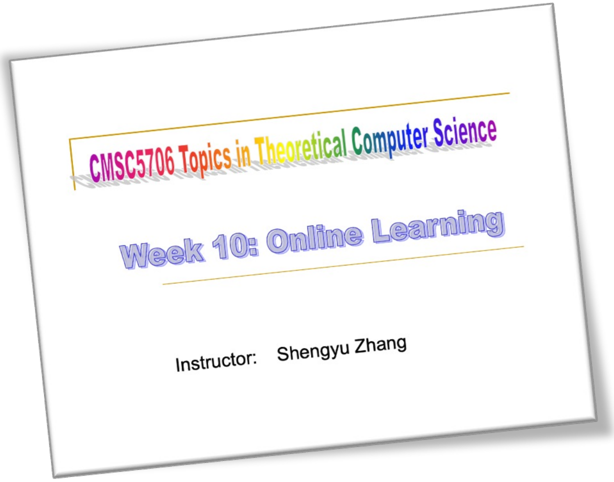
https://link.springer.com/book/10.1007/978-3-319-28929-8

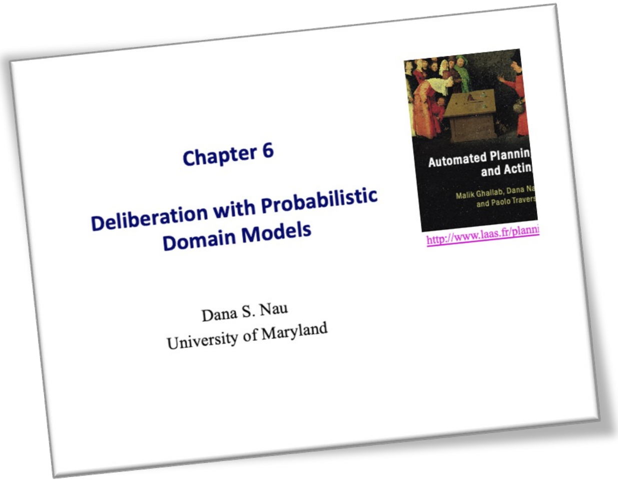
https://link.springer.com/book/10.1007/978-3-031-03767-2

# Acknowledgements

- Slides based on material provided by Dana Nau, Ralf Möller, and Shengyu Zhang
  – In part based on *AIMA Book, Chapters 16, 17, 21*








http://people.eecs.berkeley.edu/~russell/talks/2020/russell-aaai20-hntdtwwai-4x3.pptx
http://rbr.cs.umass.edu/camato/decpomdp/overview.html

# Decision Making under Uncertainty

- Goal-based: binary distinction between *happy* and *unhappy*
- Utility as a distribution over possible states
  - Essentially an internalisation of a performance measure
    - If internal utility function *agrees with* external performance measure:
    - Agent that chooses actions to maximize its utility will be *rational* according to the external performance measure
      - Rationality as a measure of intelligence

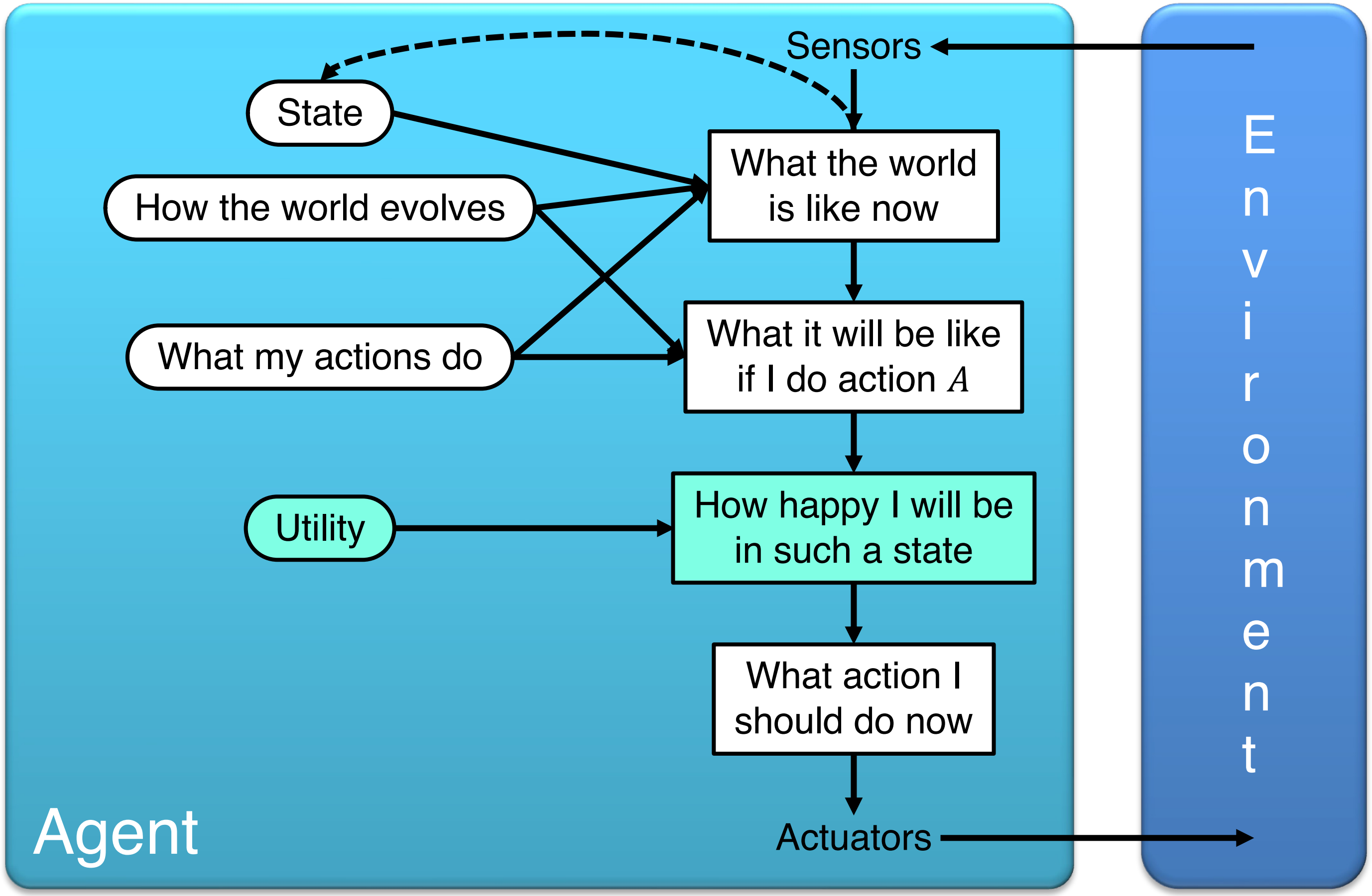



Figure: AIMA, Russell/Norvig

# Setting

- Agent can perform actions in an environment
  - Environment
    - Outcomes of actions not unique
    - Associated with probabilities (→ probabilistic model)
  - Agent has preferences over states/action outcomes
    - Encoded in utility or utility function → Utility theory
- "Decision theory = Utility theory + Probability theory"
  - Model the world with a probabilistic model
  - Model preferences with a utility (function)
  - Find action that leads to the maximum expected utility, also called decision making

# Outline: Decision Making – Foundations

***Utility Theory***

- Preferences
- Utilities
- Preference structure

*Markov Decision Process / Problem (MDP)*

- Sequence of actions, history, policy
- Value iteration, policy iteration

*Reinforcement Learning (RL)*

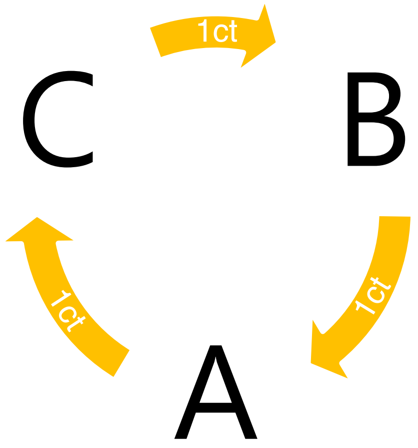
- Passive and active, model-free and model-based RL
- Multi-armed bandit

# Preferences

- An agent chooses among prizes ($A$, $B$, etc.) and lotteries, i.e., situations with uncertain prizes
  - Outcome of a nondeterministic action is a lottery
- Lottery $L = [p, A; (1-p), B]$
  - $A$ and $B$ can be lotteries again
  - Prizes are special lotteries: $[1, R; 0, \text{not } R]$
  - More than two outcomes:
    - $L = [p_1, S_1; p_2, S_2; \cdots; p_M, S_M], \sum_{i=1}^{M} p_i = 1$
- Notation
  - $A \succ B$ $A$ preferred to $B$
  - $A \sim B$ indifference between $A$ and $B$
  - $A \succsim B$ $B$ not preferred to $A$

# Rational Preferences

- Idea: preferences of a rational agent must obey constraints
  - As prerequisite for reasonable preference relations
- Rational preferences → behaviour describable as maximisation of expected utility
- Violating constraints leads to self-evident irrationality
  - Example
    - An agent with intransitive preferences can be induced to give away all its money
      - If $B \succ C$, then an agent who has $C$ would pay (say) 1 cent to get $B$
      - If $A \succ B$, then an agent who has $B$ would pay (say) 1 cent to get $A$
      - If $C \succ A$, then an agent who has $A$ would pay (say) 1 cent to get $C$

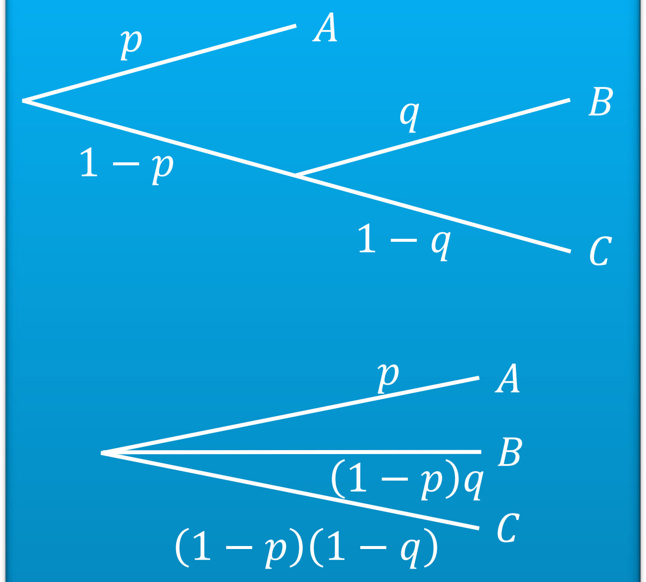


# Axioms of Utility Theory

1. Orderability
   - $(A \succ B) \vee (A \prec B) \vee (A \sim B)$
     - $\{\prec, \succ, \sim\}$ jointly exhaustive, pairwise disjoint
2. Transitivity
   - $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
3. Continuity
   - $A \succ B \succ C \Rightarrow \exists p\, [p, A;\ 1-p, C] \sim B$
4. Substitutability
   - $A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$
     - Also holds if replacing $\sim$ with $\succ$
5. Monotonicity
   - $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; 1-p, B] \succsim [q, A; 1-q, B])$
6. Decomposability
   - $\left[p, A;\ 1-p, [q, B;\ 1-q, C]\right] \sim [p, A;\ (1-p)q, B; (1-p)(1-q), C]$

Decomposability:
There is no fun in gambling.

Equivalent lotteries:



# And Then There Was Utility

- Theorem (Ramsey, 1931; von Neumann and Morgenstern, 1944):
  - Given preferences satisfying the constraints, there exists a real-valued function $U$ such that

$$U(A) \geq U(B) \Leftrightarrow A \gtrsim B$$

    - Existence of a utility function
  - Expected utility of a lottery:

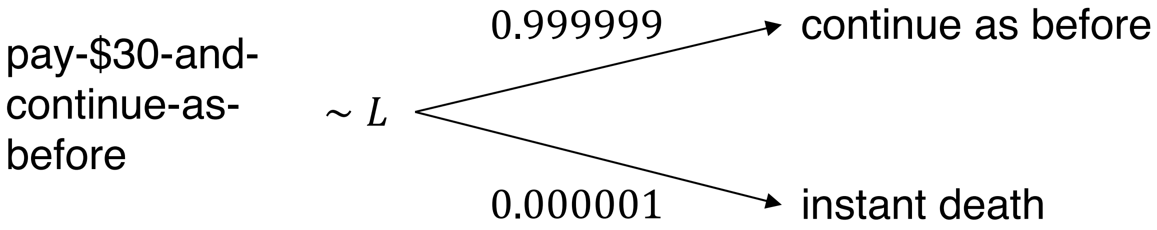
$$U([p_1, S_1; \dots; p_M, S_M]) = \sum_{i=1}^{M} p_i U(S_i)$$

- MEU principle
  - Choose the action that maximises expected utility

# Utilities

- Utilities map states to real numbers.
  Which numbers?

