Machine Intelligence

13. Planning under Uncertainty: Markov Decision Processes

Álvaro Torralba

AALBORG UNIVERSITET

Fall 2022

Thanks to Thomas D. Nielsen and Jörg Hoffmann for slide sources

In previous chapters of the MI Lecture...

- State-space search **Chapter 2**:
  →The basic algorithms under the hood
- Classical Planning **Chapter 11**:
  →Automated Decision Making with domain-independent techniques. The user only specifies how the environment works and by analyzing the environment (delete-relaxation in our case), we can solve it efficiently!

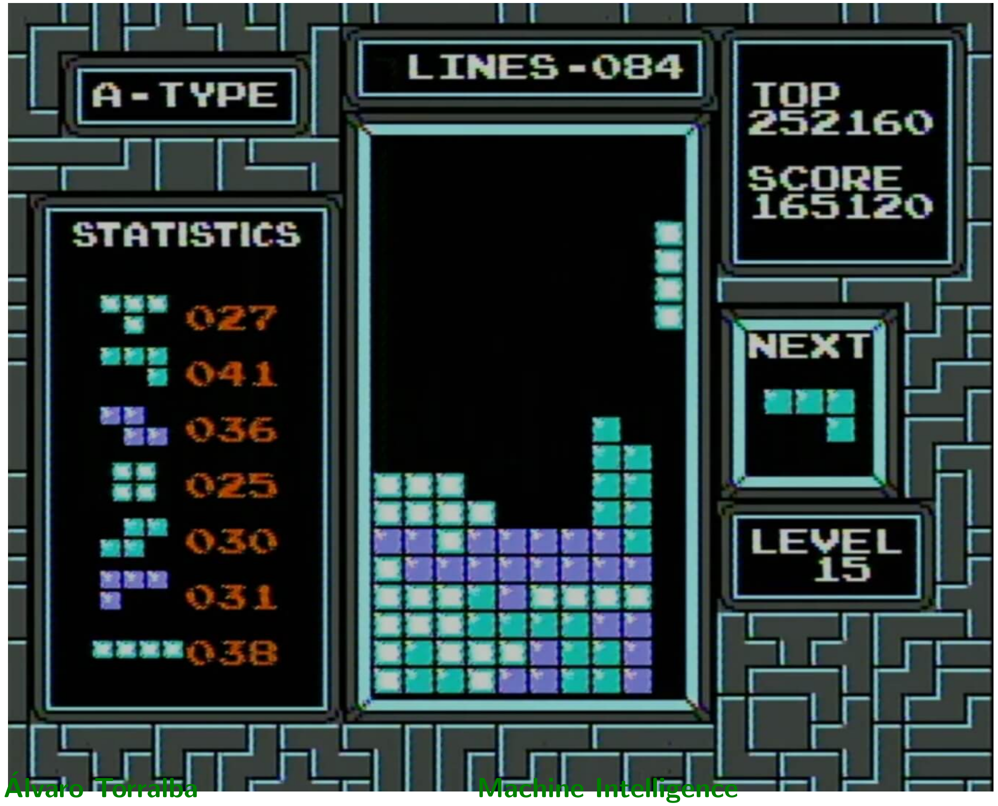
Those techniques give us a **plan to achieve a goal** on environments that are:

- single-agent → we studied multi-agent environments on **Chapter 12**
- deterministic → **(This Chapter)**: non-determinism (MDPs)
- goal-oriented → **(This Chapter)**: reward-oriented (MDPs)
- fully-observable → partially-observable (POMDPs) (not in this lecture)
- discrete
- ...

Decision problems with an unbounded time horizon

Characteristics.

- at each step we are faced with a decision,
- at each step we are given a certain reward (possibly negative) determined by the chosen decision and the state of the world,
- the outcome of a decision may be uncertain (we will assume we know the probabilities associated to each outcome),
- the time horizon of the decision problem is unbounded.



Markov Decision Processes

Definition (Markov Decision Process). *An* **MDP** *is a 6-tuple* $\Theta = (S, A, r, P, I, S^T)$ *where:*

