ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

07
Chapiter

Markov Decision Processes

2I1AE1: Artificial Intelligence

Régis Clouard, ENSICAEN - GREYC

“L'homme de science le sait bien, lui,
que seule la science, a pu, au fil des siècles,
lui apporter l'horloge pointeuse et le parcmètre automatique
sans lesquels il n'est pas de bonheur terrestre possible.”
Pierre Desproges

In this chapter

- Markov Decision Processes
  - Mathematical framework for modeling decision making in situations where outcomes are partly random.

1) Problem formulation: Markov Decision Process
2) Example of solution: Value Iteration algorithm

Andreï Markov (1856-1922)
Russian mathematician.
Founder of the theory of stochastic processes.

Planning in Stochastic Domains

- So far, we have studied search in state space.
- We use search to solve simple planning problems (ie, path finding problems)?
  - e.g. robot planning using A*
- But, only in deterministic domains…

Planning in Stochastic Domains

- A* doesn't work so well in stochastic environments where the result of action is not guaranteed…

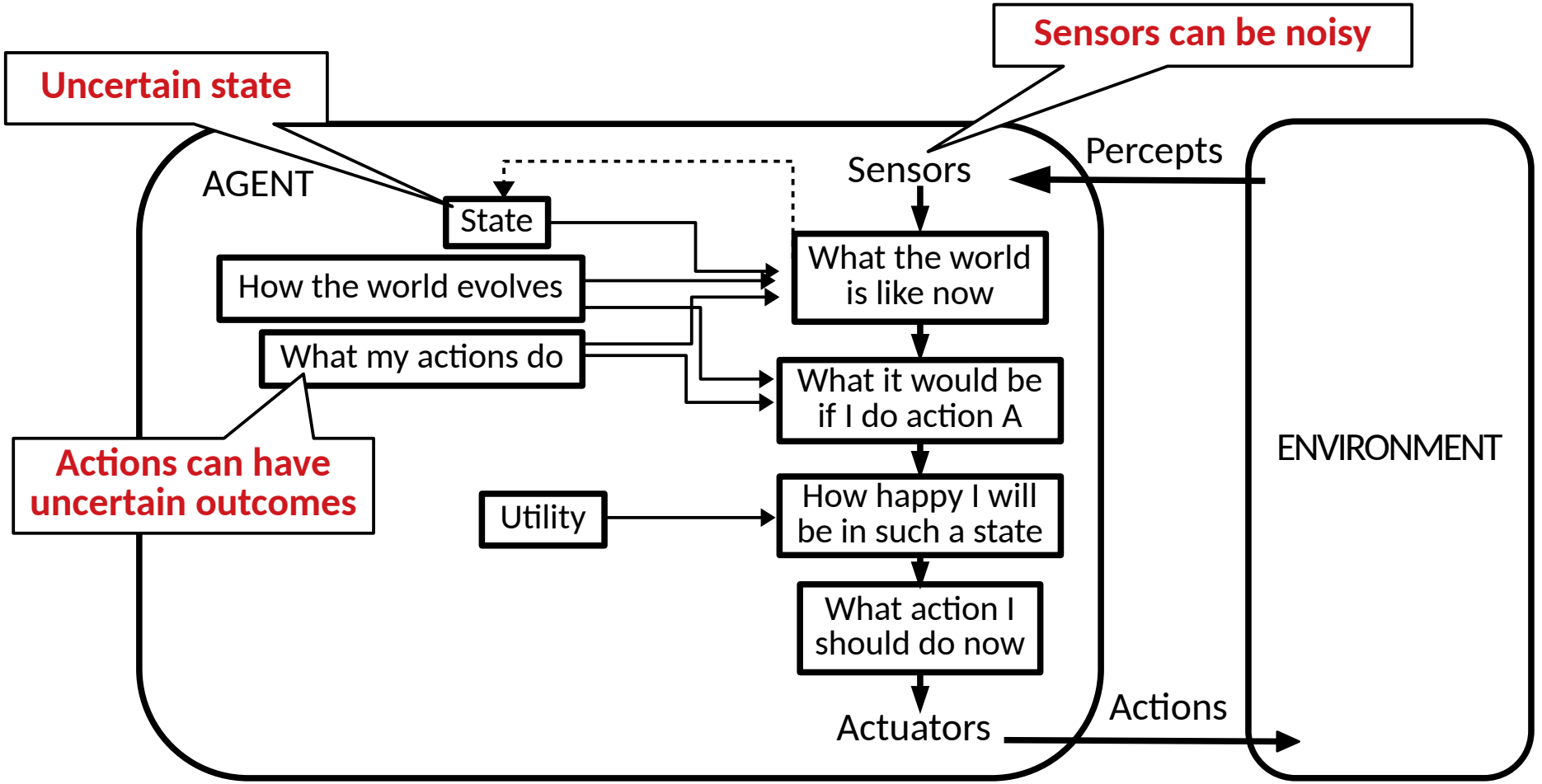
- We are going to introduce a new framework for encoding path finding problems with stochastic dynamics: the Markov Decision Processes (MDP)

Making Complex Sequential Decisions

- Markov Decision Processes (MDPs) provide a framework for finding optimal solution for **sequential decision-making** problems under **uncertainty**.
  - Sequential decisions:
    - List of actions to go from the current state to a final expected state.
  - Stochastic search:
    - How do you plan when your actions might fail?