- Standard approach to assessment of human utilities:
  - Compare a given state $A$ to a standard lottery $L_p$ that has
    - “best possible outcome” ⊤ with probability $p$
    - ”worst possible catastrophe” ⊥ with probability $(1 - p)$
  - Adjust lottery probability $p$ until $A \sim L_p$



# Utility Scales

- Normalised utilities: $u_{\top} = 1.0, u_{\bot} = 0.0$
  - Utility of lottery $L \sim$ (pay-$30-and-continue-as-before): $U(L) = u_{\top} \cdot 0.999999 + u_{\bot} \cdot 0.000001 = 0.999999$
- Micromorts: one-millionth chance of death
  - Useful for Russian roulette, paying to reduce product risks, etc.
  - Example for low risk
    - Drive a car for 370km ≈ 1 micromort → lifespan of a car: 150,000km ≈ 400 micromorts
    - Studies showed that many people appear to be willing to pay US$10,000 for a safer car that halves the risk of death → US$50/micromort
- QALYs: quality-adjusted life years
  - Useful for medical decisions involving substantial risk
- In planning: task becomes minimisation of cost instead of maximisation of utility

# Money

- Money does not behave as a utility function
- Given a lottery $L$ with expected monetary value $EMV(L)$, usually $U(L) < U\left(S_{EMV(L)}\right)$, i.e., people are risk-averse
  - $S_M$: state of possessing total wealth $\$M$
  - Utility curve
    - For what probability $p$ am I indifferent between a prize $x$ and a lottery $[p, \$M; (1-p), \$0]$ for large $M$?
    - Right: Typical empirical data, extrapolated with risk-prone behaviour for negative wealth

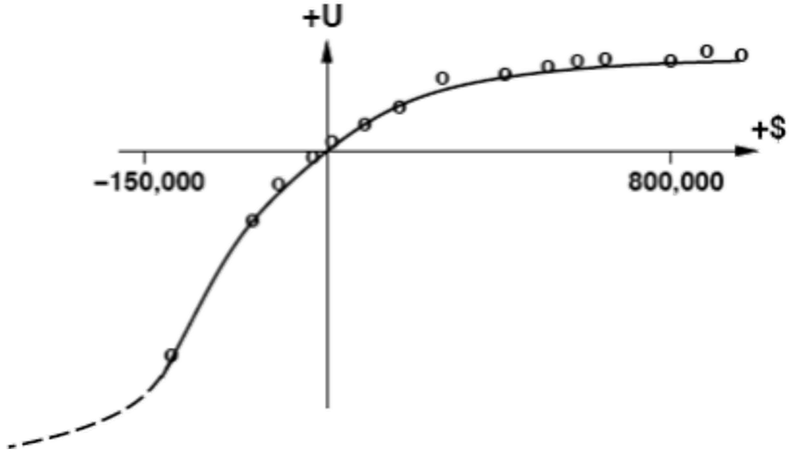
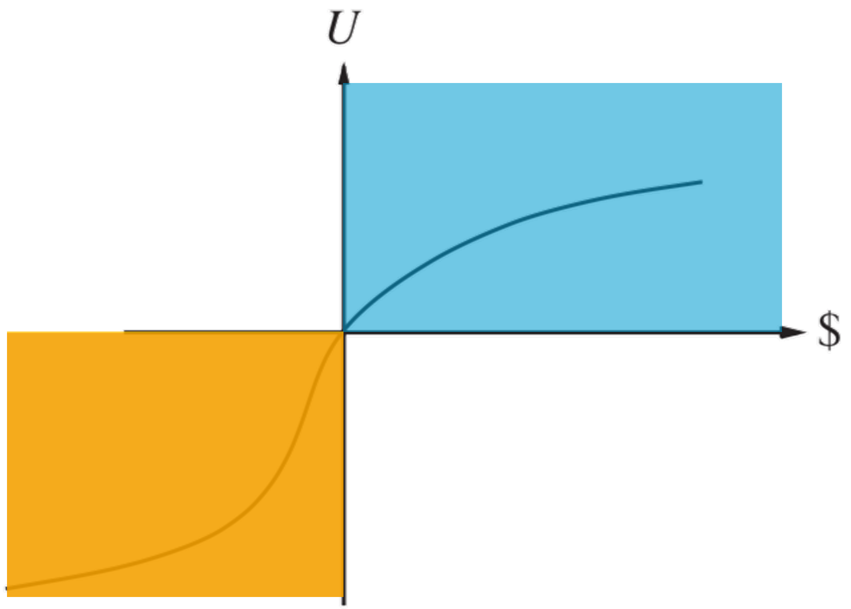



Figure: AIMA, Russell/Norvig

# Money Versus Utility

- Money ≠ Utility
  - More money is better, but not always in a linear relationship to the amount of money
- Expected Monetary Value
  - Risk-averse
    - $U(L) < U(S_{EMV(L)})$
  - Risk-seeking
    - $U(L) > U(S_{EMV(L)})$
  - Risk-neutral
    - $U(L) = U(S_{EMV(L)})$
    - Linear curve
    - For small changes in wealth relative to current wealth



# Utility Scales

- Behaviour is invariant w.r.t. positive linear transformation
$$U'(r) = k_1 U(r) + k_2$$
  - No unique utility function; $U'(r)$ and $U(r)$ yield same behaviour
- With deterministic prizes only (no lottery choices), only ordinal utility can be determined, i.e., total order on prizes
  - Ordinal utility function also called value function
  - Provides a ranking of alternatives (states), but not a meaningful metric scale (numbers do not matter)
- Note:
  An agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities
  - E.g., a lookup table for perfect tic-tac-toe

# Multi-attribute Utility Theory

- A given state may have multiple utilities
  - ...because of multiple evaluation criteria
  - ...because of multiple agents (interested parties) with different utility functions

- There are:
  - Cases in which decisions can be made *without* combining the attribute values into a single utility value
    - Strict dominance
  - Cases in which the utilities of attribute combinations can be specified very concisely
    - Preference structure

# Preference Structure

- To specify the complete utility function $U(r_1, \dots, r_M)$, we need $d^M$ values in the worst case
  - $M$ attributes
  - each attribute with $d$ distinct possible values
  - Worst case meaning: Agent's preferences have no regularity at all
- Supposition in multi-attribute utility theory
  - Preferences of typical agents have much more structure
- Approach
  - Identify regularities in the preference behaviour
  - Use so-called representation theorems to show that an agent with a certain kind of preference structure has a utility function
    $$U(r_1, \dots, r_M) = \Xi[f_1(r_1), \dots, f_M(r_M)]$$
    - where $\Xi$ is hopefully a simple function such as *addition*

# Preference Independence

- $R_1$ and $R_2$ preferentially independent (PI) of $R_3$ iff
  - Preference between $\langle r_1, r_2, r_3 \rangle$ and $\langle r_1', r_2', r_3 \rangle$ does not depend on $r_3$
  - E.g., $\langle Noise, Cost, Safety \rangle$
    - $\langle 20{,}000\ suffer, \$4.6\ billion, 0.06\ deaths/month \rangle$
    - $\langle 70{,}000\ suffer, \$4.2\ billion, 0.06\ deaths/month \rangle$

- Theorem (Leontief, 1947)
  - If every pair of attributes is PI of its complement, then every subset of attributes is PI of its complement
    - Called mutual PI (MPI)

# Preference Independence

- Theorem (Debreu, 1960):
  - MPI ⇒ ∃ *additive* value function

$$V(r_1, \dots, r_M) = \sum_{i=1}^{M} V_i(r_i)$$

  - Hence assess $M$ single-attribute functions
    - Decomposition of $V$ into a set of summands (additive semantics)

  similar to

    - Decomposition of $P_{\boldsymbol{R}}$ into a set of factors (multiplicative semantics)
  - Often a good approximation
  - Example:

$$V(Noise, Cost, Deaths) = -Noise \cdot 10^4 - Cost - Deaths \cdot 10^{12}$$

# Interim Summary

- Preferences
  - Preferences of a rational agent must obey constraints
- Utilities
  - Rational preferences = describable as maximisation of expected utility
  - Utility axioms
  - MEU principle
- Multi-attribute utility theory
  - Preference structure
  - (Mutual) preferential independence

# Outline: Decision Making – Foundations

*Utility Theory*

- Preferences
- Utilities
- Preference structure

***Markov Decision Process / Problem (MDP)***

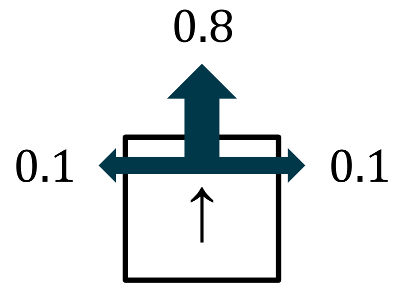
- Sequence of actions, history, policy
- Value iteration, policy iteration

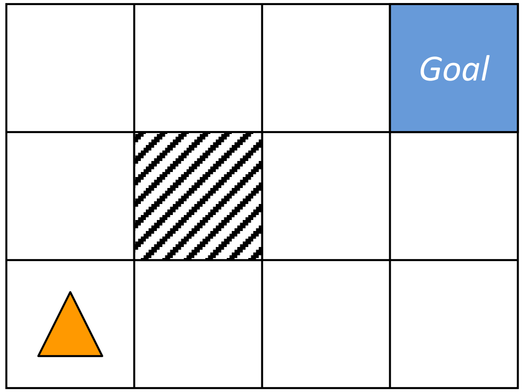
*Reinforcement Learning (RL)*

- Passive and active, model-free and model-based RL
- Multi-armed bandit

# Simple Robot Navigation Problem

- In each state, the possible actions are U, D, R, and L
- The effect of action U is as follows (transition model):
  - With probability 0.8, move up one square
    - If already in top row or blocked, no move
  - With probability 0.1, move right one square
    - If already in rightmost row or blocked, no move
  - With probability 0.1, move left one square
    - If already in leftmost row or blocked, no move
- Same transition model holds for D, R, and L and their respective directions



# Markov Property

The transition properties depend only on the current state, not on previous history (how that state was reached).

- Also known as Markov-$k$ with $k = 1$
  - $k \leq t$

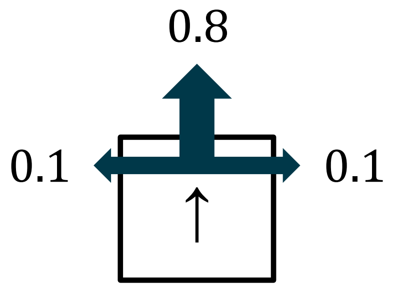
$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(0)}\right) = P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(t-k+1)}\right)$$

  - $k = 1$

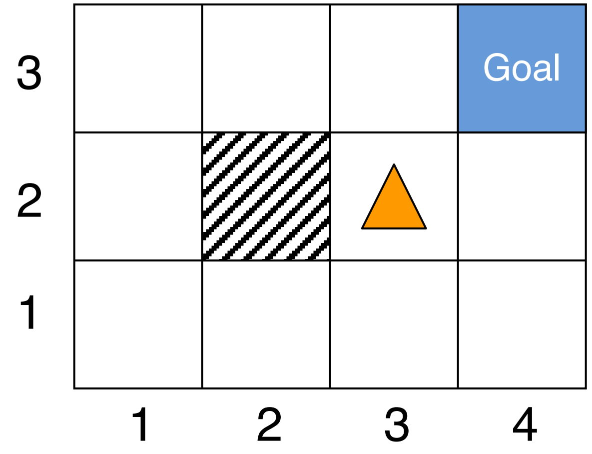
$$P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(0)}\right) = P\left(x^{(t+1)} \middle| x^{(t)}\right)$$

# Sequence of Actions

- In each state, the possible actions are U, D, R, and L; the transition model for each action is (pictured):
- Current position: [3,2]
- Planned sequence of actions: (U, R)

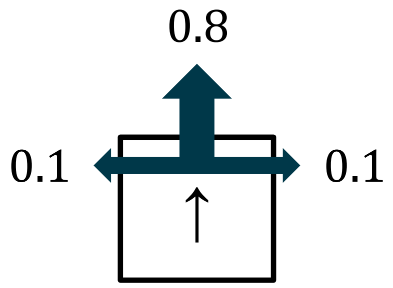


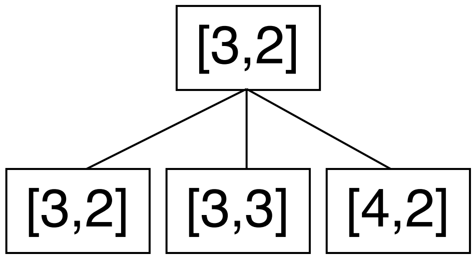

[3,2]

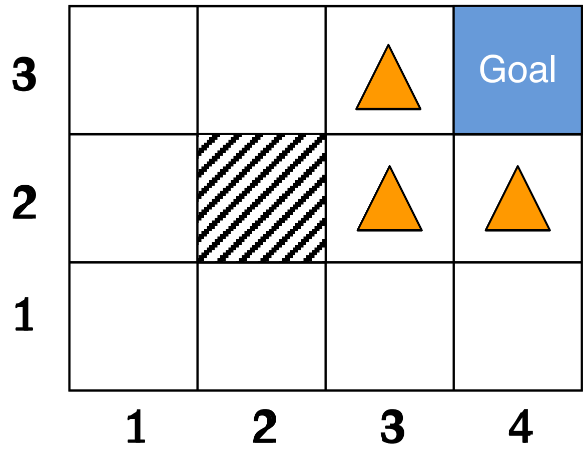


# Sequence of Actions

- In each state, the possible actions are U, D, R, and L; the transition model for each action is (pictured):
- Current position: [3,2]
- Planned sequence of actions: (U, R)
  - U is executed

