Machine Learning I
MATH80629A

Apprentissage Automatique I
MATH80629

Sequential Decision Making I
— Week #12

Today

- Motivation and introduction
  - Toward Reinforcement learning
- Planning
  - Markov Decision Process (MDP)
    - Value iteration
    - Policy iteration
- Next week: Reinforcement learning

Reinforcement Learning
Motivation

Recall

Three main components

- Task (T)
- Performance measure (P)
- Experience (E)

Supervised learning

- Experience a fixed data set
- Fit a model using this data
- Use the model to make predictions about unseen data (and to understand the data)
- Predictions may be used downstream to inform decision-making (e.g., Operations Research)

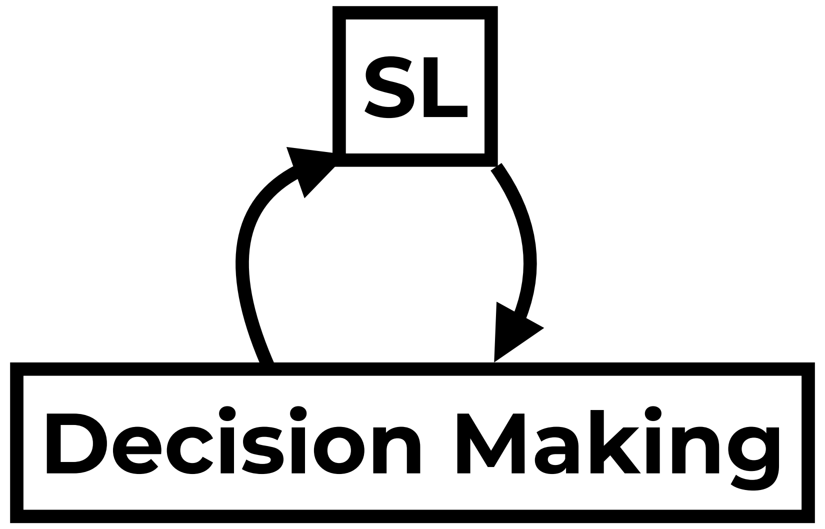
An example of learning and decision making

An example of learning and decision making

- Imagine building a robot that must navigate autonomously
  - The robot has wheels and a camera

An example of learning and decision making

- Imagine building a robot that must navigate autonomously
  - The robot has wheels and a camera
- You think about using a two-stage approach:
  1. Use supervised learning to identify objects in scenes
  2. Given scene content have a decision-making module that controls its wheels

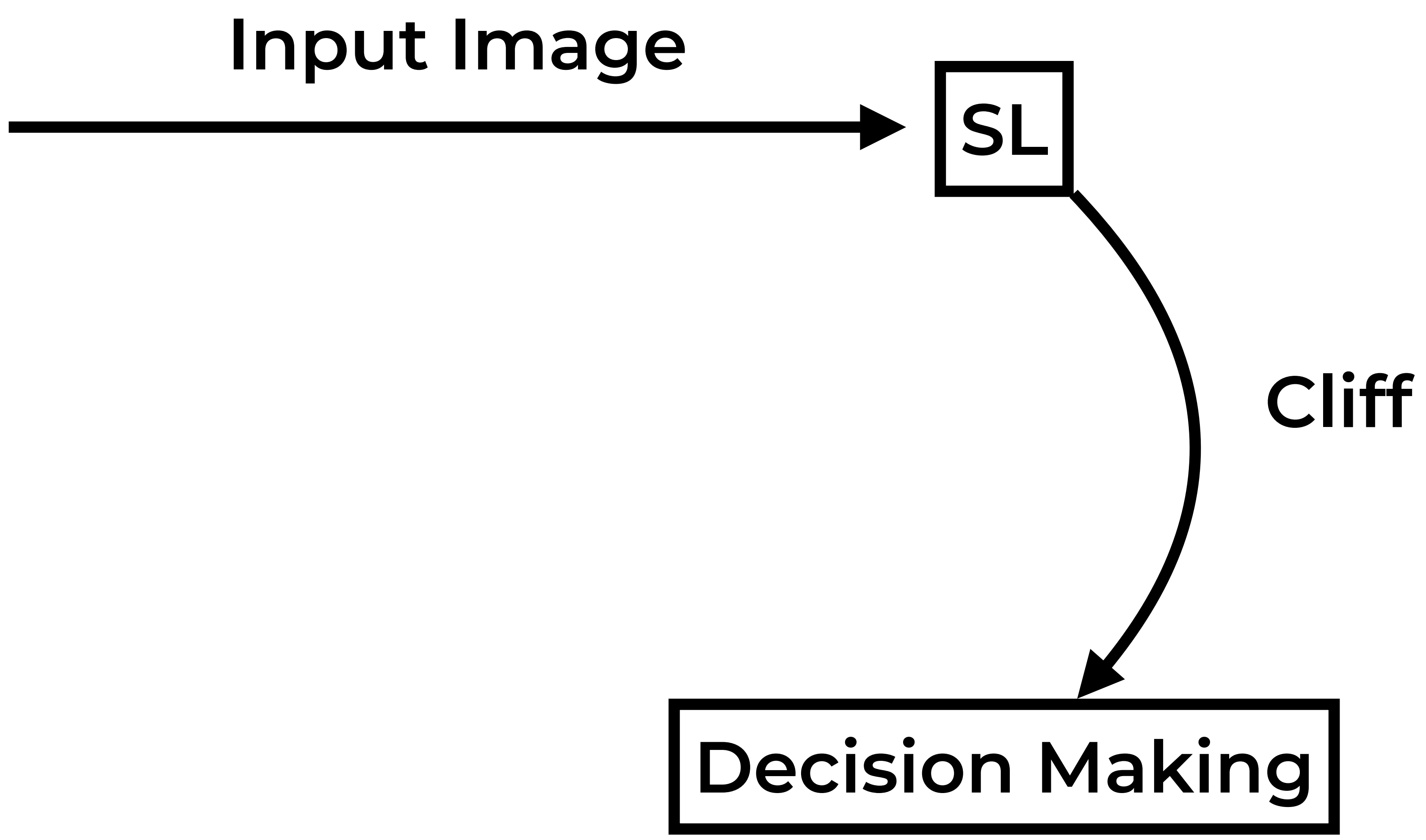


View from the robot's camera

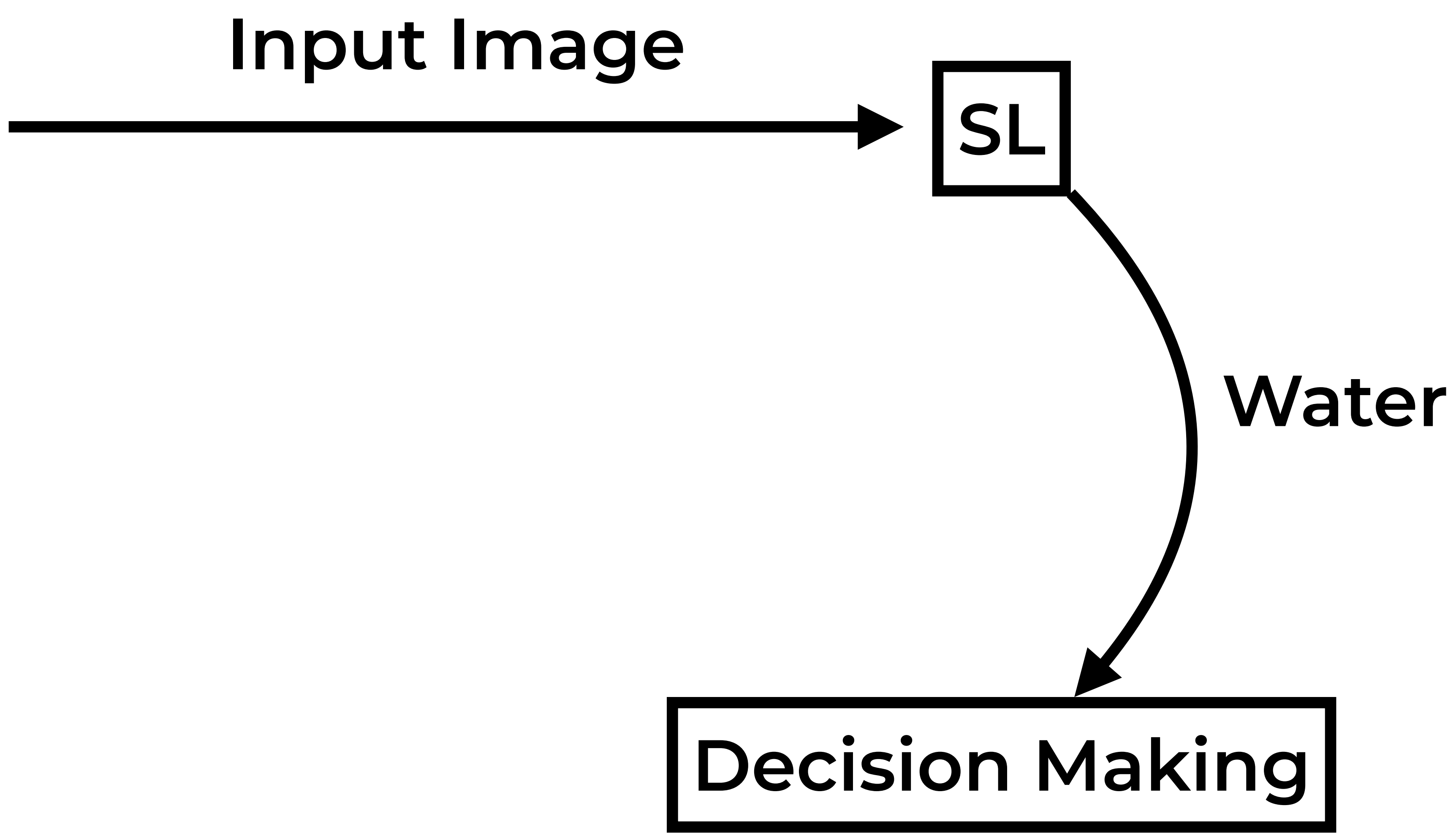
View from the robot's camera

Input Image

View from the robot’s camera



View from the robot’s camera



Limitations of two-stage approach

- Supervised learning doesn't know about the decision-making
  - Its objective is, for example, to maximize accuracy

Limitations of two-stage approach

- Supervised learning doesn't know about the decision-making
  - Its objective is, for example, to maximize accuracy
- For decision making, different errors have different costs
  - E.g., missing the cliff could have dire consequences. missing sky less so.
  - Incorporating these costs into the learning objective is tough

Limitations of two-stage approach

- Supervised learning doesn't know about the decision-making
  - Its objective is, for example, to maximize accuracy
- For decision making, different errors have different costs
  - E.g., missing the cliff could have dire consequences. missing sky less so.
  - Incorporating these costs into the learning objective is tough
- Several other limitations:
  - need labeled data
  - improvements in SL do not necessarily lead to improvements in decision making
  - ...

Alternative: Reinforcement learning (RL)

- Incorporates both stages in a single framework
- Incorporates the ideas of:
  - state (observation)
  - action
  - reward

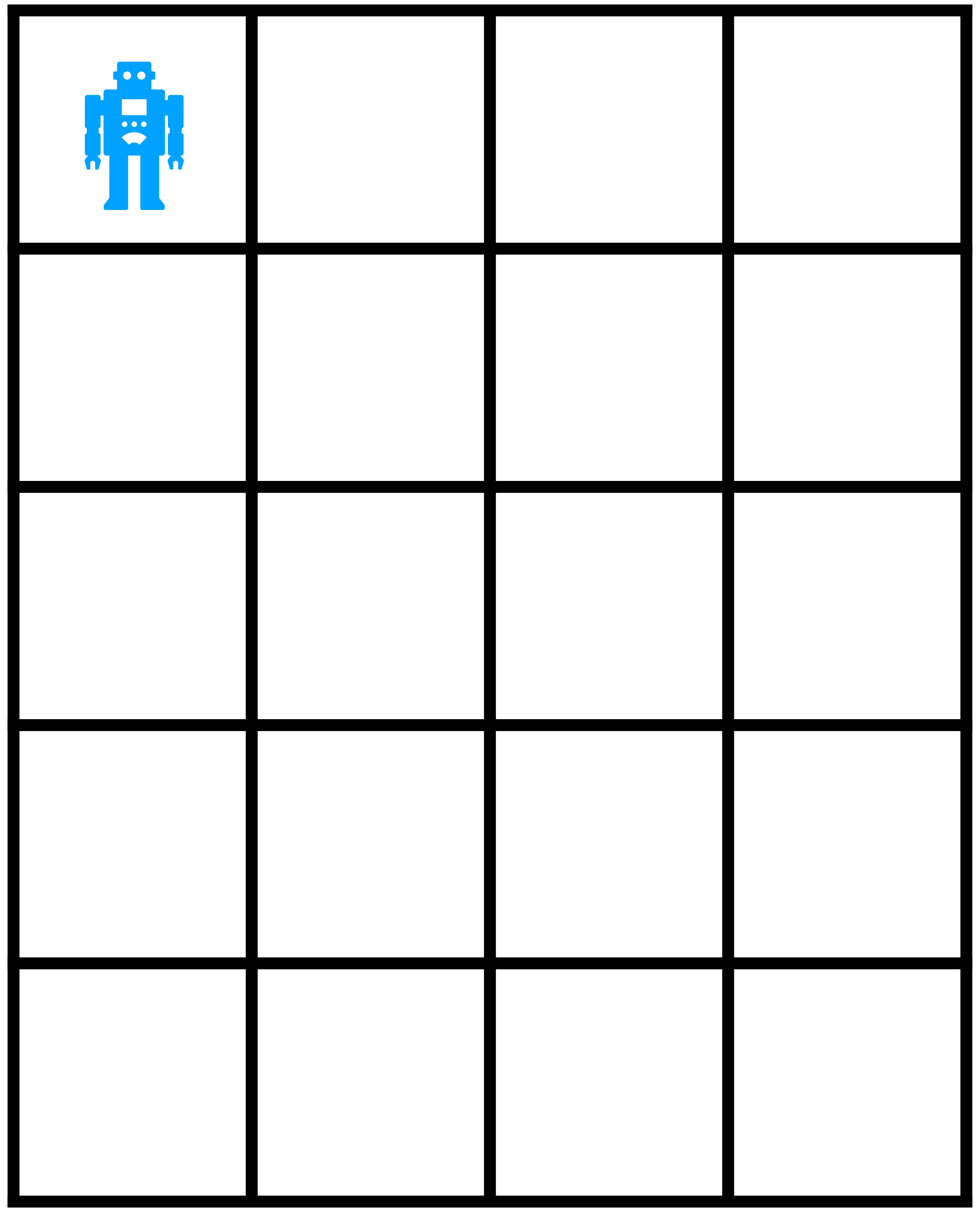
Planning:
A first step towards reinforcement learning

Alternative: Reinforcement learning (RL)

Recall

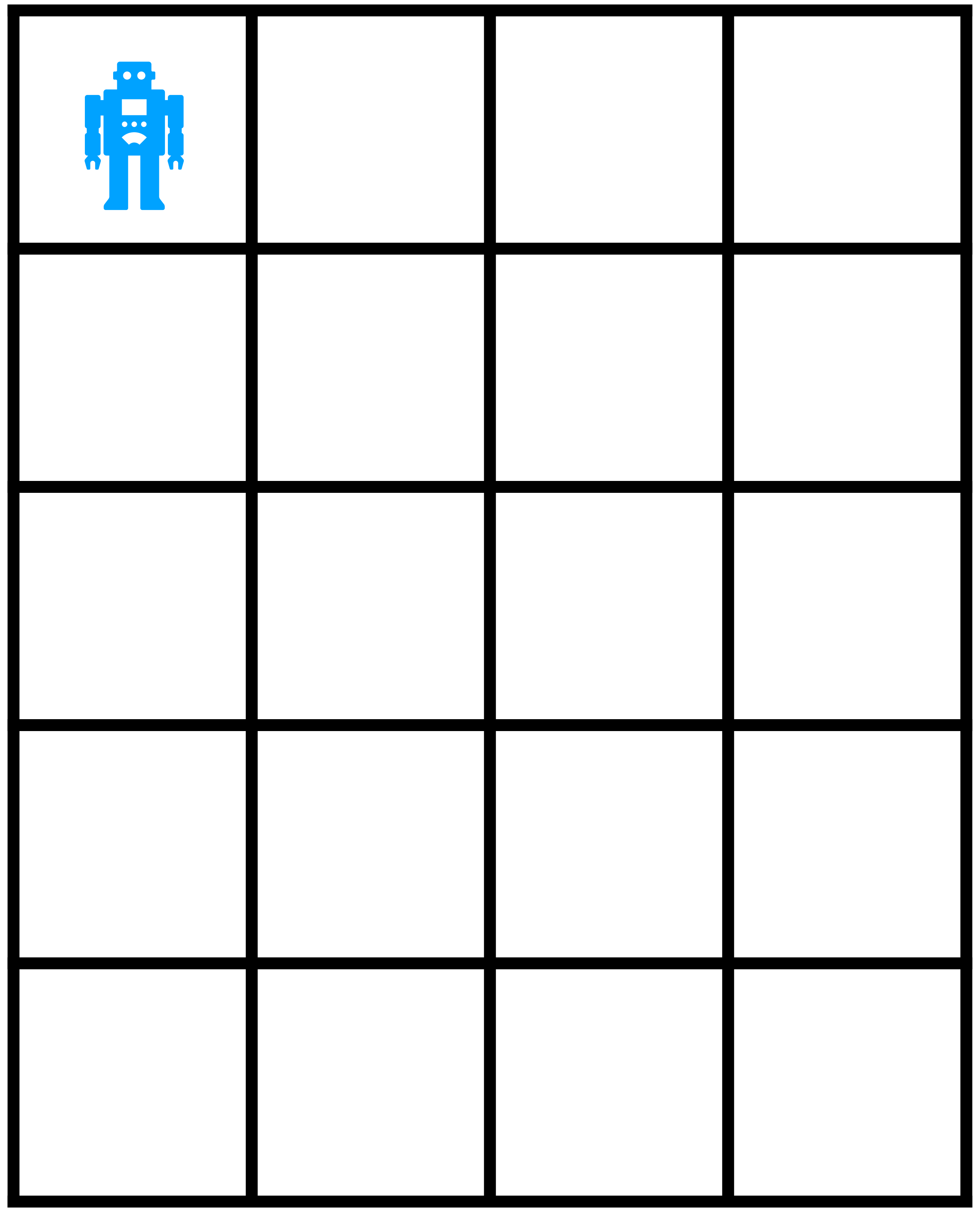
- Incorporates both stages in a single framework
- Incorporates the ideas of:
  - state (observation)
  - action
  - reward

