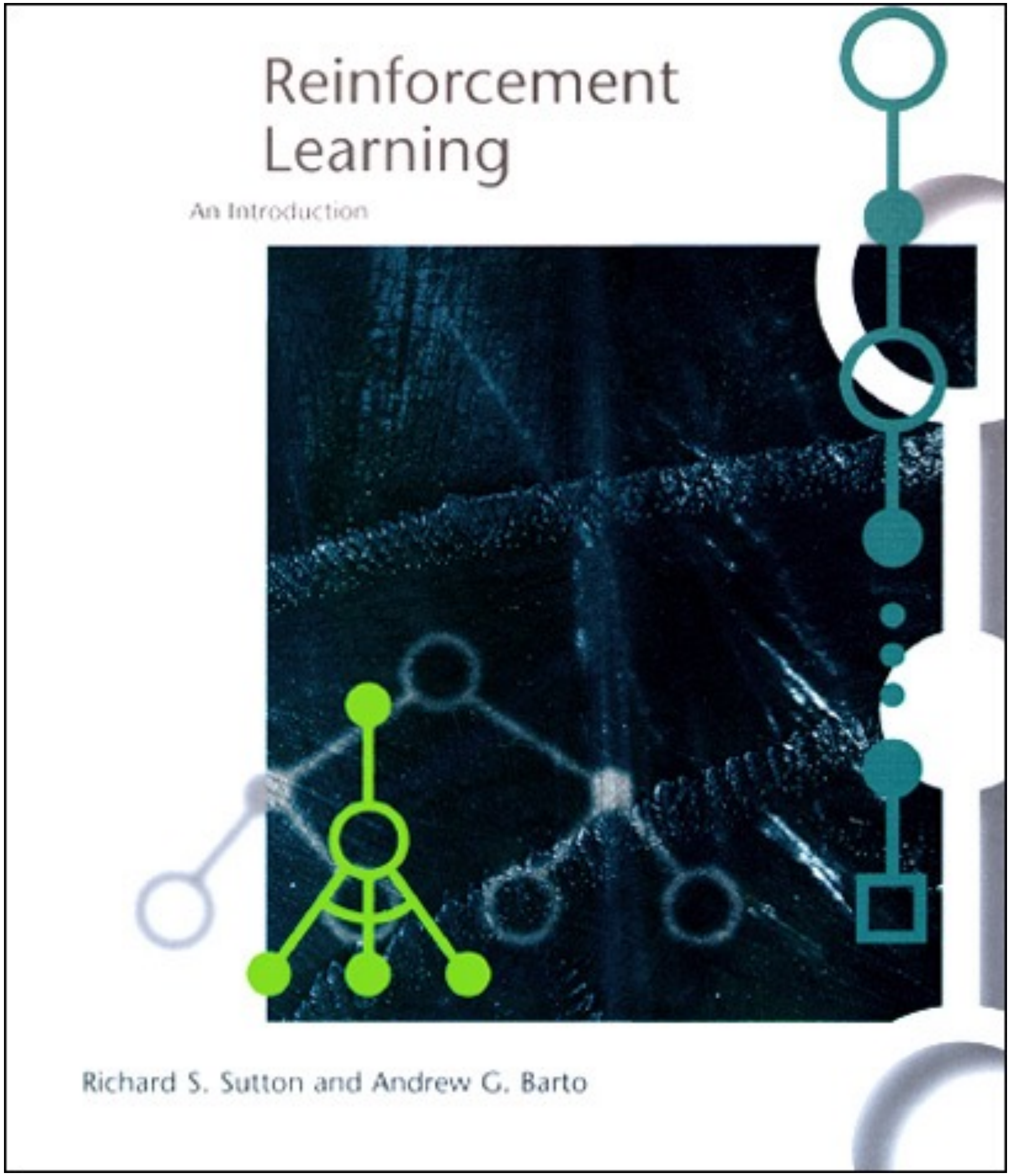
# CMSC 471:
# Reinforcement Learning

Some slides courtesy Cynthia Matuszek and Frank Farrero, with some material from Marie desJardin, Lise Getoor, Jean-Claude Latombe, and Daphne Koller

# There's an entire book!

http://incompleteideas.net/book/the-book-2nd.html



# The Big Idea

- "Planning": Find a sequence of steps to accomplish a goal.
  - Given start state, transition model, goal functions…
- This is a kind of **sequential decision making**.
  - Transitions are deterministic.
- What if they are stochastic (probabilistic)?
  - One time in ten, you drop your sock
- Probabilistic Planning: Make a plan that accounts for probability by **carrying it through the plan**.

# Review: Formalizing Agents

- Given:
  - A state space S
  - A set of actions $a_1, \ldots, a_k$ including their results
  - Reward value at the end of each trial (series of action) (may be positive or negative)
- Output:
  - A **mapping from states to actions**

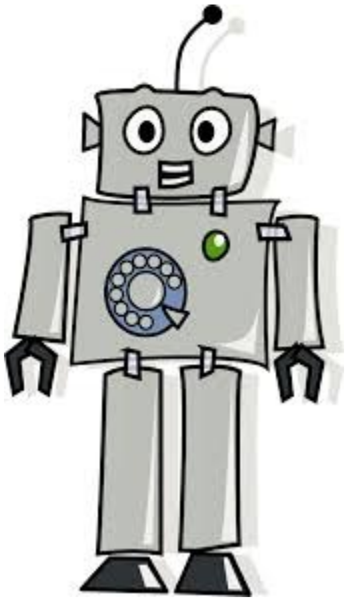
# Review: Formalizing Agents

- Given:
  - A state space S
  - A set of actions $a_1, \ldots, a_k$ including their results
  - Reward value at the end of each trial (series of action) (may be positive or negative)
- Output:
  - A **mapping from states to actions**
  - Which is a **policy**, $\pi$

# Reinforcement Learning

- We often have an agent which has a **task** to perform
  - It takes some actions in the world
  - At some later point, gets feedback on how well it did
  - The agent performs the same task repeatedly
- This problem is called **reinforcement learning**:
  - The agent gets positive reinforcement for tasks done well
  - And gets negative reinforcement for tasks done poorly
  - Must somehow figure out which actions to take next time

# Reinforcement Learning



agent

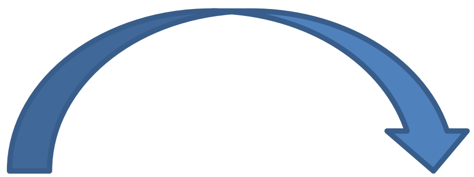
environment

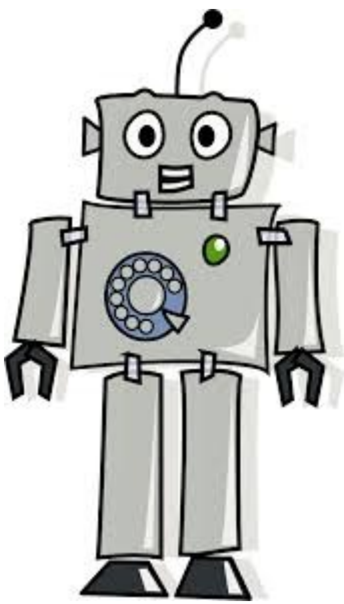
Robot image: openclipart.org

https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Reinforcement Learning

take action





agent

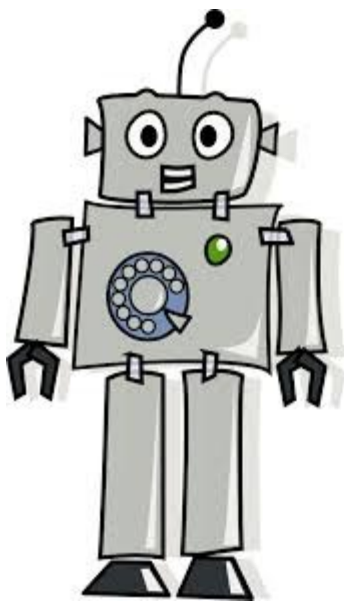
environment

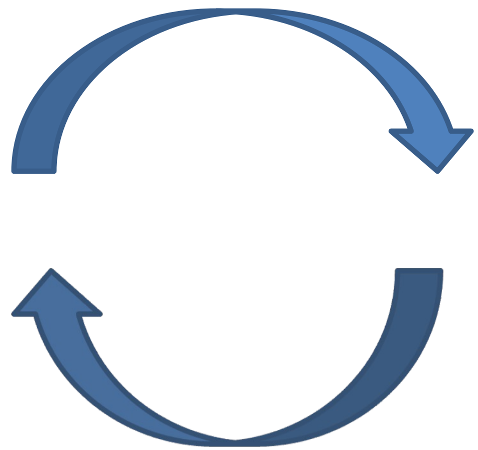
Robot image: openclipart.org

https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Reinforcement Learning

take action





get new state
and/or reward

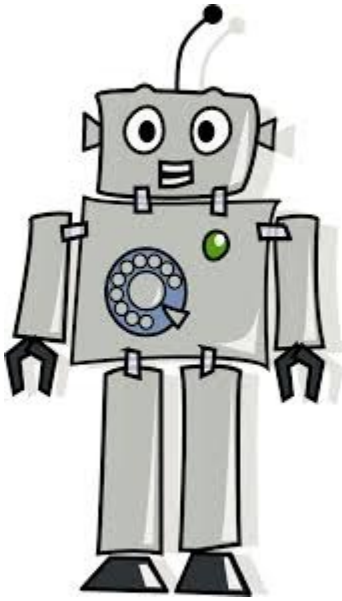
environment

agent

Robot image: openclipart.org

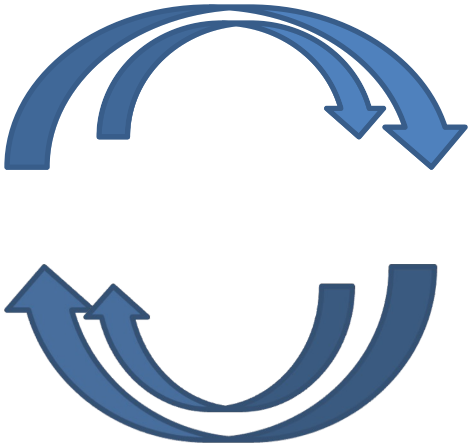
https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Reinforcement Learning



agent

take action



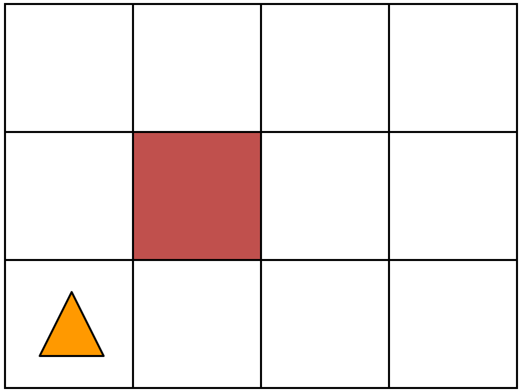
get new state
and/or reward

environment

Robot image: openclipart.org

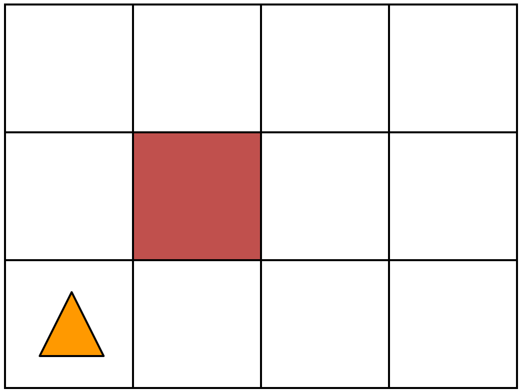
https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Simple Robot Navigation Problem



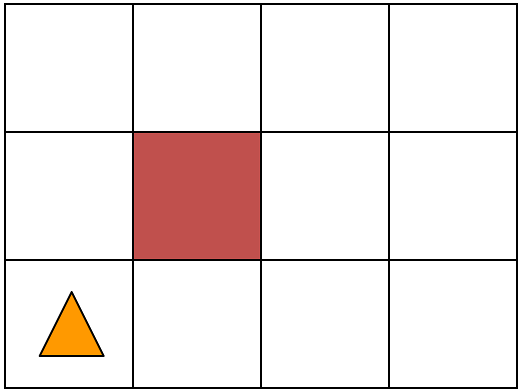
- In each state, the possible actions are U, D, R, and L