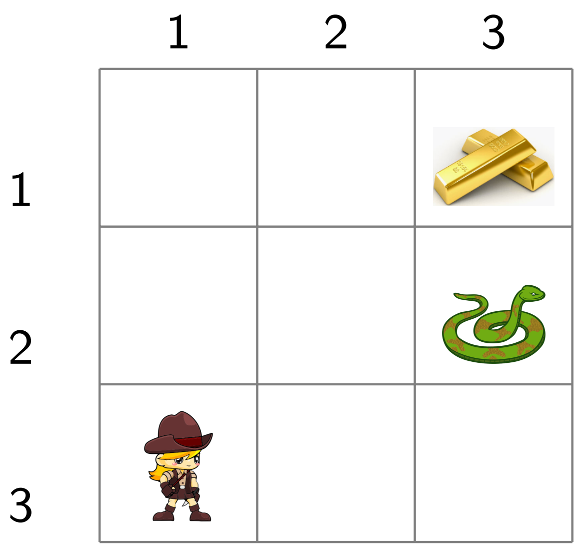
- S *is a finite set of* **states***.*
- A *is a finite set of* **actions***.*
- $r : S \times A \times S \mapsto \mathbb{R}$ *is the* **reward function***.*
- $P : S \times A \times S \mapsto [0, 1]$ *is the* **transition probability function***, representing* $P(s' \mid s, a)$*, the probability that by applying action* a *in state* s*, we end in state* s'*.*
- $I \in S$ *is the* **initial state***.*
- $S^T \subseteq S$ *is the set of* **terminal states***. The set of terminal states could be empty.*

$\rightarrow$ We select actions in order to maximize the reward obtained

Similar to the definition of state-space **(c.f. Chapters 2/11)**, but:

- Instead of a cost function, we have a reward function.
- Instead of a transition relation, we have a probability distribution over possible outgoing states

Markov Decision Process Example



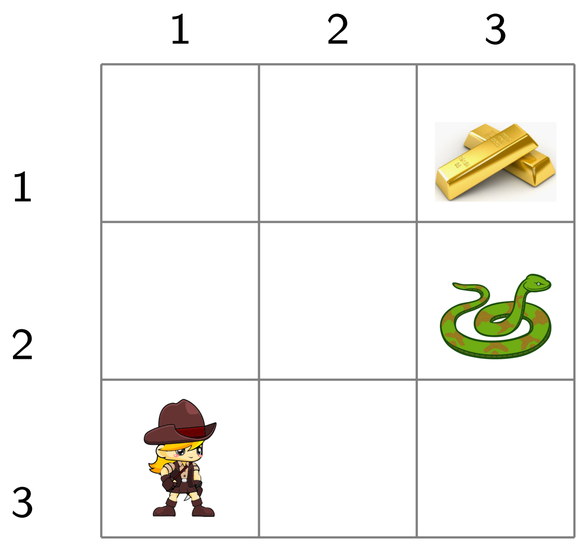
- The adventurer wants to take the gold located in (3,1).
- But she should avoid the snake.
- She has a complete knowledge of the environment.
- But paths are slipery and there is a small probability of failing one step and moving to the cell to the right.

To determine what's the best course of action, let's encode the problem as an MDP!

Transition Probabilities in an MDP

The **transition probabilities** $P(S_i \mid A_{i-1}, S_{i-1})$ tell us what are the possible outcomes of executing each action on each state

- The adventurer can move north, east, south, and west.
- A move succeeds with probability 0.8; otherwise it moves ot the location immediately to the right.
- An attempt to move outside the boundaries of the grid results in staying in the current square.