Initial example with grid world



- Each cell is a state (S)
- Actions indicate which movements are possible: $A := \{L, R, U, D\}$
- Rewards encode the task: $R(s)$
- Transition probabilities encode the outcome of an action: $P(s' \mid s, a)$

Initial example with grid world



- Each cell is a state (S)
- Actions indicate which movements are possible: $A := \{L, R, U, D\}$
- Rewards encode the task: $R(s)$
- Transition probabilities encode the outcome of an action: $P(s' \mid s, a)$

Planning

This week we discuss a version of RL where these are observed

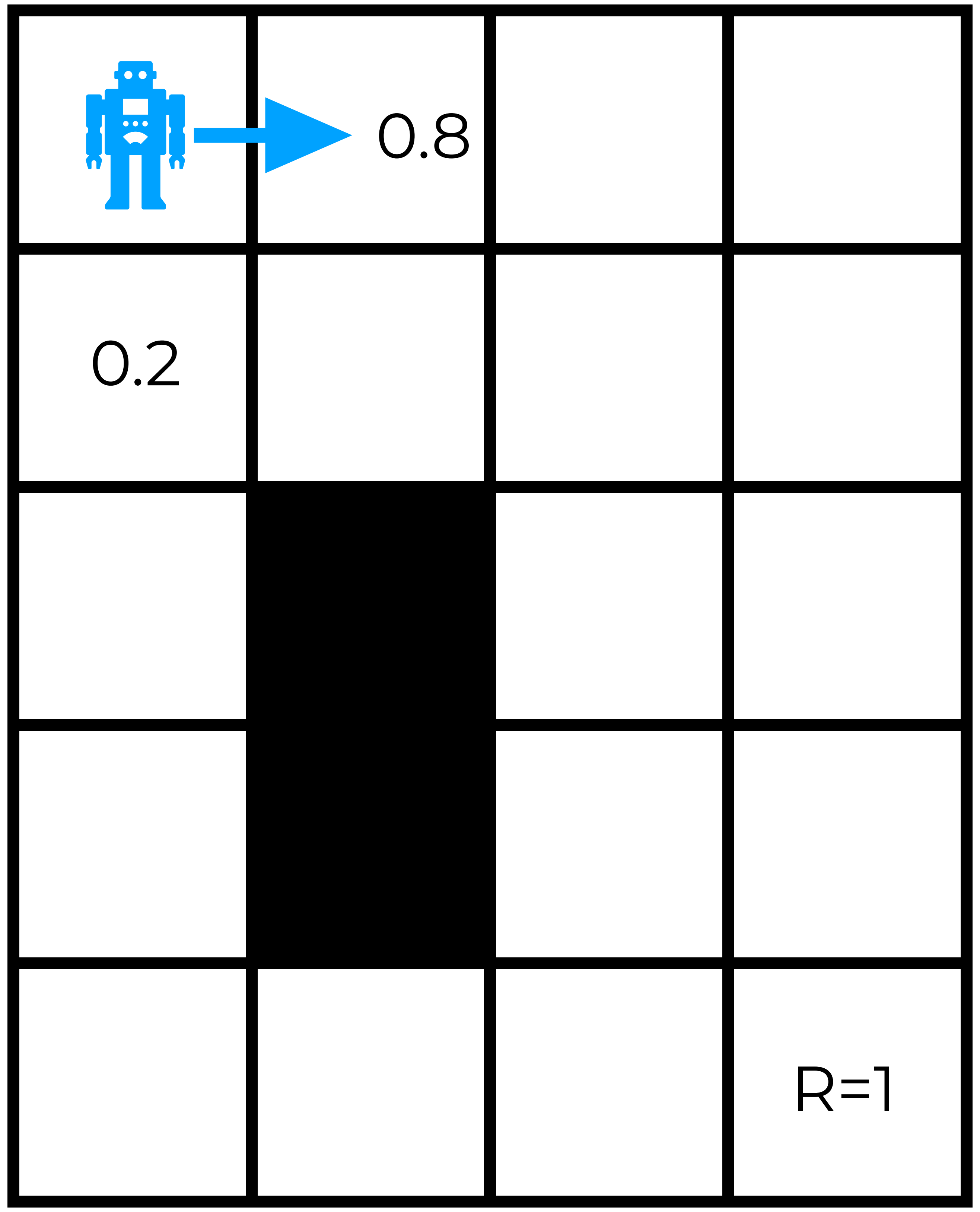
- 20 states. Start state is top-left
  - Bottom right is absorbing

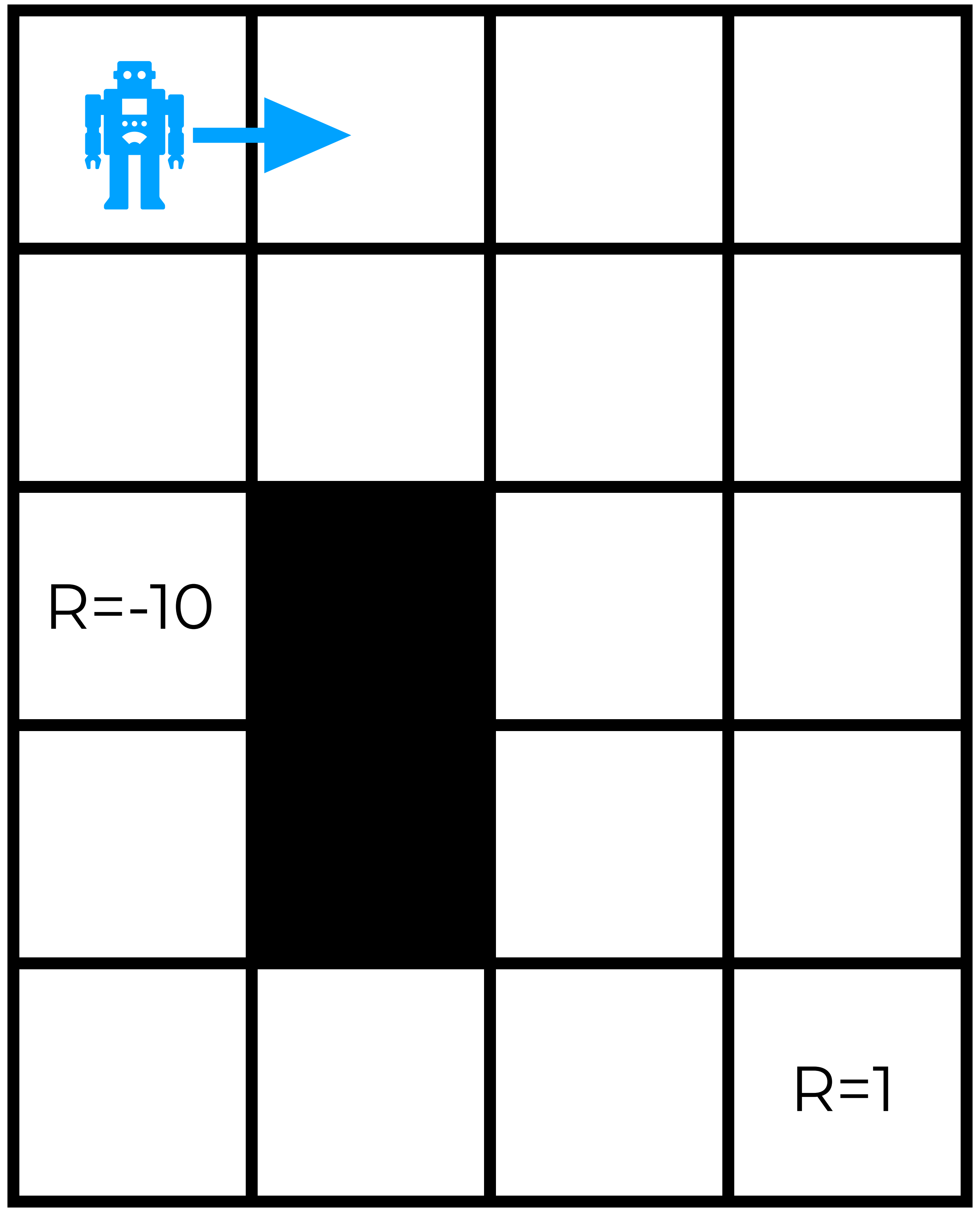
$$P(s'|s_{\text{absorbing}}, a) = \begin{cases} 1 & \text{if } s' = s, \\ 0 & \text{otherwise.} \end{cases}$$

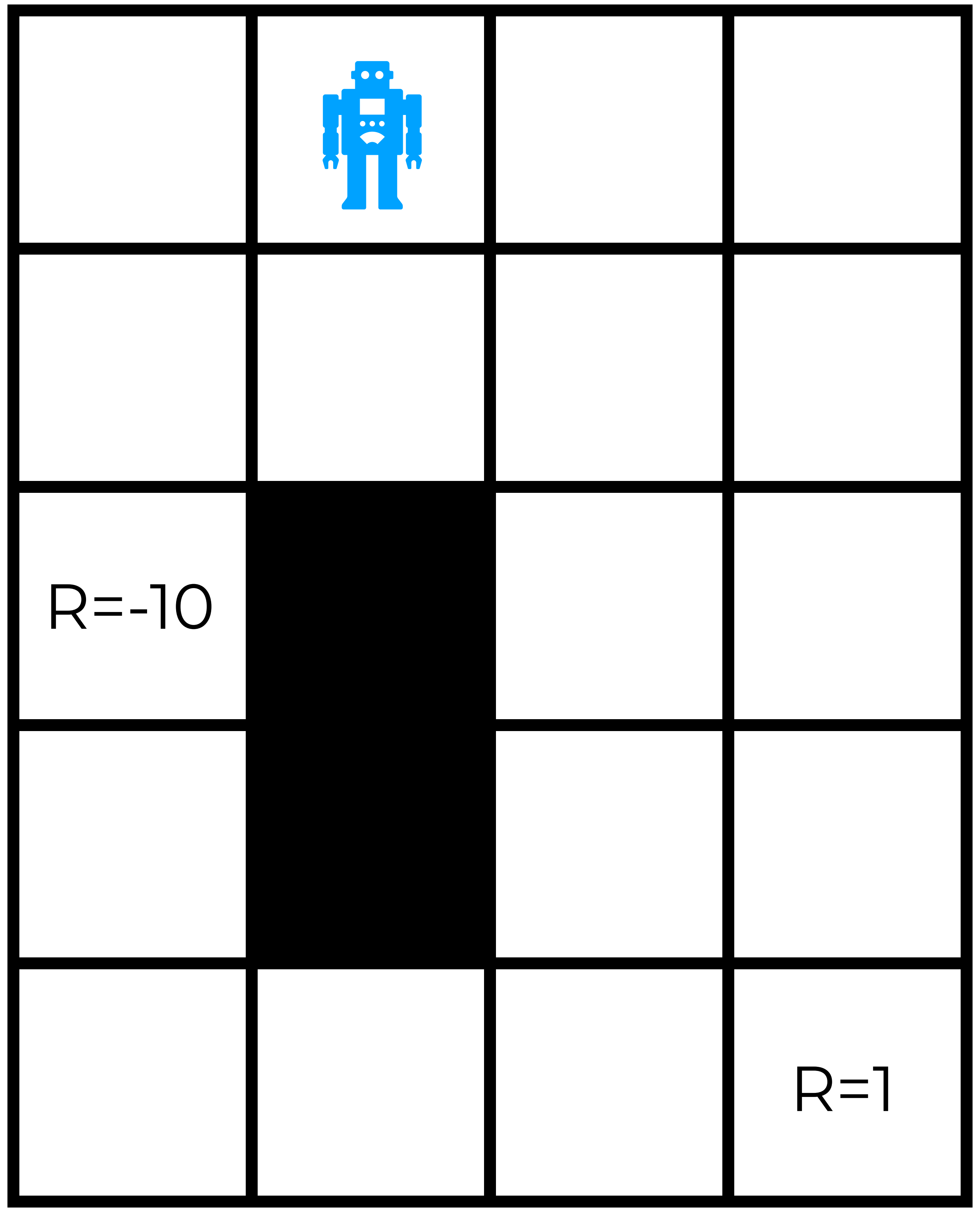
- All rewards are 0 except for the bottom-right state (goal state)