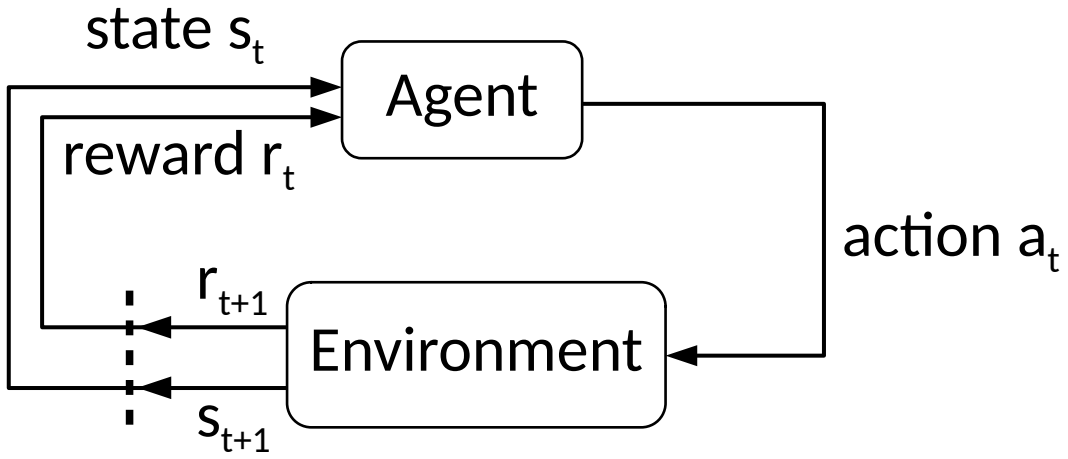
Utility-Based Agents

- Where does uncertainty come from?
  - Rational Agent



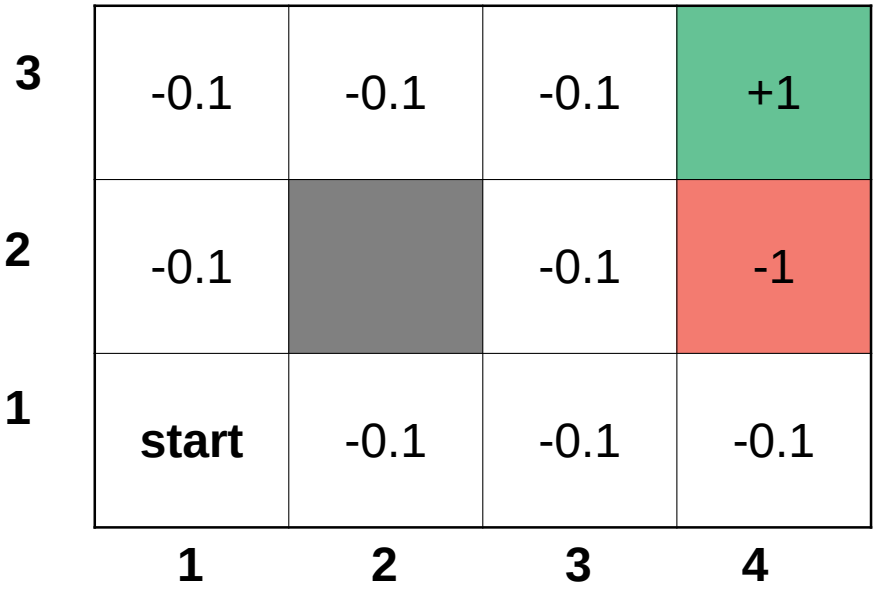
Markov Decision Process

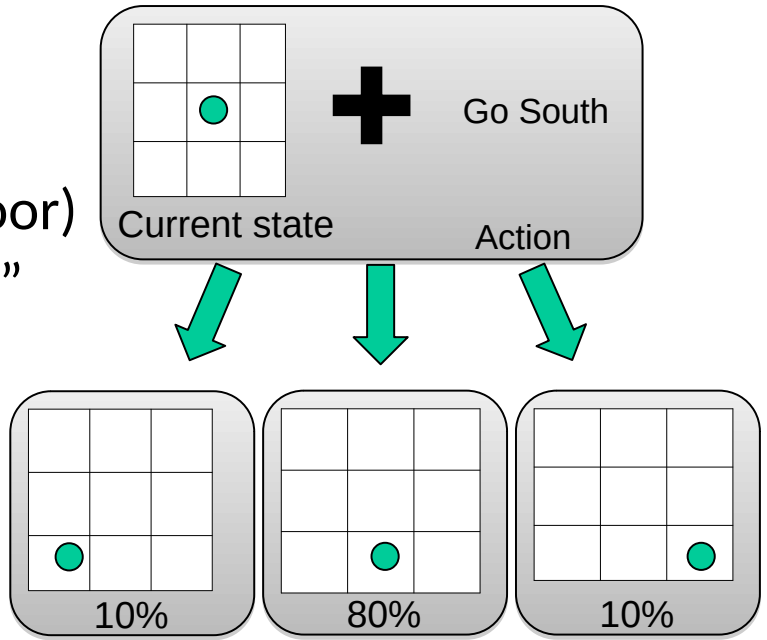
- Planning with Markov Decision Process
  - An agent must decide how to act taking into account a **utility function** expressed as **rewards** (*aka reinforcements*).
  - At each time step *t*, the agent is in **state** s_t
  - For executing the action, it may choose any **action** a_t that is available in state s_t
  - From the sensors, the agent receives an observation of the environment which typically includes the **reward** r_t
- **The agent's goal is to maximize the sum of rewards** ($\Sigma\ r_t$).



Example

- **A maze-like problem**:
  - The agent lives in a grid.
  - It receives rewards each time step:
    - Small living reward each step (can be negative).
    - Big reward come at the end (good or bad).
- **Goal**:
  - At each moment, the agent must select an action to optimize the overall rewards.
- **Problems:**
  - Non deterministic actions (e.g. slippery floor)
    - 80% of the time, the action “Go South” takes the agent south.
    - But, 10% of the time, “Go South” takes the agent East, 10% West.

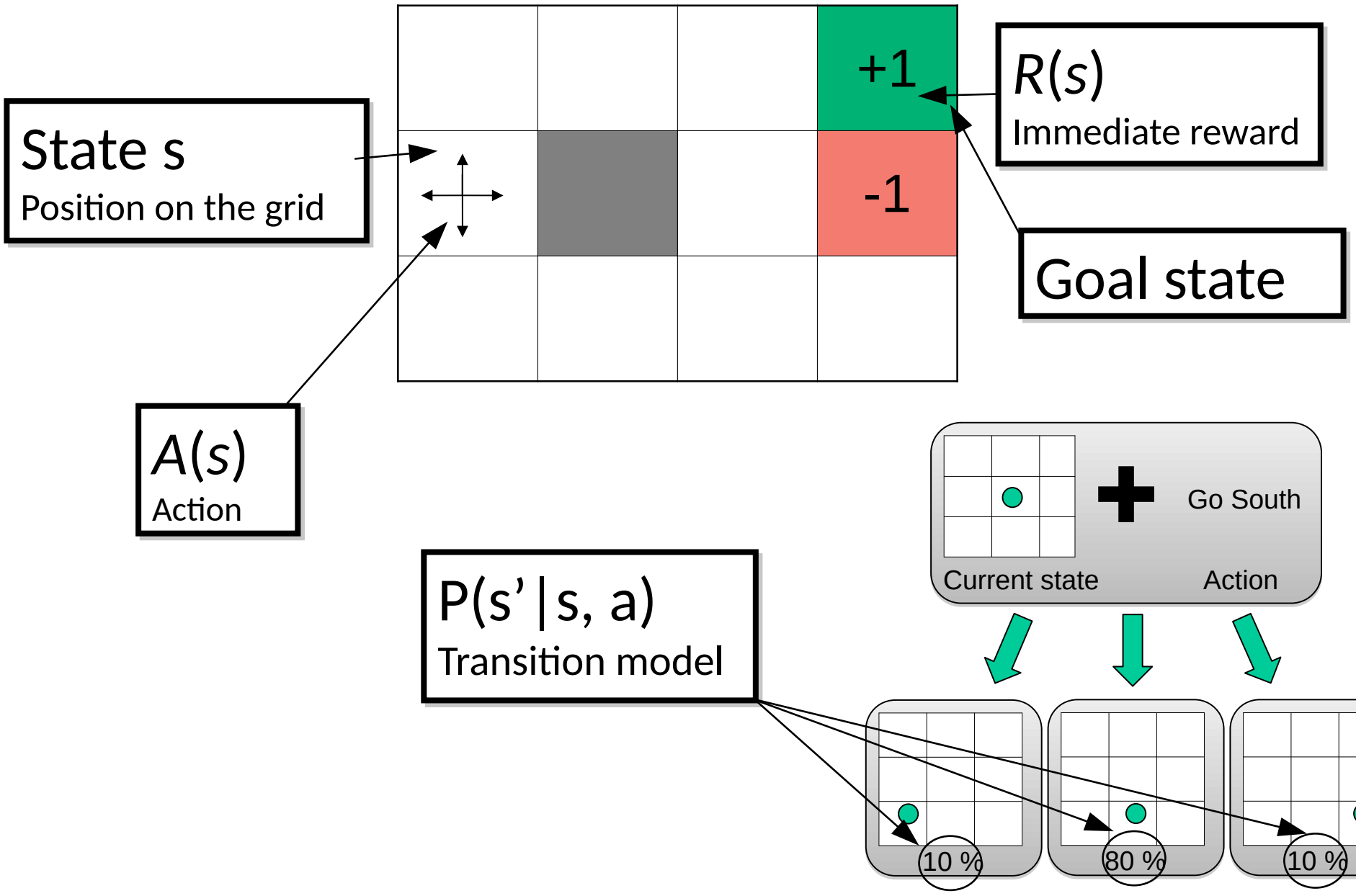




Formal Definition

- A Markov decision process (MDP) is a search problem:
  - **State: S** is a finite set of **states**
  - **Initial state**: s_0
  - **Actions:** A(s) is the finite set of actions available from state *s*
  - **Goal:** A test that decides whether a target state is reached.
  - **Type**: path finding
- What is new:
  - **P(s'|s, a)** is the **transition model** which is the probability that action *a* in state *s* at time *t* will lead to state *s'* at time *t+1*.
  - **R(s)** is the immediate **reward** received after transitioning from state *s* to state *s'*
    - Measures how good is it from the agent perspective to be in state *s*.
- Markov assumption
  - The probability of going to *s'* from *s* with action *a* depends only on *s* and not on any of the previous states.
    - Means that given the present state, the future and the past are independent.