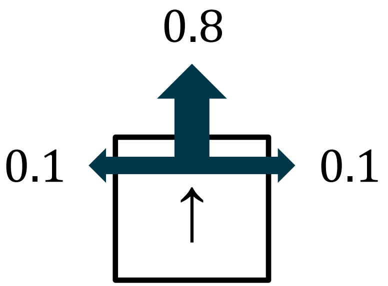


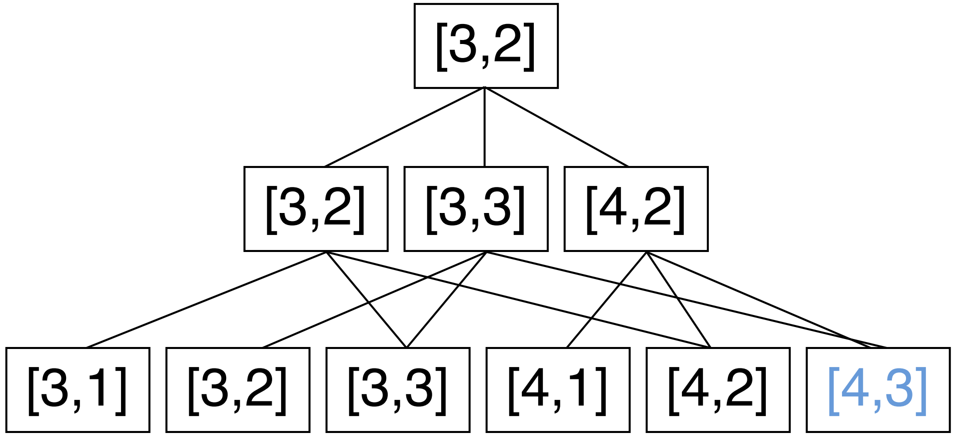



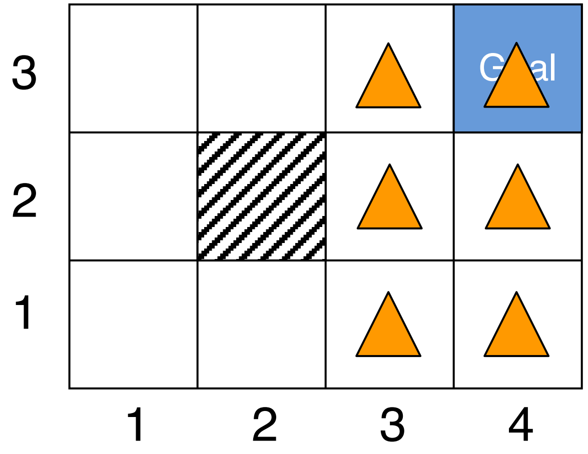



# Sequence of Actions

- In each state, the possible actions are U, D, R, and L; the transition model for each action is (pictured):
- Current position: [3,2]
- Planned sequence of actions: (U, R)
  - U has been executed
  - R is executed

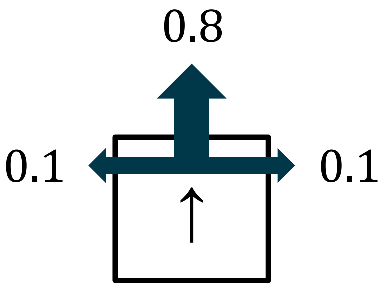


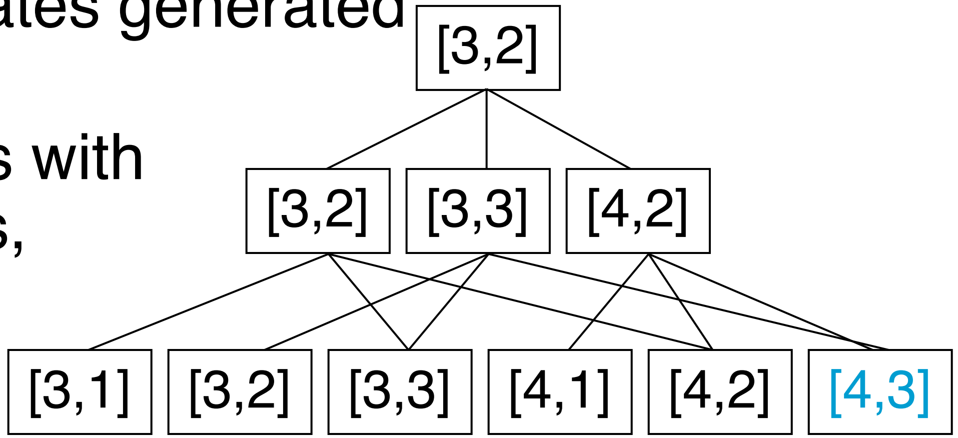



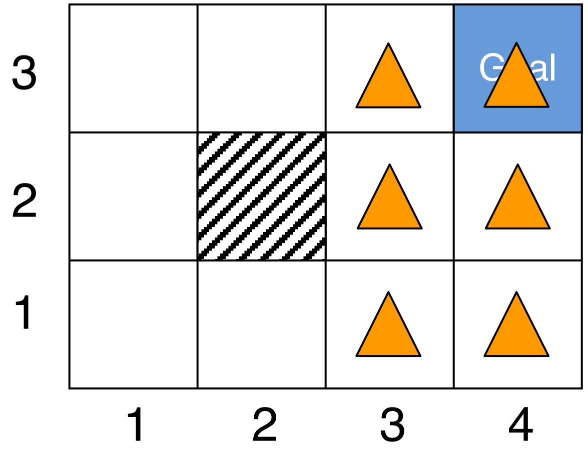



# Histories

- In each state, the possible actions are U, D, R, and L; the transition model for each action is (pictured):
- Current position: [3,2]
- Planned sequence of actions: (U, R)
  - U has been executed
  - R is executed
- History: sequence of states generated by sequence of actions
  - 9 possible sequences with 6 possible final states, only1 of which is a goal state

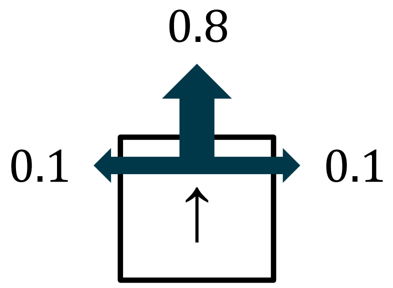


# Probability of Reaching the Goal

- In each state: possible actions U, D, R, L; trans. model:




$P([4,3] \mid (U,R).[3,2]) =$

$P([4,3] \mid R.[3,3]) \cdot P([3,3] \mid U.[3,2])$

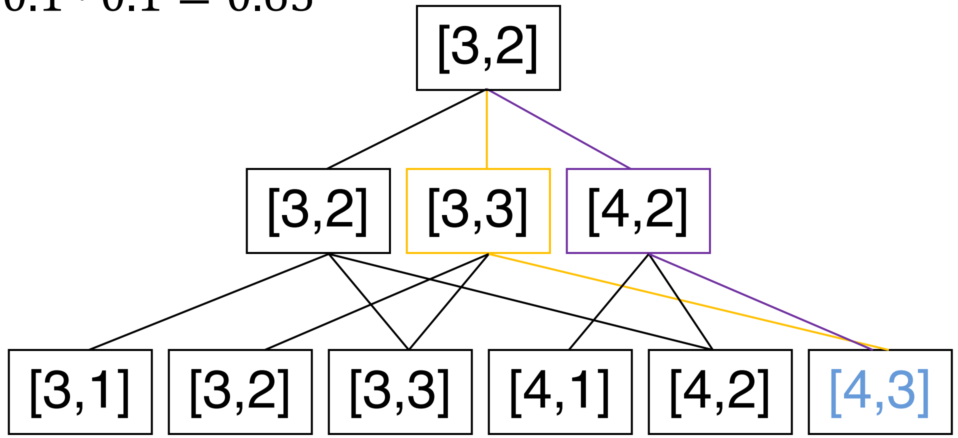
$+P([4,3] \mid R.[4,2]) \cdot P([4,2] \mid U.[3,2])$

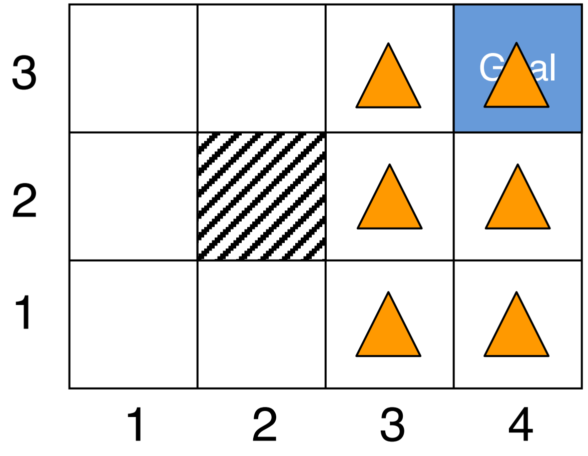
$P([4,3] \mid R.[3,3]) = 0.8$ $P([3,3] \mid U.[3,2]) = 0.8$

$P([4,3] \mid R.[4,2]) = 0.1$ $P([4,2] \mid U.[3,2]) = 0.1$

$P([4,3] \mid (U,R).[3,2]) = 0.8 \cdot 0.8 + 0.1 \cdot 0.1 = 0.65$

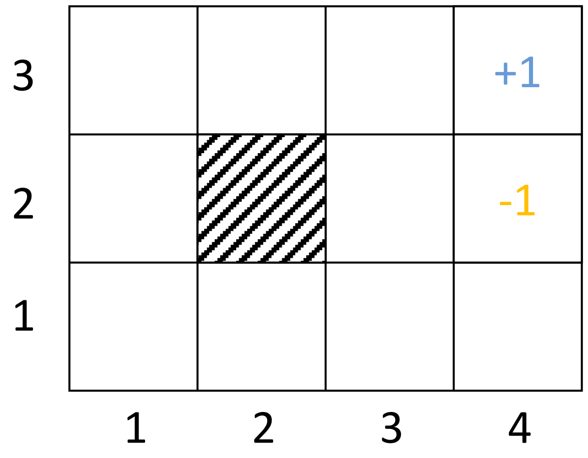
Note importance of Markov property in this derivation



# Utility Function

- [4,3] : power supply (stops the run)
- [4,2] : sand area the robot cannot escape (stops the run)
- Goal: robot needs to recharge its batteries
- [4,3] and [4,2] are terminal states
- In this example, we define the utility of a history by
  - The utility of the last state (+1 or –1) minus $0.04 \cdot n$
    - $n$ is the number of moves
    - I.e., each move costs $0.04$, which provides an incentive to reach the goal fast



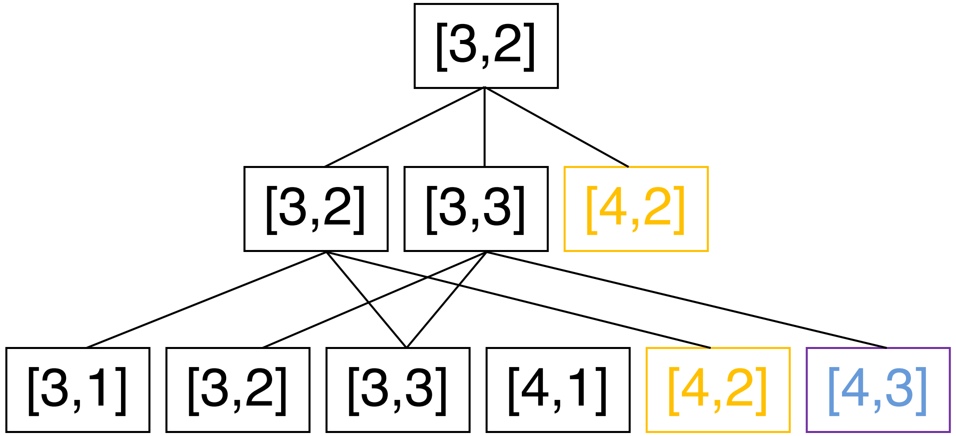
# Utility of an Action Sequence

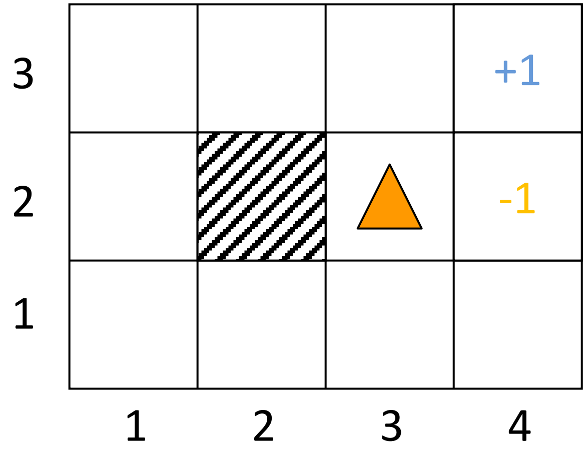
- Consider the action sequence $\boldsymbol{a} = $ (U,R) from [3,2]
- A run produces one of 7 possible histories, each with a probability
- Utility of the sequence is the expected utility of histories $h$:

$$U(\boldsymbol{a}) = \sum_h U_h P(h)$$

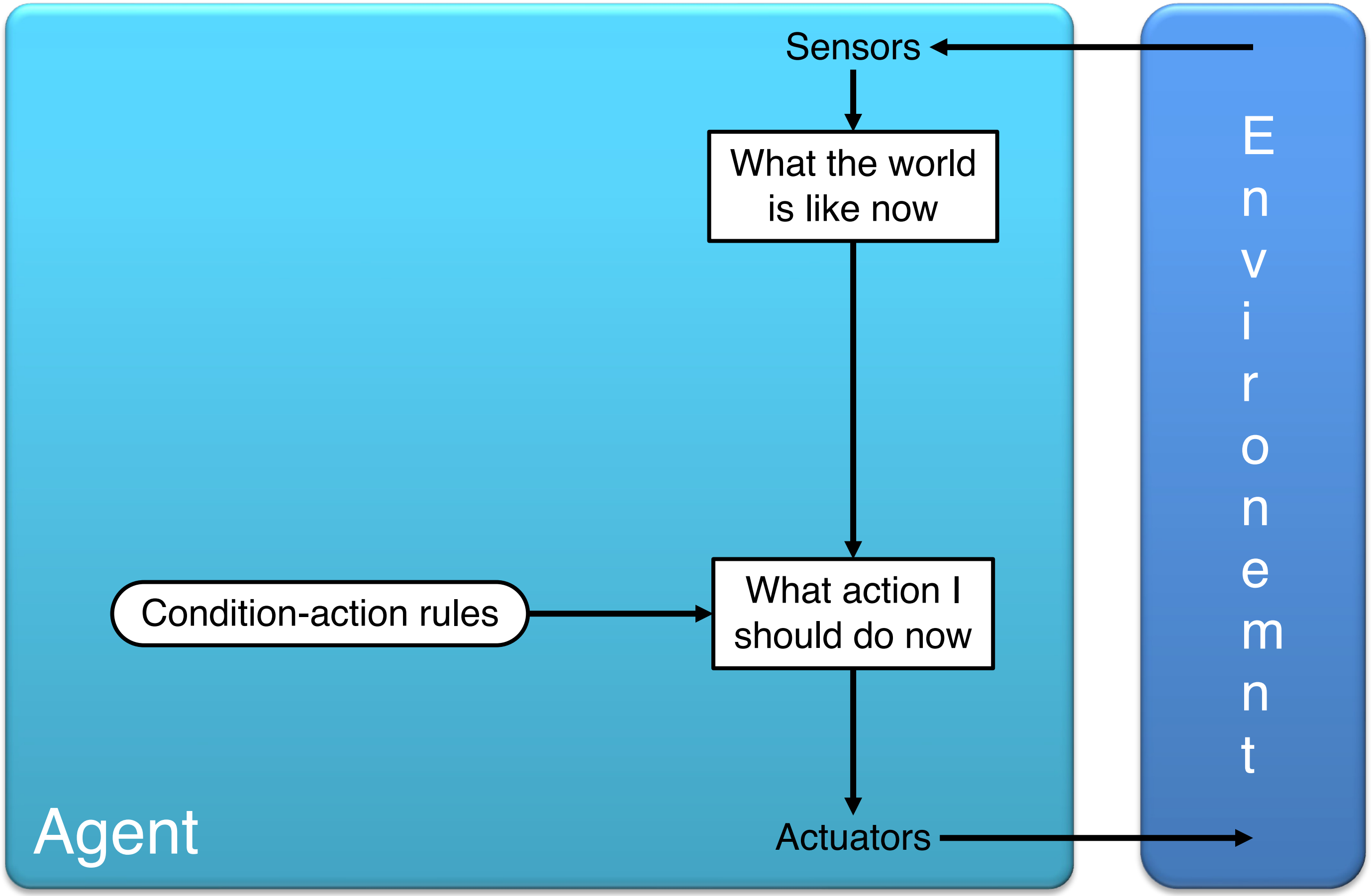
- Optimal sequence = the one with maximum utility

Is the optimal sequence what we want?