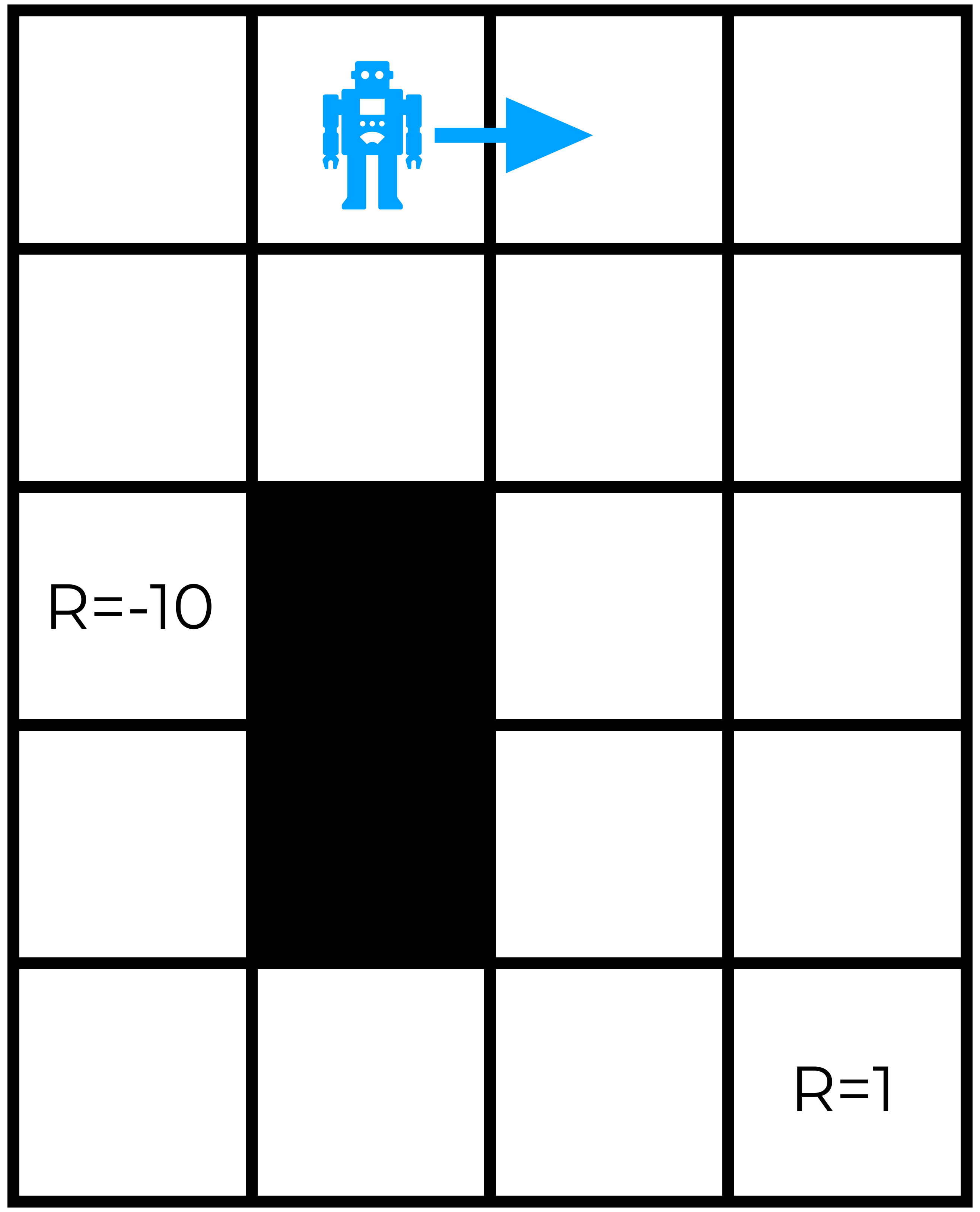
- Actions: $A := \{L, R, U, D\}$

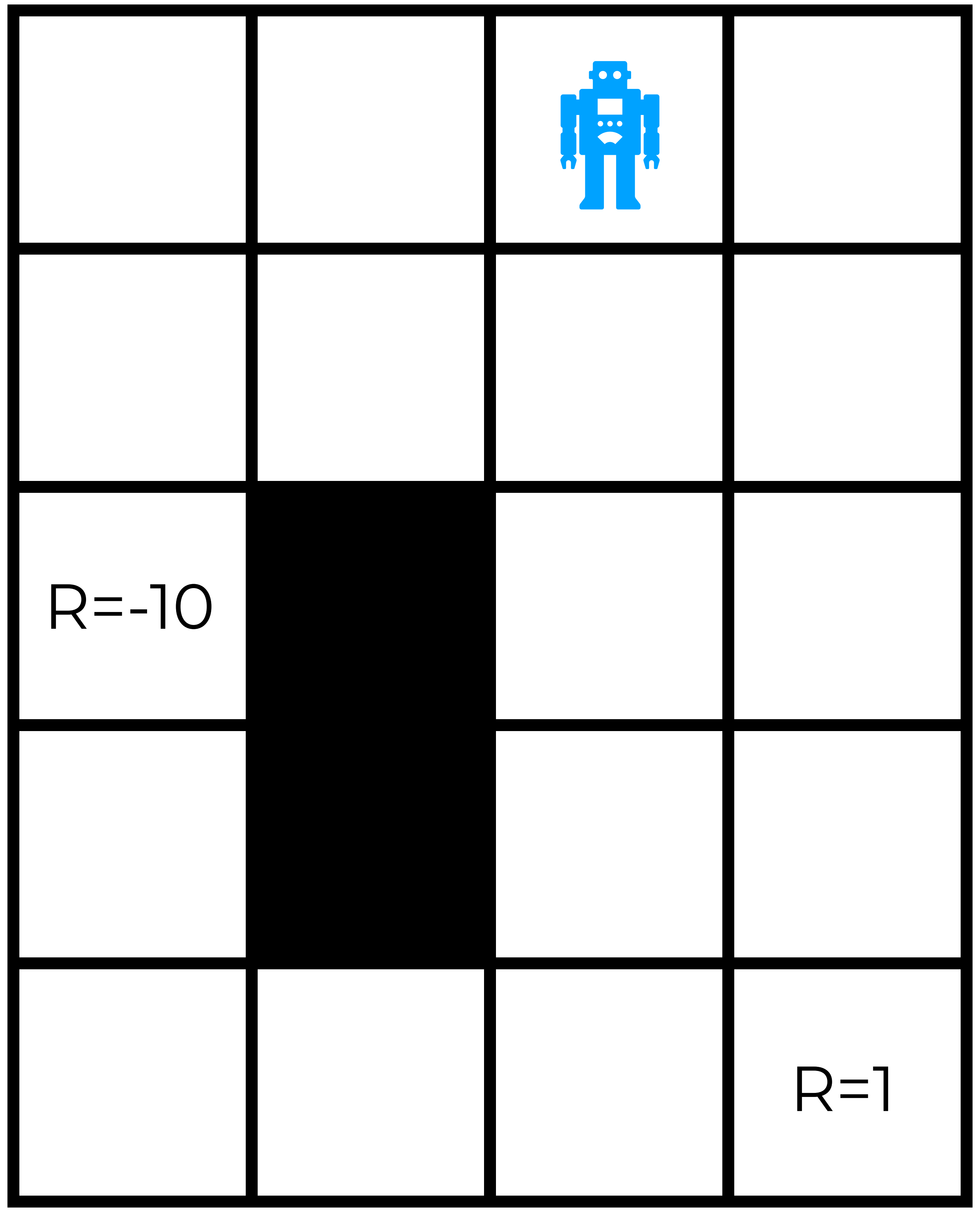
- 80% of the time actions lead to where they are supposed to.
  - The rest of the time (20%) they lead to a random adjacent state