# Probabilistic Transition Model



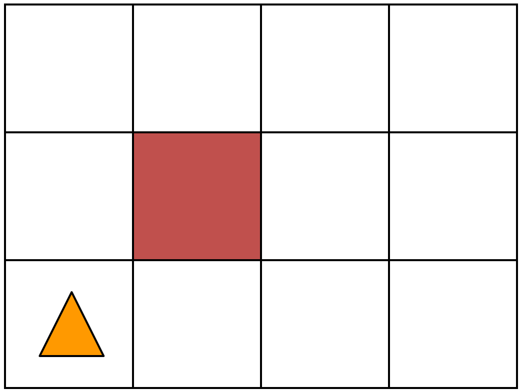
- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)

# Probabilistic Transition Model



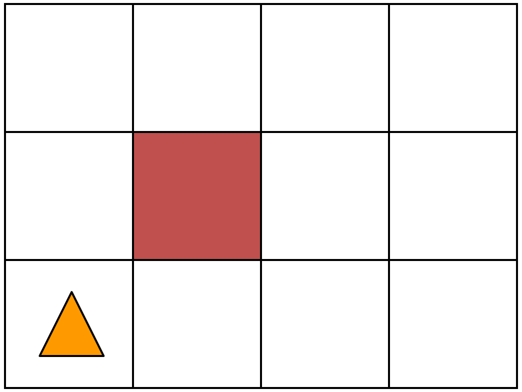
- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)
  - With probability 0.1, the robot moves right one square (if the robot is already in the rightmost row, then it does not move)

# Probabilistic Transition Model



- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)
  - With probability 0.1, the robot moves right one square (if the robot is already in the rightmost row, then it does not move)
  - With probability 0.1, the robot moves left one square (if the robot is already in the leftmost row, then it does not move)

# Probabilistic Transition Model



- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)
  - With probability 0.1, the robot moves right one square (if the robot is already in the rightmost row, then it does not move)
  - With probability 0.1, the robot moves left one square (if the robot is already in the leftmost row, then it does not move)
- D, R, and L have similar probabilistic effects

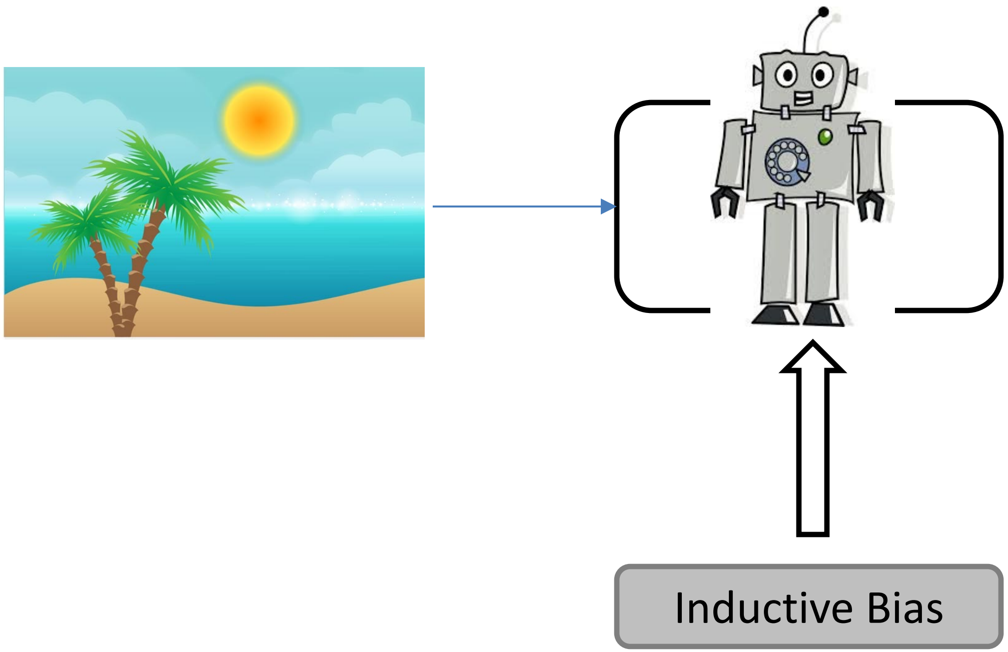
# Markov Property

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

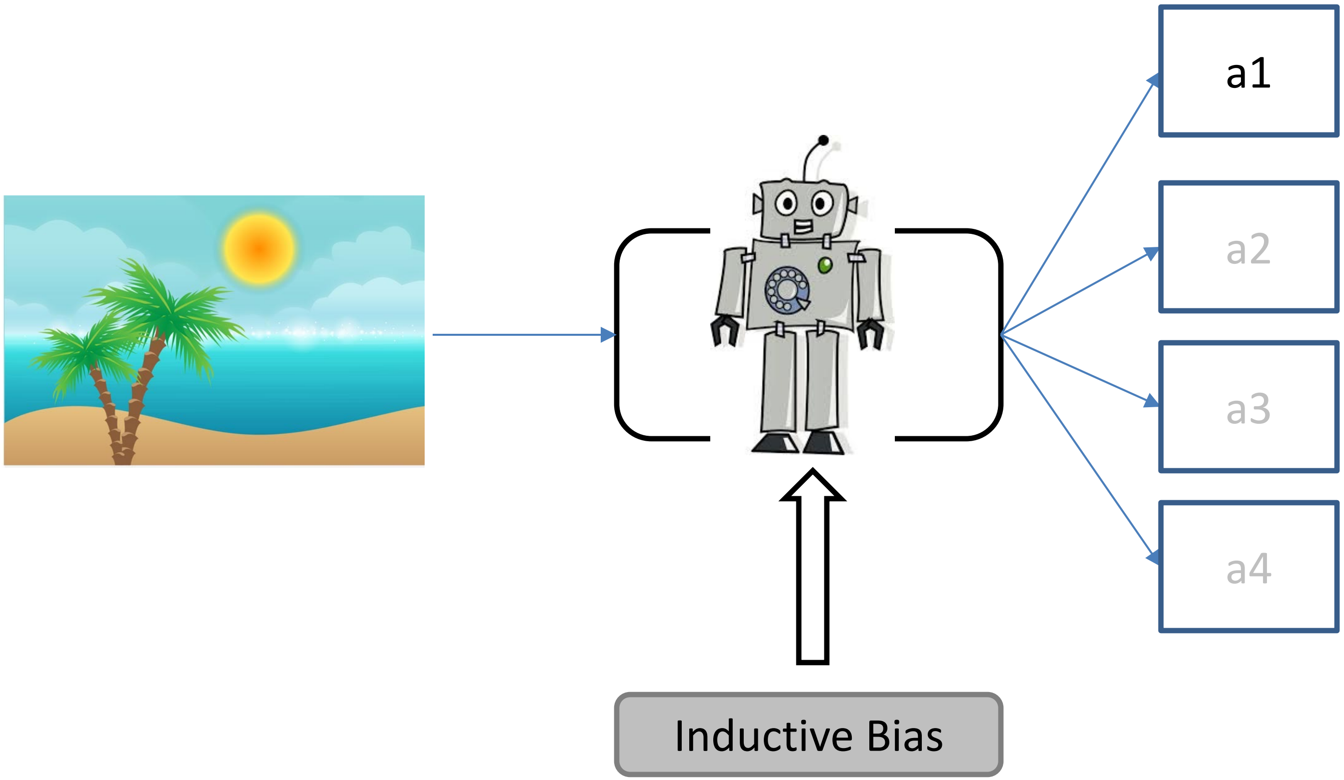
Markov assumption generally: current state only ever depends on previous state (or finite set of previous states).

But what about the **learning** part of reinforcement learning?