# Reactive Agent Algorithm




```
Act()
    repeat
        s ← sensed state
        if s is terminal then
            exit
        a ← choose action (given s)
        perform a
```

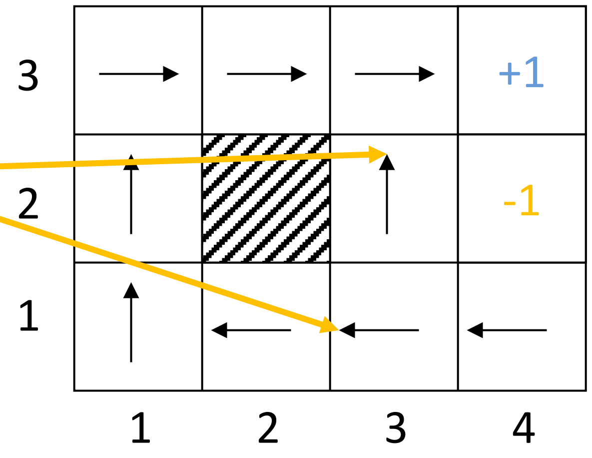
Figure: AIMA, Russell/Norvig

# Policy (Reactive/Closed-loop Strategy)

- Policy $\pi$
  - *Complete* mapping from states to actions
- Optimal policy $\pi^*$
  - Always yields a history (ending at terminal state) with maximum expected utility
    - Due to Markov property

```
Act()
    repeat
        s ← sensed state
        if s is terminal then
            exit
        a ← π(s)
        perform a
```

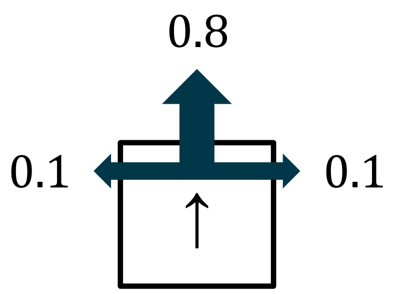
Note that [3,2] is a “dangerous” state that the optimal policy tries to avoid




How to compute $\pi^*$?
Solving a Markov Decision Process

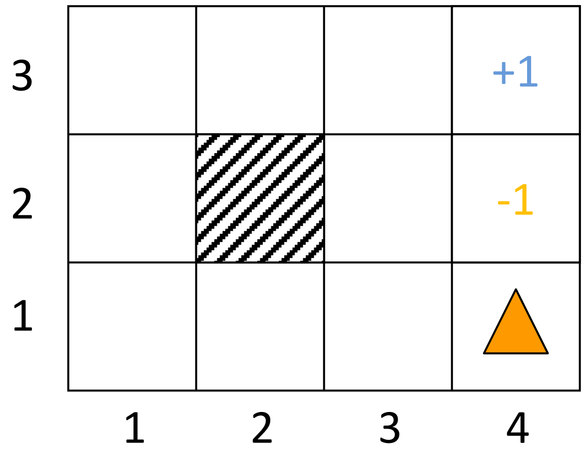
# Markov Decision Process / Problem (MDP)

- *Sequential* decision problem
  for a fully observable, stochastic environment
  with a Markovian transition model
  and additive rewards (next slide)
- MDP is a four-tuple $(S, A, T, R)$ with
  - $S$ a random variable whose domain is a set of states (with an initial state $s^0$)
  - For each $s \in \text{dom}(S)$
    - a set $A(s)$ of actions
    - a transition model $T(s', s, a) = P(s'|s, a)$
    - a reward function $R(s)$ (also with $a$ possible)
- Robot navigation example to the right




U, D, L, R

each move costs 0.04



# Additive Utility

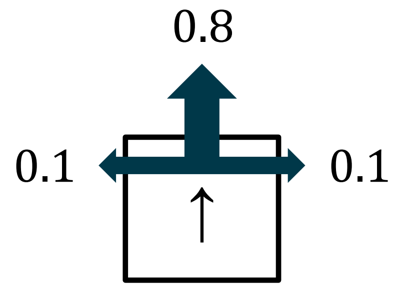
- History $h = \left(s^{(0)}, s^{(1)}, \dots, s^{(T)}\right)$
- In each state $s$, agent receives reward $R(s)$
- Utility of $h$ is additive iff

$$U\left(s^{(0)}, s^{(1)}, \dots, s^{(T)}\right) = R\left(s^{(0)}\right) + U\left(s^{(1)}, \dots, s^{(T)}\right)$$
$$= \sum_{t=0}^{T} R\left(s^{(t)}\right)$$

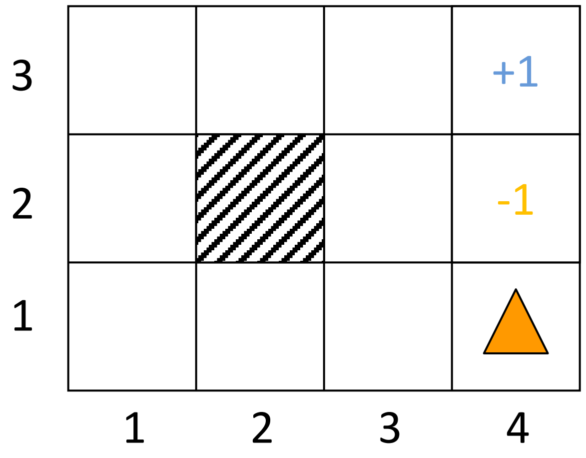
  – Discount factor $\gamma \in ]0,1]$:

$$U\left(s^{(0)}, s^{(1)}, \dots, s^{(T)}\right) = \sum_{t=0}^{T} \gamma^{t} R\left(s^{(t)}\right)$$

    - Close to 0: future rewards insignificant
    - Corresponds to interest rate $1-\gamma/\gamma$



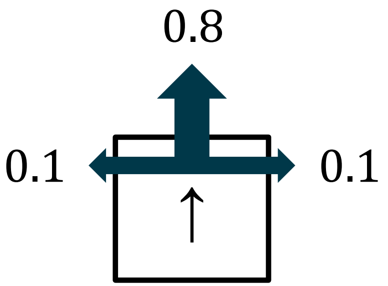

U, D, L, R each move costs 0.04



# Principle of MEU

- Bellman equation:

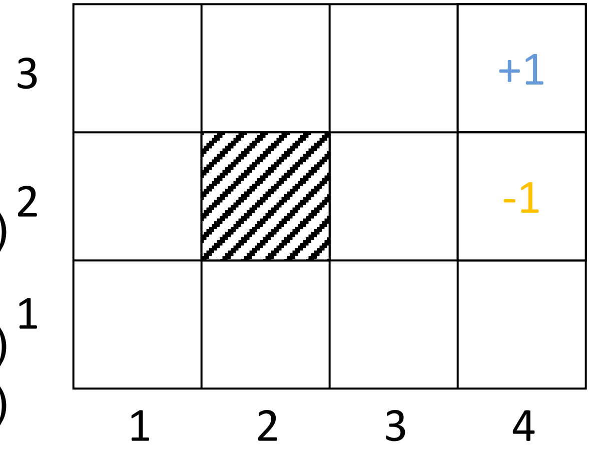
$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \text{dom}(S)} P(s'|a,s)U(s')$$




- Optimal policy:

$$\pi^*(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s' \in \text{dom}(S)} P(s'|a,s)U(s')$$

U, D, L, R each move costs 0.04



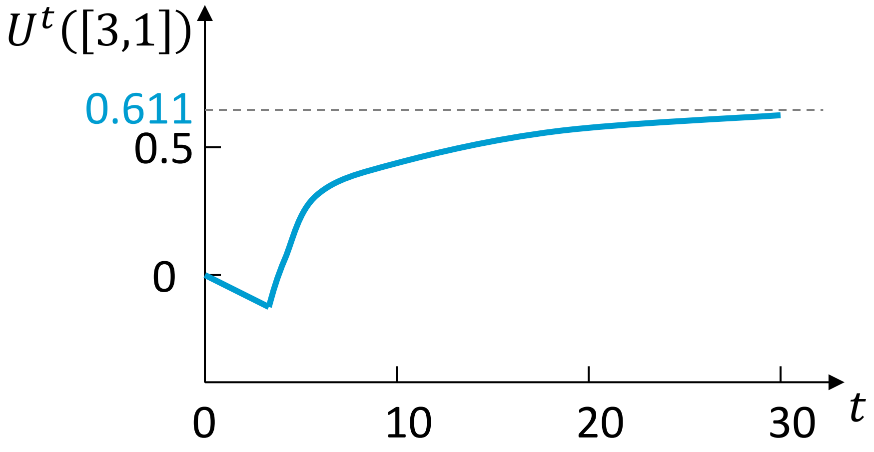

– Bellman equation for $[1,1]$ with $\gamma = 1$ as discount factor
  - $U(1,1) = -0.04 + \gamma \max_{U,L,D,R} \{ 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1),$ (U)
    $0.8U(1,1) + 0.1U(1,1) + 0.1U(1,2),$ (L)
    $0.8U(1,1) + 0.1U(2,1) + 0.1U(1,1),$ (D)
    $0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1) \}$ (R)

# Value Iteration

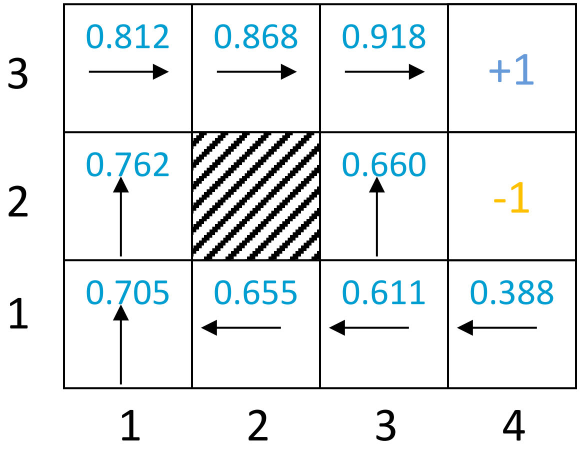
- Initialise the utility of each non-terminal state $s$ to $U^{(0)}(s) = 0$
- For $t = 0, 1, 2, \ldots$, do

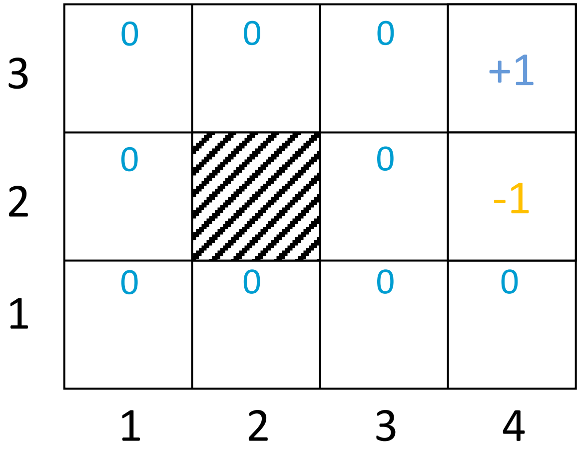
$$U^{(t+1)}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \mathrm{dom}(S)} P(s'|a,s) U^{(t)}(s')$$

  – So called Bellman update




Note the importance of terminal states and connectivity of the state-transition graph



# Value Iteration: Algorithm

- Returns a policy $\pi$ that is optimal
- Inputs
  - MDP $mpd$
    - Set of states $S$
    - For each $s \in S$
      - Set $A(s)$ of applicable actions
      - Transition model $P(s'|s,a)$
      - Reward function $R(s)$
  - Maximum error allowed $\epsilon$

```
function value-iteration(mdp,ϵ)
    U′ ← 0, π ← ⟨⟩
    repeat
        U ← U′
        δ ← 0
        for each state s ∈ S do
            U′[s] ← R(s) + γ max_{a∈A(s)} Σ_{s′} P(s′|a.s) U[s′]
            if |U′[s] - U[s]| > δ then
                δ ← |U′[s] - U[s]|
    until δ < ϵ(1-γ)/γ
    for each state s ∈ S do
        π(s) ← argmax_{a∈A(s)} Σ_{s′} P(s′|a.s) U[s′]
    return π
```