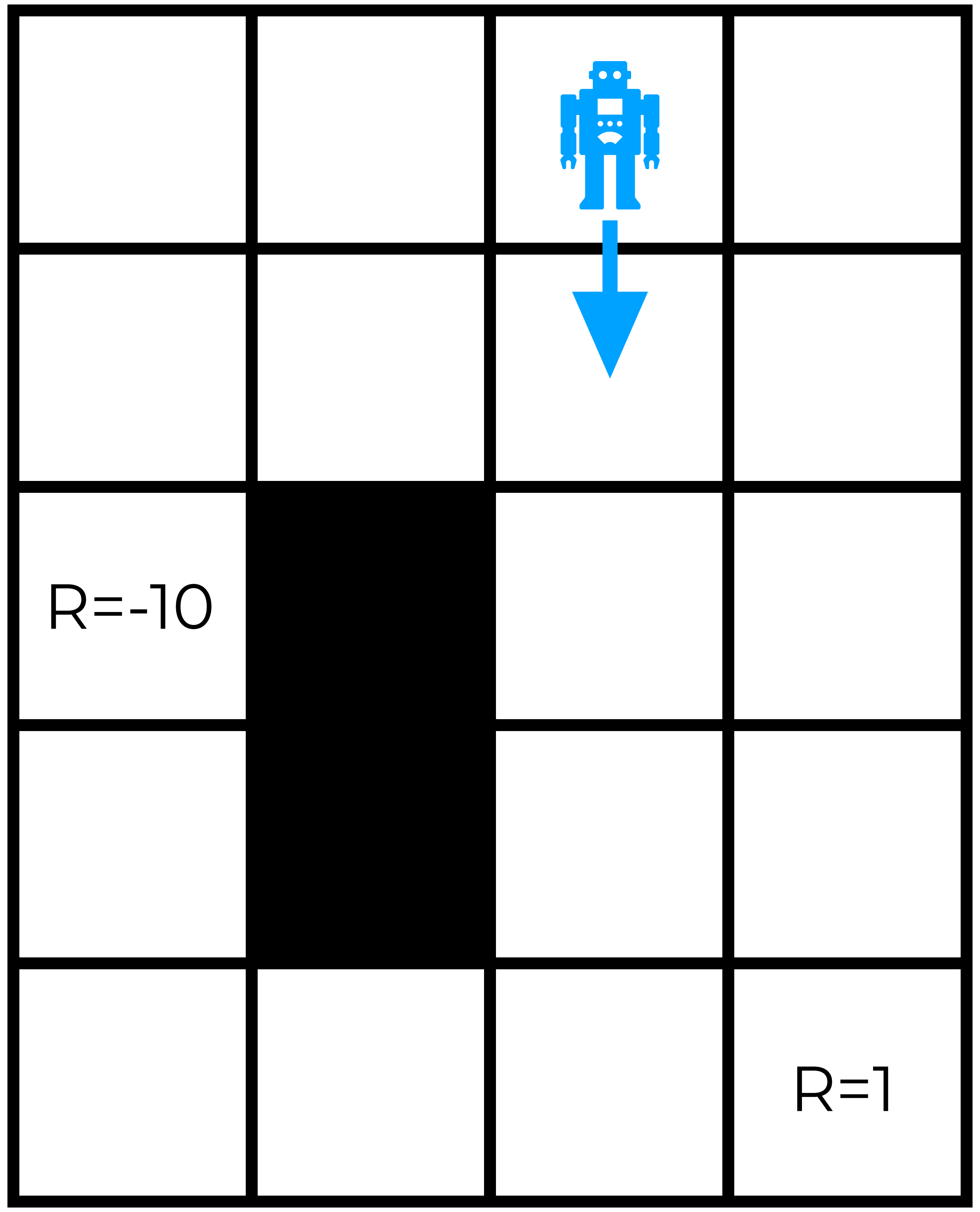


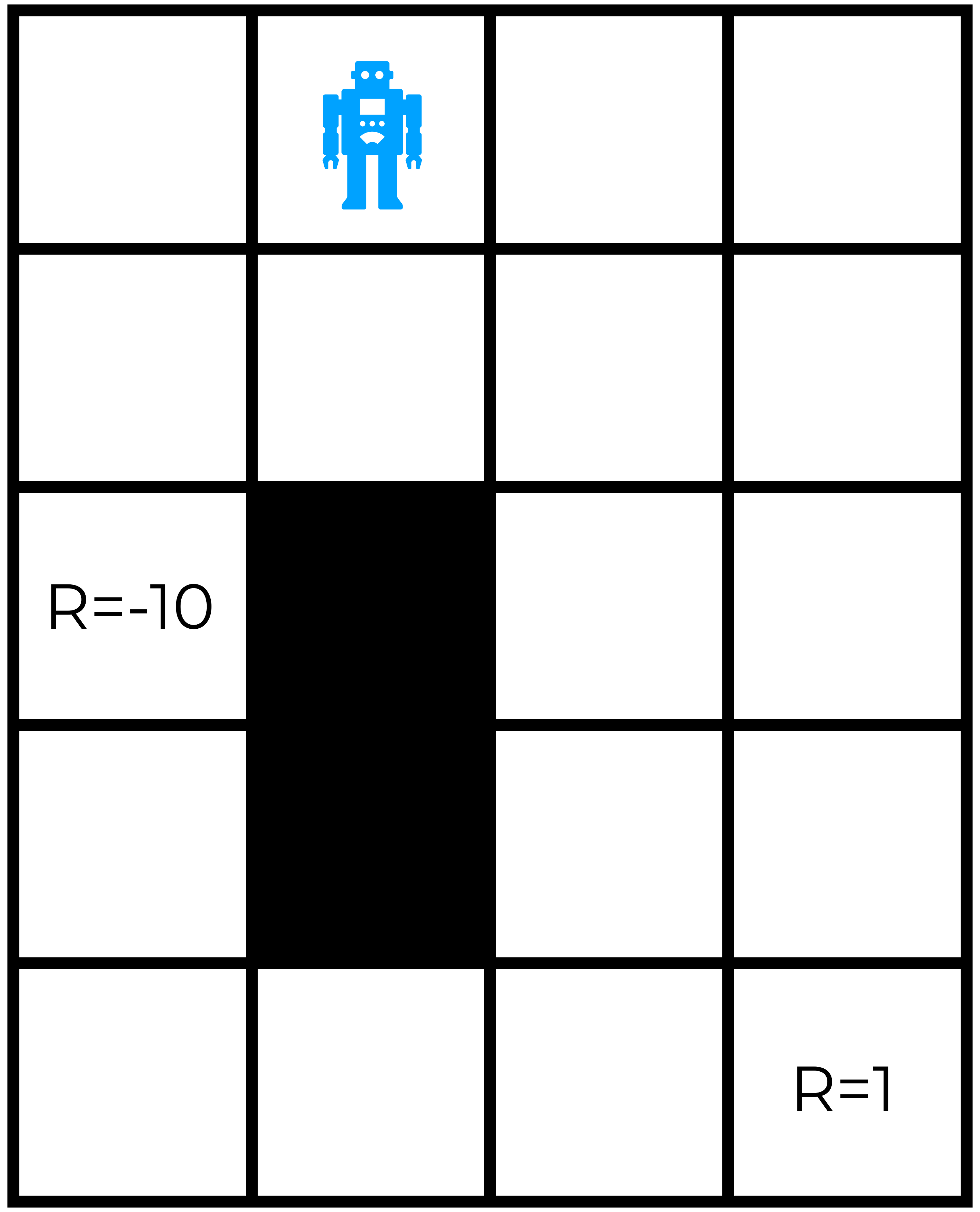

R=-10
R=1

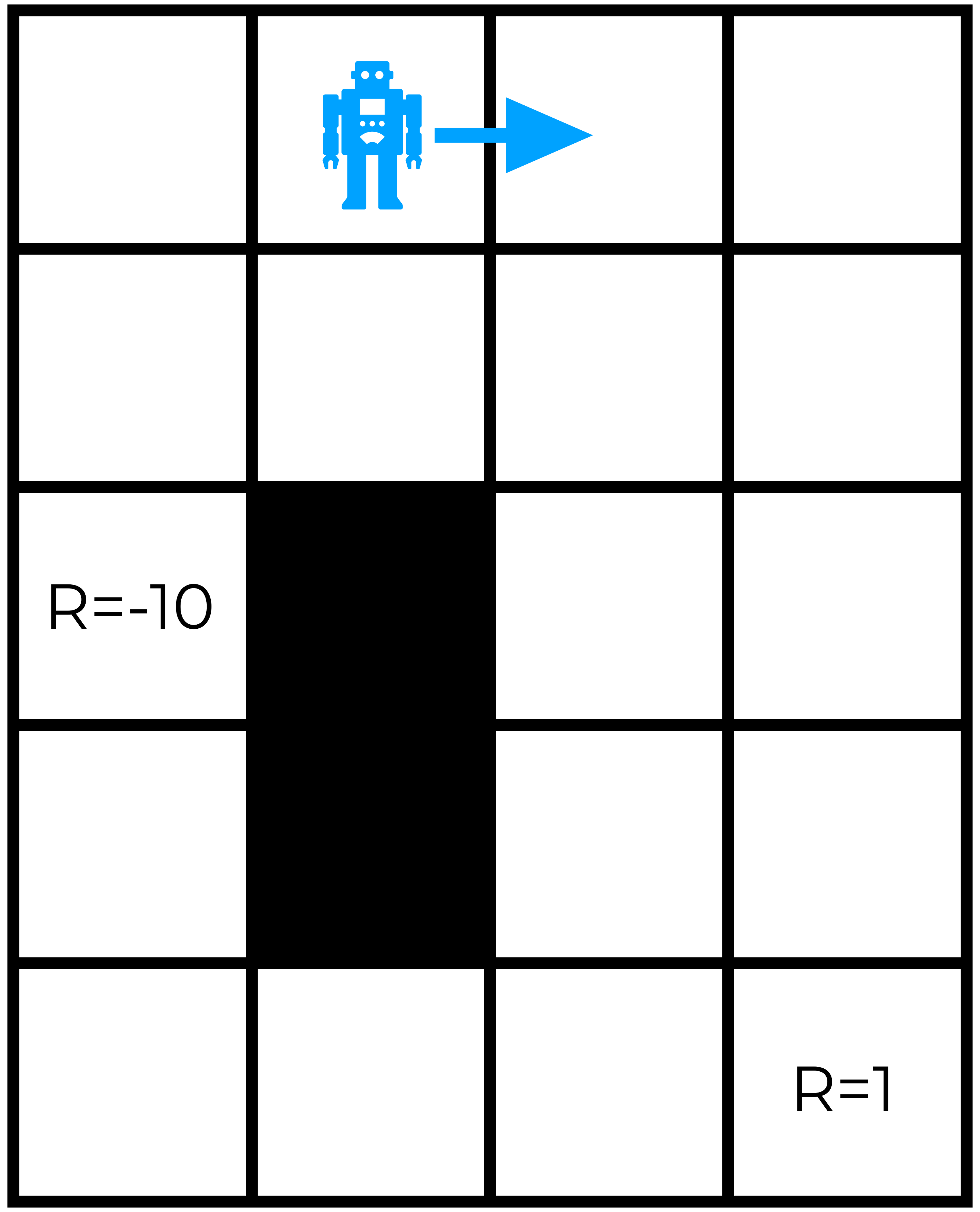

R=-10
R=1

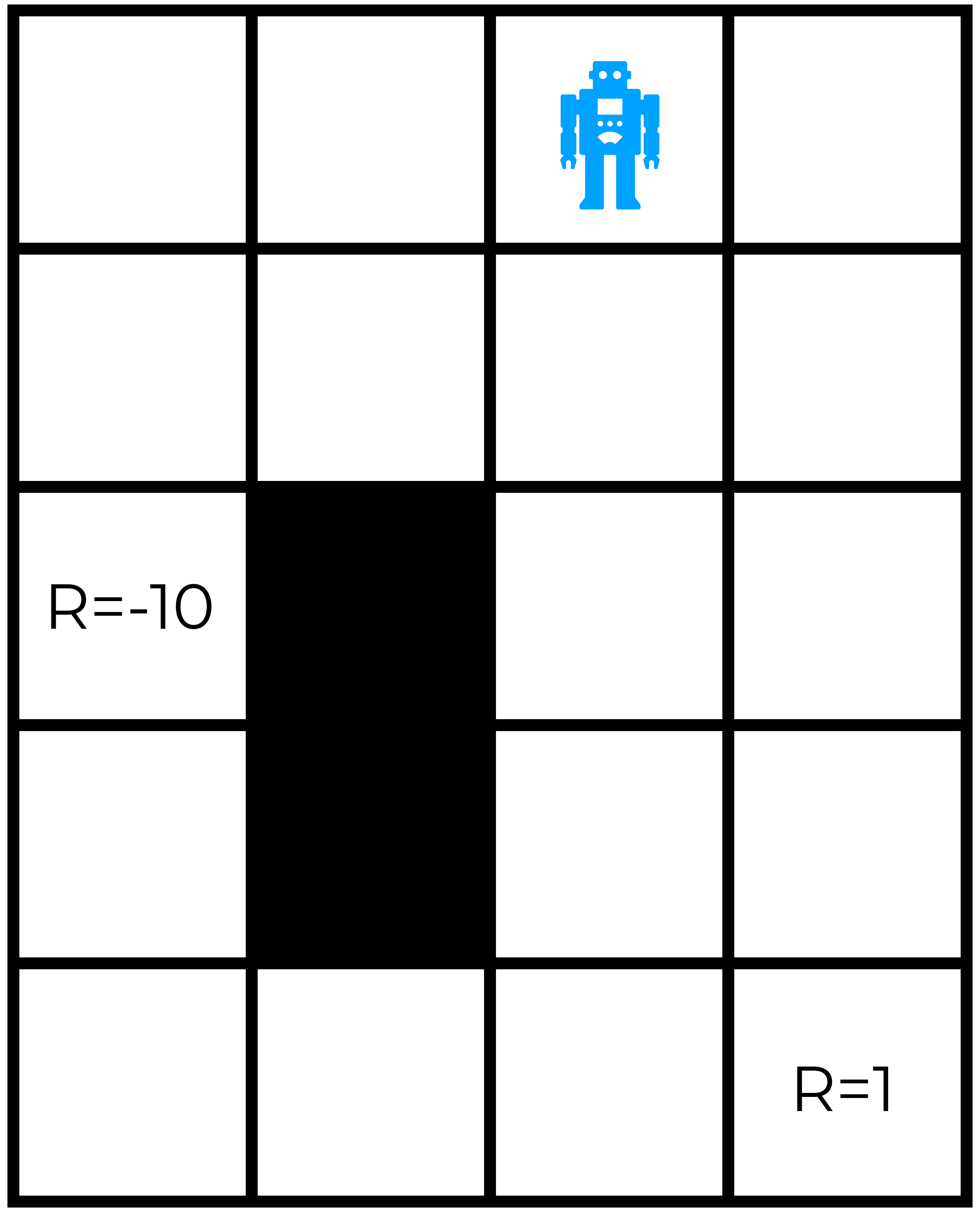

R=-10
R=1

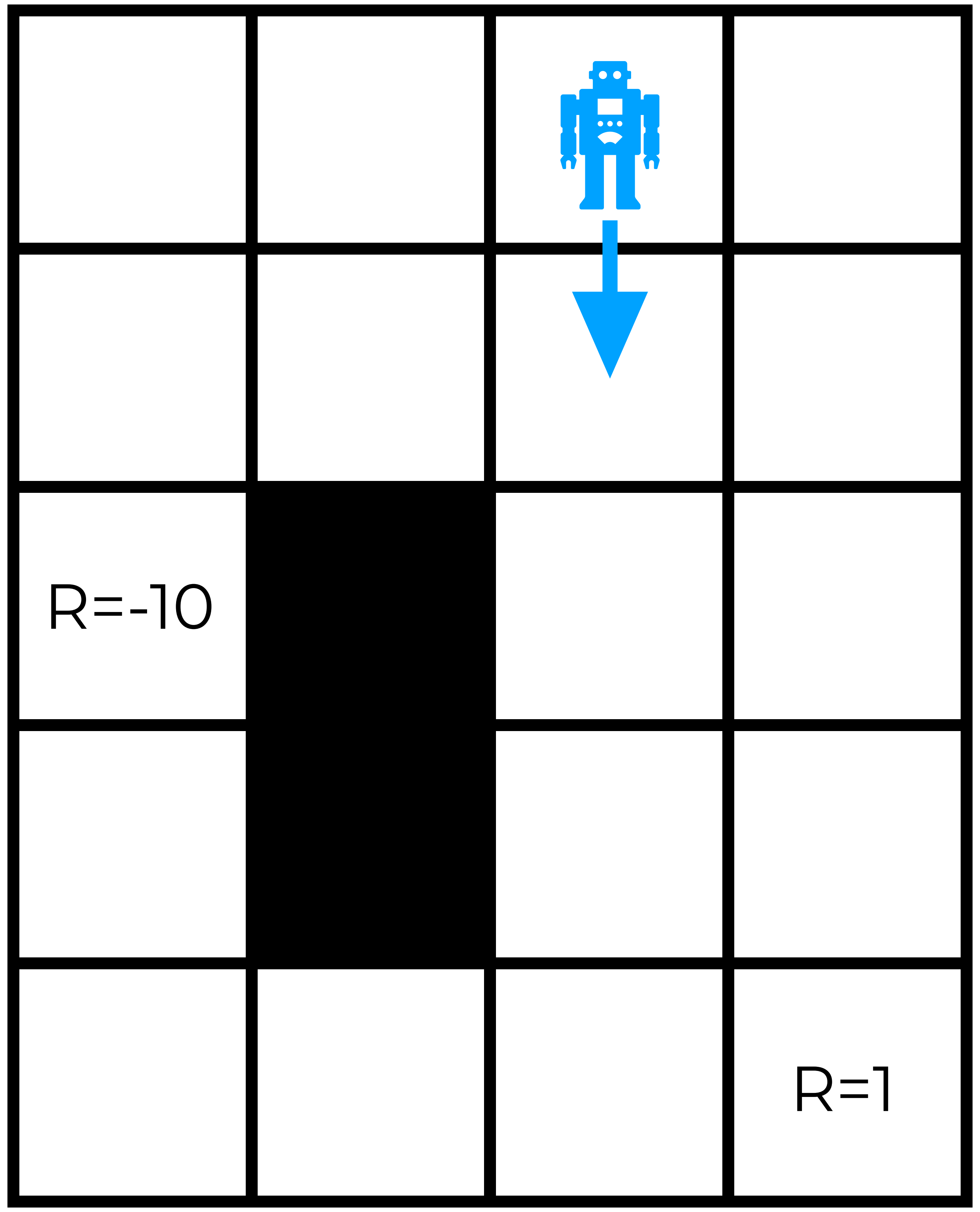

R=-10
R=1

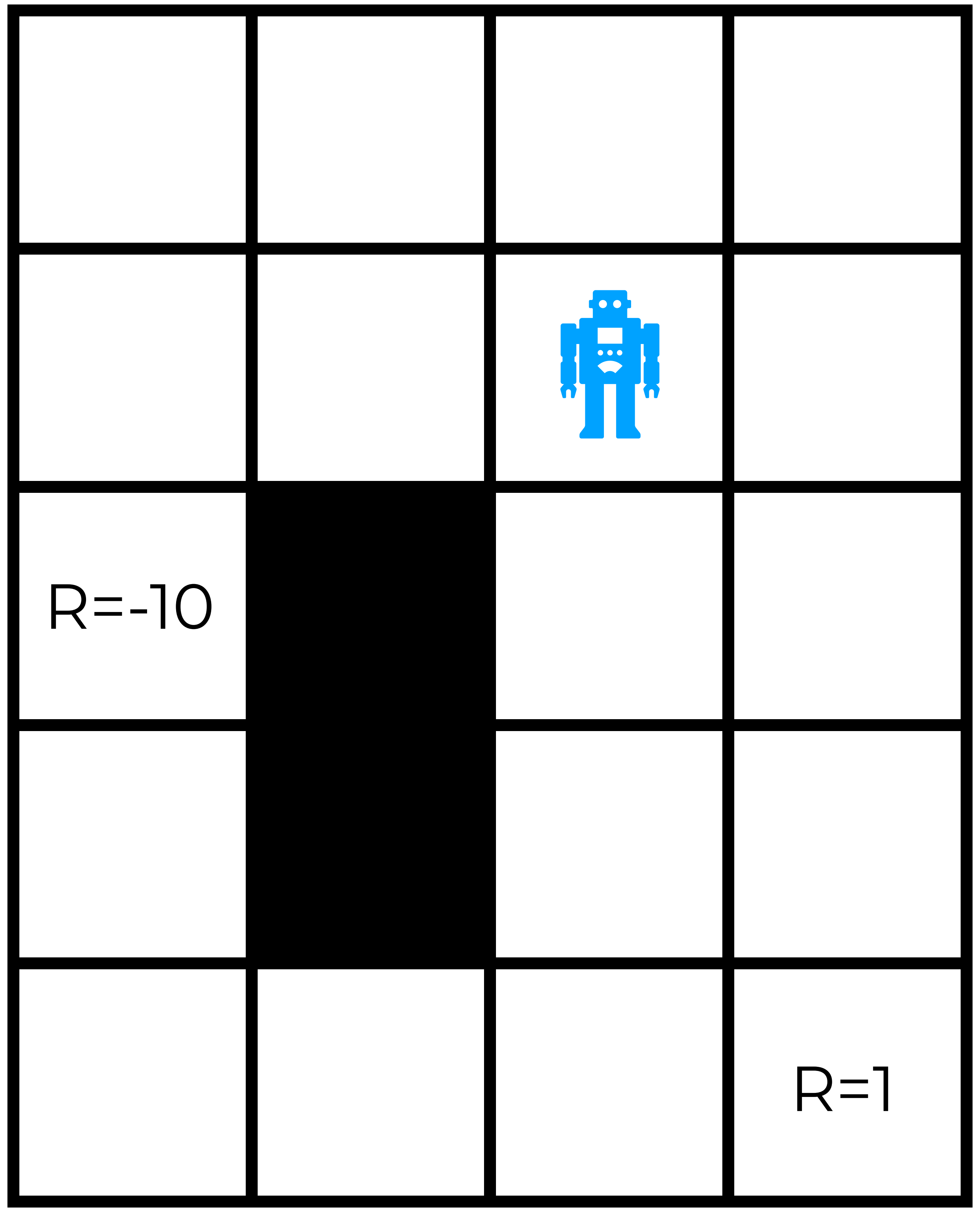

R=-10
R=1

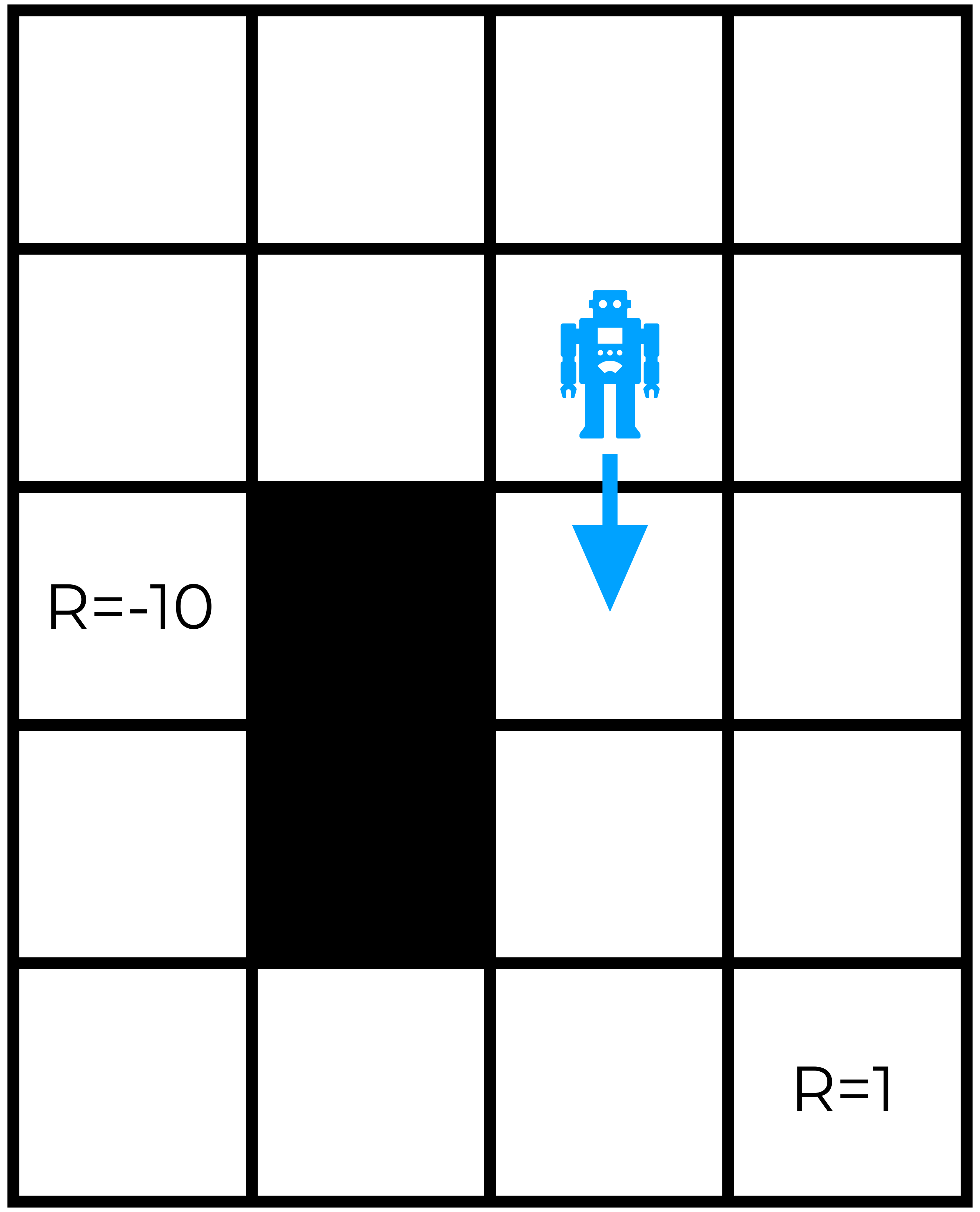

R=-10
R=1

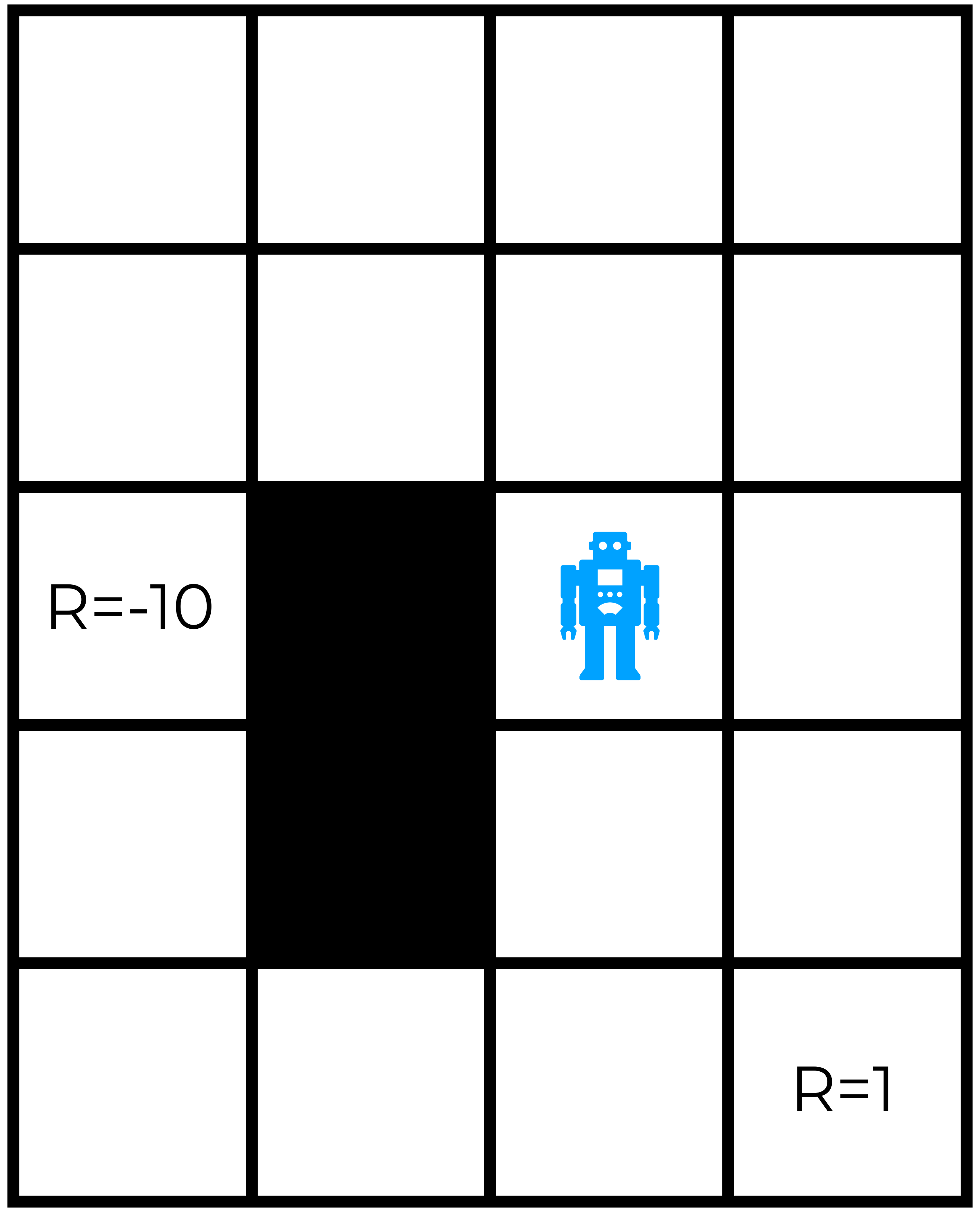

R=-10
R=1

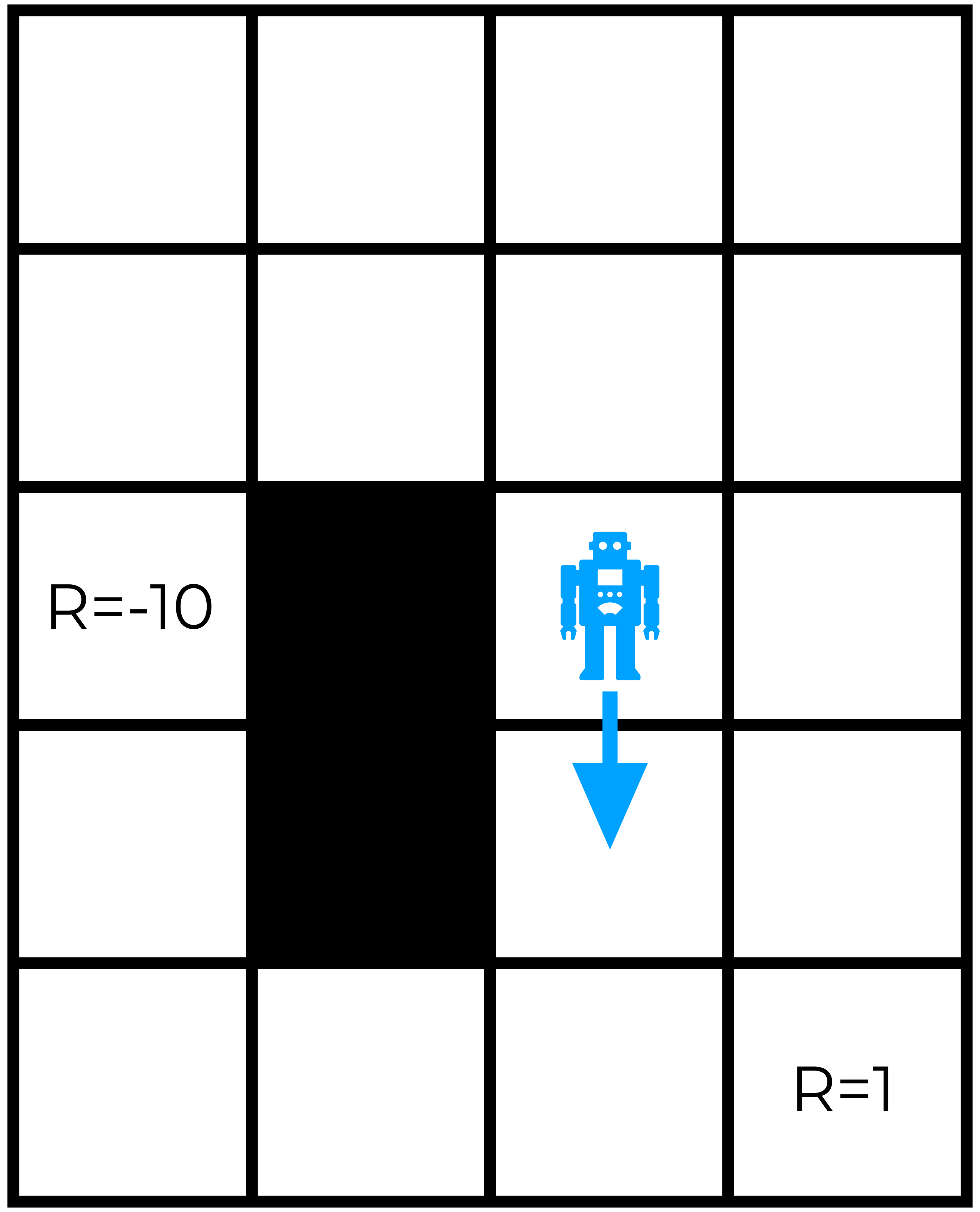

R=-10
R=1

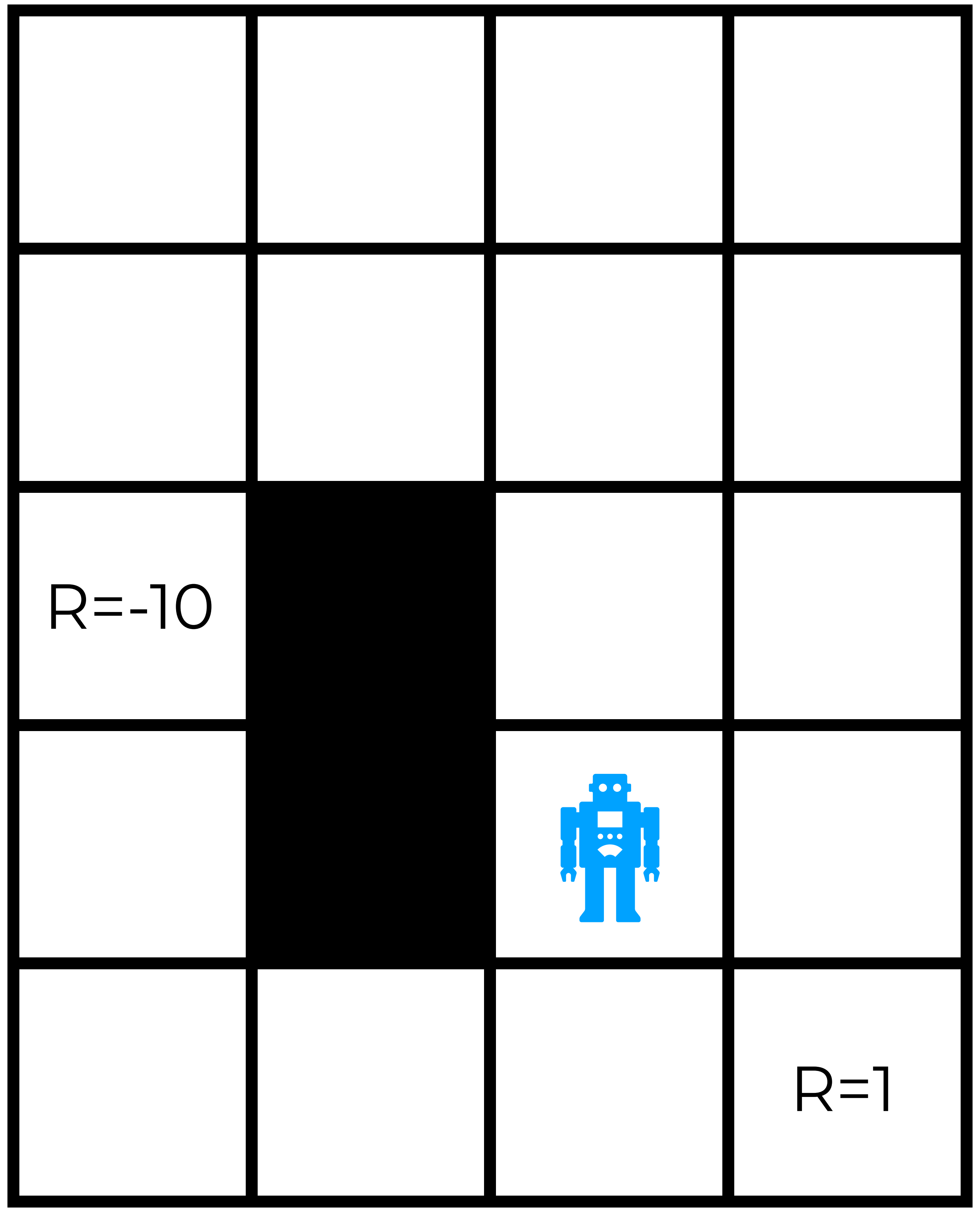