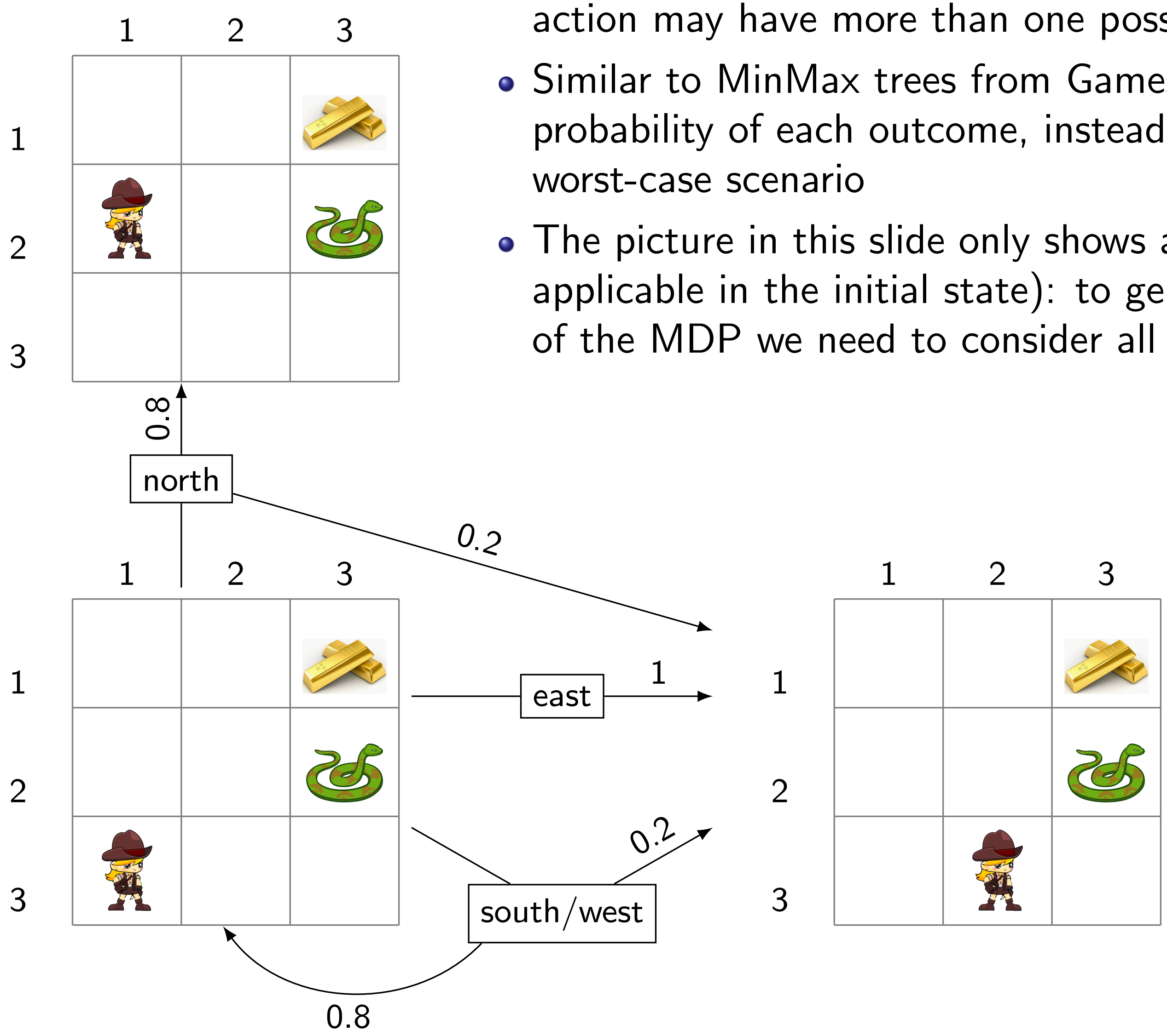


What are the outcomes of moving north in the initial state?

For example, for the north action we have that $P(S_{i+1} \mid \textit{north}, S_i)$:

	(column,row)	S_i: $(1,1)$	$(1,2)$	$(1,3)$	$(2,1)$	$(2,2)$	$(2,3)$	$(3,1)$	$(3,2)$	$(3,3)$
S_{i+1}	$(1,1)$	0.8	0.8	0	0	0	0	0	0	0
	$(1,2)$	0	0	0.8	0	0	0	0	0	0
	$(1,3)$	0	0	0	0	0	0	0	0	0
	$(2,1)$	0.2	0	0	0.8	0.8	0	0	0	0
	$(2,2)$	0	0.2	0	0	0	0.8	0	0	0
	$(2,3)$	0	0	0.2	0	0	0	0	0	0
	$(3,1)$	0	0	0	0.2	0	0	1	0.8	0
	$(3,2)$	0	0	0	0	0.2	0	0	0	0.8
	$(3,3)$	0	0	0	0	0	0.2	0	0	0

Markov Decision Process as a Graph

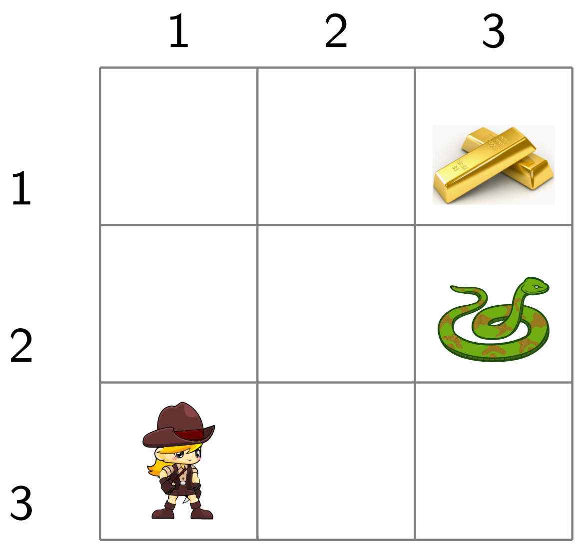


- The state space of an MDP is an hypergraph where each action may have more than one possible outcome
- Similar to MinMax trees from Games, but here we have the probability of each outcome, instead of always picking the worst-case scenario
- The picture in this slide only shows a small portion (the actions applicable in the initial state): to generate the full state-space of the MDP we need to consider all actions in all states.