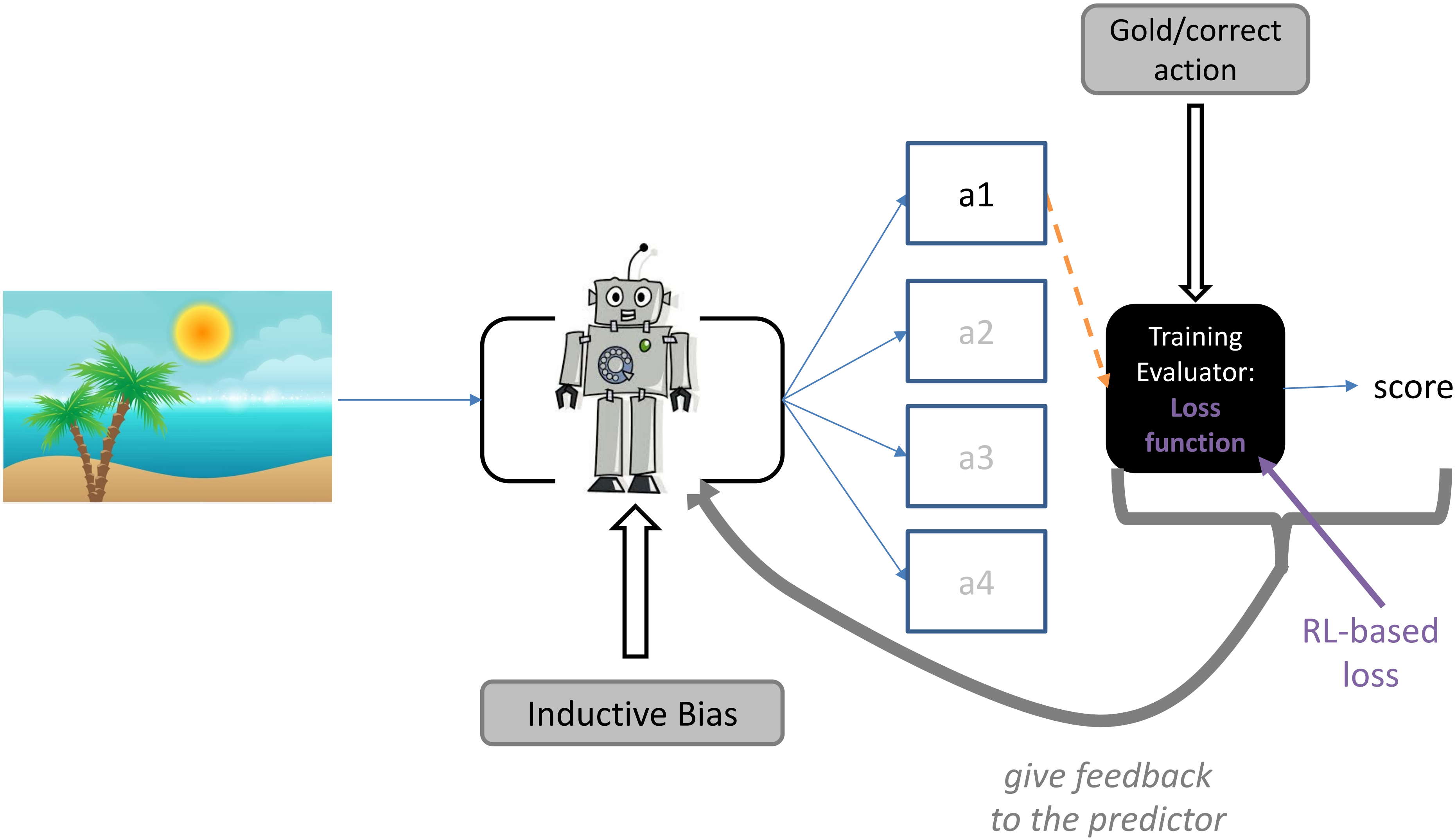
# RL, in our ML framework



# RL, in our ML framework

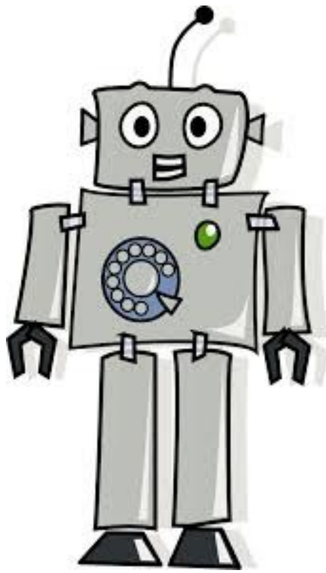


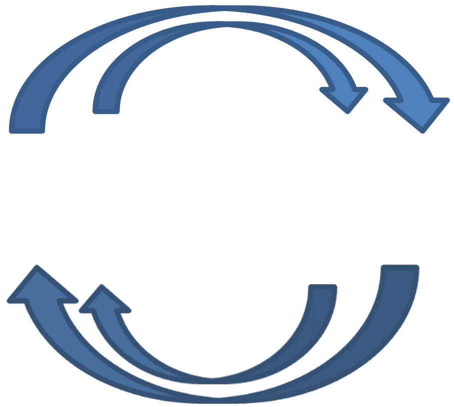
# RL, in our ML framework

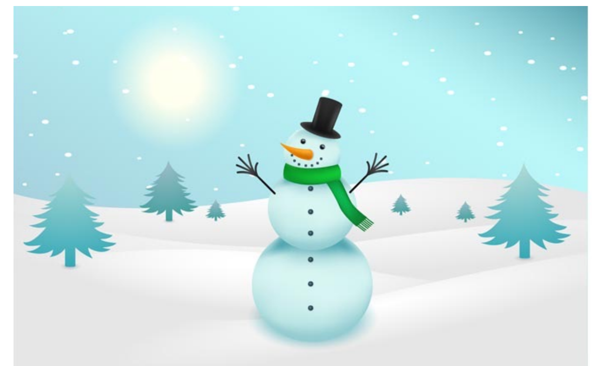


# Markov Decision Process: Formalizing Reinforcement Learning

take action







get new state and/or reward

environment

agent

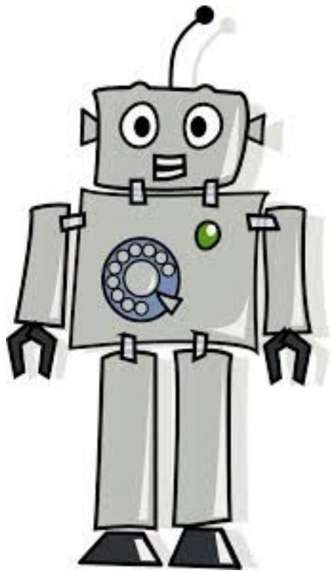
**Markov Decision Process:** $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

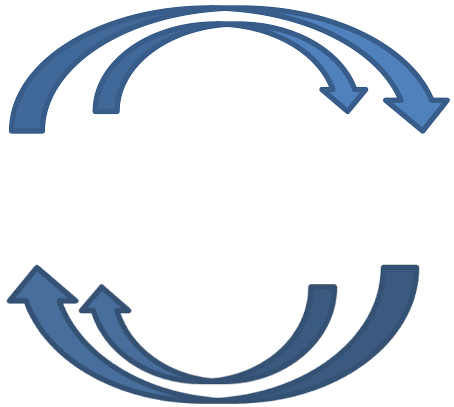
Robot image: openclipart.org

https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Markov Decision Process: Formalizing Reinforcement Learning

take action





get new state
and/or reward

environment

agent

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

$\mathcal{S}$: set of possible states

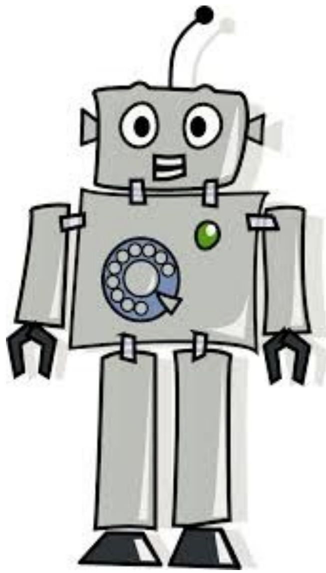
$\mathcal{A}$: set of possible actions

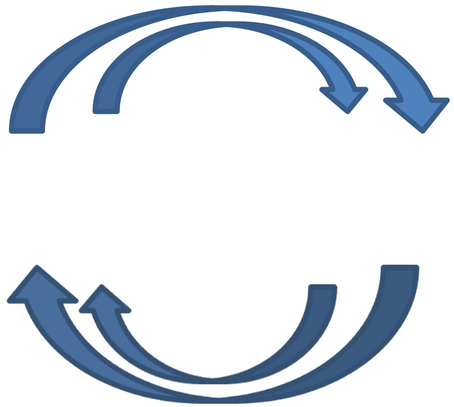
Robot image: openclipart.org

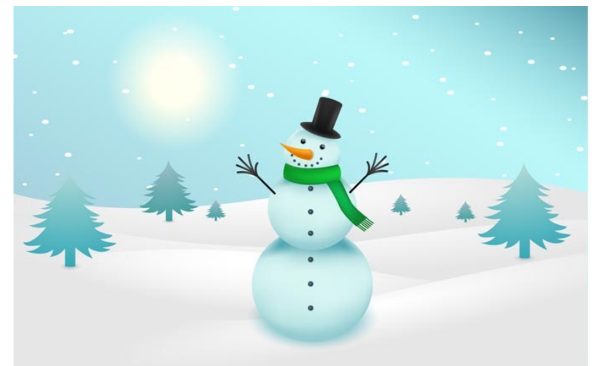
https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Markov Decision Process: Formalizing Reinforcement Learning

take action







get new state and/or reward

environment

agent

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

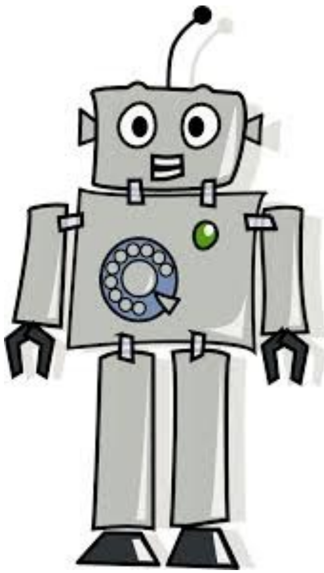
- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs

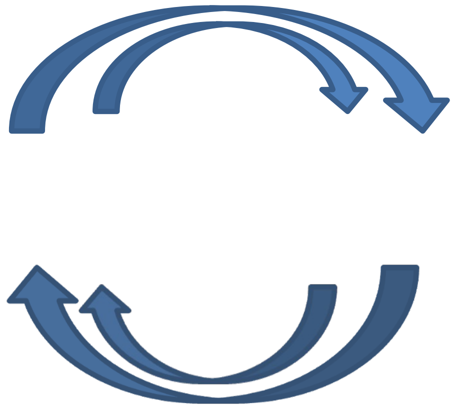
Robot image: openclipart.org

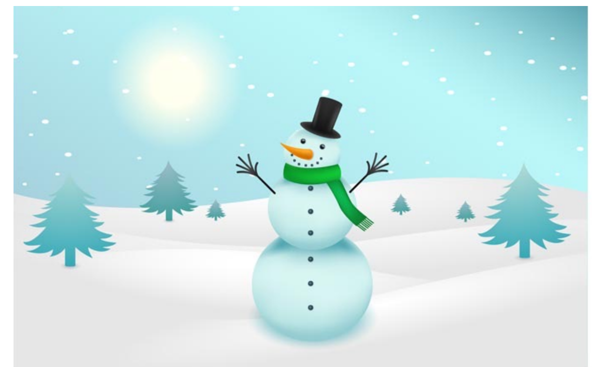
https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Markov Decision Process: Formalizing Reinforcement Learning

take action







get new state and/or reward

environment

agent

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

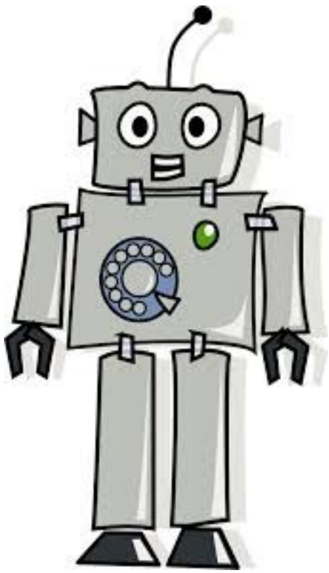
- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution

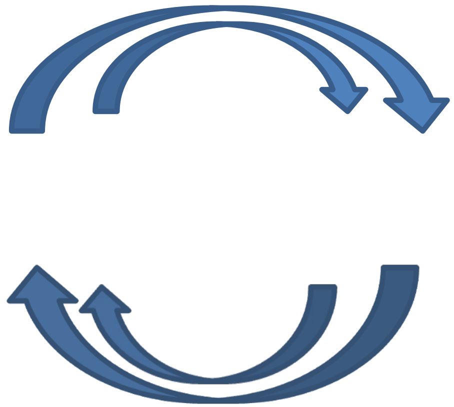
Robot image: openclipart.org

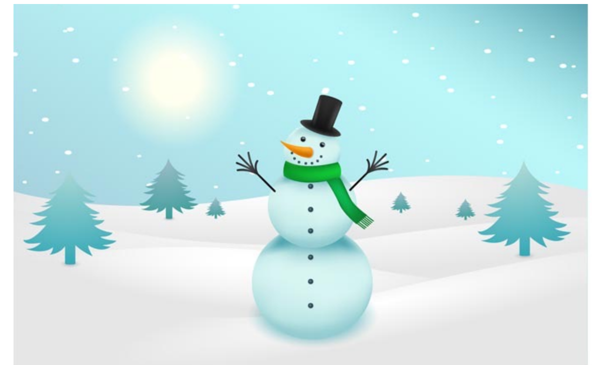
https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

# Markov Decision Process: Formalizing Reinforcement Learning

take action







get new state and/or reward

environment

agent

Markov Decision Process:

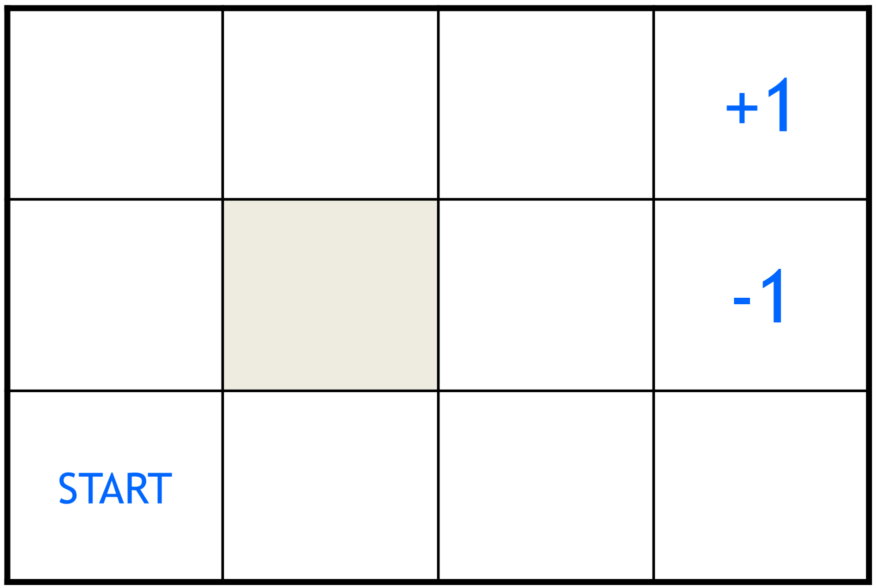
$$(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Robot image: openclipart.org

https://static.vecteezy.com/system/resources/previews/000/090/451/original/four-seasons-landscape-illustrations-vector.jpg