R=-10
R=1

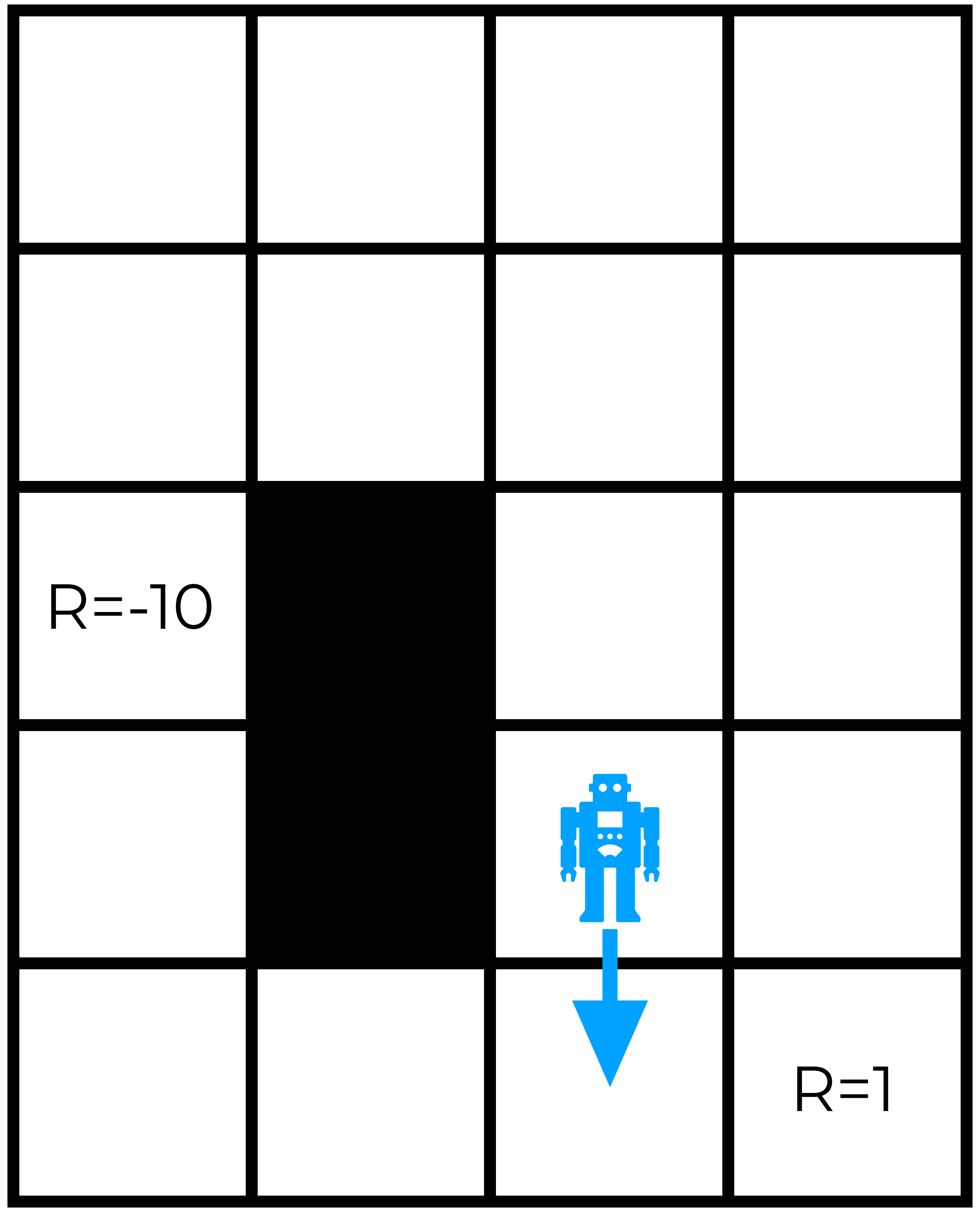

R=-10
R=1

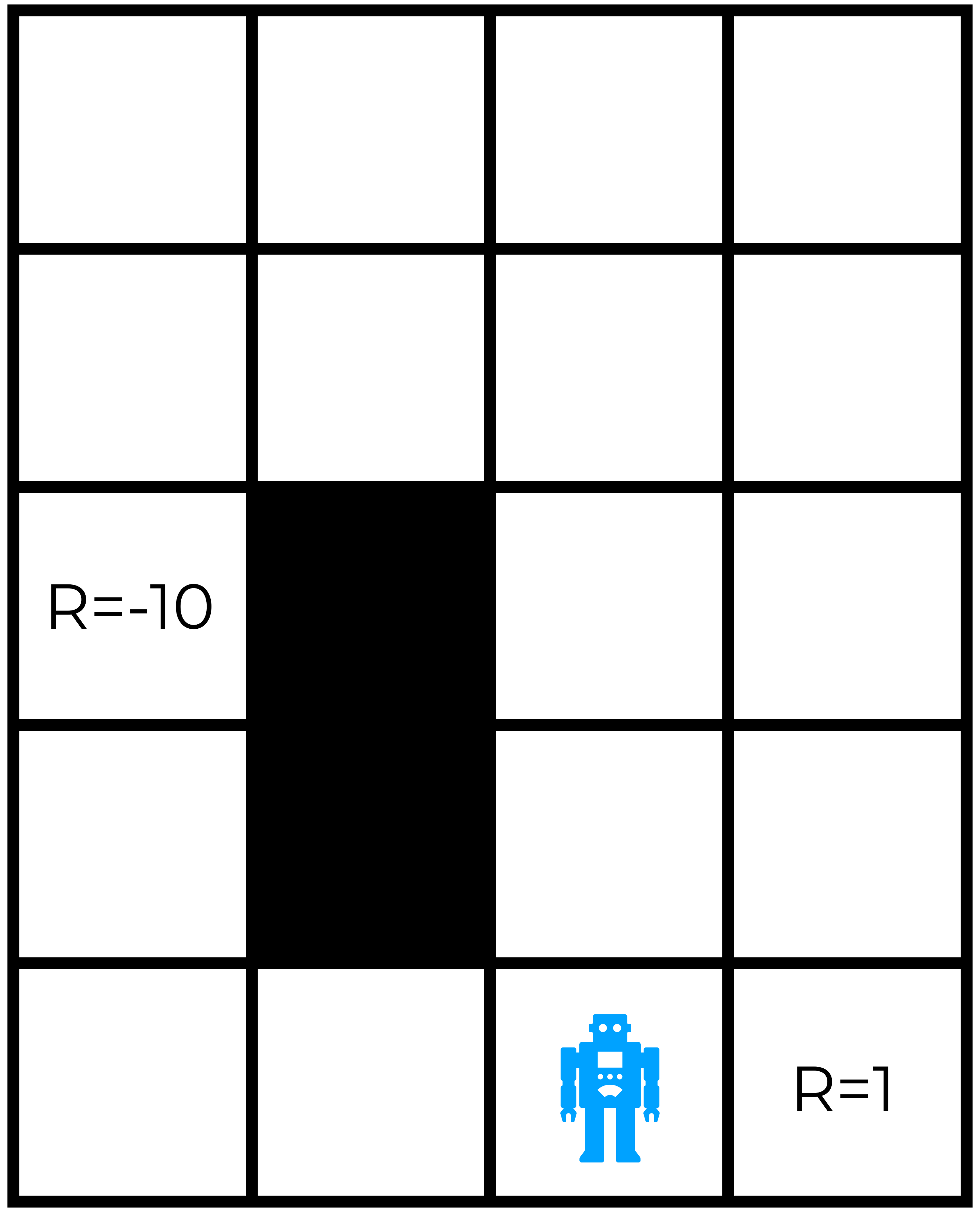

R=-10
R=1

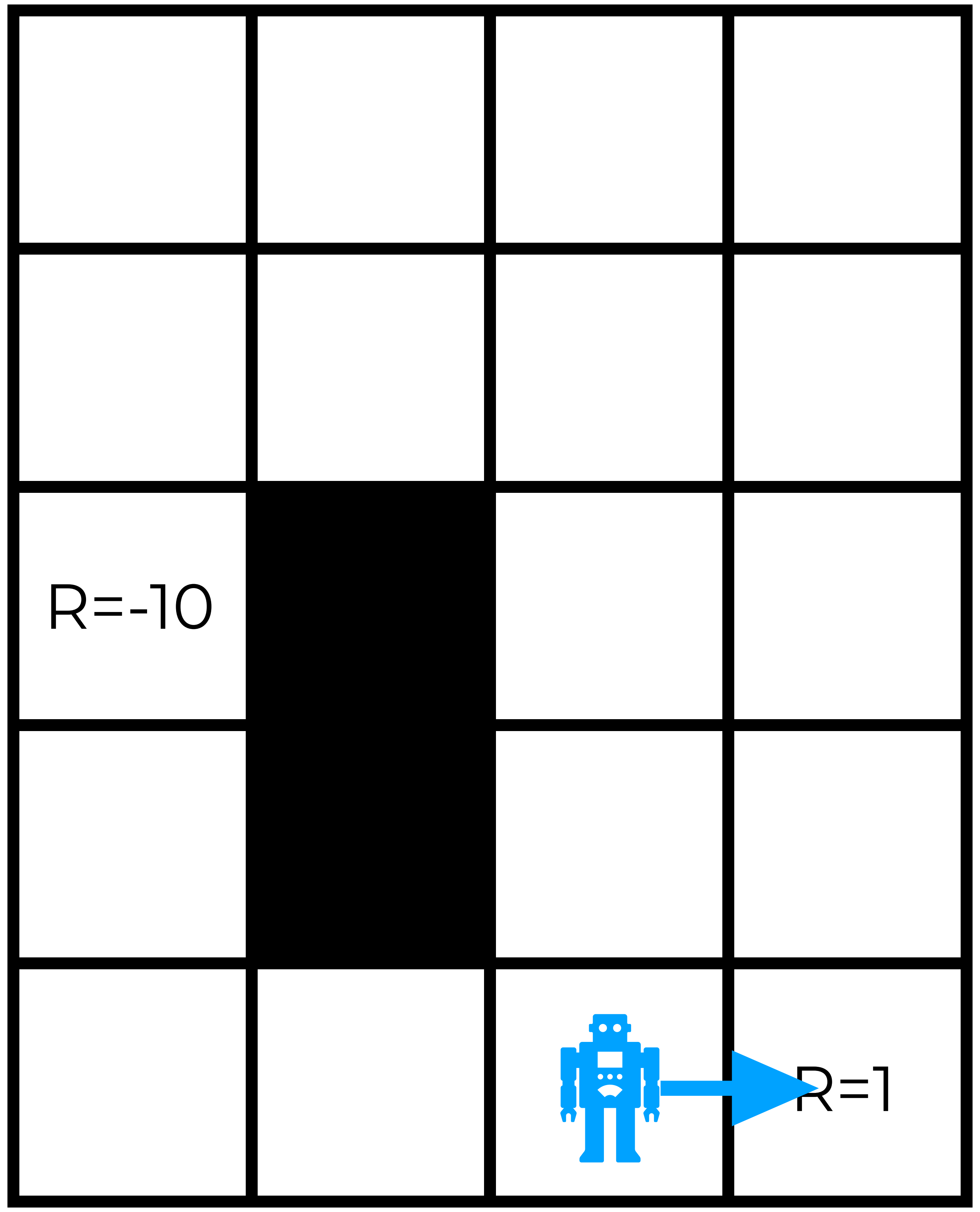

R=-10
R=1

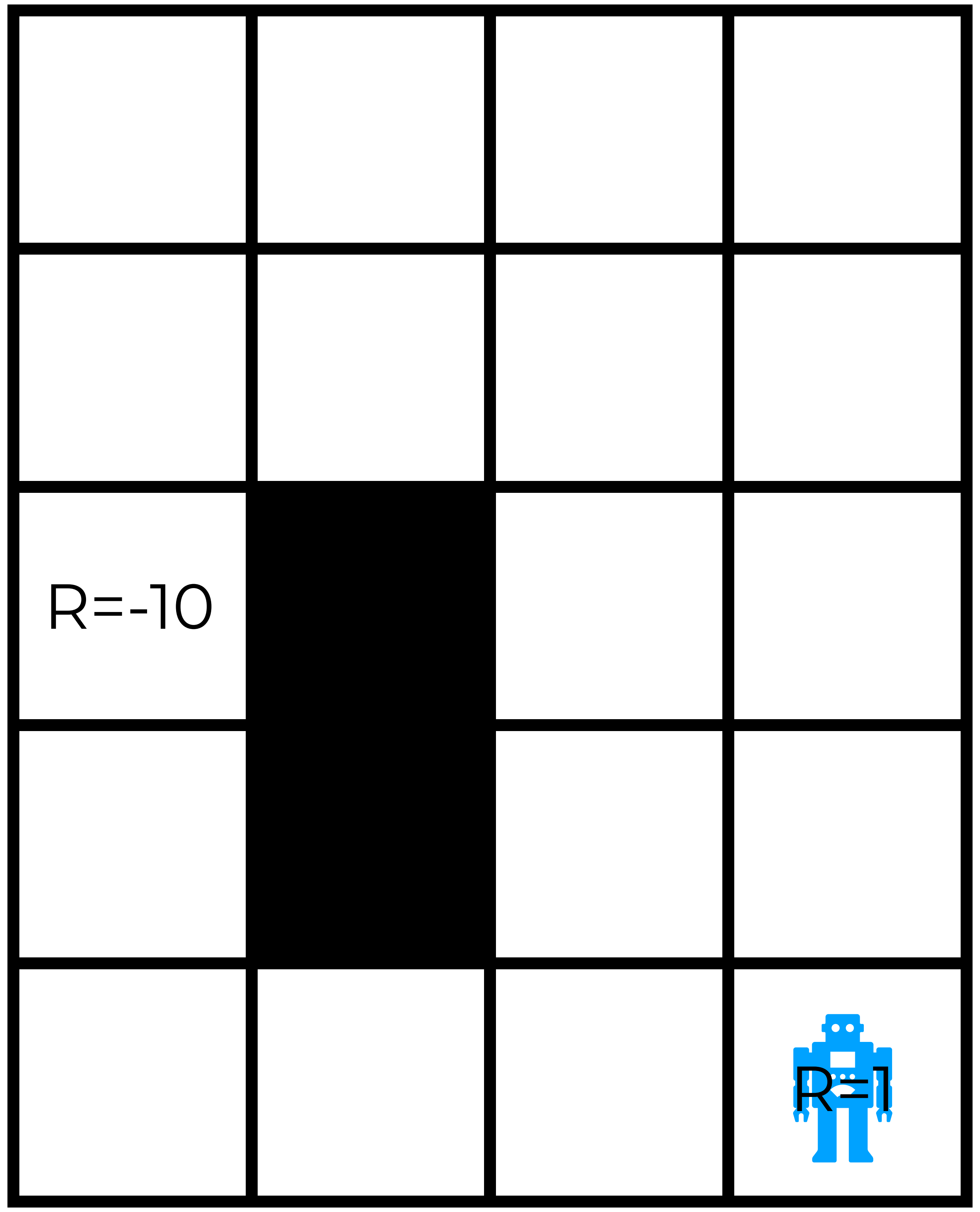

R=-10
R=1


R=-10
R=1


R=-10
R=1

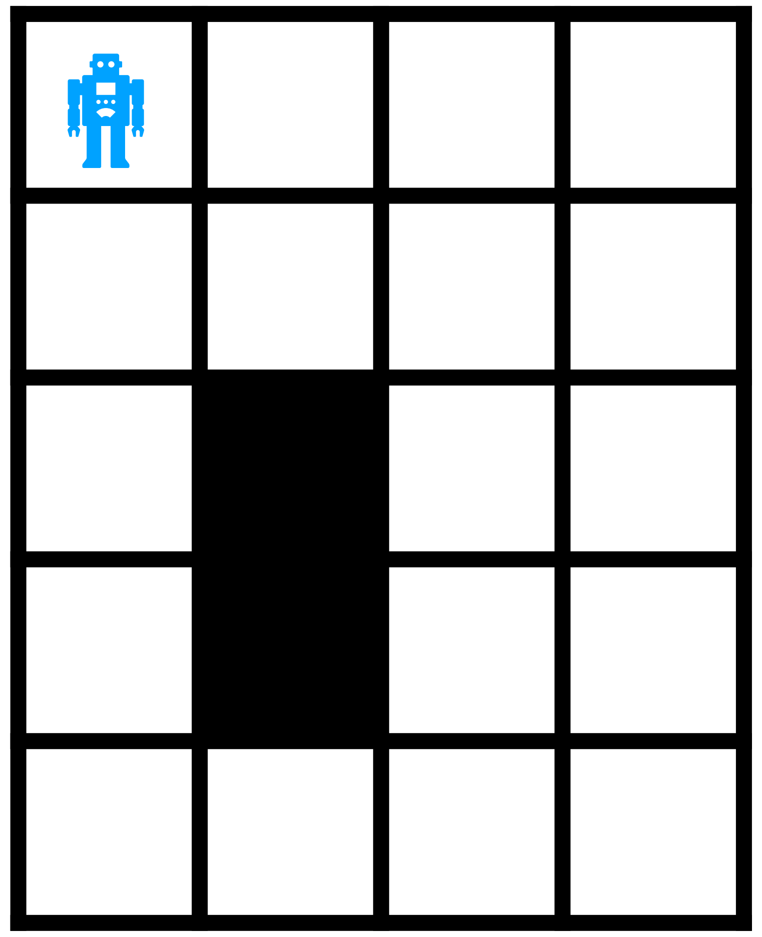

R=-10
R=1


R=-10
R=1


R=-10
R=1

Markov Decision Process (MDP)

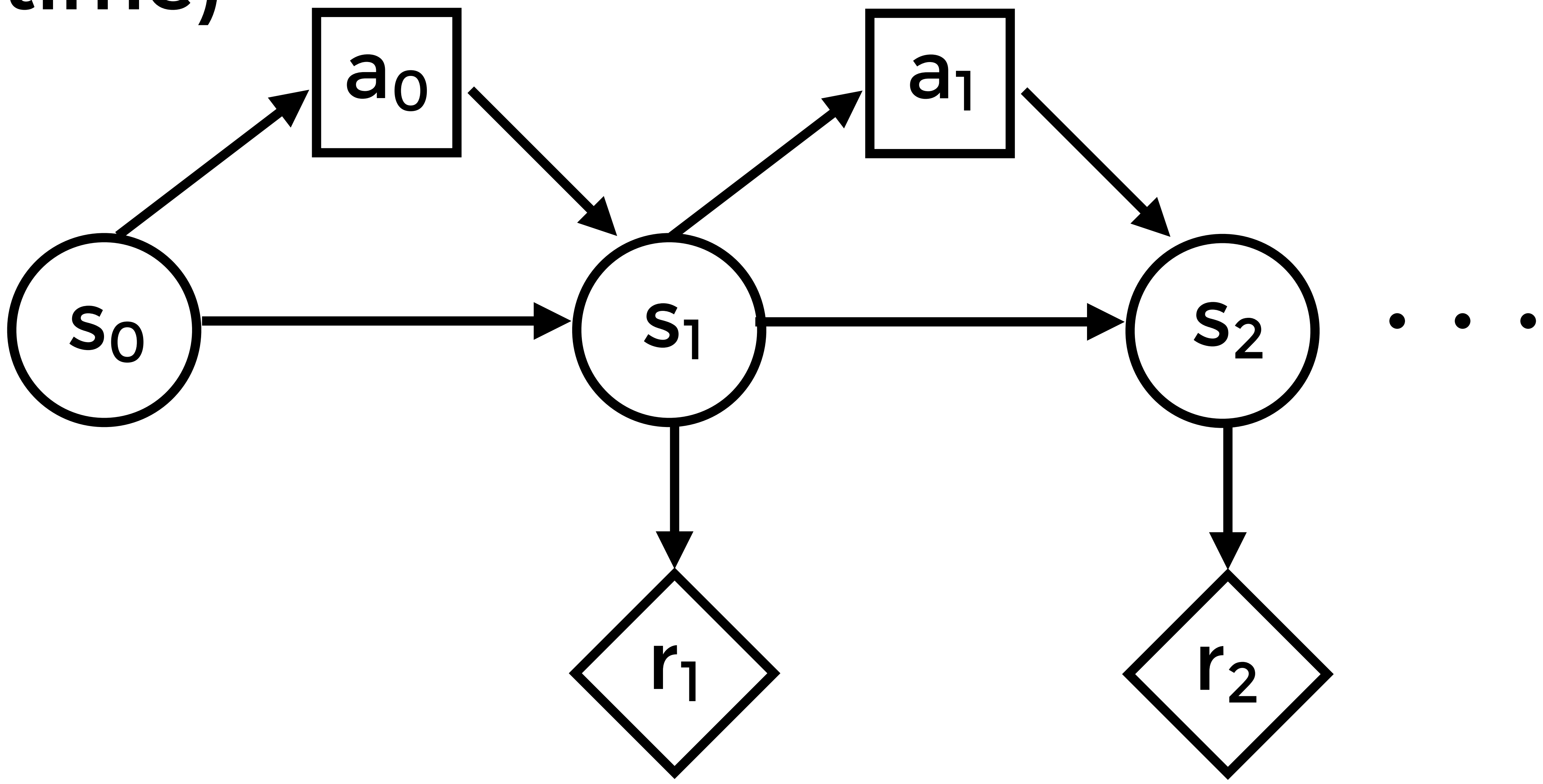
- Provide a framework for decision-making under uncertainty
  - Markov process with decisions and utilities