Notation

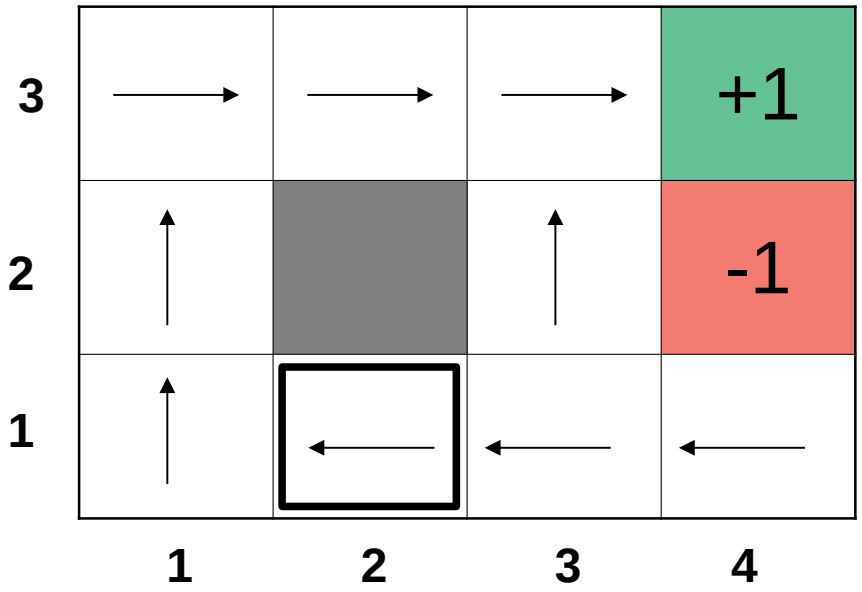

State s
Position on the grid
+1
-1
R(s)
Immediate reward
Goal state
A(s)
Action
P(s'|s, a)
Transition model
Go South
Current state
Action
10 %
80 %
10 %

What is a Solution to a MDP?

- MDP Planning Problem:
  - Input: an MDP (S, A, P, R)
  - Output: a strategy for acting so as to maximize the future rewards.
- Should the solution to a MDP be just a sequence of actions such as (a_1, a_2, a_3, …), → **Plan**?
  - No! In general an action sequence is not sufficient:
    - Actions have stochastic effects, so the state we end up in is uncertain.
- A solution should tell us what the best action is for any possible situation/state that might arise → **Policy**.
  - In deterministic single-agent search problems:
    - We want an **optimal plan** from start to a goal.
  - In stochastic single-agent search problems such as Markov Decision Problem:
    - We want an **optimal policy** $\pi^*: S \rightarrow A$.

What is a Policy?

- A policy is a mapping from perceived states of the environment to actions to be taken when in those states.
  - π(s) tells us what action to take at state *s*.



Example of policy.
At each state (cell), the policy tells us the next action to do.
For example, **π(1,2)** tell us to go East.

Policies versus Plans

- Policies are more general than plans
- Plan:
  - Specifies a sequence of actions to execute from the starting state.
  - Cannot react to unexpected outcome.
- Policy:
  - Tells you what action to take from any state at any moment.
  - Can handle unexpected outcome.

Execution of Policy

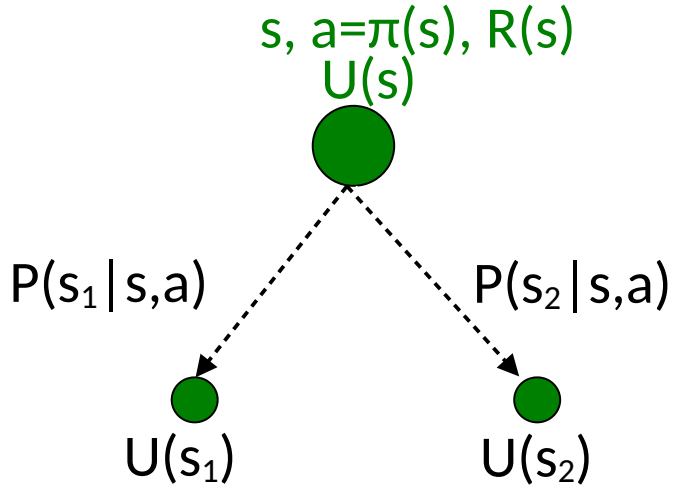
- Given a policy π, and a state *s* of the environment the agent is now simply a **reflex agent** (strategy is in the policy):

DO
1. *s* = current state of the environment perceived by sensors
2. action *a* = *π*(s)
3 execute action *a*

WHILE *s* is not a terminal node

Utility of a Policy

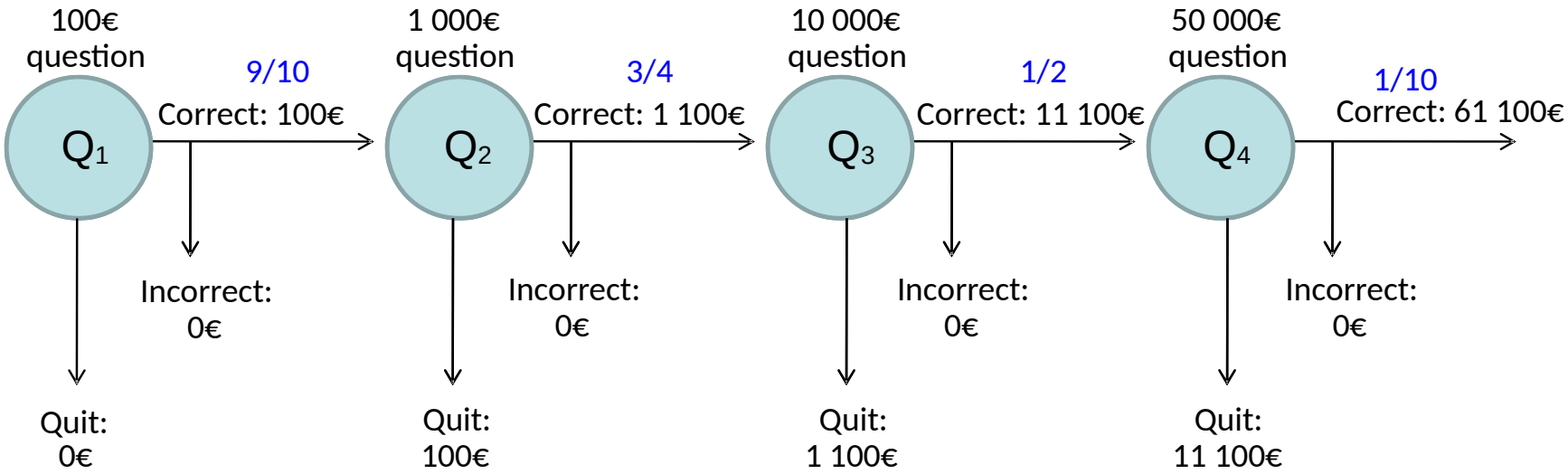
- Utility is used to compute a policy
- The utility of policy π is defined as the **expected** sum of rewards obtained over all possible consequent state sequences.
  - $U^{\pi} = [R_0 + R_1 + R_2 + \ldots + R_t]$
- Utility of policy π at state *s* is:
  - $U^{\pi}(s) = \underbrace{R(s)}_{\text{immediate reward}} + \underbrace{\Sigma_{s' \epsilon S} P(s'|s, \pi(s)) U^{\pi}(s')}_{\text{Sum of the rewards in future}}$