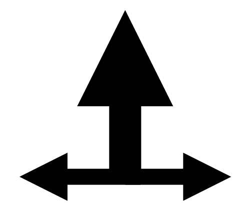
# Robot in a room




actions: UP, DOWN, LEFT, RIGHT

**UP**

80% move UP
10% move LEFT
10% move RIGHT

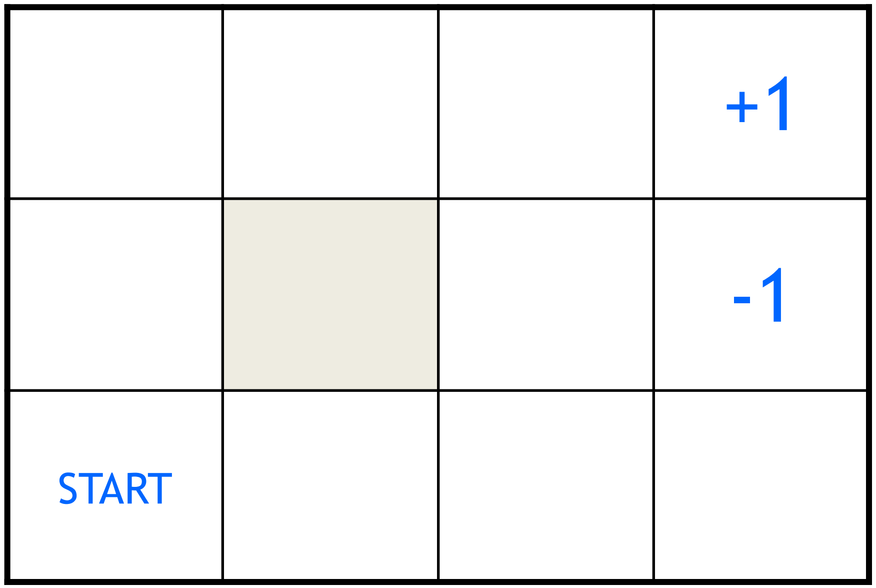


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

**Goal**: what's the strategy to achieve the maximum reward?

Slide courtesy Peter Bodík

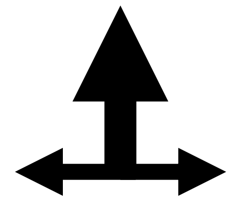
# Robot in a room




actions: UP, DOWN, LEFT, RIGHT

**UP**

80% move UP
10% move LEFT
10% move RIGHT



reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

states: current location
actions: where to go next
rewards

what is the solution? Learn a mapping from (state, action) pairs to new states

Slide courtesy Peter Bodík

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
  choose action $a_t$

# Markov Decision Process: Formalizing Reinforcement Learning

set of possible actions — state-action transition distribution

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

set of possible states — reward of (state, action) pairs — discount factor

Start in initial state $s_0$
for t = 1 to …:
  choose action $a_t$
  “move” to next state $s_t \sim \pi(\cdot | s_{t-1}, a_t)$

$Policy$
$\pi$: S → A

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
  choose action $a_t$
  "move" to next state $s_t \sim \pi(\cdot | s_{t-1}, a_t)$
  get reward $r_t = \mathcal{R}(s_t, a_t)$

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
  choose action $a_t$
  "move" to next state $s_t \sim \pi(\cdot | s_{t-1}, a_t)$
  get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: choose action over time to maximize time-discounted reward

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
   choose action $a_t$
   "move" to next state $s_t \sim \pi(\cdot | s_{t-1}, a_t)$
   get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: choose action over time to maximize discounted reward

Consider all possible future times t

Reward at time t

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
 choose action $a_t$
 "move" to next state $s_t \sim \pi(\cdot | s_{t-1}, a_t)$
 get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize discounted reward

Consider all possible future times t

Discount at time t

Reward at time t

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$

for t = 1 to …:

  choose action $a_t$

  "move" to next state $s_t \sim \pi(\cdot|s_{t-1}, a_t)$

  get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize discounted reward

$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

- $\sum_{t>0}$: Consider all possible future times t
- $\gamma^t$: Discount at time t
- $r_t$: Reward at time t

# Example of Discounted Reward

objective: maximize discounted reward

$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

Consider all possible future times t

Discount at time t

Reward at time t

- If the discount factor $\gamma = 0.8$ then reward

$$0.8^0 r_0 + 0.8^1 r_1 + 0.8^2 r_2 + 0.8^3 r_3 + \cdots + 0.8^n r_n + \ \ldots$$

- Allows you to consider all possible rewards in the future but preferring current vs. future self

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
  choose action $a_t$
  "move" to next state $s_t \sim \pi(\cdot|s_{t-1}, a_t)$
  get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize discounted reward

$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

"solution": the policy $\pi^*$ that maximizes the expected (average) time-discounted reward

# Markov Decision Process: Formalizing Reinforcement Learning

Markov Decision Process: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$

- $\mathcal{S}$: set of possible states
- $\mathcal{A}$: set of possible actions
- $\mathcal{R}$: reward of (state, action) pairs
- $P$: state-action transition distribution
- $\gamma$: discount factor

Start in initial state $s_0$
for t = 1 to …:
  choose action $a_t$
  "move" to next state $s_t \sim \pi(\cdot | s_{t-1}, a_t)$
  get reward $r_t = \mathcal{R}(s_t, a_t)$

objective: maximize discounted reward

$$\max_{\pi} \sum_{t>0} \gamma^t r_t$$

"solution" $$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t>0} \gamma^t r_t \,; \pi\right]$$

# Expected Value of a Random Variable

*random variable*

$$X \sim p(\cdot)$$

# Expected Value of a Random Variable

*random variable*

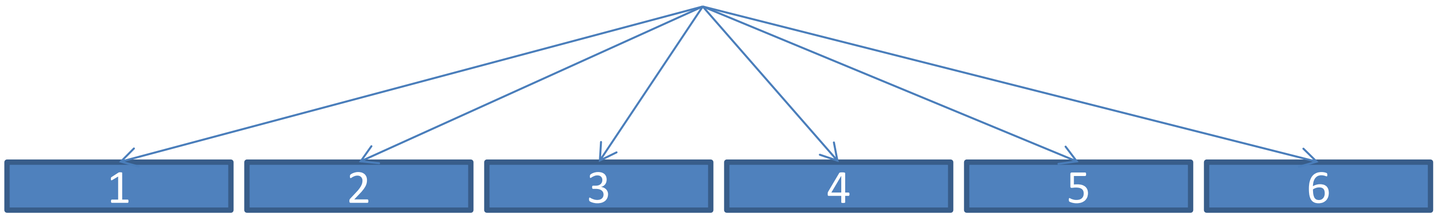
$$X \sim p(\cdot)$$

$$\mathbb{E}[X] = \sum_{x} x\, p(x)$$

*expected value (distribution p is implicit)*

# Expected Value: Example

*uniform distribution of number of cats I have*



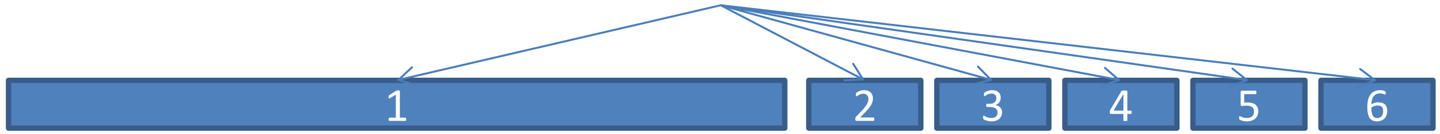

$$\mathbb{E}[X] = \sum_x x\, p(x)$$

1/6 * 1 +
1/6 * 2 +
1/6 * 3 +
1/6 * 4 +
1/6 * 5 +
1/6 * 6

= 3.5

# Expected Value: Example 2

*non-uniform distribution of number of cats a normal cat person has*




$$\mathbb{E}[X] = \sum_x x\, p(x)$$

1/2 * 1 +
1/10 * 2 +
1/10 * 3 +
1/10 * 4 +
1/10 * 5 +
1/10 * 6

= 2.5

# Expected Value of a Function of a Random Variable

$$X \sim p(\cdot)$$

$$\mathbb{E}[X] = \sum_{x} x\, p(x)$$

$$\mathbb{E}[f(X)] = ???$$

# Expected Value of a Function of a Random Variable

$$X \sim p(\cdot)$$

$$\mathbb{E}[X] = \sum_x x\, p(x)$$

$$\mathbb{E}[f(X)] = \sum_x f(x)\, p(x)$$

# Some Challenges

1. Representing states (and actions)

2. Defining our reward

3. Learning our policy

# Overview: Learning Strategies

Dynamic Programming

Q-learning

Monte Carlo approaches

# Dynamic programming

use value functions to structure the search for good policies

Value function is defined as $V^{\pi}$

# Dynamic programming

use value functions to structure the search for good policies

Value function is defined as $V^{\pi}$

policy evaluation: compute $V^{\pi}$ from $\pi$

policy improvement: improve $\pi$ based on $V^{\pi}$

# Dynamic programming

use value functions to structure the search for good policies

Value function is defined as $V^\pi$

policy evaluation: compute $V^\pi$ from $\pi$

policy improvement: improve $\pi$ based on $V^\pi$

start with an arbitrary policy

repeat evaluation/improvement until convergence

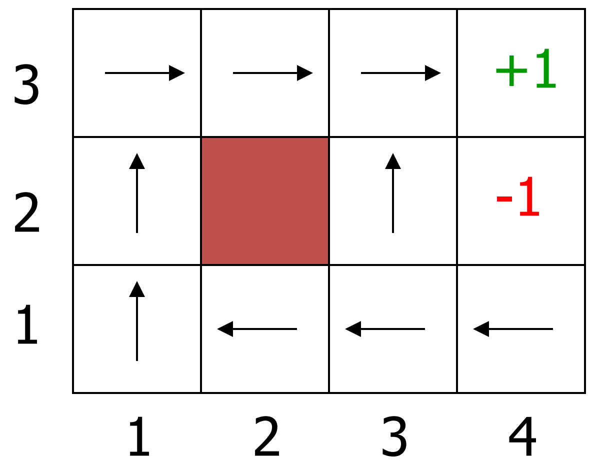
# Reactive Agent Algorithm

Repeat:

- s ← sensed state
- If s is a terminal state then exit
- a ← choose action (given s)
- Perform a

Accessible or observable state

# Policy (Reactive/Closed-Loop Strategy)



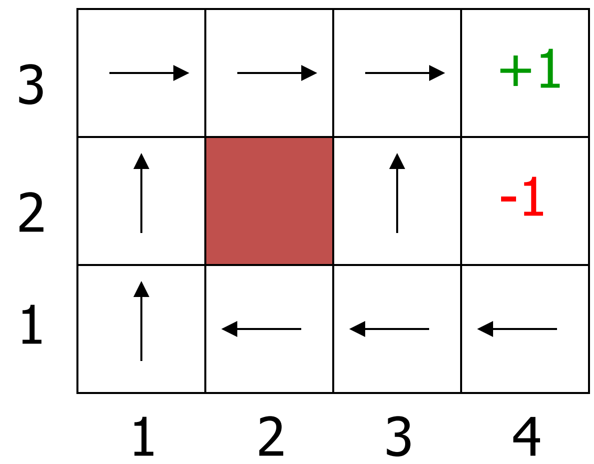

- In every state, we need to know what to do
- The **goal** doesn't change
- A policy (Π) is a complete mapping from *states* to *actions*
  - "If in [3,2], go up; if in [3,1], go left; if in…"

# Reactive Agent Algorithm

Repeat:

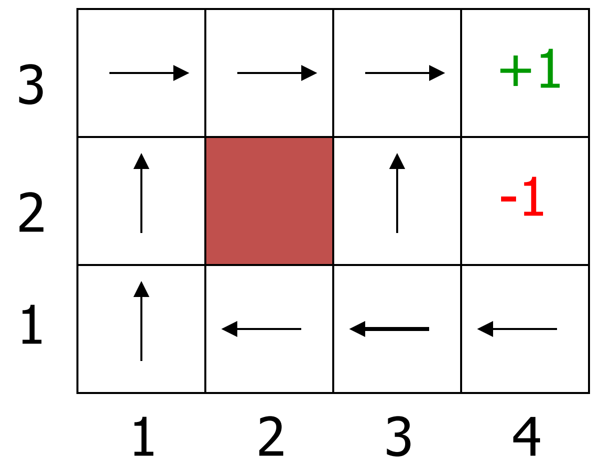
- s ← sensed state
- If s is terminal then exit
- a ← $\Pi$(s)
- Perform a

# Optimal Policy




- A policy Π is a complete mapping from states to actions
- The optimal policy Π* is the one that always yields a history (sequence of steps ending at a terminal state) with maximal ***expected*** utility

# Optimal Policy




- A policy Π is a comp ... ns
- The optimal policy Π ...
  history with maximal expected utility

This problem is called a
Markov Decision Problem (MDP)

How to compute Π*?

# Defining Value Function

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions

# Defining Value Function

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions
  - We've defined value function retroactively (i.e., the value function/utility of a history/sequence of states is known *once we finish it*)

# Defining Value Function

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions
  - We've defined value function retroactively (i.e., the value function/utility of a history/sequence of states is known *once we finish it*)
  - What is the value function of a particular ***state*** in the middle of decision making?