  - Assumes stationarity (i.e., transitions are fixed across time)

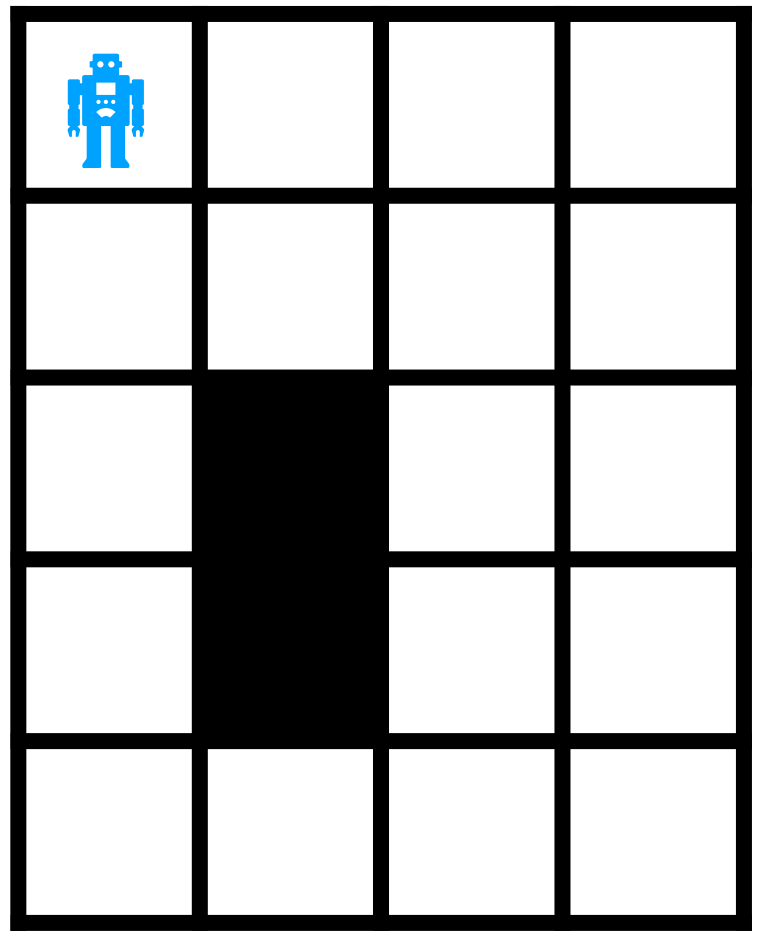


Markov Decision Process (MDP)

- Provide a framework for decision-making under uncertainty
  - Markov process with decisions and utilities
  - Assumes stationarity (i.e., transitions are fixed across time)

- Square nodes: decisions
- Circle nodes: States
- Diamond nodes: utility



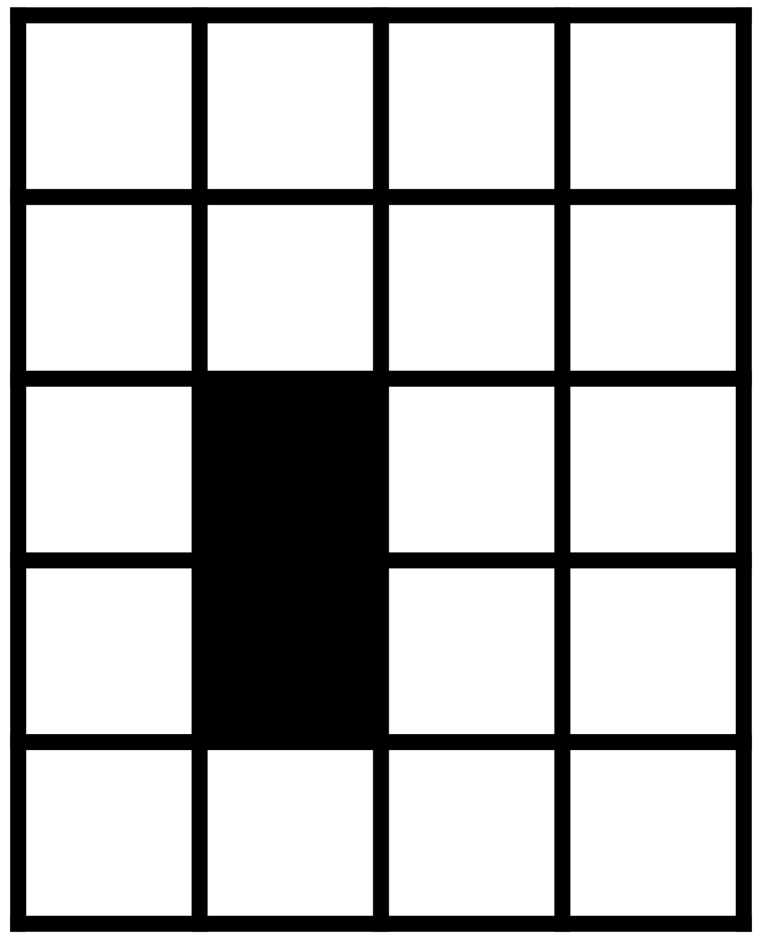


The objective of MDPs

Markov Decision Process (MDP)