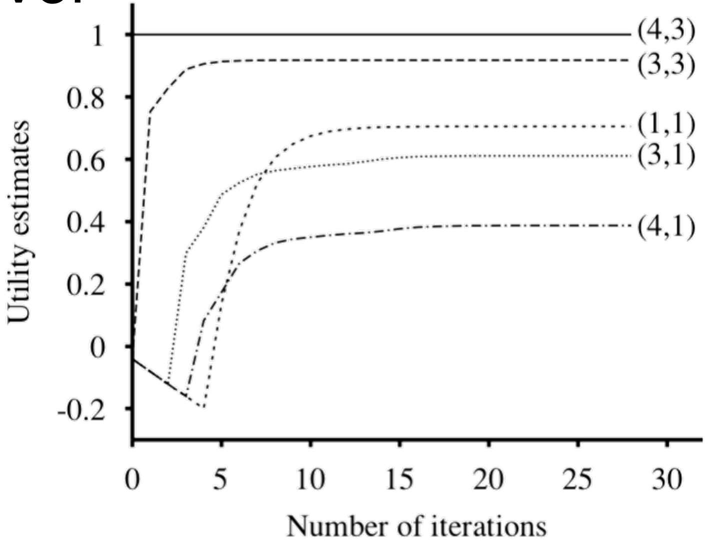
- Local variables
  - $U, U'$ vectors of utilities for states in $S$
  - $\delta$ maximum change in utility of any state in an iteration

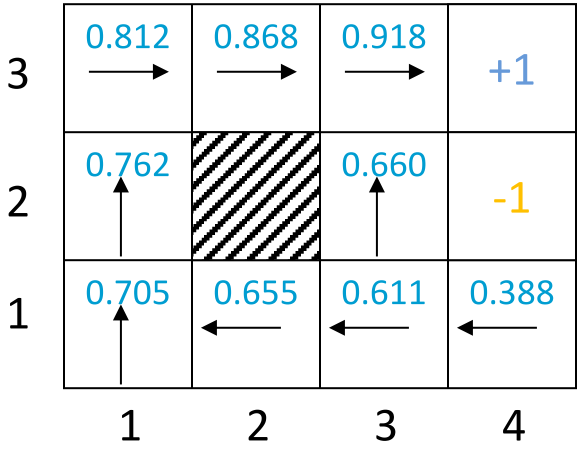
# Evolution of Utilities

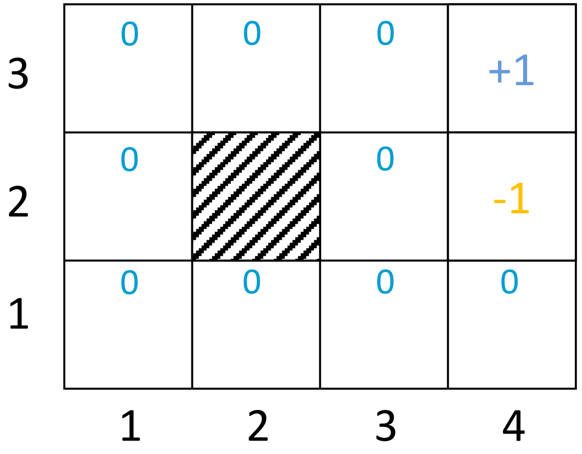
- For $t = 0, 1, 2, \ldots,$ do

$$U^{(t+1)}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \mathrm{dom}(S)} P(s'|a, s) U^{(t)}(s')$$

- Value iteration ≈ information propagation
  – Argmax action may change over time due to utilities changing



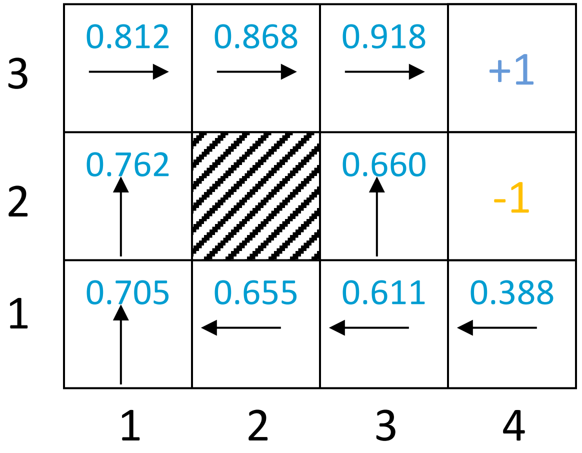



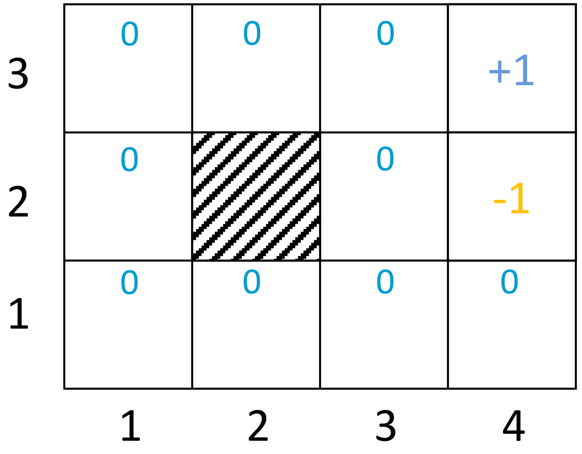
# Effect of Rewards

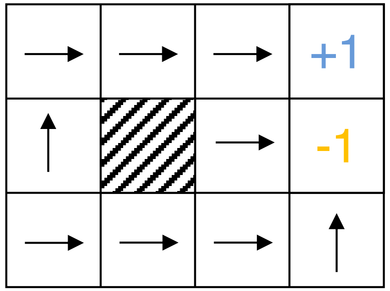
- For $t = 0, 1, 2, \ldots$, do

$$U^{(t+1)}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \mathrm{dom}(S)} P(s'|a,s) U^{(t)}(s')$$

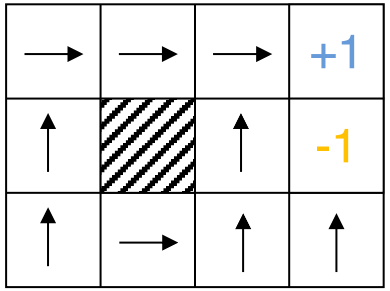
- Optimal policies for different rewards:
  - For $R(s) = -0.04$, see right ⇢



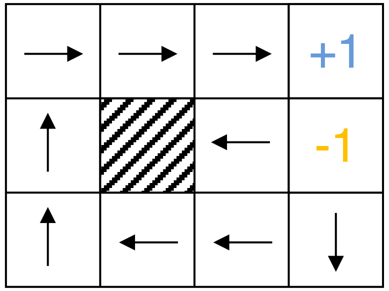



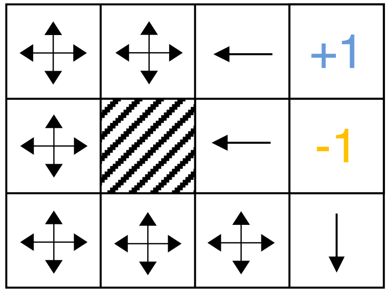

$R(s) < -1.6284$




$-0.4278 < R(s) < -0.0850$




$-0.0221 < R(s) < 0$




$R(s) > 0$



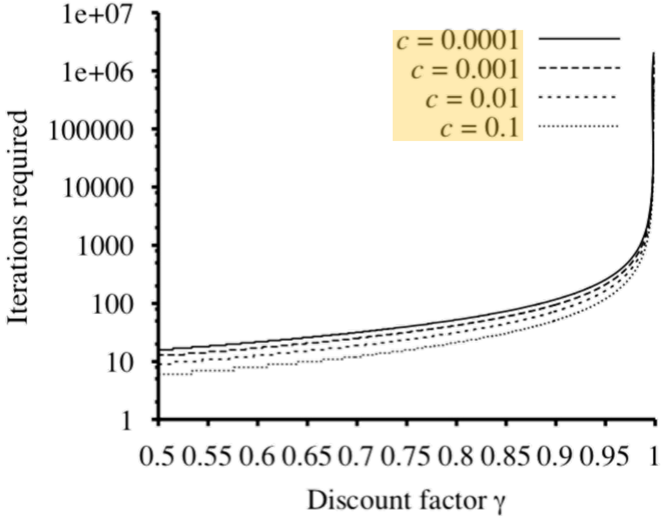
Data for figures: AIMA, Russell/Norvig

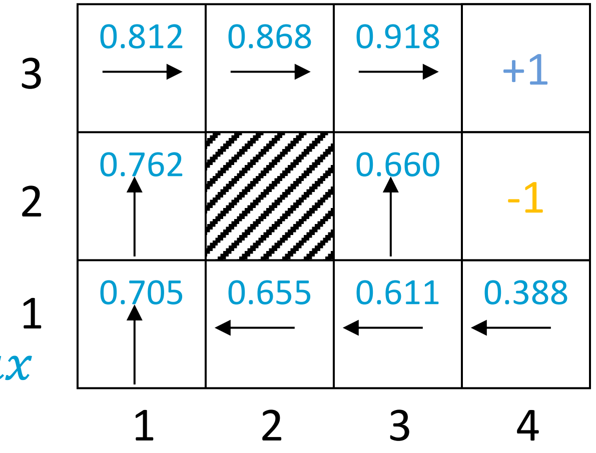
# Effect of Allowed Error & Discount

- For $t = 0, 1, 2, \ldots,$ do

$$U^{(t+1)}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \mathrm{dom}(S)} P(s'|a,s) U^{(t)}(s')$$

- Iterations required to ensure a maximum error of $\varepsilon = c \cdot R_{max}$
  - $R_{max}$ maximum reward



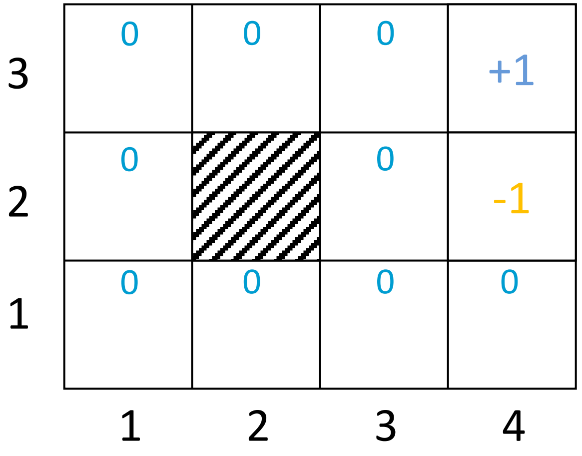




Figure right: AIMA, Russell/Norvig


UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME
Marcel Gehrke

# Policy Iteration

- Pick a policy $\pi_0$ at random
- Repeat:
  - Policy evaluation: Compute the utility of each state for $\pi_t$
    - $U^{(t)}(s) = R(s) + \gamma \sum_{s' \in \text{dom}(S)} P(s'|a,s) U^{(t)}(s')$
      - No longer involves a $\max$ operation as action is determined by $\pi_t$
  - Policy improvement: Compute the policy $\pi_{t+1}$ given $U_t$
    - $\pi^{(t+1)}(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s' \in \text{dom}(S)} P(s'|a,s) U^{(t)}(s')$
  - If $\pi^{(t+1)} = \pi^{(t)}$, then return $\pi^{(t)}$

Solve the set of linear equations:

$$U(s) = R(s) + \gamma \sum_{s' \in \text{dom}(S)} P(s'|a,s) U(s')$$

(often a sparse system)

# Policy Iteration: Algorithm

- Returns a policy $\pi$ that is optimal
  - Inputs: MDP $mpd$
    - Set of states $S$
    - For each $s \in S$
      - Set $A(s)$ of applicable actions
      - Transition model $P(s'|s,a)$
      - Reward function $R(s)$

```
function policy-iteration(mdp)
    repeat
        U ← policy-evaluation(π,U,mdp)
        unchanged ← true
        for each state s ∈ S do
            if max_{a∈A(s)} Σ_{s'} P(s'|a.s) U[s'] > Σ_{s'} P(s'|π[s].s) U[s'] then
                π[s] ← argmax_{a∈A(s)} Σ_{s'} P(s'|a.s) U[s']
                unchanged ← false
    until unchanged
    return π
```

- Local variables
  - $U$ vectors of utilities for states in $S$, initially $0$
  - $\pi$ a policy vector indexed by state, initially random

# Policy Evaluation

- Compute the utility of each state for $\pi$
  - $U^{(t)}(s) = R(s) + \gamma \sum_{s' \in \mathrm{dom}(S)} P(s'|a,s) U^{(t)}(s')$
- Complexity of policy evaluation: $O(n^3)$, $n = |\mathrm{dom}(S)|$
  - For $n$ states, $n$ linear equations with $n$ unknowns
  - Prohibitive for large $n$
- Approximation of utilities
  - Perform $k$ value iteration steps with fixed policy $\pi_t$, return utilities
    - Simplified Bellman update: $U^{(t+1)}(s) = R(s) + \gamma \sum_{s' \in \mathrm{dom}(S)} P(s'|a,s) U^{(t)}(s')$
  - Asynchronous policy iteration (next slide)
    - Pick any subset of states

# Asynchronous Policy Iteration

- Further approximation of policy iteration
  - Pick any subset of states and do one of the following
    - Update utilities
      - Using simplified value iteration as described on previous slide
    - Update the policy
      - Policy improvement as before
- Is not guaranteed to converge to an optimal policy
  - Possible if each state is still visited infinitely often, knowledge about unimportant states, etc.
- Freedom to work on any states allows for design of domain-specific heuristics
  - Update states that are likely to be reached by a good policy

# Intermediate Summary

- MDP
  - Markov property
    - Current state depends only on previous state
  - Sequence of actions, history, policy
    - Sequence of actions may yield multiple histories, i.e., sequences of states, with a utility
    - Policy: complete mapping of states to actions
    - Optimal policy: policy with maximum expected utility
  - Value iteration, policy iteration
    - Algorithms for calculating an optimal policy for an MDP