  - where
    - **S:** a finite set of states
    - **R(s)**: reward for state *s*
    - **π(s)**: action to perform at state *s* according to policy π
    - **P(s'|s, π(s))**: probability that action π(s) in state *s* leads to state *s'*

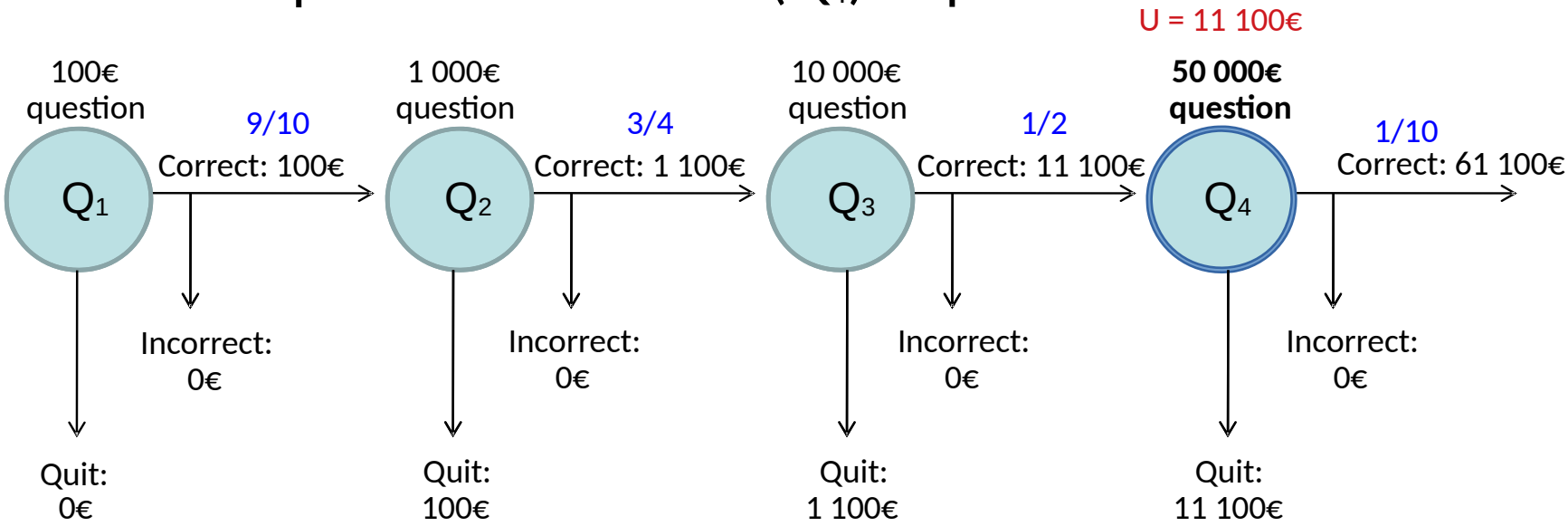
Example: Game Show

- A series of questions with increasing level of difficulty and increasing payoff.
- Decision: at each step
  - Take your earnings and quit, or go for the next question.
  - If you answer wrong, you lose everything.



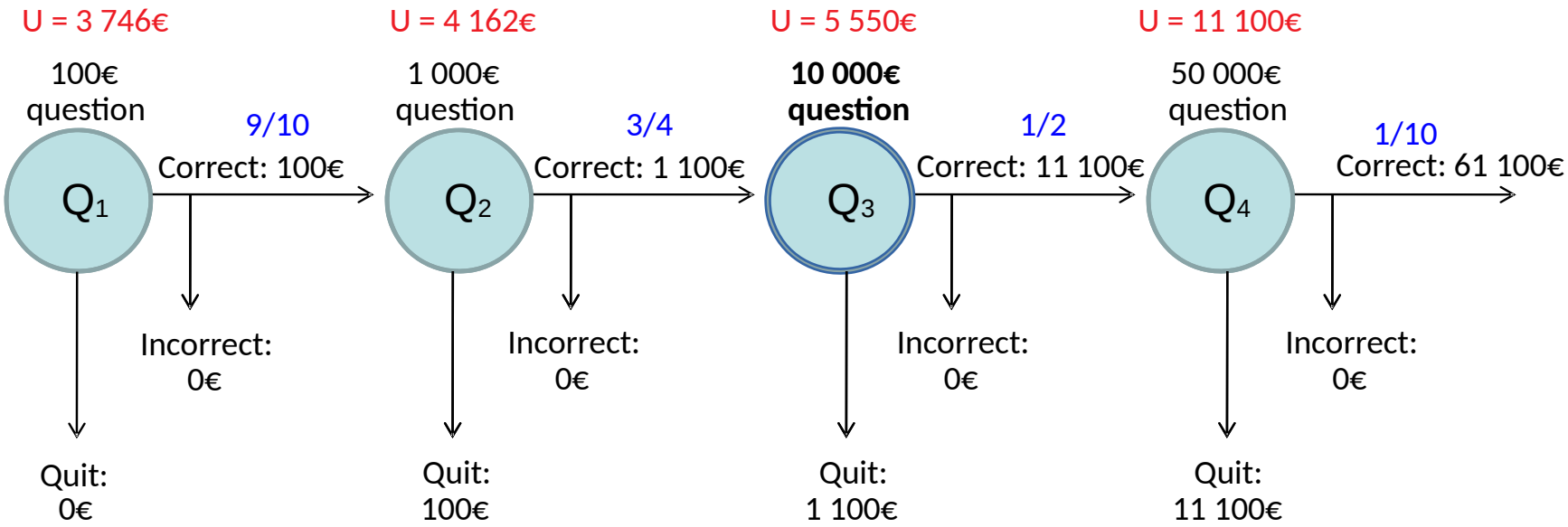
Example: Game Show

- Consider question Q_4
  - Probability of guessing correctly: 1/10
  - Quit or go for the question?
- What is the expected payoff utility for continuing?
  - 0.1 * 61 100€ + 0.9 * 0€ = 6 110€
- What is the expected payoff for quitting?
  - 11 100€
- What is the optimal decision? $\pi(Q_4)$ = quit