Rewards

The **reward function** $r(s, a, s')$ specifies “how many points we get” every time we apply an action.
In our example:

- Getting gold: $+10$
- Meeting snake: -5
- Moving elsewhere: -0.1 (so that shorter paths are preferred)

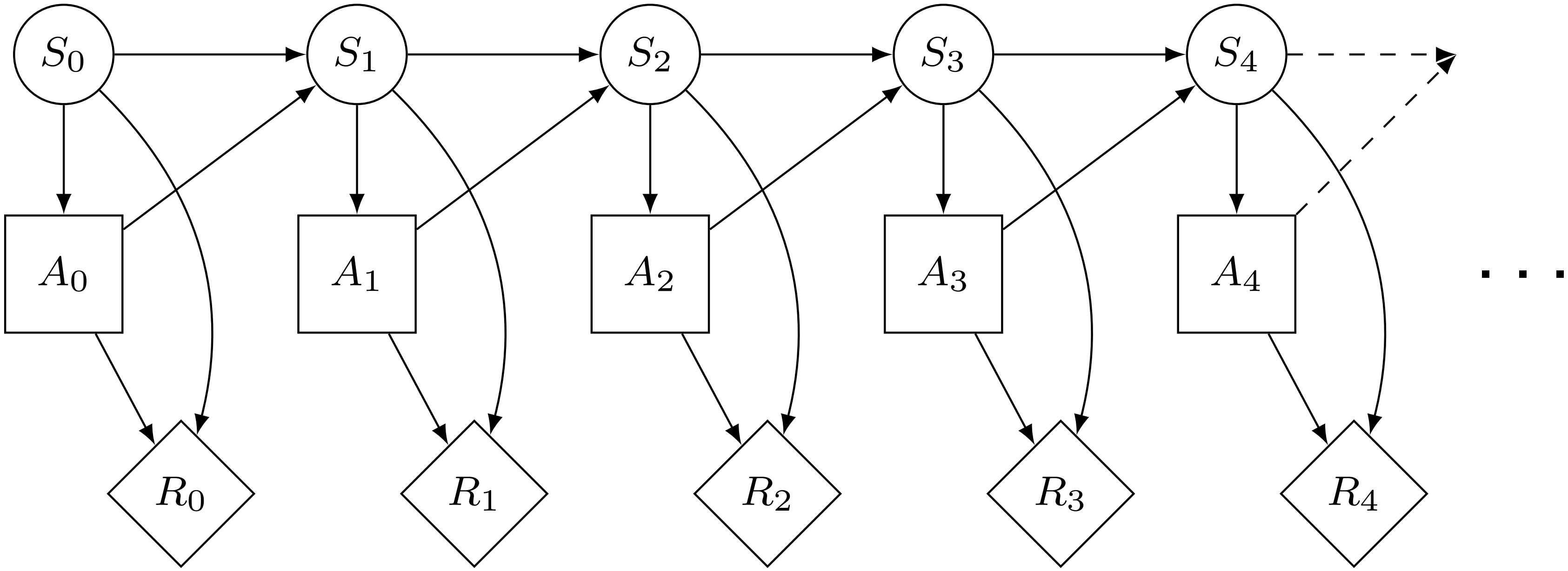


Markov Assumption

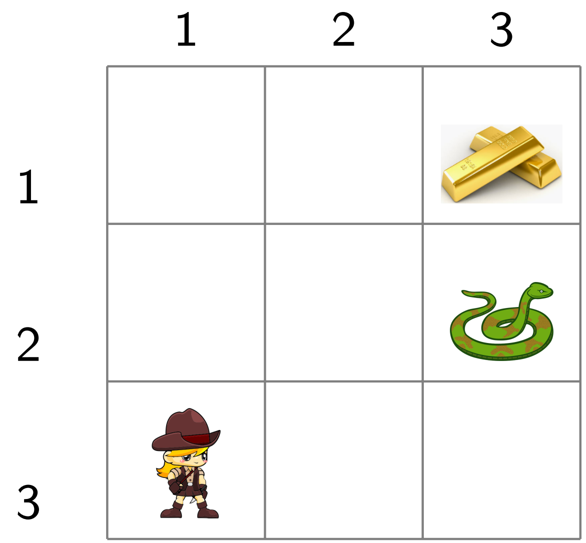
Markov Assumption: given the present, the future does not depend on the past
Where do we see the Markov assumption in Markov Decision Processes?