# Outline: Decision Making – Foundations

*Utility Theory*

- Preferences
- Utilities
- Preference structure

*Markov Decision Process / Problem (MDP)*

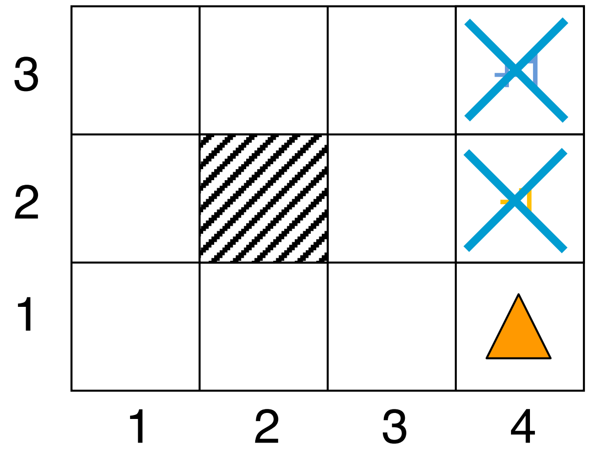
- Sequence of actions, history, policy
- Value iteration, policy iteration

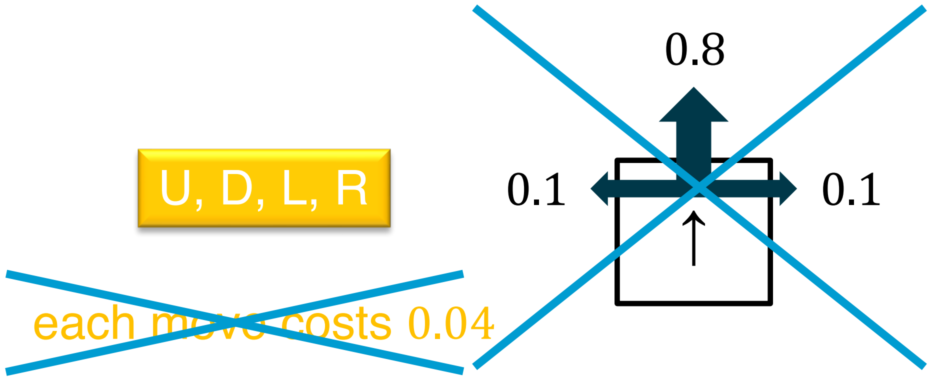
***Reinforcement Learning (RL)***

- Passive and active, model-free and model-based RL
- Multi-armed bandit

# Acting as Reinforcement Learning (RL)

- Agent, placed in an environment, must learn to act optimally in it
- Assume that the world behaves like an MDP, except
  - Agent can act but does not know the transition model
  - Agent observes its current state and its reward but does not know the reward function
- Goal: learn an optimal policy



# Factors That Make RL Hard

- Actions have non-deterministic <u>effects</u>
  – which are initially <u>unknown</u> and must be learned
- Rewards / punishments can be infrequent
  – Often at the end of long sequences of actions
  – How does an agent determine what action(s) were really responsible for reward or punishment?
    - Credit assignment problem
  – World is large and complex

# Passive vs. Active Learning

- Passive learning
  - Agent acts based on a fixed policy $\pi$ and tries to learn how good the policy is by observing the world go by
  - Analogous to policy iteration (without the optimisation part)
- Active learning
  - Agent attempts to find an optimal (or at least good) policy by exploring different actions in the world
  - Analogous to solving the underlying MDP

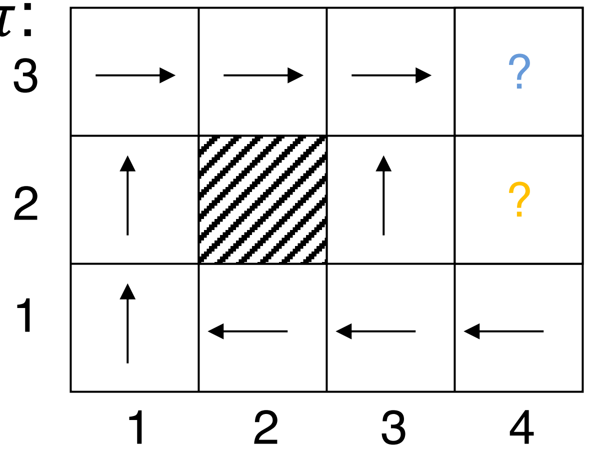
# Model-based vs. Model-free RL

- Model-based approach to RL
  - Learn the MDP model ($P(s'|s,a)$ and $R$), or an approximation of it
  - Use it to find the optimal policy
- Model-free approach to RL
  - Derive the optimal policy without explicitly learning the model

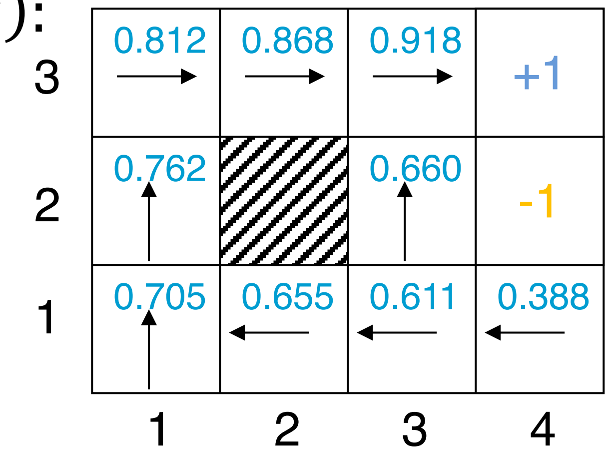
# Passive RL

- Suppose the agent is given a policy
- Wants to determine how good it is

- Given $\pi$:



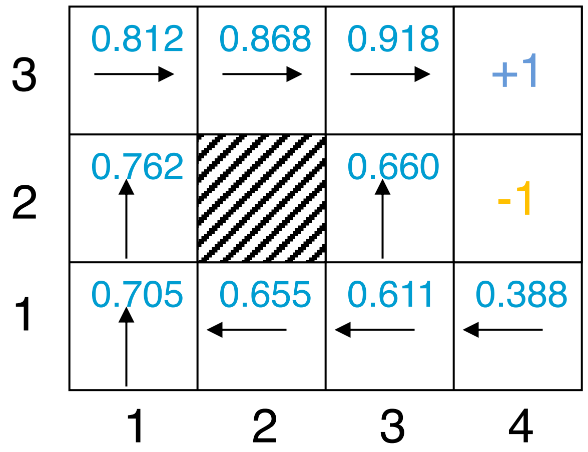

Need to learn $U^{\pi}(s)$:




UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME
Marcel Gehrke

# Passive RL

- Given policy $\pi$:
  - Estimate $U^{\pi}(s)$
- Not given
  - Transition model $P(s'|s,a)$
  - Reward function $R(s)$
- Simply follow the policy for many epochs
  - Epochs: training sequences / trials




$$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (4,3) + 1$$
$$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (4,3) + 1$$
$$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2) - 1$$

  - Assumption: restart or reset possible (or no terminal states with the end of an epoch given by the receipt of a reward)

# Direct Utility Estimation (DUE)

- Model-free approach
  - Estimate $U^{\pi}(s)$ as average total reward of epochs containing $s$
    - Calculating from $s$ to end of epoch
- Reward-to-go of a state $s$
  - The sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed reward-to-go of the state as the direct evidence of the actual expected utility of that state

# DUE: Example

- Suppose the agent observes the following trial:
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
- The total reward starting at $(1,1)$ is $0.72$
  - I.e., a sample of the observed-reward-to-go for $(1,1)$
- For $(1,2)$, there are two samples of the observed-reward-to-go
  - Assuming $\gamma = 1$

1. $(1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
   [Total: $0.76$]
2. $(1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
   [Total: $0.84$]

# DUE: Convergence

- Keep a running average of the observed reward-to-go for each state
  - E.g., for state $(1,2)$, it stores $\frac{(0.76+0.84)}{2} = 0.8$
- As the number of trials goes to infinity, the sample average converges to the true utility

# DUE: Problem

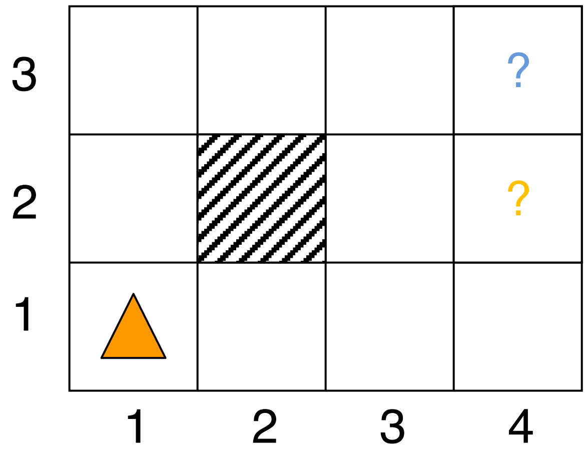
- Big problem: it converges very slowly!
- Why?
  - Does not exploit the fact that utilities of states are not independent
  - Utilities follow the Bellman equation

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

Dependence on neighbouring states

# DUE: Problem

- Using the dependence to your advantage
  - Suppose you know that state $(3,3)$ has a high utility
  - Suppose you are now at $(3,2)$
  - Bellman equation would be able to tell you that $(3,2)$ is likely to have a high utility because $(3,3)$ is a neighbour
- DUE cannot tell you that until the end of the trial



# Adaptive Dynamic Programming (ADP)

- Model-based approach
- Given policy $\pi$:
  - Estimate $U^{\pi}(s)$
  - All while acting in the environment

How?

- Basically learns the transition model $P(s'|s,a)$ and the reward function $R(s)$
  - Takes advantage of constraints in the Bellman equation
- Based on $P(s'|s,a)$ and $R(s)$, performs policy evaluation (part of policy iteration)

# Recap: Policy Iteration

- Pick a policy $\pi_0$ at random
- Repeat:
  – Policy evaluation: Compute the utility of each state for $\pi_t$
    - $U^{(t)}(s) = R(s) + \gamma \sum_{s' \in \text{dom}(S)} P(s'|a,s) U^{(t)}(s')$
      – No longer involves a $\max$ operation as action is determined by $\pi_t$
  – Policy improvement: Compute the policy $\pi_{t+1}$ given $U_t$
    - $\pi^{(t+1)}(s) = \underset{a \in A(s)}{\text{argmax}} \sum_{s' \in \text{dom}(S)} P(s'|a,s) U^{(t)}(s')$
  – If $\pi^{(t+1)} = \pi^{(t)}$, then return $\pi^{(t)}$

Solve the set of linear equations:

$$U(s) = R(s) + \gamma \sum_{s' \in \text{dom}(S)} P(s'|a,s) U(s')$$

(often a sparse system)

Can be solved in $O(n^3)$, where $n = |S|$

# ADP: Estimate the Utilities

- Make use of policy evaluation to estimate the utilities of states
- To use policy equation

$$U^{(t+1)}(s) = R(s) + \gamma \sum_{s' \in \mathrm{dom}(S)} P(s'|\pi(s), s) U^{(t)}(s')$$

  agent needs to learn $P(s'|s,a)$ and $R(s)$
- How?

# ADP: Learn the Model

- Learning $R(s)$
  - Easy because it is deterministic
  - Whenever you see a new state, store the observed reward value as $R(s)$
- Learning $P(s'|s, a)$
  - Keep track of how often you get to state $s'$ given that you are in state $s$ and do action $a$
  - E.g., if you are in $s = (1,3)$ and you execute R three times and you end up in $s' = (2,3)$ twice, then $P(s'|\text{R}, s) = \frac{2}{3}$

# ADP: Algorithm

Update reward function

Update transition model

```
function passive-ADP-agent(percept)
  returns an action
  input: percept, indicating current state s′, reward r′
  static:
     π, fixed policy
     mdp, MDP with P[s′|s,a], R(s), γ
     U, table of utilities, initially empty
     N_sa, table of freq. for s-a pairs, initially 0
     N_sas′, table of freq. for s-a-s′ triples, initially 0
     s,a, previous state and action, initially null
  if s′ is new then
     U[s′] ← r′
     R[s′] ← r′
  if s is not null then
     increment N_sa[s,a] and N_sas′[s,a,s′]
     for each t s.t. N_sas′[s,a,t] ≠ 0 do
        P[t|s,a] ← N_sas′[s,a,t] / N_sa[s,a]
  U ← Policy-evaluation(π,U,mdp)
  if Terminal?(s′) then
     s,a ← null
  else
     s,a ← s′,π[s′]
  return a
```

# ADP: Problem

- Need to solve a system of simultaneous equations – costs $O(n^3)$
  - Very hard to do if you have $10^{50}$ states like in Backgammon
  - Could make things a little easier with modified policy iteration
- Can the agent avoid the computational expense of full policy evaluation?