# Defining Value Function

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions
  - We've defined value function retroactively (i.e., the value function/utility of a history/sequence of states is known *once we finish it*)
  - What is the value function of a particular ***state*** in the middle of decision making?
  - Need to compute ***expected value function*** of possible future histories/states

# Defining Value Function

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions
  - We've defined value function retroactively (i.e., the value function/utility of a history/sequence of states is known *once we finish it*)
  - What is the value function of a particular ***state*** in the middle of decision making?
  - Need to compute ***expected value function*** of possible future histories/states

# Defining Value Function

$$V^\pi(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right].$$

$V^\pi(s)$ is simply the expected sum of discounted rewards upon starting in state $s$, and taking actions according to $\pi$.[1]

Given a fixed policy $\pi$, its value function $V^\pi$ satisfies the **Bellman equations**:

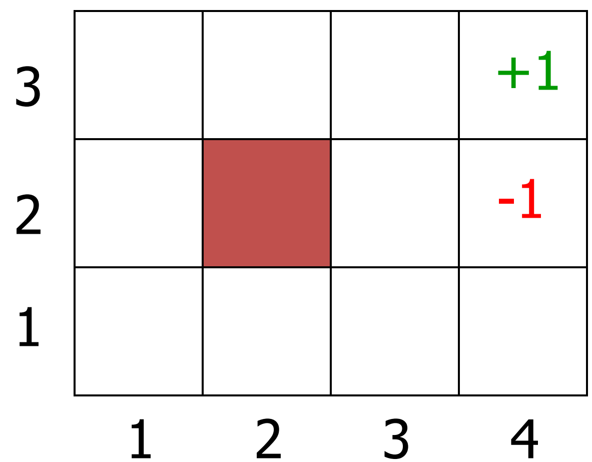
$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

# Value Iteration

**Algorithm 4** Value Iteration

For each state $s$, initialize $V(s) := 0$.
**for** until convergence **do**
    For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \tag{15.4}$$



# Value Iteration

---

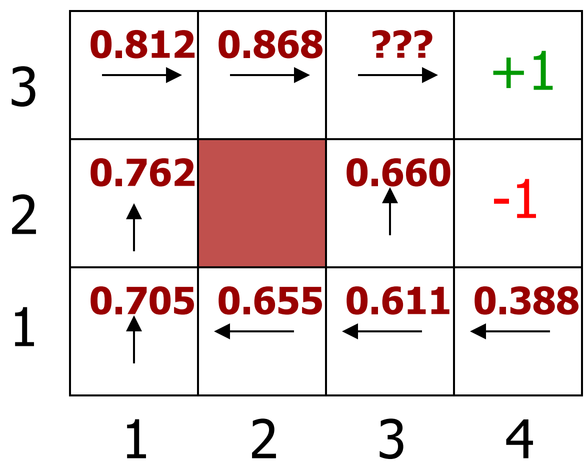
**Algorithm 4** Value Iteration

---

1: For each state $s$, initialize $V(s) := 0$.
2: **for** until convergence **do**
3: For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \tag{15.4}$$

---

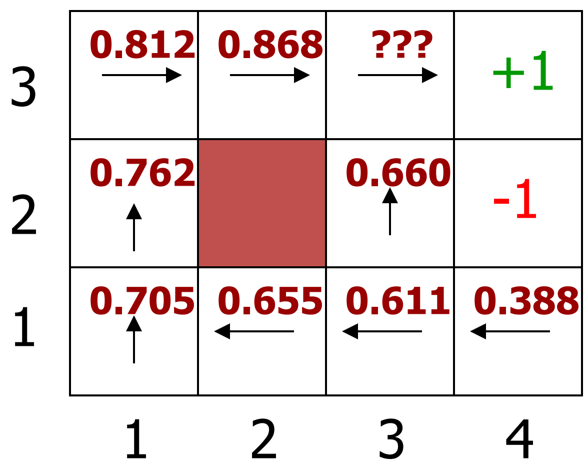


# Value Iteration

**Algorithm 4** Value Iteration

1: For each state $s$, initialize $V(s) := 0$.
2: **for** until convergence **do**
3: For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \tag{15.4}$$



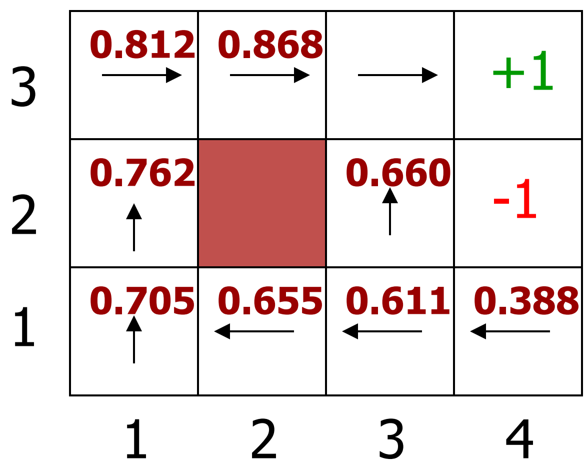

EXERCISE: What is V*([3,3]) (assuming that the other V* are as shown)?

# Value Iteration

**Algorithm 4** Value Iteration

For each state $s$, initialize $V(s) := 0$.
**for** until convergence **do**
    For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \tag{15.4}$$



# Value Iteration

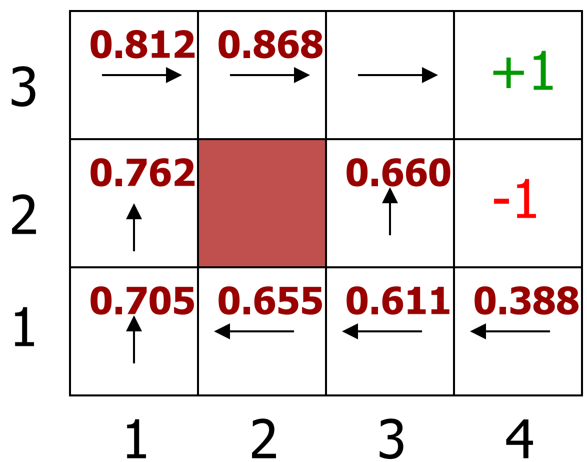
---

**Algorithm 4** Value Iteration

1: For each state $s$, initialize $V(s) := 0$.
2: **for** until convergence **do**
3: For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \tag{15.4}$$

---




$V^*_{3,3} =$
$R_{3,3} +$
$[P_{3,2}\, V^*_{3,2} + P_{3,3}\, V^*_{3,3} + P_{4,3}\, V^*_{4,3}]$

# Value Iteration

---

**Algorithm 4** Value Iteration

---

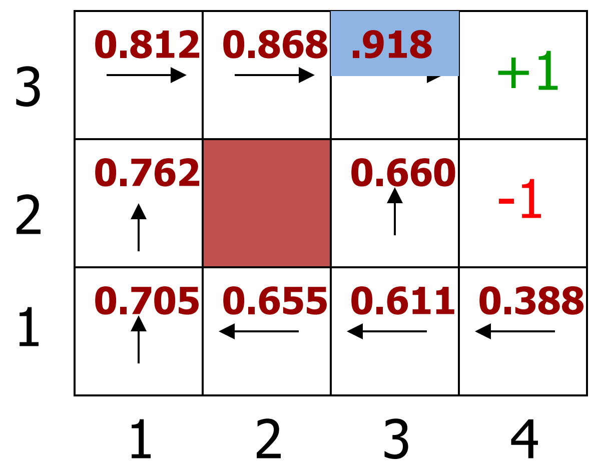
1: For each state $s$, initialize $V(s) := 0$.

2: **for** until convergence **do**

3: For every state, update

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \tag{15.4}$$

---




$V^*_{3,3} =$

$R_{3,3} +$

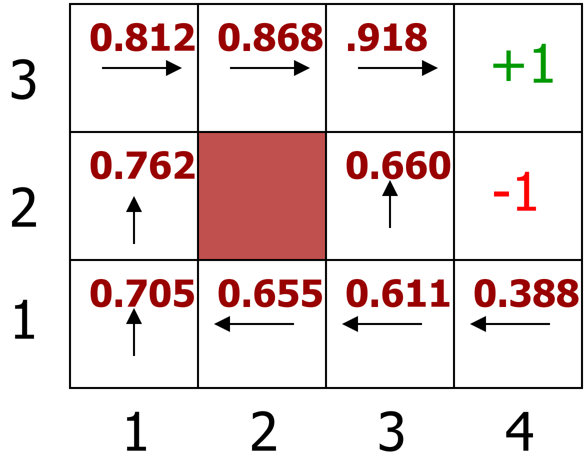
$[P_{3,2}\ V^*_{3,2} + P_{3,3}\ V^*_{3,3} + P_{4,3}\ V^*_{4,3}]$

From (3, 3), 3 options: (3, 2), (4, 3), (3, 4) => but there is no (3,4) but wall, so bounced off and remains at (3, 3)

# Value Iteration

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V^*(s').$$

What's next action for (3, 1)??

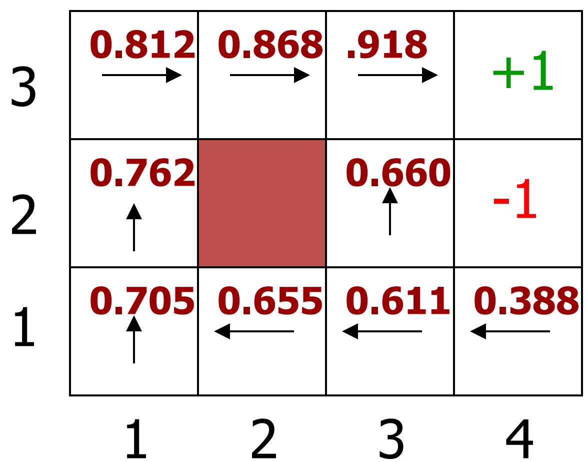



Whichever is higher becomes next action for (3, 1)

# Value Iteration

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s').$$

What's next action for (3, 1)??




$\pi^*_{3,1}$ being (←) =
$P_{up}\ V^*_{1,2} + P_{left}\ V^*_{3,3}$ (Bounced off) $+ P_{right}\ V^*_{3,2}$
= 0.8 * 0.655 + 0.1 * 0.611 + 0.1 * 0.66

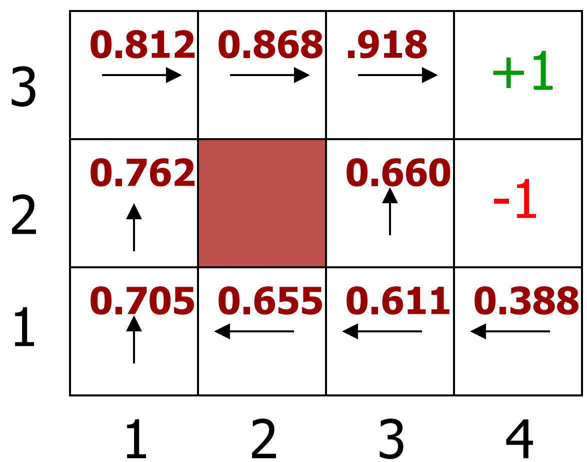
Whichever is higher becomes next action for (3, 1)

# Value Iteration

$$\pi^*(s) = \arg\max_{a\in A} \sum_{s'\in S} P_{sa}(s')V^*(s').$$

What's next action for (3, 1)??

$\pi^*_{3,1}$ being (←) =
$P_{up}\ V^*_{1,2} + P_{left}\ V^*_{3,3}$ (Bounced off) $+ P_{right}\ V^*_{3,2}$
= 0.8 * 0.655 + 0.1 * 0.611 + 0.1 * 0.66

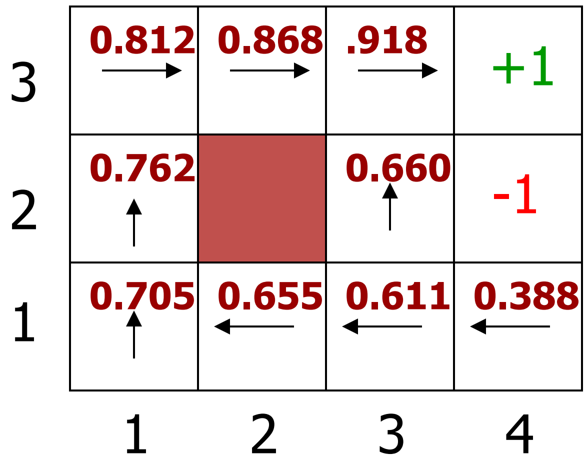



Whichever is higher becomes next action for (3, 1)

# Value Iteration

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V^*(s').$$

What's next action for (3, 1)??