- Transition probabilities $P(S_i \mid A_{i-1}, S_{i-1})$
  $\rightarrow$ The probability that we reach certain state only depends on the present (what state we were and what action we applied)
- Reward function $r(s, a, s')$
  $\rightarrow$ Again, it does only depend on the present

Example of problem that does not satisfy the Markov assumption?



Solving MDPs



Question

What's the plan?

Decision policies

A decision policy for MDPs is a function mapping non-terminal states to actions:

$$\pi : S \mapsto A$$

	1	2	3
1	$\rightarrow$	$\rightarrow$	
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\rightarrow$	$\uparrow$

In MDPs it suffices to consider this "simple" pure policies. In other settings policies could be more complex:

- A policy could depend on the entire history from the initial state to s $(\pi(s_0, a_0, s_1, a_1, \dots, s_i) \mapsto A)$

  $\rightarrow$ We do not need to take the past into account due to the Markov property

  In other words, if we visit the same state twice, we'd like to do the same action
- A policy could be mixed (assigning probabilities to actions)

  $\rightarrow$ We do not need that here (in MDPs there is always some optimal policy that is pure).

Executing a Policy

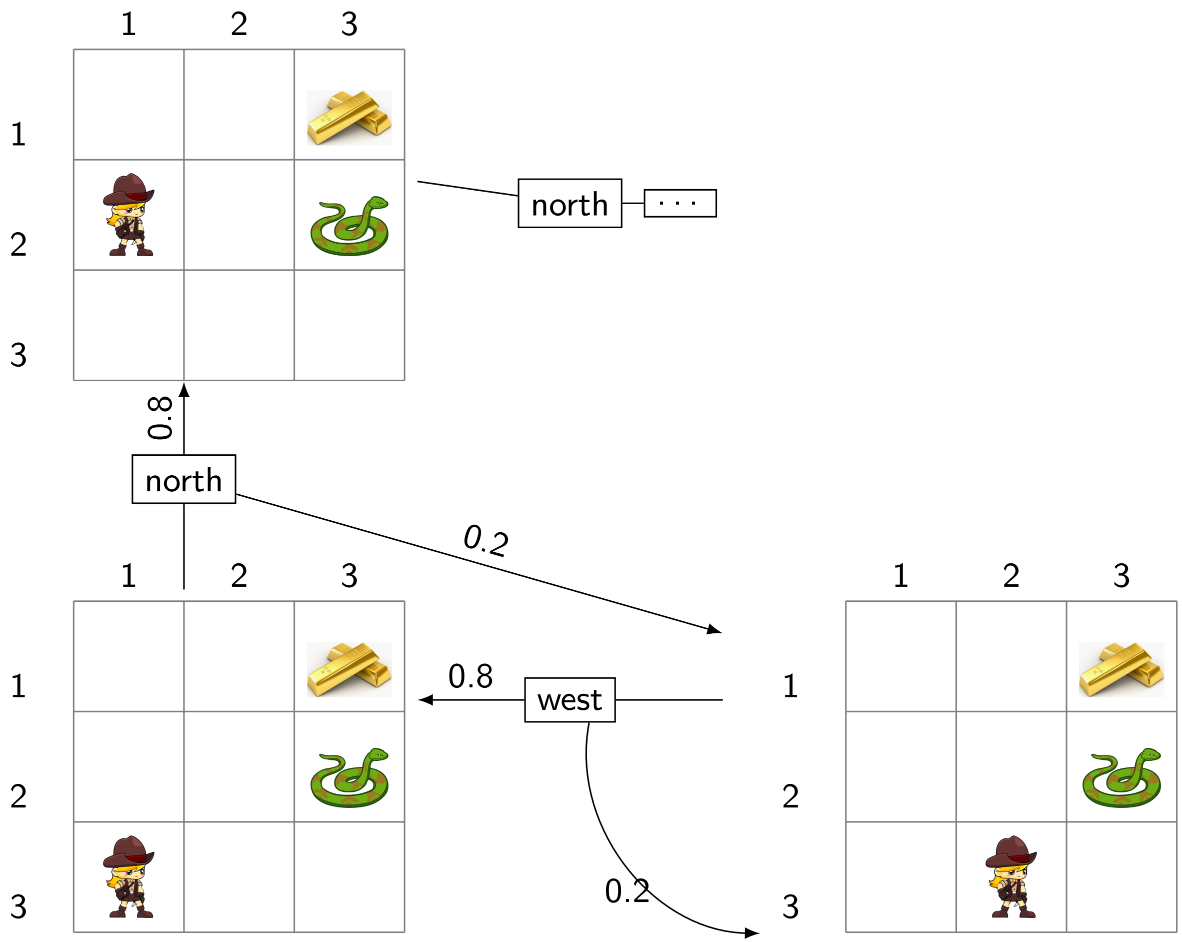
Given a policy, we can simulate its execution to obtain all possible traces it could have and their corresponding probability and utility.