# Temporal Difference Learning (TD)

- Instead of calculating the exact utility for a state, can the agent approximate it and possibly make it less computationally expensive?
- Yes, it can! Using TD:

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

  - Instead of doing the sum over all successors, only adjust the utility of the state based on the successor observed in the trial
  - Does not estimate the transition model – model-free

# TD: Example

- Suppose you see that $U^\pi(1,3) = 0.84$ and $U^\pi(2,3) = 0.92$
- If the transition $(1,3) \rightarrow (2,3)$ happens all the time, you would expect to see:

$$U^\pi(1,3) = R(1,3) + U^\pi(2,3)$$
$$\Rightarrow U^\pi(1,3) = -0.04 + U^\pi(2,3)$$
$$\Rightarrow U^\pi(1,3) = -0.04 + 0.92 = 0.88$$

- Since you observe $U^\pi(1,3) = 0.84$ in the first trial and it is a little lower than $0.88$, so you might want to "bump" it towards $0.88$

# Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers
  - E.g., to estimate the mean of a random variable from a sequence of samples

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \left(\frac{1}{n+1}\sum_{i=1}^{n} x_i\right) + \frac{1}{n+1}x_{n+1} = \left(\frac{n}{n(n+1)}\sum_{i=1}^{n} x_i\right) + \frac{1}{n+1}x_{n+1}$$

average of $n+1$ samples

$$= \left(\frac{n+1-1}{n(n+1)}\sum_{i=1}^{n} x_i\right) + \frac{1}{n+1}x_{n+1} = \left(\frac{n+1}{n(n+1)}\sum_{i=1}^{n} x_i\right) - \left(\frac{1}{n(n+1)}\sum_{i=1}^{n} x_i\right) + \frac{1}{n+1}x_{n+1}$$

$$= \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) - \left(\frac{1}{(n+1)}\cdot\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{n+1}x_{n+1} = \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i\right)$$

$$= \hat{X}_n + \frac{1}{n+1}\left(x_{n+1} - \hat{X}_n\right)$$

learning rate

sample $n+1$

Given a new sample $x_{n+1}$, the new mean is the old estimate (for $n$ samples) plus the weighted difference between the new sample and old estimate

# TD Update

- TD update for transition from $s$ to $s'$

$$U^\pi(s) = U^\pi(s) + \alpha\big(R(s) + \gamma U^\pi(s') - U^\pi(s)\big)$$

learning rate

new (noisy) sample of utility based on next state

  - Similar to one step of value iteration
  - Equation called backup
- So, the update is maintaining a "mean" of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g., $1/n$), then the utility estimates will eventually converge to true values

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

# TD: Convergence

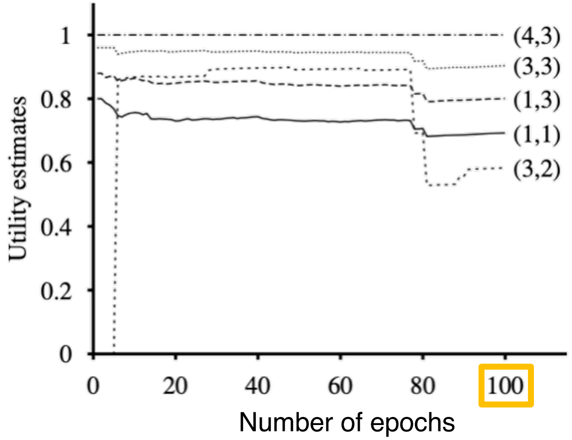
- Since TD uses the observed successor $s'$ instead of all the successors, what happens if the transition $s \longrightarrow s'$ is very rare and there is a big jump in utilities from $s$ to $s'$?
  - How can $U^{\pi}(s)$ converge to the true equilibrium value?
- Answer:
  The average value of $U^{\pi}(s)$ will converge to the correct value
  - This means the agent needs to observe enough trials that have transitions from $s$ to its successors
  - Essentially, the effects of the TD backups will be averaged over a large number of transitions
  - Rare transitions will be rare in the set of transitions observed

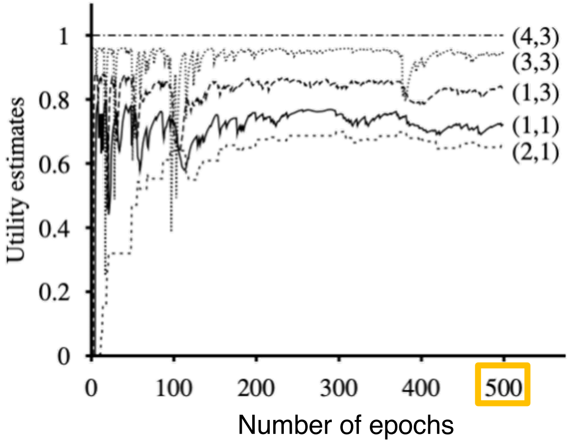
# Comparison between ADP and TD

- Advantages of ADP
  - Converges to true utilities in fewer iterations
  - Utility estimates do not vary as much from the true utilities
- Advantages of TD
  - Simpler, less computation per observation
  - Crude but efficient first approximation to ADP
  - Do not need to build a transition model to perform its updates

# ADP and TD

- Utility estimates for 4x3 grid
  - ADP, given optimal policy (above)
    - Notice the large changes occurring around the 78th trial—this is the first time that the agent falls into the –1 terminal state at (4,2)
  - TD (below)
    - More epochs required
    - Faster runtime per epoch



# Overall comparisons

- DUE (model-free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints and converges slowly
- ADP (model-based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of epochs)
- TD (model-free)
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman constraints – adjusts state to “agree” with observed successor
    - Not all possible successors
  - Convergence in between DUE and ADP

# Passive Learning: Disadvantage

- Learning $U^{\pi}(s)$ does not lead to an optimal policy, why?
  - Only evaluated $\pi$ (no optimisation)
  - Models are incomplete/inaccurate
  - Agent has only tried limited actions, cannot gain a good overall understanding of $P(s'|s, a)$
- Solution: Active learning

# Goal of Active Learning

- Assume that the agent still has access to some sequence of trials performed by the agent
  - Agent is not following any specific policy
  - Assume for now that the sequences should include a thorough exploration of the space
  - We will talk about how to get such sequences later

- The goal is to learn an optimal policy from such sequences
  - Active RL agents
    - Active ADP agent
    - Q-learner (based on TD algorithm)

# Active ADP Agent

- Model-based approach
- Using the data from its trials, agent estimates a transition model $\hat{T}$ and a reward function $\hat{R}$
  - With $\hat{T}(s,a,s')$ and $\hat{R}(s)$, it has an estimate of the underlying MDP
  - Like passive ADP using policy evaluation
- Given estimate of the MDP, it can compute the optimal policy by solving the Bellman equations using value or policy iteration

$$U(s) = \hat{R}(s) + \gamma \max_{a \in A(s)} \sum_{s' \in \mathrm{dom}(S)} \hat{T}(s,a,s')U(s')$$

- If $\hat{T}$ and $\hat{R}$ are accurate estimations of the underlying MDP model, agent can find the optimal policy this way

# Issues with ADP Approach

- Need to maintain MDP model
- $T$ can be very large, $O(|S|^2 \cdot |A|)$
- Also, finding the optimal action requires solving the Bellman equation – time consuming
- Can the agent avoid this large computational complexity both in terms of time and space?

# Q-learning

- So far, focus on utilities for states
  - $U(s)$ = utility of state $s$ = expected maximum future rewards
- Alternative: store Q-values
  - $Q(a,s)$ = utility of taking action $a$ at state $s$
    = expected maximum future reward if action $a$ taken at state $s$

- Relationship between $U(s)$ and $Q(a,s)$?

$$U(s) = \max_{a \in A(s)} Q(a,s)$$

# Q-learning can be model-free

- Note that after computing $U(s)$, to obtain the optimal policy, the agent needs to compute

$$\pi(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s' \in \operatorname{dom}(S)} T(s, a, s') U(s')$$

  - Requires $T$, model of the world
  - Even if it uses TD learning (model-free), it still needs the model to get the optimal policy
- However, if the agent successfully estimates $Q(a, s)$ for all $a$ and $s$, it can compute the optimal policy without using the model

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

# Q-learning

- At equilibrium when Q-values are correct, we can write the constraint equation:

$$Q(a, s) = R(s) + \gamma \sum_{s' \in \text{dom}(S)} T(s, a, s') U(s')$$

Reward at state $s$

Expected value for action-state pair $(a, s)$

Expected value averaged over all possible states $s'$ that can be reached from $s$ after executing action $a$

# Q-learning

- At equilibrium when Q-values are correct, we can write the constraint equation:

$$Q(a, s) = R(s) + \gamma \sum_{s' \in \mathrm{dom}(S)} T(s, a, s') \max_{a' \in A(s')} Q(a', s')$$

Expected value for action-state pair $(a, s)$

Reward at state $s$

Expected value averaged over all possible states $s'$ that can be reached from $s$ after executing action $a$

Best value at the next state = max over all actions in state $s'$

# Q-learning without a Model

- Q-update: after moving from $s$ to state $s'$ using action $a$

$$Q(a,s) \leftarrow Q(a,s) + \alpha\left(R(s) + \gamma \max_{a' \in A(s')} Q(a',s') - Q(a,s)\right)$$

New estimate of $Q(a,s)$

Old estimate of $Q(a,s)$

Learning rate $0 < \alpha < 1$

Difference between old estimate $Q(a,s)$ and the new noisy sample after taking action $a$

  – TD approach
  – Transition model does not appear anywhere!
  – Once converged, optimal policy can be computed without transition model
    • Completely model-free learning algorithm

# Q-learning: Convergence

- Guaranteed to converge to true Q-values given enough exploration
- Very general procedure
  - Because it is model-free
- Converges slower than ADP agent
  - Because it is completely model-free and it does not enforce consistency among values through the model

# Exploitation vs. Exploration

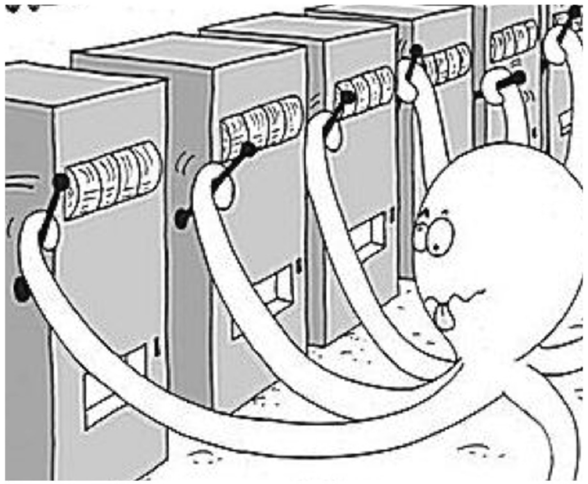
- Actions are always taken for one of the two following purposes
  - Exploitation: Execute the current optimal policy to get high payoff
  - Exploration: Try new sequences of (possibly random) actions to improve the agent's knowledge of the environment even though current model does not show they have a high payoff
- Pure exploitation: gets stuck in a rut
- Pure exploration: not much use if you do not put that knowledge into practice

# Multi-Arm Bandit Problem

- So far, we assumed that the agent has a set of epochs of sufficient exploration

- Multi-arm bandit problem:
  Statistical model of sequential experiments
  - Name comes from a traditional slot machine (one-armed bandit)

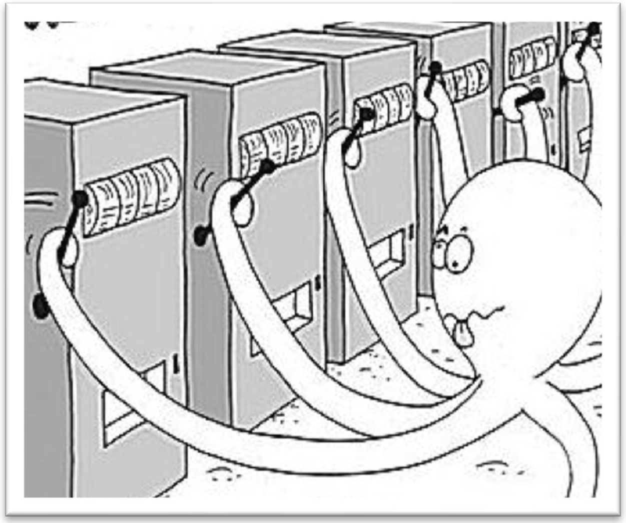
- Question:
  Which machine to play?



# Actions

- $n$ arms, each with a fixed but unknown distribution of reward
  - In terms of actions: Multiple actions $a_1, a_2, \dots, a_n$
    - Each $a_i$ provides a reward from an unknown (but stationary) probability distribution $p_i$
    - Specifically, expectation $\mu_i$ of machine $i$'s reward unknown
      - If all $\mu_i$'s were known, then the task is easy: just pick $\underset{i}{\operatorname{argmax}}\, \mu_i$

- With $\mu_i$'s unknown, question is which arm to pull



# Formal Model

- At each time step $t = 1, 2, \ldots, T$:
  - Each machine $i$ has a random reward $X_i^{(t)}$
    - $E\left[X_i^{(t)}\right] = \mu_i$ independent of the past (Markov property again)
  - Pick a machine $I_t$ and get reward $X_{I_t}^{(t)}$
  - Other machines' rewards hidden

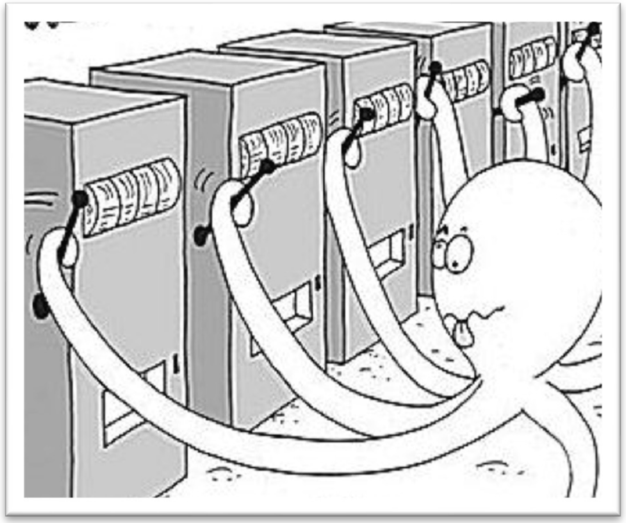
- Over $T$ time steps, the agent has a total reward of $\sum_{t=1}^{T} X_{I_t}^{(t)}$
  - If all $\mu_i$'s known, it would have selected $\arg\max_i \mu_i$ at each time $t$
    - Expected total reward $T \cdot \max_i \mu_i$
- Agent's "regret": $T \cdot \max_i \mu_i - \sum_{t=1}^{T} X_{I_t}^{(t)}$

best machine's reward (in expectation)

agent's reward

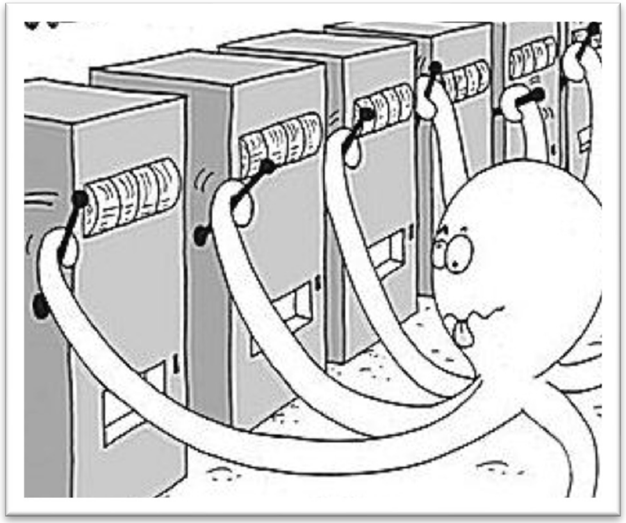
# Exploitation vs. Exploration Reprise

- Exploration: to find the best
  - Overhead: big loss when trying the bad arms
- Exploitation: to exploit what the agent has discovered
  - Weakness: there may be better ones that it has not explored and identified
- *Question*:
  *With a fixed budget, how to balance exploration and exploitation such that the total loss (or regret) is small?*



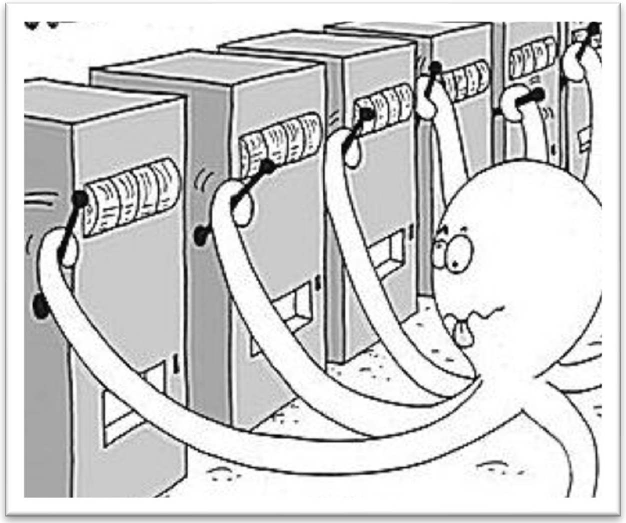
# Where Does the Loss Come from?