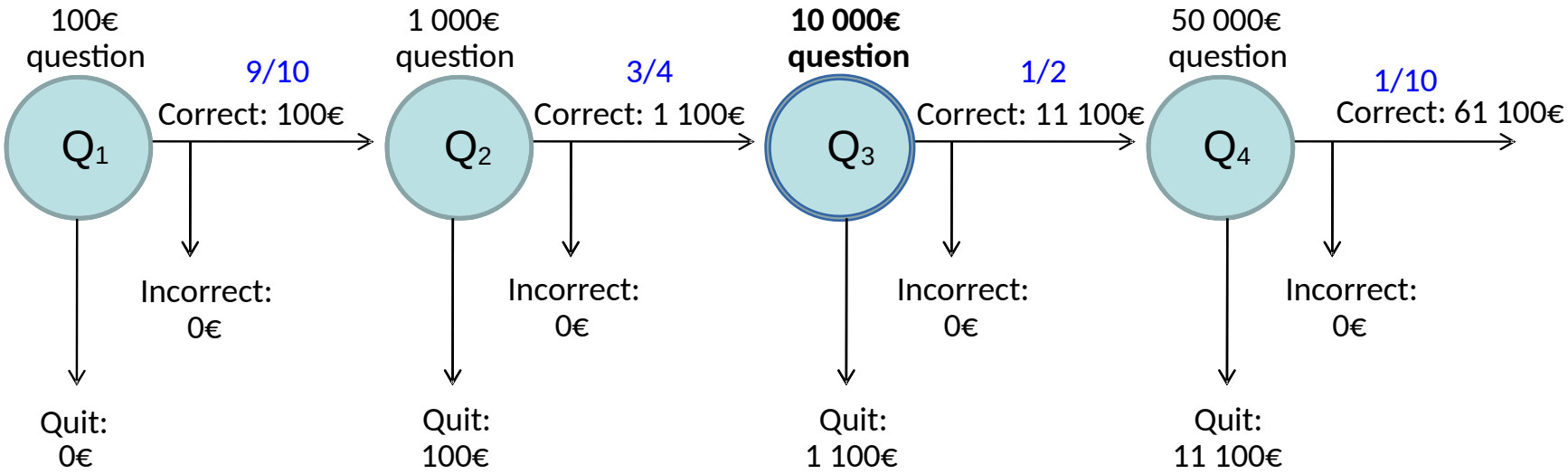
Example: Game Show

- What should we do in Q_3? $\pi(Q_3)$ = continue
  - Payoff for quitting: 1 100€
  - Payoff for continuing: 1/2 * 11 100€ + 1/2 * 0€ = 5 550€
- What about Q_2? $\pi(Q_2)$ = continue
  - 100€ for quitting vs. 5 550€ * 3/4 = 4 162€ for continuing
- What about Q_1? $\pi(Q_1)$ = continue
  - 0€ for quitting vs. 4 162€ * 9/10 = 3 746€ for continuing



Example: Game Show

- So, the policy π is:
  - π(Q1): Continue
  - π(Q2): Continue
  - π(Q3): Continue
  - π(Q4): Quit



Finite / Infinite Horizon

- Value function: $U^{\pi}(s) = \mathbb{E}_{\pi} \Sigma_{i \in \{0, \infty\}} [r_{t+i} | s_t = s]$
  - Intractable for infinite or long state sequences!
- **Solution 1**: Finite horizon *T* (*similar to Depth-Limited Search*)
  - Maximize expected future reward over the next *t* timesteps (nothing is important after *T* steps!)
    - $U^{\pi}(s) = \mathbb{E}_{\pi} \Sigma_{i \in \{0, T\}} [r_{t+i} | s_t = s]$
  - Problem: give non stationary policies (π highly depends on the value of *T*).
- **Solution 2**: Infinite horizon
  - Discount the individual state rewards by a factor γ ∈ [0, 1[.
    - $U^{\pi}(s) = \mathbb{E}_{\pi} \Sigma_{i \in \{0, \infty\}} [\boldsymbol{\gamma}^i r_{t+i} | s_t = s] = R(s_t) + \boldsymbol{\gamma}\, R(s_{t+1}) + \boldsymbol{\gamma}^2 R(s_{t+2}) + \boldsymbol{\gamma}^3 R(s_{t+3}) + \ldots$
    - Discount factor **γ** represents the difference in importance between future rewards and present rewards: sooner rewards count more than later ones.
  - So the utility of policy *π* at state *s* becomes:
    - $U^{\pi}(s) = R(s) + \boldsymbol{\gamma} \Sigma_{s' \in S} P(s' | s, \pi(s))\, U^{\pi}(s')$

Discount factor γ

- The discount factor domain
  - γ ϵ [0,1[to have finite sum (ie, $\boldsymbol{\gamma}^i$ becomes infinitely small as *i* increases)
    - If the discount factor ≥ 1, the action values may diverge.
- The discount factor γ determines the importance of future rewards.
  - A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards.
  - A factor approaching 1 will make it strive for a long-term high reward.
- Generally and empirically, we use γ=0.1, which favors recent rewards.

Optimal Policy

- The optimal policy π^* is the policy that always leads to maximizing the expected sum of rewards collected in future states:
  - $\pi^*(s) = \arg\max_\pi U^\pi(s)$

    $= \arg\max_\pi \Sigma_{s' \epsilon S} P(s' | s, \pi(s)) U^\pi(s')$

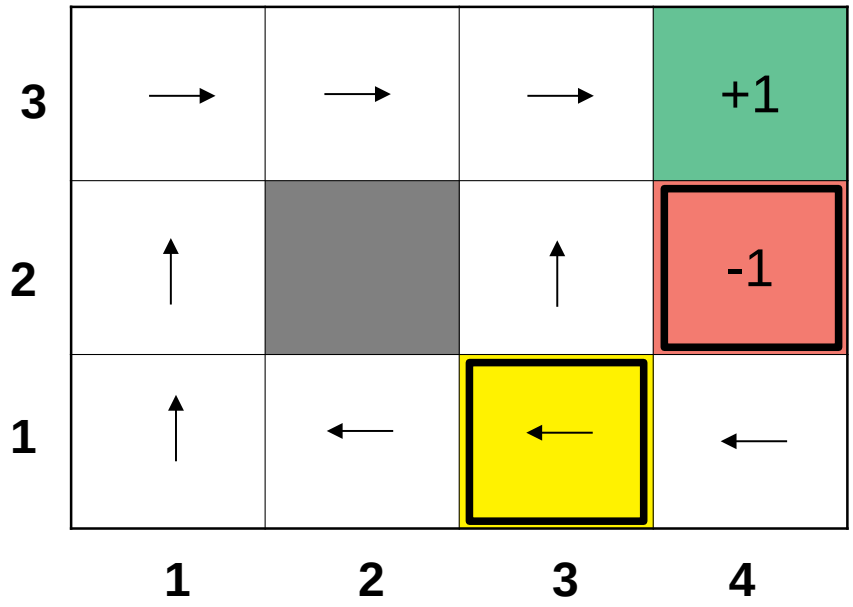
Optimal Policy and State Utilities for MDP

- Example of optimal policy for:
  - **R(s*) = -0.04** for non terminal states
  - R(3,4) = +1
  - R(2,4) = -1

U^*

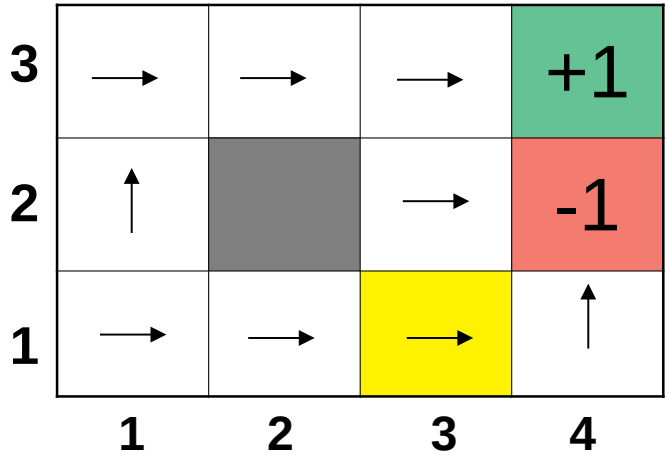
3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