$\pi^*_{3,1}$ being (←) =
$P_{up}\ V^*_{1,2} + P_{left}\ V^*_{3,3}$ (Bounced off) $+ P_{right}\ V^*_{3,2}$

= 0.8 * 0.655 + 0.1 * 0.611 + 0.1 * 0.66

$\pi^*_{3,1}$ being (↑) =
$P_{up}\ V^*_{3,2} + P_{left}\ V^*_{2,1} + P_{right}\ V^*_{1,4}$

Whichever is higher becomes next action for (3, 1)

# Value Iteration: Summary

- Initialize state values (expected utilities) randomly
- Repeatedly update state values using best action, according to current approximation of state values
- Terminate when state values stabilize
- Resulting policy will be the best policy because it's based on accurate state value estimation

# Overview: Learning Strategies

Dynamic Programming

Q-learning

Monte Carlo approaches

# Monte Carlo policy evaluation

don't need full
knowledge of
environment (just
(simulated) experience)

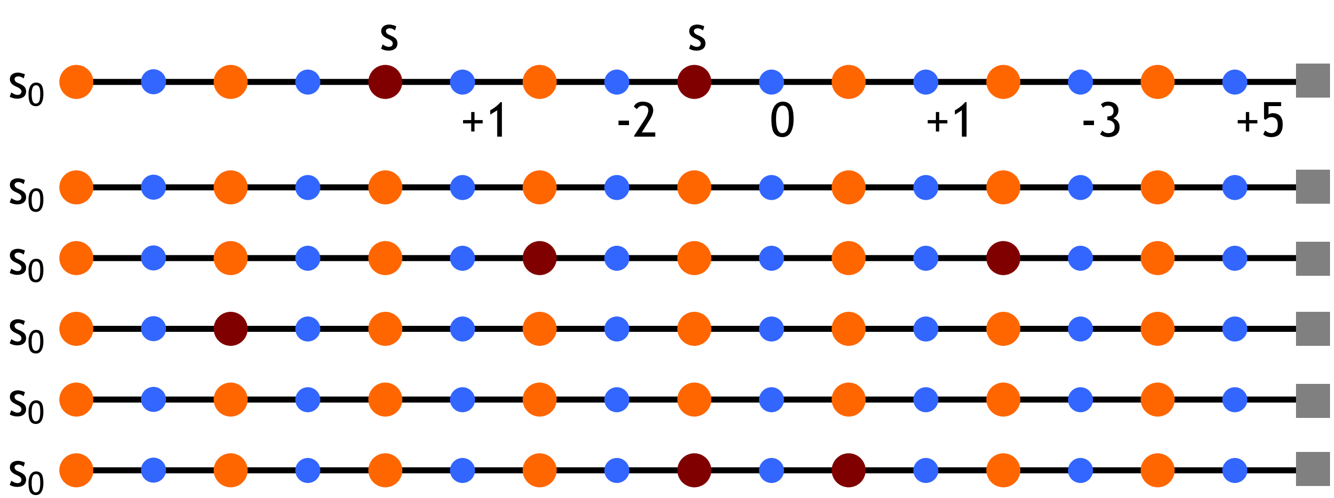
want to estimate $V^{\pi}(s)$

# Monte Carlo policy evaluation

don't need full knowledge of environment (just **(simulated)** experience)

want to estimate $V^{\pi}(s)$

- expected return starting from s and following $\pi$
- estimate as average of observed returns in state s




$R_1(s) = +2$

$R_2(s) = +1$

$R_3(s) = -5$

$R_4(s) = +4$

$$V^{\pi}(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$$

# RL Summary 1:

- **Reinforcement learning systems**
  - Learn **series** of actions or decisions, rather than a single decision
  - Based on feedback given at the end of the series
- A reinforcement learner has
  - A goal
  - Carries out trial-and-error search
  - Finds the best paths toward that goal

# RL Summary 2:

- A typical reinforcement learning system is a reactive agent, interacting with its environment.
- It must balance:
  - Exploration: trying different actions and sequences of actions to discover which ones work best
  - Exploitation (achievement): using sequences which have worked well so far
- Must learn **successful sequences of actions** in an uncertain environment

# RL Summary 3

- Very hot area of research at the moment
- There are **many** more sophisticated RL algorithms
  - Most notably: probabilistic approaches
- Applicable to game-playing, search, finance, robot control, driving, scheduling, diagnosis, …

# EXTRA SLIDES

# Some Challenges

1. Representing states (and actions)

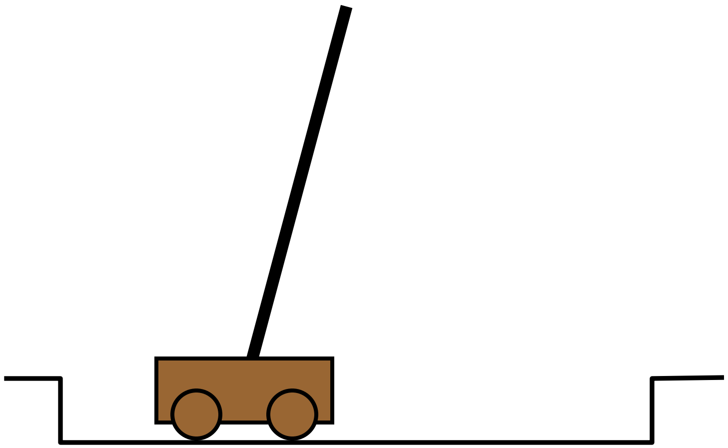
2. Defining our reward

3. Learning our policy

# State Representation

Task: pole-balancing

state representation?



move car left/right to
keep the pole balanced

# State Representation

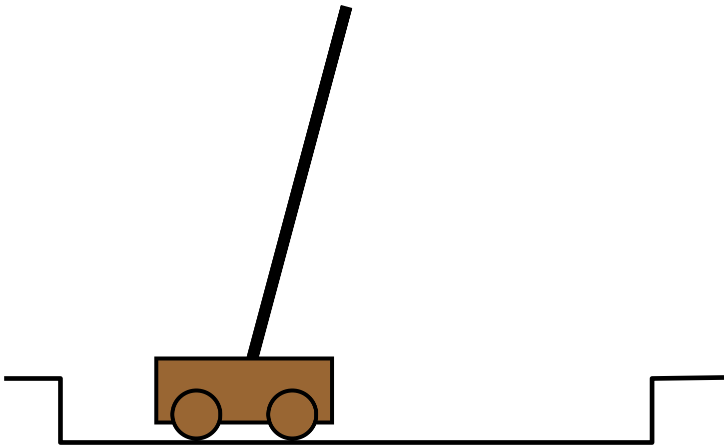
Task: pole-balancing

state representation

position and velocity of car

angle and angular velocity of pole

what about *Markov property*?



move car left/right to
keep the pole balanced

Slide courtesy/adapted Peter Bodík

# State Representation

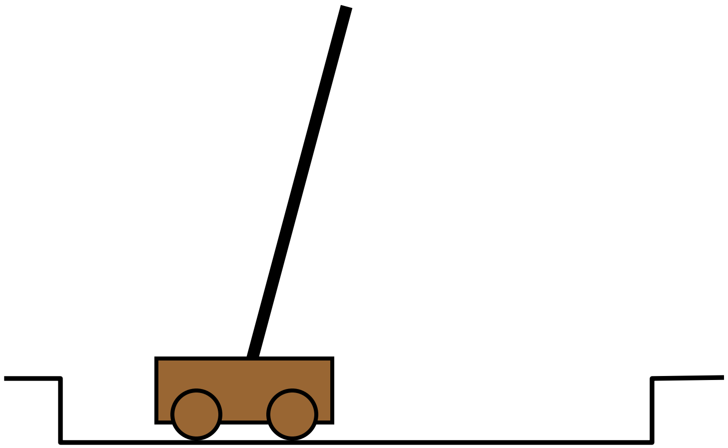
Task: pole-balancing

state representation

- position and velocity of car
- angle and angular velocity of pole

what about *Markov property*?

- would need more info
- noise in sensors, temperature, bending of pole



move car left/right to keep the pole balanced

# Some Challenges

1. Representing states (and actions)
2. Defining our reward
3. Learning our policy

# Designing Rewards

robot in a maze

- episodic task, not discounted, +1 when out, 0 for each step

chess

- GOOD: +1 for winning, -1 losing
- BAD: +0.25 for taking opponent's pieces
  - high reward even when lose

# Designing Rewards

robot in a maze

- episodic task, not discounted, +1 when out, 0 for each step

chess

- GOOD: +1 for winning, -1 losing
- BAD: +0.25 for taking opponent's pieces
  - high reward even when lose

rewards

- rewards indicate what we want to accomplish
- NOT how we want to accomplish it

Slide courtesy/adapted: Peter Bodík

# Designing Rewards

robot in a maze

- episodic task, not discounted, +1 when out, 0 for each step

chess

- GOOD: +1 for winning, -1 losing
- BAD: +0.25 for taking opponent's pieces
  - high reward even when lose

rewards

- rewards indicate what we want to accomplish
- NOT how we want to accomplish it

shaping

- positive reward often very "far away"
- rewards for achieving subgoals (domain knowledge)
- also: adjust initial policy or initial value function

Slide courtesy/adapted: Peter Bodík

# Simple Reinforcement Learning

- Feedback is at the end, on a **series** of actions.
- Very early concept in Artificial Intelligence!
- Arthur Samuels' checker program was a simple reinforcement based learner, initially developed in 1956.
- In 1962 it beat a human checkers master.

*www-03.ibm.com/ibm/history/ibm100/us/en/icons/ibm700series/impacts/*

# Example: Animals Game

– Human: I’m thinking of an animal.

# Example: Animals Game

– Human: I’m thinking of an animal.
– Computer: Is it a bird?

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?

# Example: Animals Game

– Human: I’m thinking of an animal.
– Computer: Is it a bird?
– Human: No.
– Computer: Is it a frog?
– Human: No.

# Example: Animals Game

- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.

# Example: Animals Guessing Game Architecture

- All of the parts of ML Architecture:
  - The Representation is a sequence of questions and pairs of yes/no answers (decision tree).
  - The Actor “walks” the tree, interacting with a human; at each question it chooses whether to follow the “yes” branch or the “no” branch.
  - The Evaluator/loss function is the human player telling the game whether it has guessed correctly.
  - The Learner elicits new questions and adds questions, guesses and branches to the tree.

# Reinforcement Learning (cont.)

- Goal: agent acts in the world to maximize its rewards
- Agent has to figure out what it did that made it get that reward/punishment
  - This is known as the credit assignment problem

# Example: Animals Game

– Human: I'm thinking of an animal.

# Example: Animals Game

– Human: I'm thinking of an animal.
– Computer: Is it a bird?

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.

# Example: Animals Game

- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?

# Example: Animals Game

- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.

# Example: Animals Game

- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.
- Computer: **Tell me a question that distinguishes a frog from a mouse.**

# Example: Animals Game

- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.
- Computer: **Tell me a question that distinguishes a frog from a mouse.**
- Human: Does it have fur?