- If $\mu_i$ is small, trying this arm too many times makes a big loss
  - So the agent should try it less if it finds the previous samples from it are bad
- But how to know whether an arm is good?
- The more the agent tries an arm $i$, the more information it gets about its distribution
  - In particular, the better estimate to its mean $\mu_i$



# Where Does the Loss Come from?

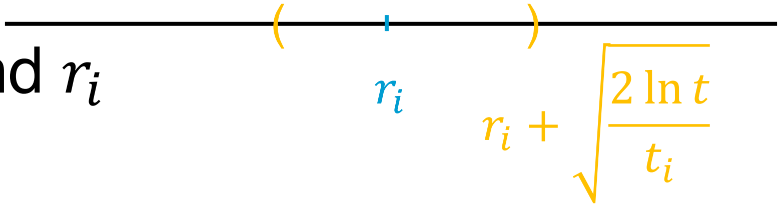
- So the agent wants to estimate each $\mu_i$ precisely, and at the same time, it does not want to try bad arms too often
  - Two competing tasks
    - Exploration vs. exploitation dilemma
- Rough idea: the agent tries an arm if
  - Either
    it has not tried it often enough
  - Or
    its estimate of $\mu_i$ so far is high

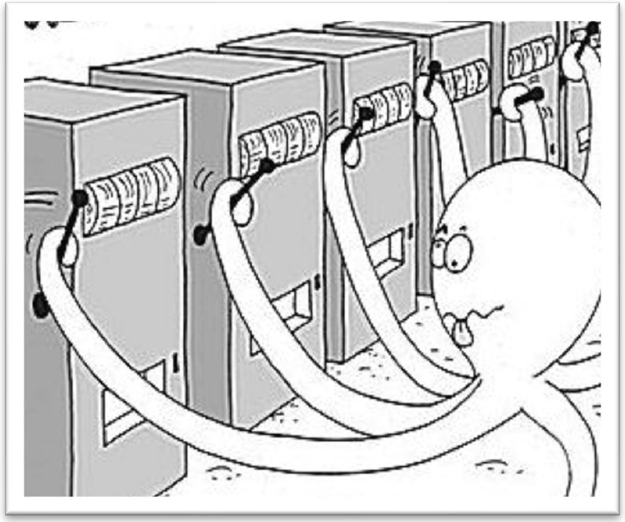


# UCB (Upper Confidence Bound) Algorithm

- Input: Set of actions $A$
- Assume rewards between 0 and 1
  - If they are not, normalise them
- For each action $a_i$ , let
  - $r_i$ = average reward from $a_i$
  - $t_i$ = number of times $a_i$ tried
- $t = \sum_i t_i$
- Confidence interval around $r_i$

```
UCB(A)
   Try each action a_i once
   loop
      choose an action a_i that has
         the highest value of r_i + √(2·ln(t)/t_i)
      perform a_i
      update r_i , t_i , t
```



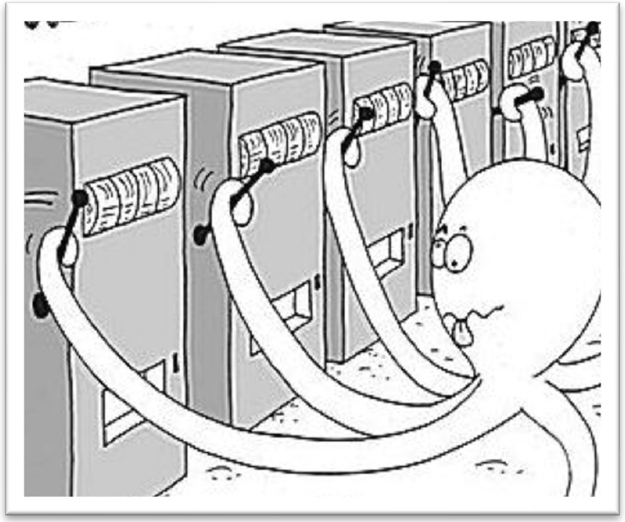
# UCB: Performance

- Theorem: If each distribution of reward has support in $[0,1]$, i.e., rewards are normalised, then the regret of the UCB algorithm is at most

$$O\left(\sum_{i:\mu_i<\mu^*} \frac{\ln T}{\Delta_i} + \sum_{j\in\{1,\dots,n\}} \Delta_j\right)$$

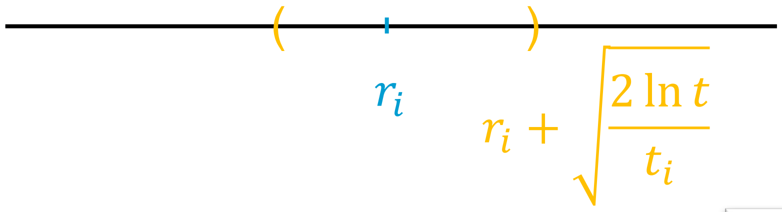
  - $\mu^* = \max_{i} \mu_i$
  - $\Delta_i = \mu^* - \mu_i$
    - Expected loss of choosing $a_i$ once
  - [without proof]

- Loss grows very slowly with $T$

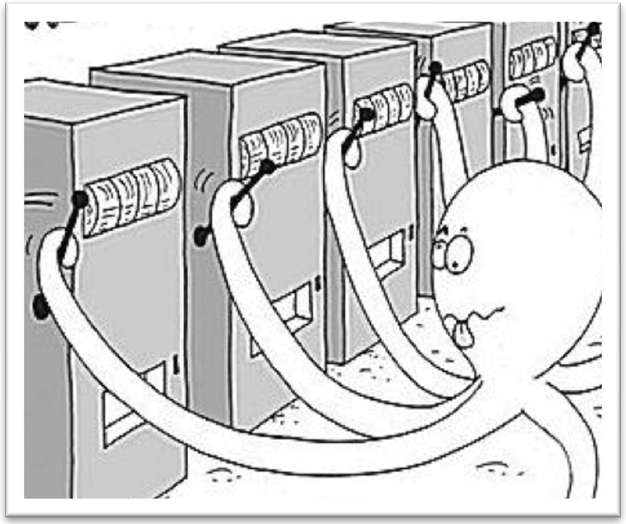


# UCB: Performance

- Uses principle of optimism in face of uncertainty
  - Agent does not have a good estimate $\hat{\mu}_i$ of $\mu_i$ before trying it many times
    - Thus give a big confidence interval $[-c_i, c_i]$ for such $i$

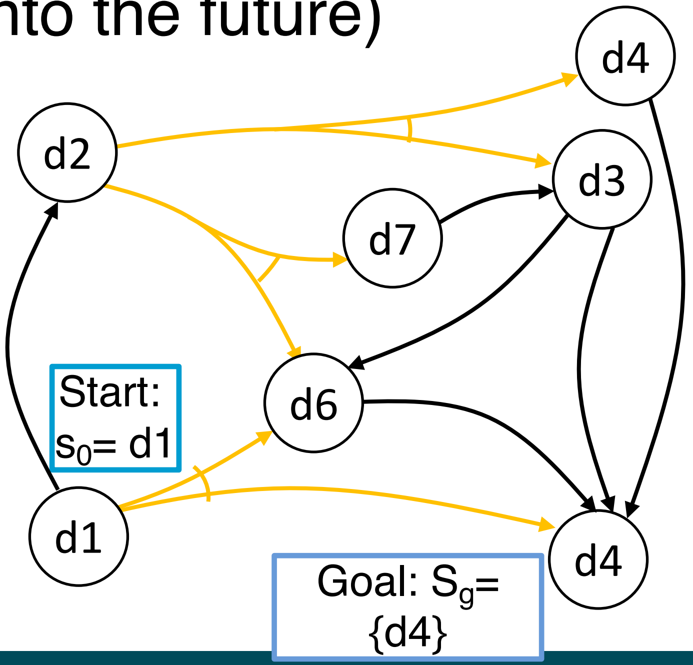



  - $c_i = \sqrt{\frac{2 \ln t}{t_i}}$
    - And select an $i$ with maximum $\mu_i + c_i$
  - If an action has not been tried many times, then the big confidence interval makes it still possible to be tried
  - I.e., in face of uncertainty (of $\mu_i$), the agent acts optimistically by giving chances to those that have not been tried enough



# UCT Algorithm

- Recursive UCB computation to compute $Q(s,a)$ for *cost*
  - Min ops instead of max
  - Planning domain $\Sigma$, state $s$
  - Horizon $h$ (steps into the future)
- Anytime algorithm:
  - Call repeatedly until time runs out
  - Then choose action $\underset{a}{\operatorname{argmin}}\, Q(s,a)$




```
UCT(Σ,s,h)
    if s ∈ S_g then
        return 0
    if h = 0 then
        return V_0(s)
    if s ∉ Envelope then
        add s to Envelope
        n(s) ← 0
        for all a ∈ Applicable(s) do
            Q(s,a) ← 0
            n(s,a) ← 0
    Untried ← {a ∈ Applicable(s)| n(s,a)=0}
    if Untried ≠ ∅ then
        ã ← Choose(Untried)
    else
        ã ← argmin_{a∈Applicable(s)}
              {Q(s,a)-C·[log(n(s))/n(s,a)]^½}
    s' ← Sample(Σ,s,ã)
    cost-rollout ← cost(s,ã) + UCT(s',h-1)
    Q(s,ã) ← [n(s,ã)·Q(s,ã)+cost-rollout]
               /(1+n(s,ã))
    n(s) ← n(s) + 1
    n(s,ã) ← n(s,ã) + 1
    return cost-rollout
```

# UCT as an Acting Procedure

- Suppose probabilities and costs unknown
- Suppose you can restart your actor as many times as you want
- Can modify UCT to be an acting procedure
  - Use it to explore the environment

```
UCT(Σ,s,h)
    if s ∈ S_g then
        return 0
    if h = 0 then
        return V_0(s)
    if s ∉ Envelope then
        add s to Envelope
        n(s) ← 0
        for all a ∈ Applicable(s) do
            Q(s,a) ← 0
            n(s,a) ← 0
    Untried ← {a ∈ Applicable(s)| n(s,a)=0}
    if Untried ≠ ∅ then
        ã ← Choose(Untried)
    else
        ã ← argmin_{a∈Applicable(s)}
              {Q(s,a)-C·[log(n(s))/n(s,a)]^½}
    s' ← Sample(Σ,s,ã)
    cost-rollout ← cost(s,ã) + UCT(s',h-1)
    Q(s,ã) ← [n(s,ã)·Q(s,ã)+cost-rollout]
                 /(1+n(s,ã))
    n(s) ← n(s) + 1
    n(s,ã) ← n(s,ã) + 1
    return cost-rollout
```

perform $\tilde{a}$; observe $s'$

# UCT as a Learning Procedure

- Suppose probabilities and costs unknown
  - But you have an accurate simulator for the environment
- Run UCT multiple times in the simulated environment
  - Learn what actions work best

```
UCT(Σ,s,h)
    if s ∈ S_g then
        return 0
    if h = 0 then
        return V_0(s)
    if s ∉ Envelope then
        add s to Envelope
        n(s) ← 0
        for all a ∈ Applicable(s) do
            Q(s,a) ← 0
            n(s,a) ← 0
    Untried ← {a ∈ Applicable(s)| n(s,a)=0}
    if Untried ≠ ∅ then
        ã ← Choose(Untried)
    else
        ã ← argmin_{a∈Applicable(s)}
              {Q(s,a)-C·[log(n(s))/n(s,a)]^½}
    s' ← Sample(Σ,s,ã)
    cost-rollout ← cost(s,ã) + UCT(s',h-1)
    Q(s,ã) ← [n(s,ã)·Q(s,ã)+cost-rollout]
                 /(1+n(s,ã))
    n(s) ← n(s) + 1
    n(s,ã) ← n(s,ã) + 1
    return cost-rollout
```

simulate $\tilde{a}$; observe $s'$

# Intermediate Summary

- Passive learning
  - DUE
  - ADP
  - TD
- Active learning
  - Active ADP
  - Q-learning
- Multi-armed bandit problem
  - UCB, UCT

# Outline: Decision Making – Foundations

*Utility Theory*

- Preferences
- Utilities
- Preference structure

*Markov Decision Process / Problem (MDP)*

- Sequence of actions, history, policy
- Value iteration, policy iteration

*Reinforcement Learning (RL)*

- Passive and active, model-free and model-based RL
- Multi-armed bandit

⟹ Next: Decision Making – Extensions