π^*

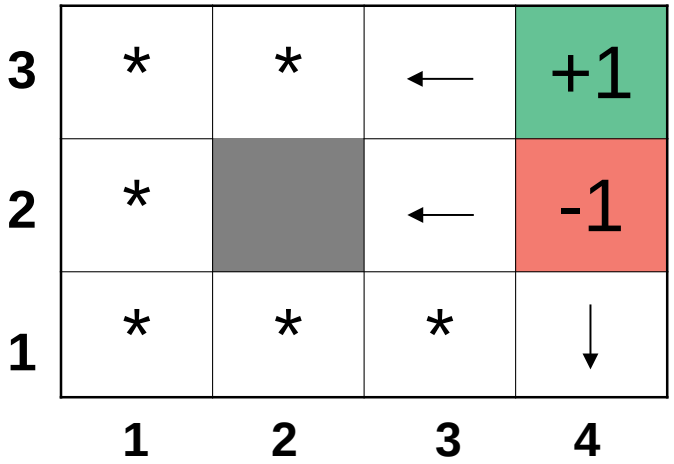


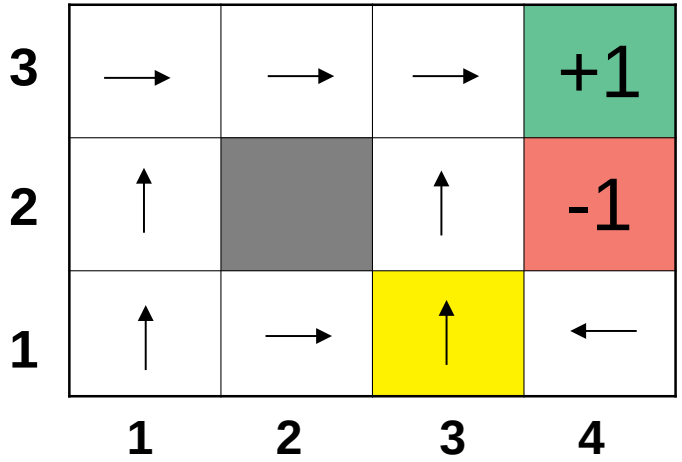
- Notice:
  - Because the cost of taking a step is small compared with the penalty for ending up in (4,2) by accident, the optimal policy recommends taking the long way round (3,1), rather than taking the shortcut and thereby risking entering (4,2).

Balance of Risk and Reward



Example 2: $R(s^*) < -1.6284$
Life is so painful that the agent heads straight for the nearest exit, even if the exit is worth.





Example 3: $-0.4278 \leq R(s^*) \leq -0.0850$
Life is quite unpleasant; the agent takes the shortest route to the +1 state and is willing to risk falling into the −1 state by accident.
In particular, the agent takes the shortcut from (3,1).

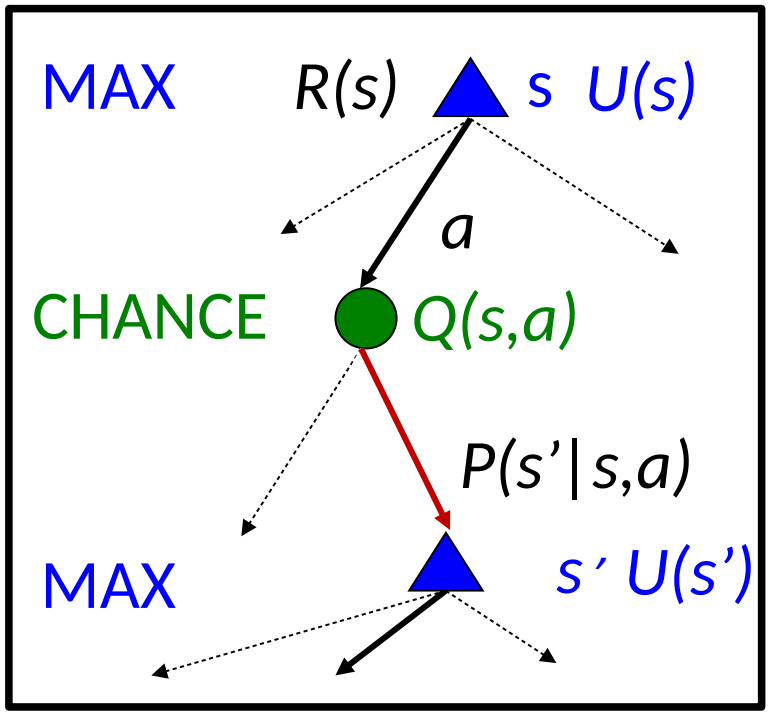
Example 4: $R(s^*) > 0$
Life is positively enjoyable and the agent avoids both exits.

Solving MDPs

- Now, we look at how to solve MDP, ie find the optimal policy, when the transition model and the reward function are known.
  - Given for each state *s*:
    - Transition model: $P(s' | s, a)$
    - Reward function: $R(s)$
  - Goal:
    - Find $U^*(s)$ and then $\pi^*(s)$

Find Optimal Utility

- Definition of optimal utility *U(s)* via ***expectimax*** recurrence gives a simple one-step look-ahead relationship among optimal utility values.
  - The utility of a state *s*:
    - *U(s)* = expected utility of a plan starting in *s*.
  - The quality value of a q-state (*s*, *a*):
    - *Q(s,a)* = expected utility of a plan having taken action *a* from state *s*.
- Recursive definition of utility:
  - What is the expected q-value of taking action *a* in state *s*?
    - $Q(s,a) = R(s) + \gamma \sum_{s' \in S} P(s'|s, a)\, U(s')$
    - $U(s) = \max_a Q(s,a)$
    - $Q(s,a) = R(s) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} Q(s',a')$
  - $U(s) = R(s) + \gamma \max_a \sum_{s' \in S} P(s'|s, a)\, U(s')$
- These are the Bellman equations (recursive definition).



The Bellman Equation (Bellman, 1957)

- The Bellman equation gives the value U^* of the optimal policies:
  - $U^*(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in S} P(s' | s, a)\, U^*(s')$

immediate reward

future reward

- For *n* states, we get *n* equations with *n* unknowns
  - Solving them solves the MDP.
  - Problem: these are **non linear equations** (use of the max operation) more difficult to solve than linear equations.
- Instead, we can use a method based on Dynamic Programming (Bellman).
  - Simplest algorithm: Value iteration

Value Iteration Algorithm

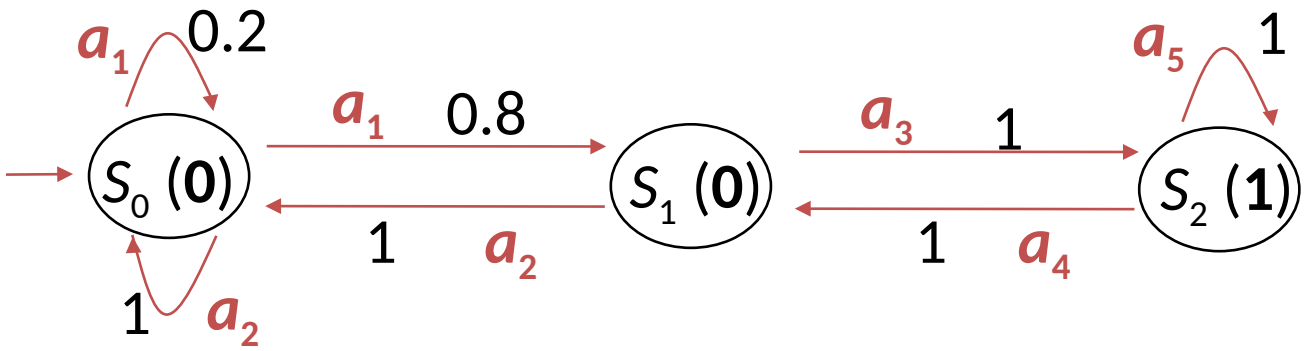

```
function Value-Iteration(MDP) returns a utility function
inputs:    P(S_i|S_j,a), a transition model
           R, a reward function on states
           γ, discount parameter
locals:    U, utility function, initially identical to R (or 0)
           U', utility function, initially identical to R (or 0)
           δ, max difference between U and U'
REPEAT
    U ← U'; δ ← 0
    FOREACH state S_i DO
        U'[S_i] ← R[S_i] + γ max_a Σ_j P(S_j|S_i,a) U[S_j]
        δ ← max(δ, |U'[S_i]–U[S_i]|)
    END
UNTIL δ < ε # until idempotence
return U
```

- Complexity:
  - Each Iteration : $O(|A|\cdot|S|^2)$
  - Number of iterations can be exponential in the discount factor γ.