# Example: Animals Game

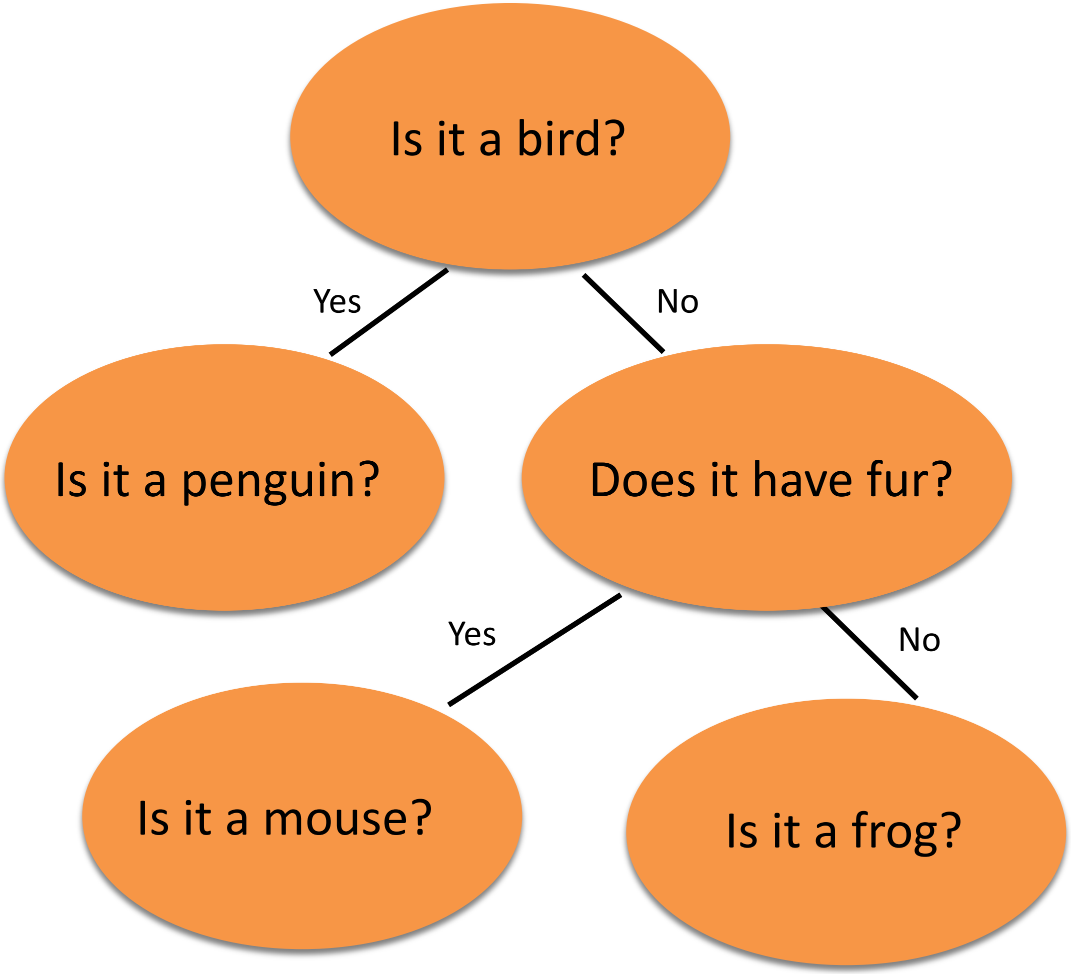
- Human: I’m thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.
- Computer: **Tell me a question that distinguishes a frog from a mouse.**
- Human: Does it have fur?
- Computer: What is the answer for a frog?

# Example: Animals Game

- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.
- Computer: **Tell me a question that distinguishes a frog from a mouse.**
- Human: Does it have fur?
- Computer: What is the answer for a frog?
- Human: No.

# Example: Animals Behind the Scene

Computer: Is it a bird?
Human: No
Computer: Is it a frog?
Human: No
Computer: What is it?
Human: A mouse
Computer: Tell me a question that distinguishes a frog from a mouse.
Human: Does it have fur?
Computer: What is the answer for a frog?
Human: no




After several rounds...

# Reinforcement Learning (cont.)

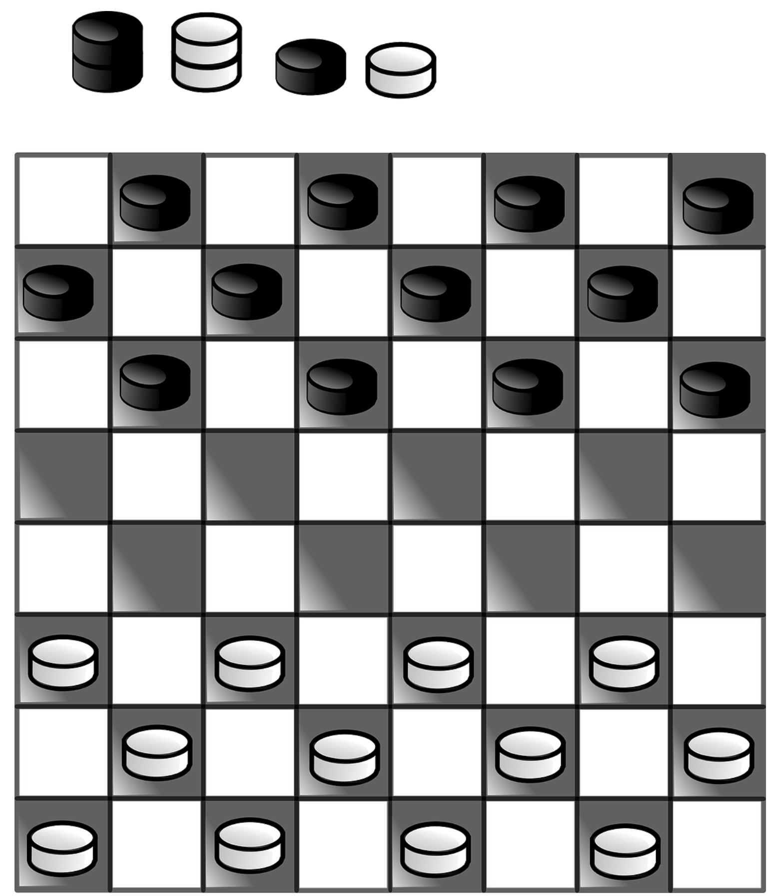
- Goal: agent acts in the world to maximize its rewards
- Agent has to figure out what it did that made it get that reward/punishment
  - This is known as the credit assignment problem
- RL can be used to train computers to do many tasks
  - Backgammon and chess playing
  - Job shop scheduling
  - Controlling robot limbs

# Reactive Agent

- This kind of agent is a reactive agent
- The general algorithm for a reactive agent is:
  - Observe some state
  - If it is a terminal state, stop
  - Otherwise choose an action from the actions possible in that state
  - Perform the action
  - Recur.

# Simple Example

- Learn to play checkers
  - Two-person game
  - 8x8 boards, 12 checkers/side
  - relatively simple set of rules: http://www.darkfish.com/checkers/rules.html
  - Goal is to eliminate all your opponent's pieces



https://pixabay.com/en/checker-board-black-game-pattern-29911

# Representing Checkers

- First we need to represent the game
- To completely describe one step in the game you need
  - A representation of the game board.
  - A representation of the current pieces
  - A variable which indicates whose turn it is
  - A variable which tells you which side is "black"
- There is no history needed
- A look at the current board setup gives you a complete picture of the state of the game

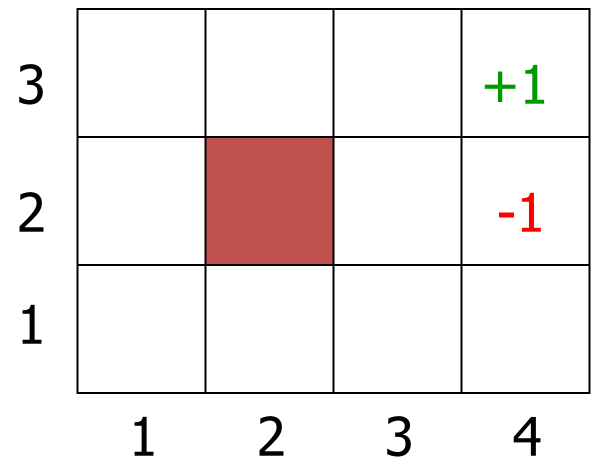
# Representing Checkers

- Second, we need to represent the rules
- Represented as a **set of allowable moves** given board state
  - If a checker is at row x, column y, and row x+1 column y±1 is empty, it can move there.
  - If a checker is at (x,y), a checker of the opposite color is at (x+1, y+1), and (x+2,y+2) is empty, the checker must move there, and remove the "jumped" checker from play.
- There are additional rules, but all can be expressed in terms of the state of the board and the checkers.
- Each rule includes the outcome of the relevant action in terms of the state.
- What's a good reward?

# A More Complex Example

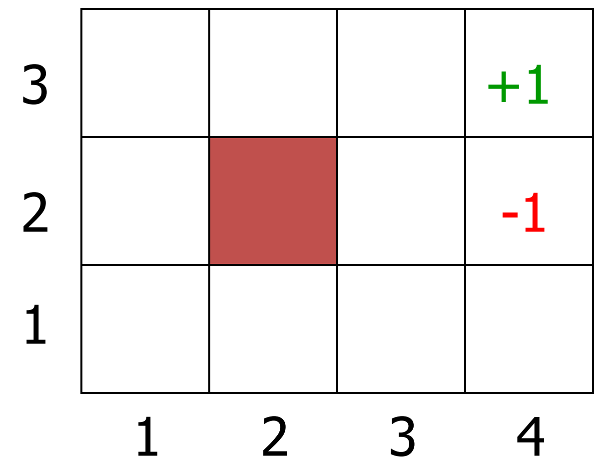
- Consider an agent which must learn to drive a car
  - State?
  - Possible actions?
  - Rewards?

# Utility Function



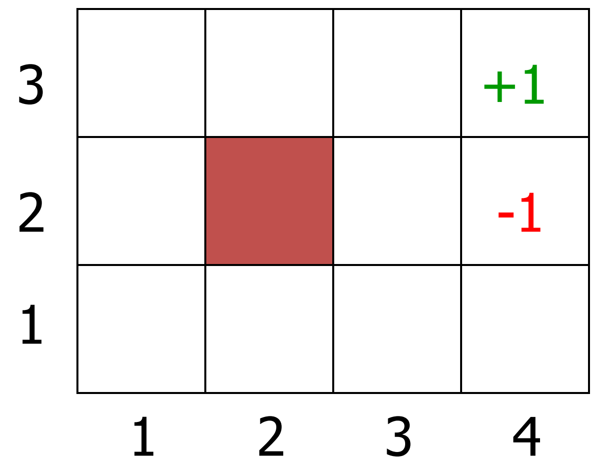

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape

# Utility Function



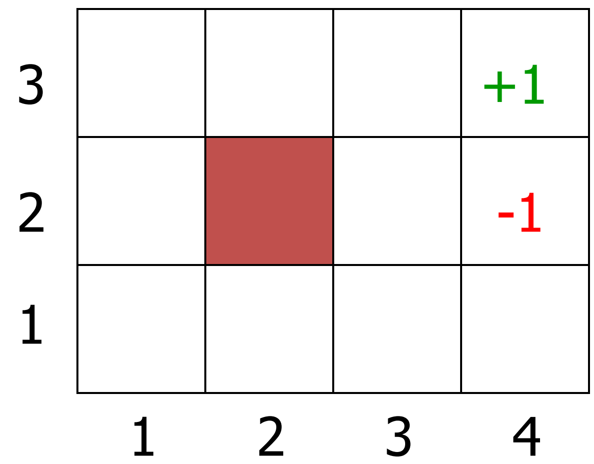

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries

# Utility Function



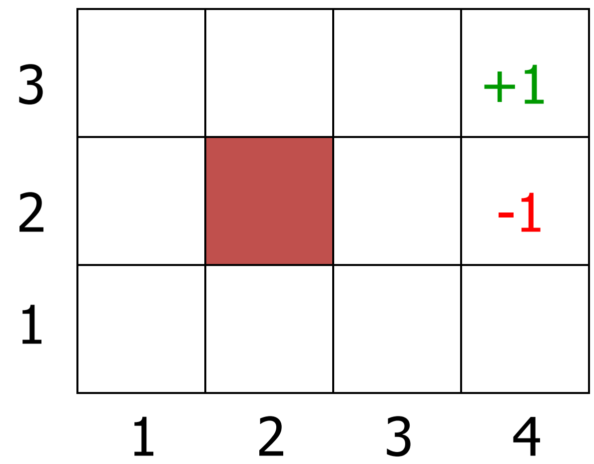

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries
- [4,3] and [4,2] are terminal states

# Utility Function



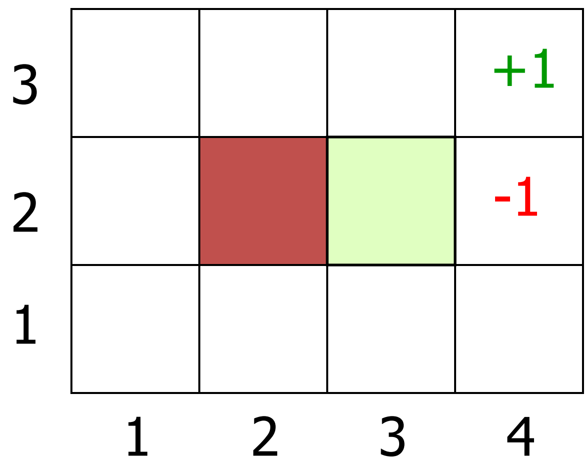

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries
- [4,3] and [4,2] are terminal states
- Histories have utility!