	1	2	3
1	$\rightarrow$	$\rightarrow$	
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\rightarrow$	$\uparrow$

Two example traces:

Markov Chains

The **Markov chain** associated to a policy π of an MDP is the (possibly infinite) set of traces with their corresponding probabilities.



The **expected utility** is the average reward obtained from each trace (weighted by its probability)

Evaluating strategies

Imagine that we have the following reward function and **there is no terminal state** and no uncertainty on the result of an action:

	1	2	3
1	-0.1	-0.1	10
2	-0.1	-0.1	-5
3	-0.1	-0.1	-0.1

Question!

Which strategy is best according to expected utility with unbounded horizon?

(A): $\pi_A =$

	1	2	3
1	$\rightarrow$	$\rightarrow$	$\leftarrow$
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\rightarrow$	$\uparrow$

(B): $\pi_B =$

	1	2	3
1	$\rightarrow$	$\rightarrow$	$\downarrow$
2	$\uparrow$	$\uparrow$	$\downarrow$
3	$\uparrow$	$\leftarrow$	$\leftarrow$

Beyond Expected Sum of Rewards

Problem: Sum of rewards over an infinite horizon is often infinite, not matter how much you wait to start gaining reward

Two solutions:

- **Finite Horizon**
- **Discounted Rewards**

The utility of an unbounded sequence: fixed horizon

Finite-Horizon MDP

Consider only the rewards obtained in the first k states:

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + R(s_1) + \ldots + R(s_k) < \infty.$$

But how do we choose k?

- For $k = 0$ we only care about the immediate reward, hence we pursue a very greedy strategy.
- For, say, $k = 5$, we only care about maximizing the reward the next 5 time steps (but we may end in a really bad state for the future)

Finite horizon MDPs = Live your life as if there were not tomorrow

Question!

Do finite-horizon MDPs satisfy the Markov property?

(A): Yes **(B):** No

Discounted rewards

Compare reward sequences

$$
\begin{array}{cccccccccccc}
-0.1 & 10.0 & -0.1 & 10.0 & -0.1 & 10.0 & -0.1 & 10.0 & -0.1 & 10.0 & -0.1 & \dots \\
-0.1 & -0.1 & -0.1 & -0.1 & 10.0 & -0.1 & -0.1 & -0.1 & -0.1 & 10.0 & -0.1 & \dots
\end{array}
$$

The utility of an unbounded sequence: discounted rewards

Weigh rewards in the immediate future higher than rewards in the distant future:

$$U(s_0, s_1, s_2, \ldots) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots ,$$

where $0 \leq \gamma \leq 1$. Possible interpretations of the discounting factor γ:

- In economics, γ may be thought of as inflation or an interest rate.
- The decision process may terminate with probability $(1 - \gamma)$ at any point in time, e.g. the robot breaking down.

Thanks to increasing the exponent in gamma after every step, the sum of rewards is always finite!
With $\gamma < 1$ we have:

$$U(s_0, s_1, s_2, \ldots) = \sum_{i=0}^{\infty} \gamma^i R(s_i) \leq \sum_{i=0}^{\infty} \gamma^i \max R = \frac{\max R}{1 - \gamma} < \infty.$$

- For $\gamma = 0$ we have a greedy strategy.
- For $\gamma = 1$ we have normal additive rewards.

→ So, **expected discounted reward** is a good metric to compare policies → Typically, we want values $\gamma < 1$ but $\gamma \approx 1$, such as 0.99

Expected utilities

The actions may be non-deterministic so a strategy may only take you to a state with a certain probability.

$$\Downarrow$$

Strategies should be compared based on the expected rewards they can produce.

Starting in state s_0 and following strategy π, the expected (discounted) reward in step i is:

$$\underbrace{\gamma^i}_{\text{discount}} \sum_{s_i} R(s_i) \underbrace{P(S_i = s_i \mid \pi, S_0 = s_0)}_{prob(*)}$$

prob (*) is the probability of reaching s_i after i steps when starting at the initial state s_0 and following policy π

The expected discounted reward of π is defined as:

$$Q(s, \pi) = \sum_{i=0}^{\infty} \gamma^i \left(\sum_{s_i} R(s_i) P(S_i = s_i \mid \pi, S_0 = s_0) \right).$$

and

$$U^\pi(s) = Q(s, \pi(s))$$

Finding optimal strategies

The maximum expected utility of starting in state s is:

$$U^*(s) = \max_{\pi} Q(s, \pi) = \max_{\pi} \sum_{i=0}^{\infty} \gamma^i \left(\sum_{S_i} R(S_i) P(S_i \mid \pi, S_0 = s_0) \right).$$