Value Iteration Algorithm

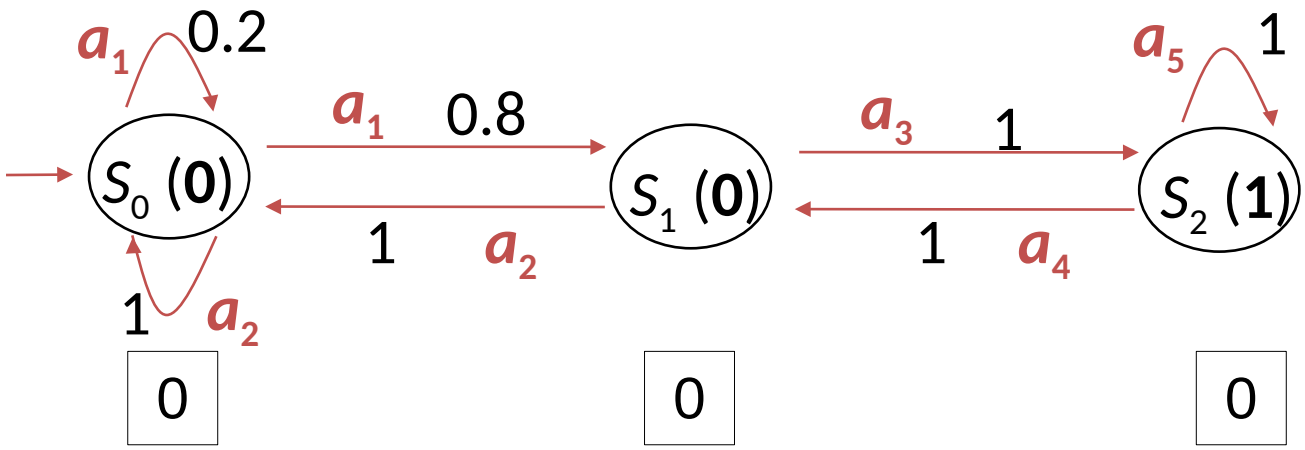
- Convergence
  - Needs $\gamma \in [0,1[$
  - With infinite number of iterations, guaranteed to find the optimal utility values
  - In practice, no need of an infinite number of iterations.

MDP Example



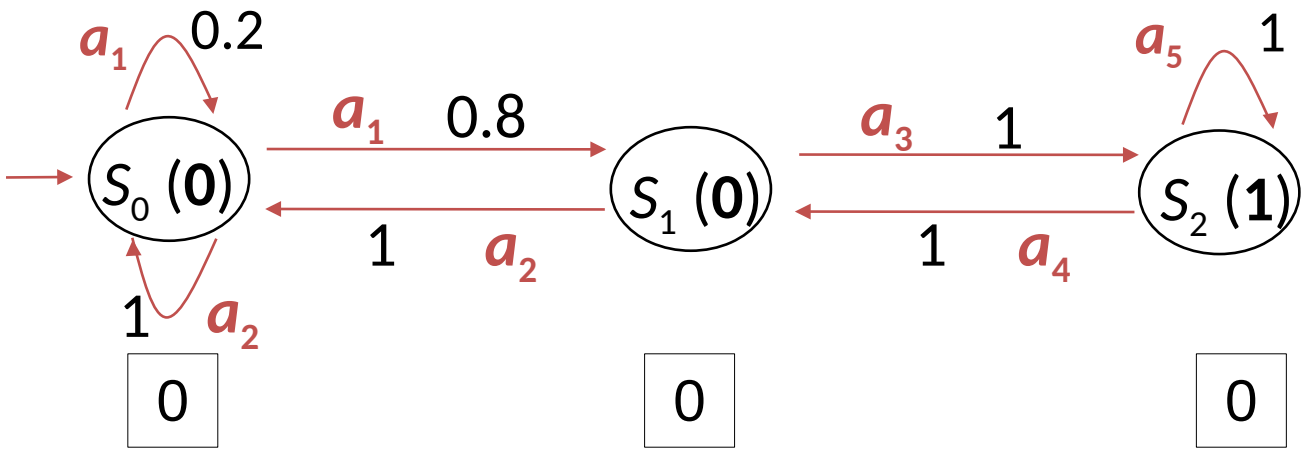
- MDP with 3 states: S = $\{s_0, s_1, s_2\}$
  - $P(s_0 | s_0, a_1) = 0.2$ $P(s_1 | s_0, a_1) = 0.8$ $P(s_0 | s_0, a_2) = 1$
  - $P(s_0 | s_1, a_2) = 1$ $P(s_2 | s_1, a_3) = 1$
  - $P(s_1 | s_2, a_4) = 1$ $P(s_2 | s_2, a_5) = 1$
- Rewards functions
  - $R(s_0) = 0$, $R(s_1) = 0$, $\mathbf{R(s_2) = 1}$ **(terminal node)**
- The discount factor:
  - $\gamma = 0.5$

Value Iteration: Initialization



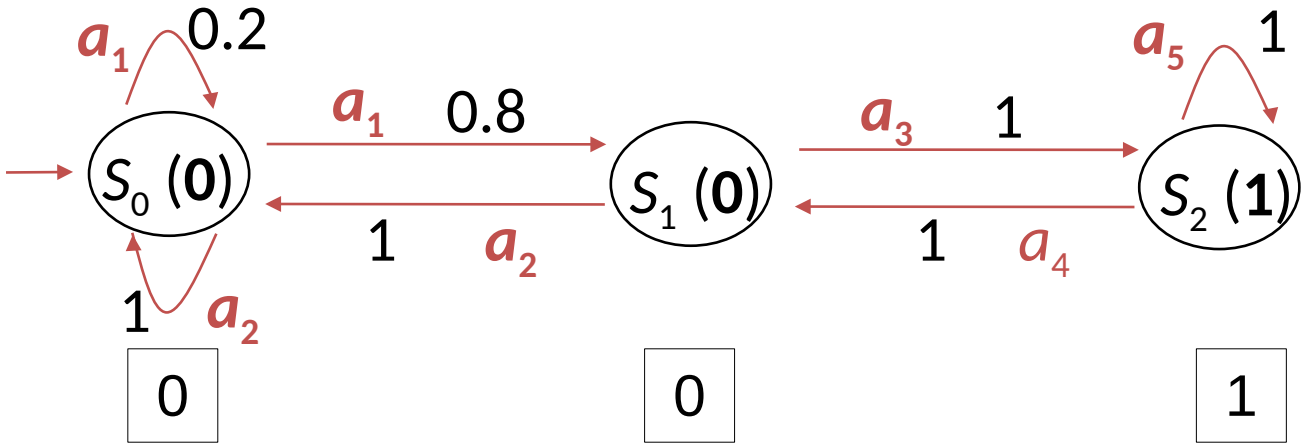
- Initial values set to 0:
  $U(s_0) = 0$
  $U(s_1) = 0$
  $U(s_2) = 0$