# Utility of a History



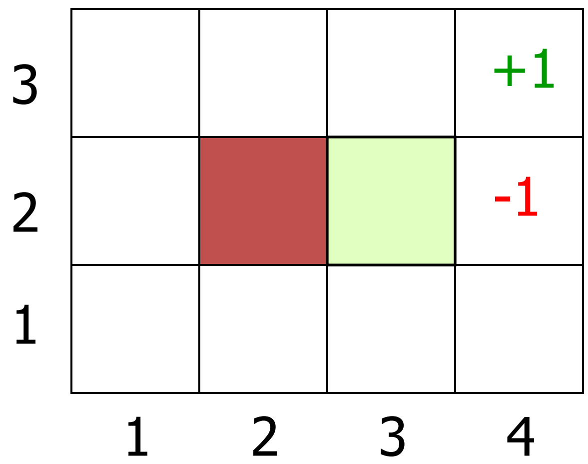

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries
- [4,3] or [4,2] are terminal states
- Histories have utility!
- The utility of a history is defined by the utility of the last state (+1 or –1) minus n/25, where n is the number of moves
  - Many utility functions possible, for many kinds of problems.

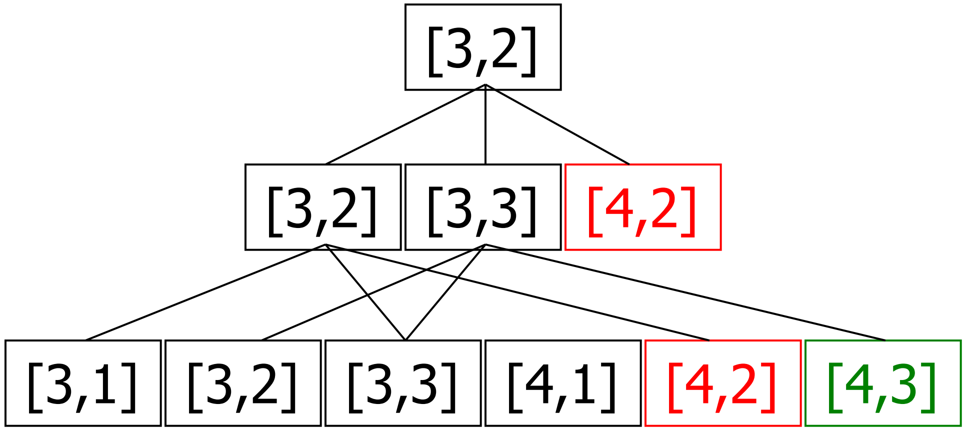
# Utility of an Action Sequence




- Consider the action sequence (U,R) from [3,2]

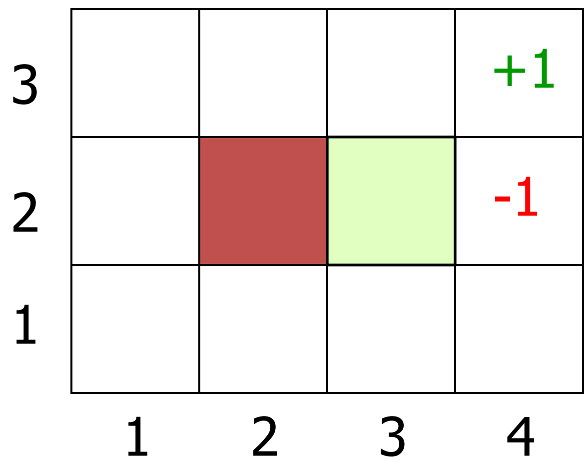
# Utility of an Action Sequence

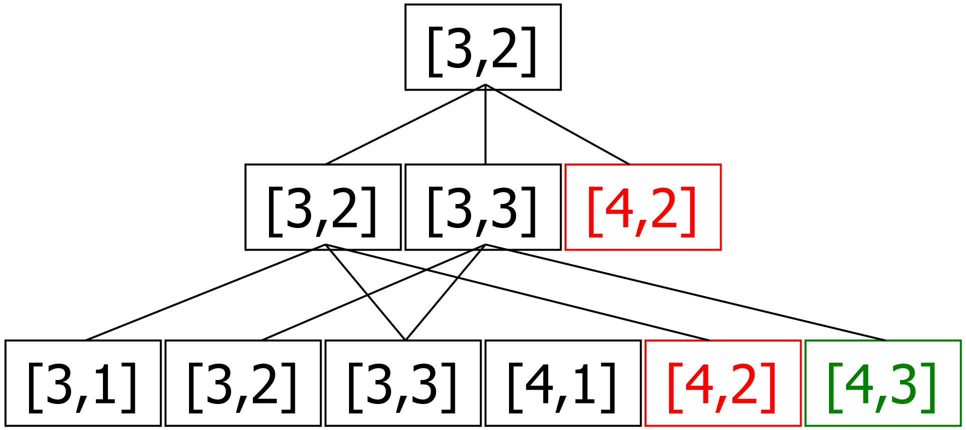



- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability

# Utility of an Action Sequence

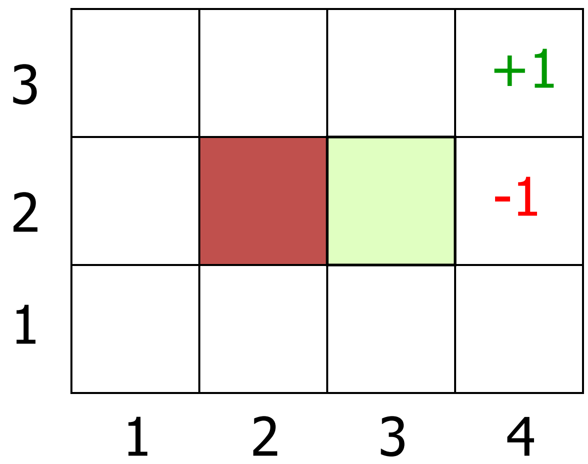


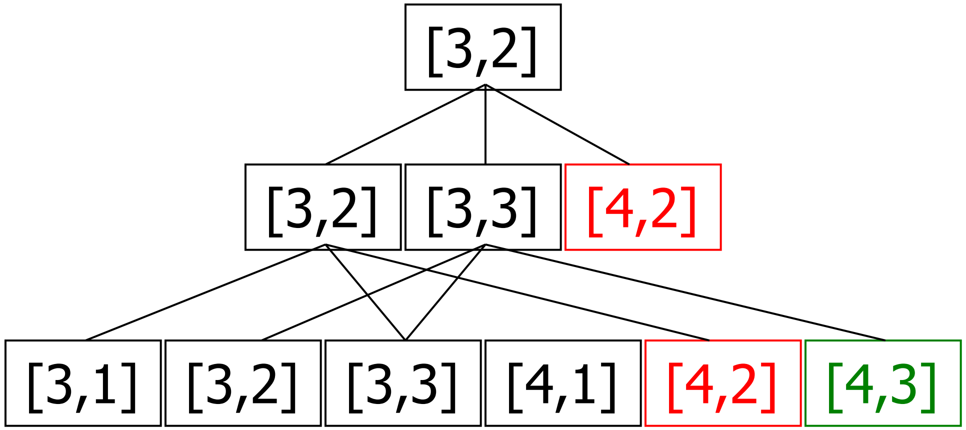




- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories:

$$\mathcal{U} = \Sigma_h \mathcal{U}_h \, \mathbf{P}(h)$$

# Optimal Action Sequence



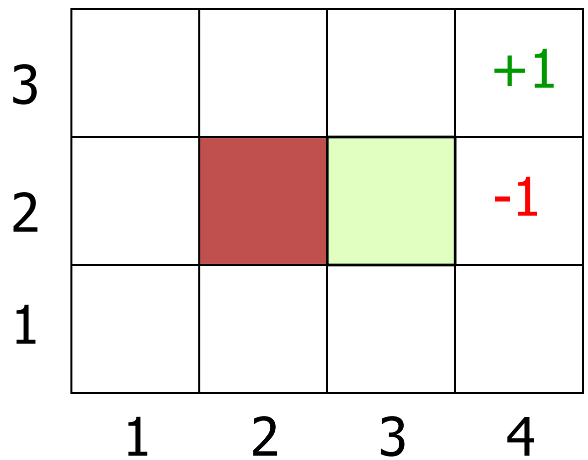



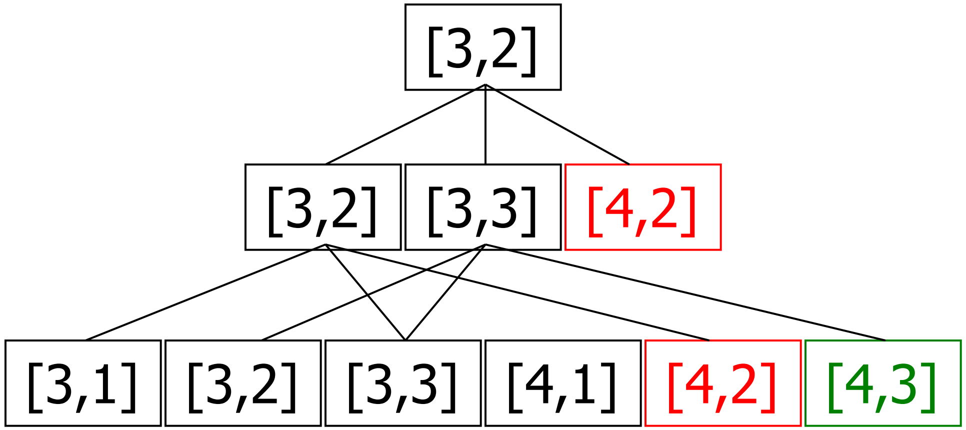

- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories:

$$\mathcal{U} = \Sigma_h \mathcal{U}_h \, \mathbf{P}(h)$$

- The optimal sequence is the one with maximal utility

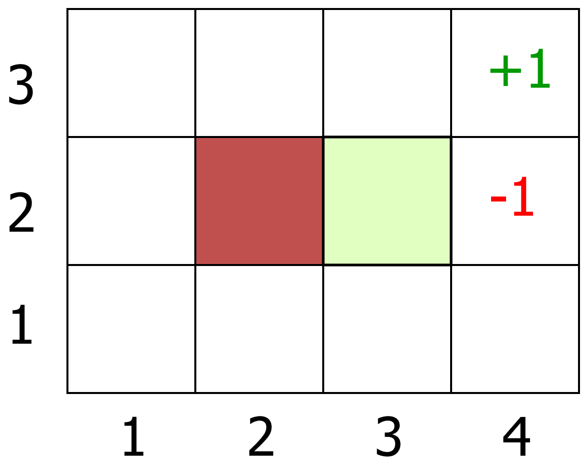
# Optimal Action Sequence

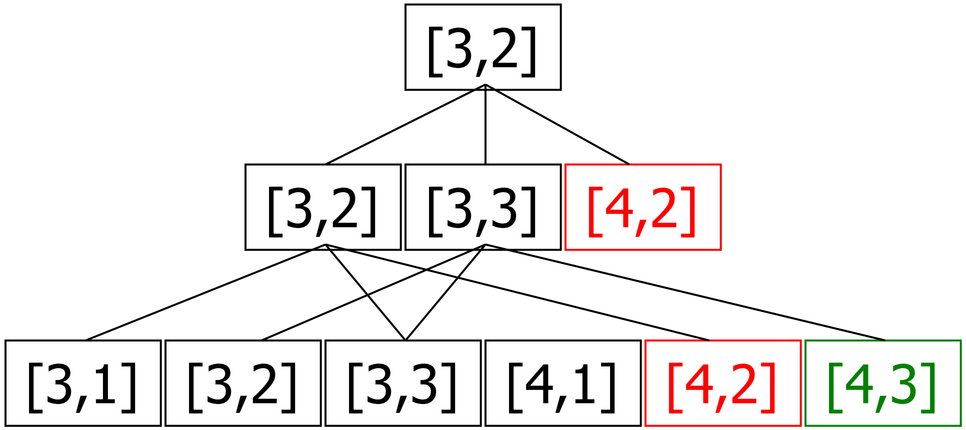



- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories
- The optimal sequence is the one with maximal utility
- **But is the optimal action sequence what we want to compute?**

# Optimal Action Sequence




- Consider the action sequence (U,R) from [3,2]
- A run produc… …ability
- The utility of the sequence is the expected utility of the histories
- The optimal sequence is the one with maximal utility
- **But is the optimal action sequence what we want to compute?**

only if the sequence is executed blindly!