$\langle \text{A}, \text{S}, \text{P}, \text{R}, \gamma \rangle$

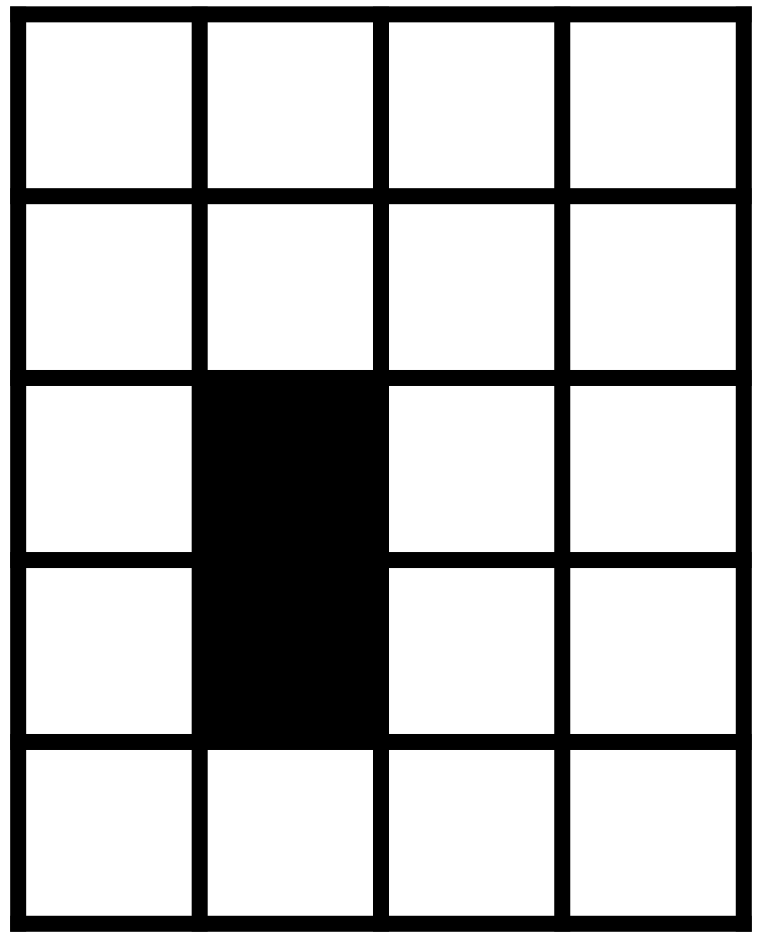
- **A: set of actions**
- **P(S' | S,A): transition probabilities**
- **R(S): reward function**
- γ **: discount factor** $\in [0, 1]$



Markov Decision Process (MDP)

$\langle \mathsf{A}, \mathsf{S}, \mathsf{P}, \mathsf{R}, \gamma \rangle$

- A: set of actions
- P(S' | S,A): transition probabilities
- R(S): reward function
- γ : discount factor $\in [0, 1]$



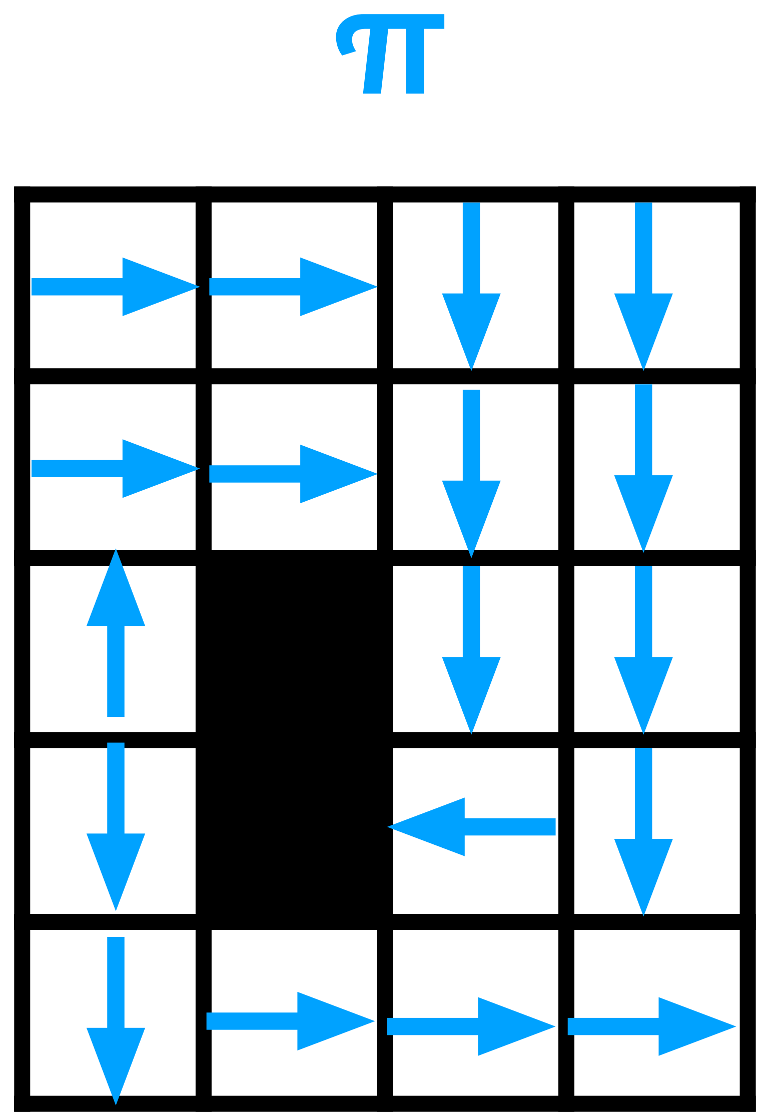
- A policy: $\pi : \mathsf{S} \rightarrow \mathsf{A}$

Markov Decision Process (MDP)

$\langle \text{A}, \text{S}, \text{P}, \text{R}, \gamma \rangle$

- A: set of actions
- P(S' | S,A): transition probabilities
- R(S): reward function
- γ : discount factor $\in [0, 1]$

- A policy: $\pi : \text{S} \rightarrow \text{A}$

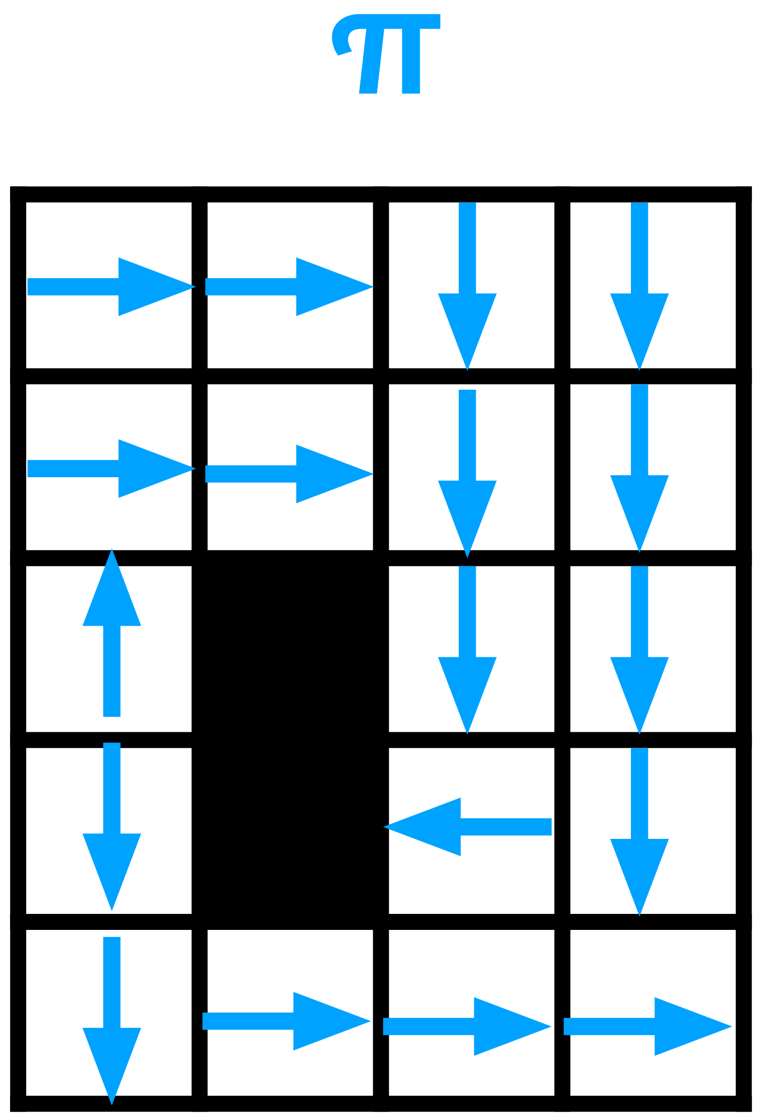


Markov Decision Process (MDP)

$\langle$A, S, P, R, $\gamma\rangle$

- A: set of actions
- P(S' | S,A): transition probabilities
- R(S): reward function
- γ : discount factor $\in [0, 1]$

- A policy: π : S $\rightarrow$ A
- **Goal:** find the optimal policy



Optimal policy?

- Agent is trying to maximize its rewards (utility)
  - Utility simply assigns a real value to a state
  - Typically combine rewards with an additive function

$$\sum_t R(s_t)$$

Discounting (γ)

- **The sum of rewards could be infinite/unbounded**

$$\lim_{T \to \infty} \sum_{t}^{T} R(s_t)$$

Discounting (γ)

- **The sum of rewards could be infinite/unbounded**

$$\lim_{T \to \infty} \sum_{t}^{T} R(s_t)$$

- **A typical solution is to use a discount factor** $0 \leq \gamma \leq 1$

$$\lim_{T \to \infty} \sum_{t}^{T} \gamma^t R(s_t)$$

Discounting (γ)

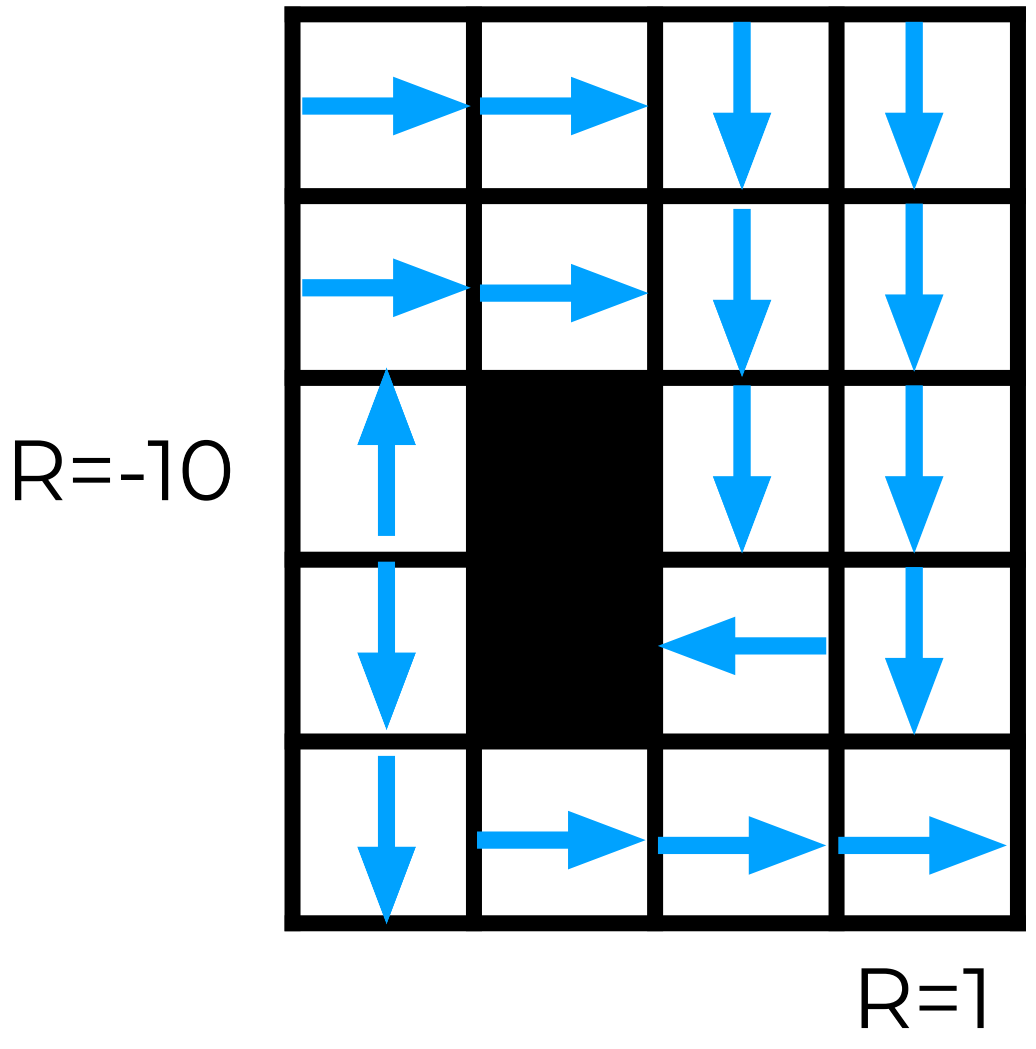
- **The sum of rewards could be infinite/unbounded**

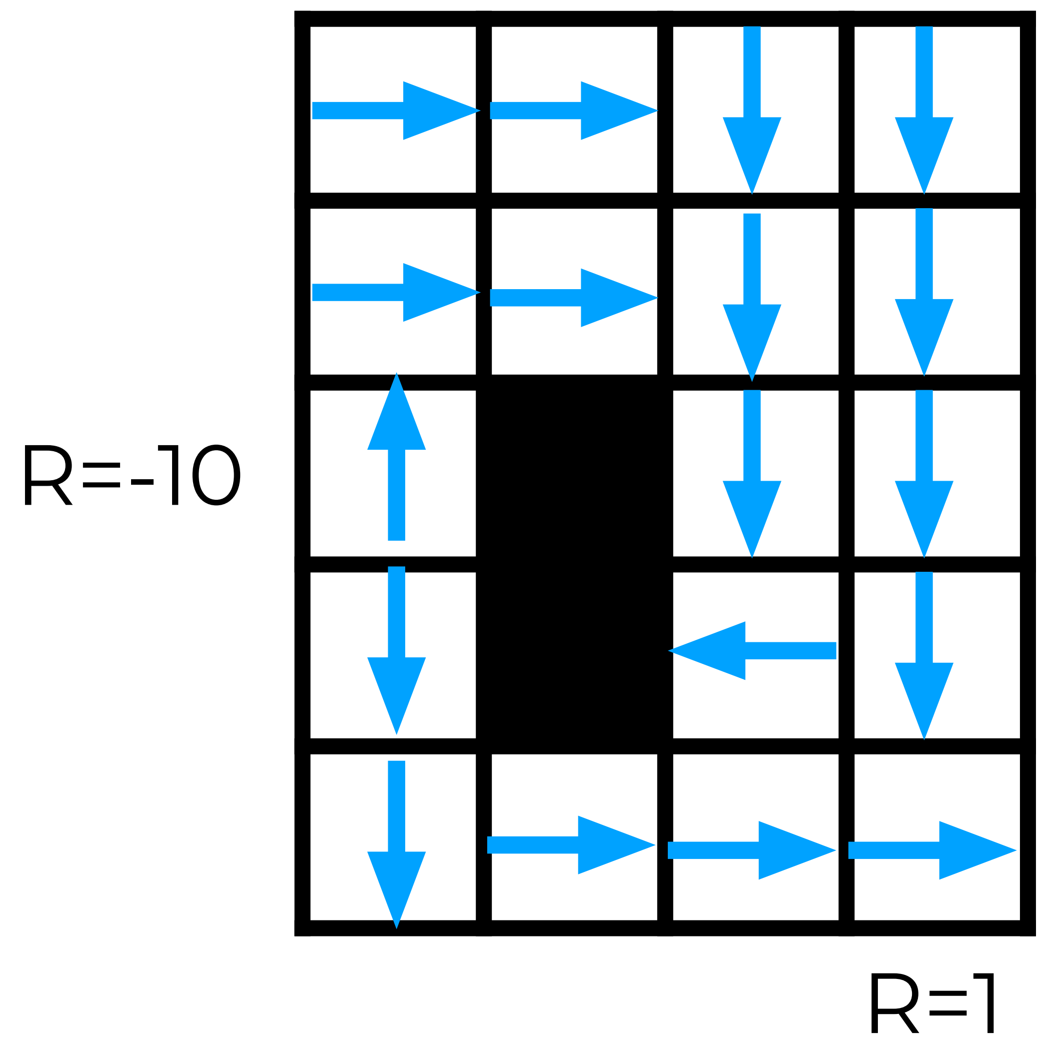
$$\lim_{T \to \infty} \sum_{t}^{T} R(s_t)$$