Value Iteration: Iteration #1



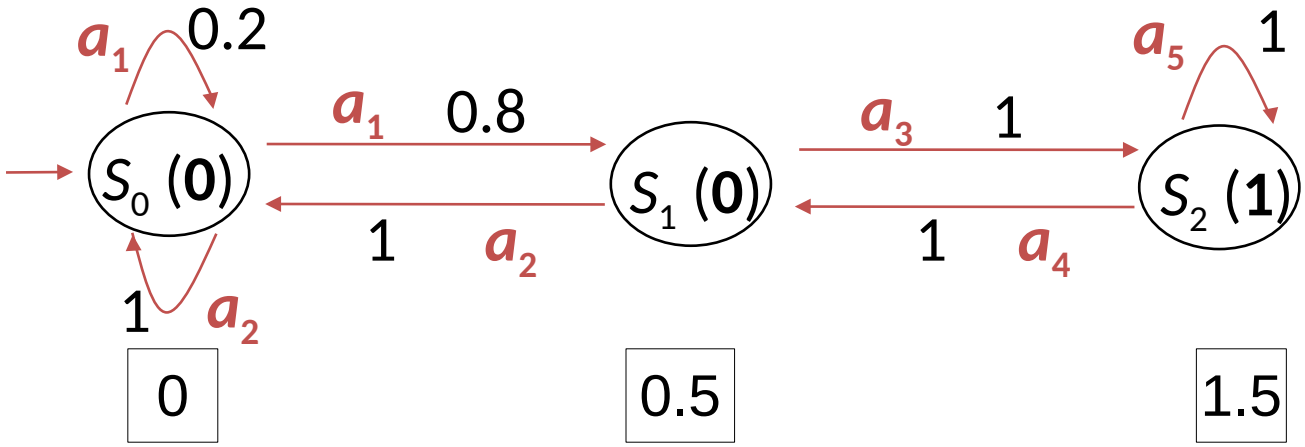
- Update: $U_{i+1}^*(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in S} P(s' | s, a)\, U_i^*(s')$
  - $U(s_0) \leftarrow 0 + 0.5 \max\{ 0.2\, U(s_0) + 0.8\, U(s_1),\ 1\, U(s_0)\} = 0 + 0.5 \max\{ 0, 0 \} = 0$
  - $U(s_1) \leftarrow 0 + 0.5 \max\{ 1\, U(s_0),\ 1\, U(s_2)\} = 0 + 0.5 \max\{ 0, 0 \} = 0$
  - $U(s_2) \leftarrow 1 + 0.5 \max\{ 1\, U(s_1),\ 1\, U(s_2)\} = 1 + 0.5 \max\{ 0, 0 \} = 1$
- The new values: $U(s_0) = 0$, $U(s_1) = 0$, $U(s_2) = 1$

Value Iteration: Iteration #2



- Update: $U_{i+1}^*(s) = R(s) + \gamma \max_{a \in A(s)} \Sigma_{s' \in S} P(s' | s, a)\, U_i^*(s')$
  - $U(0) \leftarrow 0 + 0.5 \max\{ 0.2\, U(s_0) + 0.8\, U(s_1),\ 1\, U(s_0) \} = 0 + 0.5 \max\{ 0, 0 \} = 0$
  - $U(1) \leftarrow 0 + 0.5 \max\{ 1\, U(s_0),\ 1\, U(s_2) \} = 0 + 0.5 \max\{ 0, 1 \} = 0.5$
  - $U(2) \leftarrow 1 + 0.5 \max\{ 1\, U(s_1),\ 1\, U(s_2) \} = 1 + 0.5 \max\{ 0, 1 \} = 1.5$
- The new values: $U(s_0) = 0$, $U(s_1) = 0.5$, $U(s_2) = 1.5$

Value Iteration: Iteration #3



- Update: $U_{i+1}^*(s) = R(s) + \gamma \max_{a \in A(s)} \Sigma_{s' \in S} P(s'|s, a)\, U_i^*(s')$
  - $U(s_0) \leftarrow 0 + 0.5 \max\{ 0.2\, U(s_0) + 0.8\, U(s_1), 1\, U(s_0) \} = 0 + 0.5 \max\{ 0.8 * 0.5, 0 \} = 0.2$
  - $U(s_1) \leftarrow 0 + 0.5 \max\{ 1\, U(s_0)\ 1\, U(s_2)\} = 0 + 0.5 \max\{ 0, 1.5 \} = 0.75$
  - $U(s_2) \leftarrow 1 + 0.5 \max\{ 1\, U(s_1)\ 1\, U(s_2)\} = 1 + 0.5 \max\{ 0.5, 1.5 \} = 1.75$
- The new values: $U(s_0) = 0.2$, $U(s_1) = 0.75$, $U(s_2) = 1.75$

- We stop when the change of $\forall_i U_i$ is small.

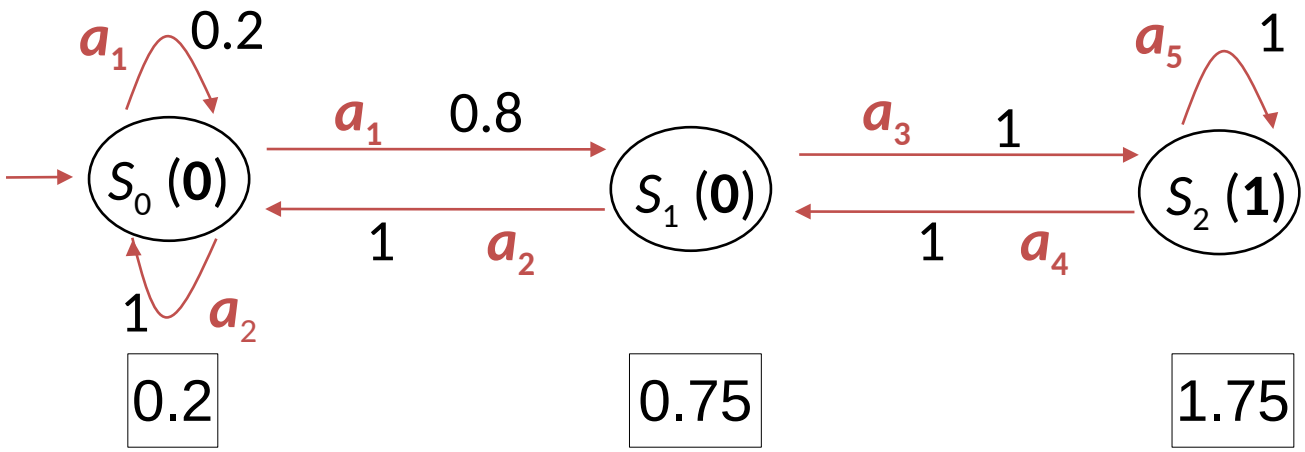
[mdp.sh]

Policy Extraction: Computing Actions from Values

- We know how to compute the optimal utility values $U^*(s)$
- Now, how to compute the optimal policy $\pi^*(s)$?
  - We need to do a mini-expectimax (one step):
    - $\pi^*(s) = \arg\max_a [\ \gamma \Sigma_{s' \epsilon S} P(s'|s, a)\ U^*(s')\ ]$

3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	**2**	**3**	**4**

Value Iteration: Optimal policy



- If we stop at iteration #3, the policy is:
  - $\pi(s_0) = \arg\max_a\{ 0.2\ U(s_0) + 0.8\ U(s_1),\ U(s_0)\} = \arg\max_a\{ 0.2 * 0.2 + 0.8 * 0.75,\ 0.2\}$
    $= \arg\max_a\{ \mathbf{0.64},\ 0.2\} = a_1$
  - $\pi(s_1) = \arg\max_a\{ 1\ U(s_0),\ 1\ U(s_2)\} = \arg\max_a\{ 0.2,\ \mathbf{1.75}\ \} = a_3$
  - $\pi(s_2) = \arg\max_a\{ 1\ U(s_1),\ 1\ U(s_2)\} = \arg\max_a\{ 0.75,\ \mathbf{1.75}\ \} = a_5$

Conclusion

- MDP is very attractive because it combines probabilistic reasoning and optimization seamlessly.
  - Very popular formal framework for mobile robots.
- Note Can be used even when the environment is deterministic (probabilities are all of 100%, no unexpected behavior)
- Limits
  - The algorithm Value-Iteration is slow with large state space.
    - Policy Iteration (better complexity)
    - Real-Time Dynamic Programming (RTPD).
    - Labeled RTDP.
  - MDP assume complete observability.
    - Partially Observable MDP (PoMDP).
  - MDP are limited to sequential decisions. For concurrent decisions:
    - Concurrent MDP (CoMPD)
    - Concurrent Probabilistic Temporal Planning (CPTP)