In any state we choose the action maximizing the expected utility:

$$\delta(s) = \arg\max_{a} \sum_{s' \in \mathrm{sp}(S)} P(s' \mid s, a) U^*(s')$$

with

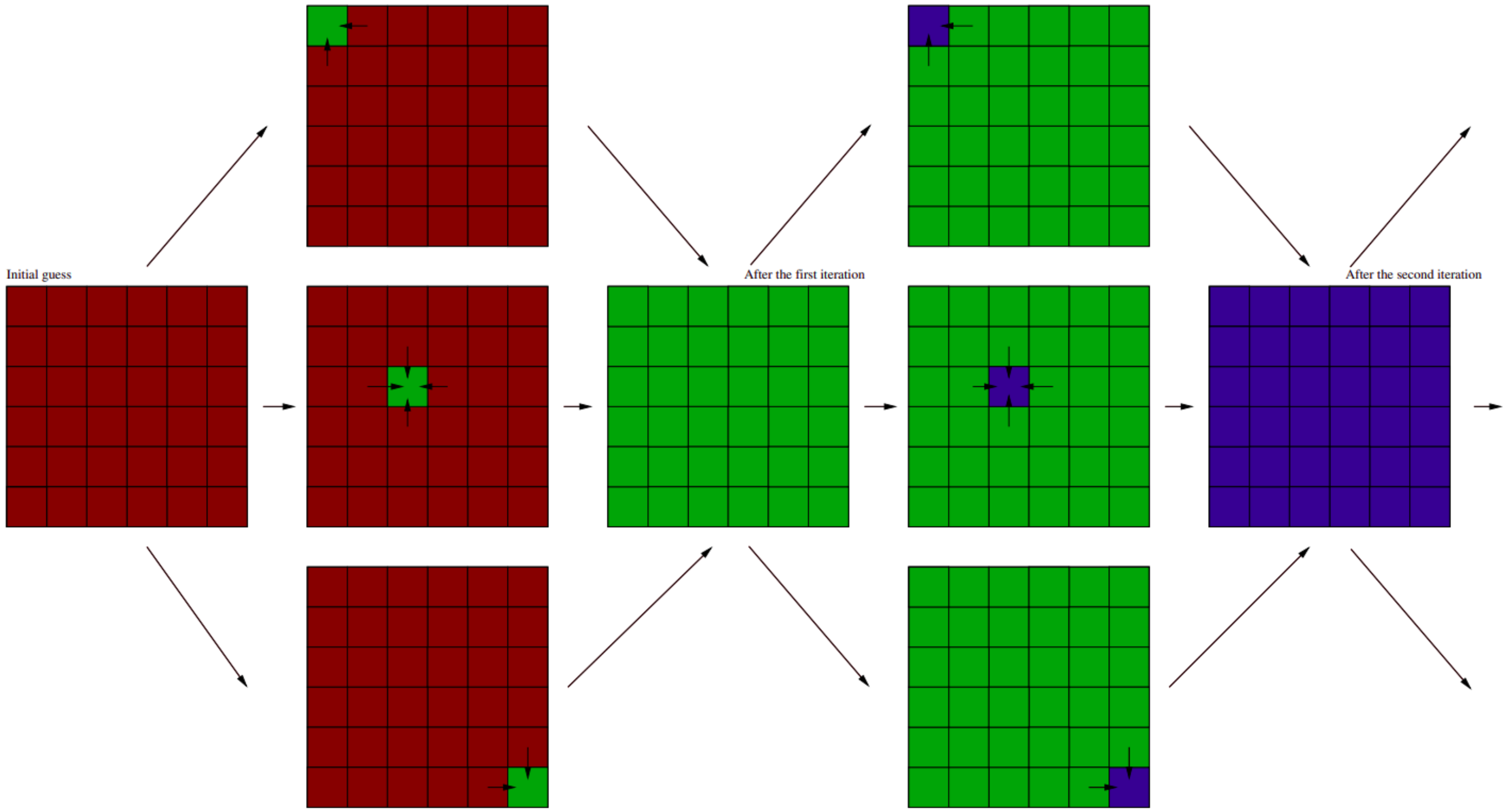
$$U^*(s) = R(s) + \gamma \max_{a} \sum_{s' \in \mathrm{sp}(S)} P(s' \mid s, a) U^*(s').$$

$\rightsquigarrow$ we need to compute the value function.

But how do we calculate this?

Value iteration

Start with an initial guess at the utility function, and iteratively refine this:



The updating function (a Bellman update):

$$U^{j+1}(s) := R(s) + \max_a \sum_{s'} P(s' \mid a, s) U^j(s').$$

Value iteration: the algorithm

1. Choose an $\epsilon > 0$ to regulate the stopping criterion.
2. Let U^0 be an initial estimate of the utility function (for example, initialized to zero for all states).
3. Set $i := 0$.
4. **Repeat**
   1. Let $i := i + 1$.
   2. **For** each $s \in \mathrm{sp}(S)$

$$U^i(s) := R(s) + \gamma \cdot \max_a \sum_{s' \in \mathrm{sp}(S)} P(s' \mid a, s) U^{i-1}(s').$$

5. **Until** Stopping criteria met (e.g. $U^i(s) - U^{i-1}(s) < \epsilon$, for all $s \in \mathrm{sp}(S)$)

Value iteration: an example

Value iteration: the impact of the discounting factor

	1	2	3
1	7.91	8.9	10
2	6.82	6.79	2.62
3	5.83	5.66	4.85

The utility function for $\gamma = 0.9$ (51 iterations)

	1	2	3
1	-0.01	0.9	10
2	-0.1	-0.11	-4.29
3	-0.11	-0.11	-0.11

The utility function for $\gamma = 0.1$(17 iterations)

	1	2	3
1	$\rightarrow$	$\rightarrow$	
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\uparrow$	$\leftarrow$

The optimal strategy for $\gamma = 0.9$

	1	2	3
1	$\rightarrow$	$\rightarrow$	
2	$\uparrow$	$\uparrow$	$\uparrow$
3	$\uparrow$	$\leftarrow$	$\leftarrow$

The optimal strategy for $\gamma = 0.1$

Value iteration: convergence

So the algorithm converges for this particular example, but does this hold in general?

Yes, it can be proven that there is only one "true" utility function and value iteration is guaranteed to converge to this utility function. Moreover,

$$m = \frac{\log(\epsilon(1-\gamma)/2R_{\max})}{\log(\gamma)}$$

is an upper bound on the number of iterations required to achieve an error less than ϵ.

Beyond Value Iteration

There are other algorithms beyond value iteration:

- Policy Iteration: Instead of converging to the optimal value function V^* and then extracting the optimal policy, try to converge by changing the policy in each iteration.
- Heuristic Search: Value Iteration enumerates the entire state space before starting to solve the algorithm (like naive implementations of Dijkstra's algorithm). As in A^*, we can reduce the portion of the state-space that we consider by using admissible heuristics.
- Online decision making: There are also algorithms that, instead of computing the entire policy in advance, try to decide what action to perform in the current state.

Summary

In general, in a **Markov decision process**:

- the world is *fully observable*, i.e., the agent can observe the true state of the world at any point in time,
- the uncertainty in the system is a result of the consequences of the actions being non-deterministic (when performing an action we make a state transition with a certain probability, but independent of the time step), and
- for each decision we get a reward (which may be negative) that may depend on the current world state but is independent of the time step.

Value Iteration can be used to find the optimal policy of any MDP.

Reading

- *Chapter 9 Planning with Uncertainty* from the book "Artificial Intelligence:Foundations of Computational Agents" (2nd edition); in particular subchapters:
  - 9.1. Preferences and Utility
  - 9.2. One-Off Decisions
  - 9.3. Sequential Decisions
  - 9.4. The value of information and control
  - 9.5. Decision Processes

Policy iteration

Instead of updating the utility function, make an initial guess at the optimal policy and perform an iterative refinement of this guess:

The updating function:

$$\pi_{i+1}(s) := \arg\max_a \sum_{s' \in \mathrm{sp}(S)} P(s' \mid a, s) U_{\pi_i}(s').$$

The evaluation function:

$$U_{\pi_i}(s) = R(s) + \gamma \sum_{s' \in \mathrm{sp}(S)} P(s' \mid \pi_i(s), s) U_{\pi_i}(s'),$$

which defines a system of linear equalities; the solution is U_{π_i}.

Policy iteration: the algorithm

1. Let π_0 be an initial randomly chosen policy.
2. Set $i := 0$.
3. **Repeat**
   1. Find the utility function U_{π_i} corresponding to the policy π_i [Policy evaluation].
   2. Let $i := i + 1$.
   3. **For** each $s \in \mathrm{sp}(S)$

$$\pi_i(s) := \arg\max_a \sum_{s' \in \mathrm{sp}(S)} P(s' \mid a, s) U_{\pi_{i-1}}(s') [\text{Policy updating}].$$

4. **Until** $\pi_i = \pi_{i-1}$

Policy evaluation can be performed both exactly ($O(n^3)$) or approximately using a simplified version of value iteration.