- **A typical solution is to use a discount factor** $0 \leq \gamma \leq 1$

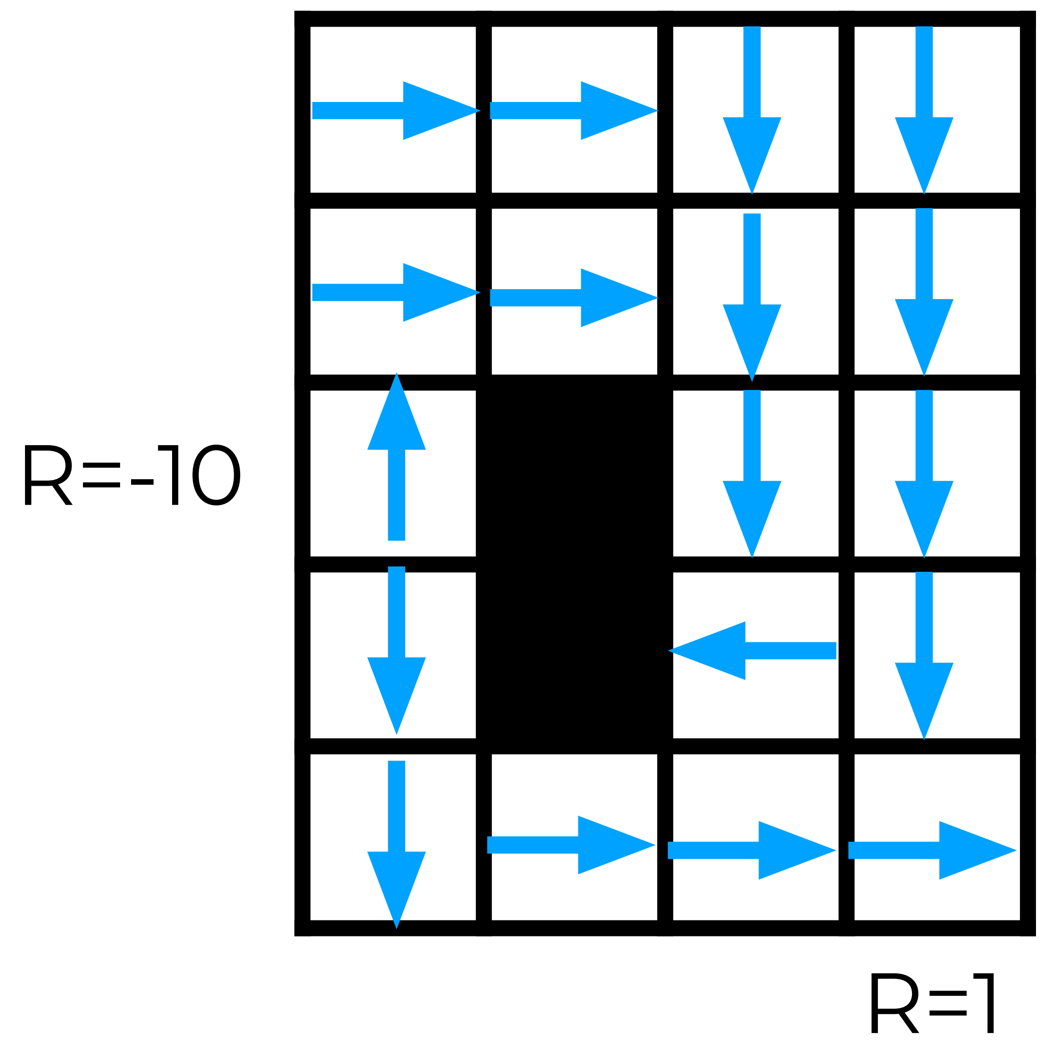
$$\lim_{T \to \infty} \sum_{t}^{T} \gamma^t R(s_t)$$

  - **Geometric series. Bounded by:** $\frac{R_{\max}}{1 - \gamma}$
  - **Intuition: would rather have rewards sooner**


R=-10
R=1

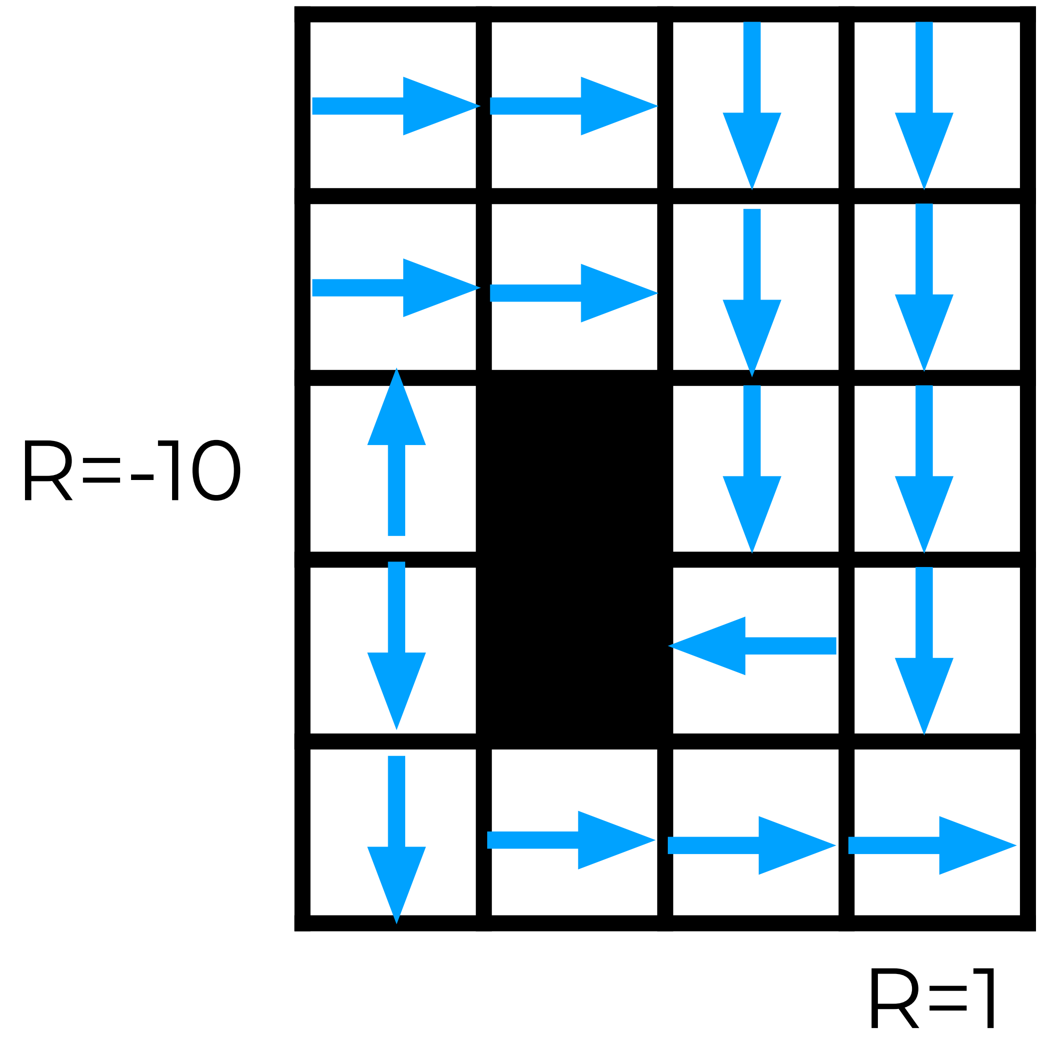


You cannot calculate the sum of the rewards directly:



You cannot calculate the sum of the rewards directly:

$$\sum_{t}^{T} \gamma^t R(s_t)$$



You cannot calculate the sum of the rewards directly:

$$\sum_{t}^{T} \gamma^{t} R(s_t)$$

Rewards are uncertain
- They depend on the transition probabilities

Maximize Expected Utility

- Maximize Expected Utility (MEU)
  - In short: optimal decision under uncertainty is the one with greatest expected utility
  - Variability comes from: environment uncertainty
- Justification for MEU: Rational agents must obey constraints which lead to optimizing expected utility

Solving an MDP

- **Find the optimal policy of an MDP**

$$\pi^*(s) \quad \forall s$$

Solving an MDP

- **Find the optimal policy of an MDP**

$$\pi^*(s) \ \ \forall s$$

- **Policies are evaluated using their expected utility:**

$$EU(\pi) = \sum_{t=0}^{\infty} \gamma^t \sum_{s_{t+1}} P(s_{t+1} \mid s_t, \pi(s_t)) R(s_{t+1})$$

Solving an MDP

- Find the optimal policy of an MDP

$$\pi^*(s) \quad \forall s$$

- Policies are evaluated using their expected utility:

$$EU(\pi) = \boxed{\sum_{t=0}^{\infty} \gamma^t} \sum_{s_{t+1}} P(s_{t+1} \mid s_t, \pi(s_t)) \boxed{R(s_{t+1})}$$

Solving an MDP

- Find the optimal policy of an MDP

$$\pi^*(s) \quad \forall s$$

- Policies are evaluated using their expected utility:

$$EU(\pi) = \boxed{\sum_{t=0}^{\infty} \gamma^t} \sum_{s_{t+1}} P(s_{t+1} \mid s_t, \pi(s_t)) \boxed{R(s_{t+1})}$$

- The optimal policy is the one with highest expected utility: $EU(\pi^*) \geq EU(\pi) \quad \forall \pi$

Solving MDPs
(obtaining the optimal policy)

Solving an MDP

- Three well-known techniques:
  1. Value iteration
  2. Policy Iteration
  3. Linear Programming

Value Function

- $V(s_t)$: The value of being in state s at time t

Value Function

- $V(s_t)$: The value of being in state s at time t

$$V(s_t) := \text{expected sum of rewards of being in s}$$

Finite horizon

- Assume that the process has T steps

Finite horizon

- Assume that the process has T steps
- The value at step T is

Finite horizon

- Assume that the process has T steps
- The value at step T is $V(s_T) = R(s_T)$

Finite horizon

- **Assume that the process has T steps**
- **The value at step T is** $V(s_T) = R(s_T)$
- **The value at step T-1 is**

$$V(s_{T-1}) = \max_{a_{T-1}} \left\{ R(s_{T-1}) + \gamma \sum_{s_T} P(s_T \mid s_{T-1}, a_{T-1}) R(s_T) \right\}$$

Finite horizon

- Assume that the process has T steps
- The value at step T is $V(s_T) = R(s_T)$
- The value at step T-1 is

$$V(s_{T-1}) = \max_{a_{T-1}} \left\{ R(s_{T-1}) + \gamma \sum_{s_T} P(s_T \mid s_{T-1}, a_{T-1}) R(s_T) \right\}$$

- The value at step t is$(0 \leq t \leq T)$

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\}$$

Finite horizon

- Assume that the process has T steps
- The value at step T is $V(s_T) = R(s_T)$
- The value at step T-1 is

$$V(s_{T-1}) = \max_{a_{T-1}} \left\{ R(s_{T-1}) + \gamma \sum_{s_T} P(s_T \mid s_{T-1}, a_{T-1}) R(s_T) \right\}$$

- The value at step t is$(0 \leq t \leq T)$

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\}$$

Bellman equation

- **Value of state s**

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\} \quad \forall s$$

Bellman equation

- Value of state s

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\} \quad \forall s$$

- Recursive equations

Bellman equation

- Value of state s

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\} \quad \forall s$$

- Recursive equations
- The value of a state only depends on the state's reward and the neighbours' value

Bellman equation

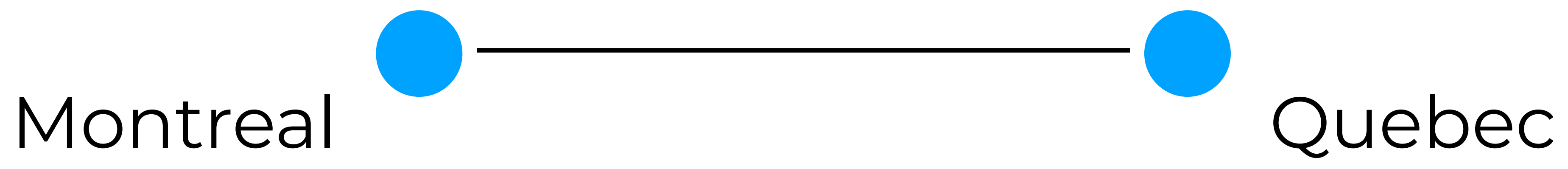
- Value of state s

$$V(s_t) = \max_{a_t} \left\{ R(s_t) + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V(s_{t+1}) \right\} \quad \forall s$$

- Recursive equations
- The value of a state only depends on the state's reward and the neighbours' value
- This is also known as a dynamic programming equation

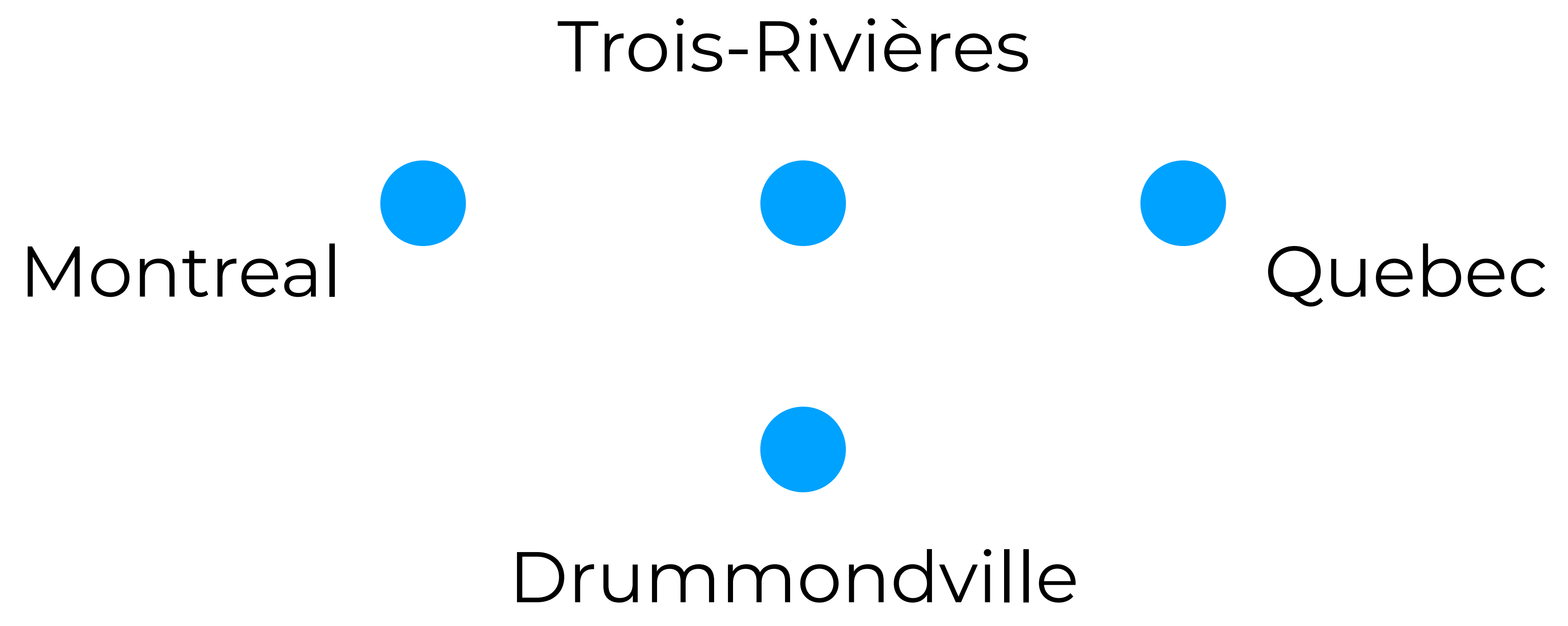
Dynamic Programming (in 1 slide)

- **Solution technique that decomposes a problem into a set of subproblems**
  - **The solution to each subproblem is part of the solution of the original problem**
  - **E.g.,**



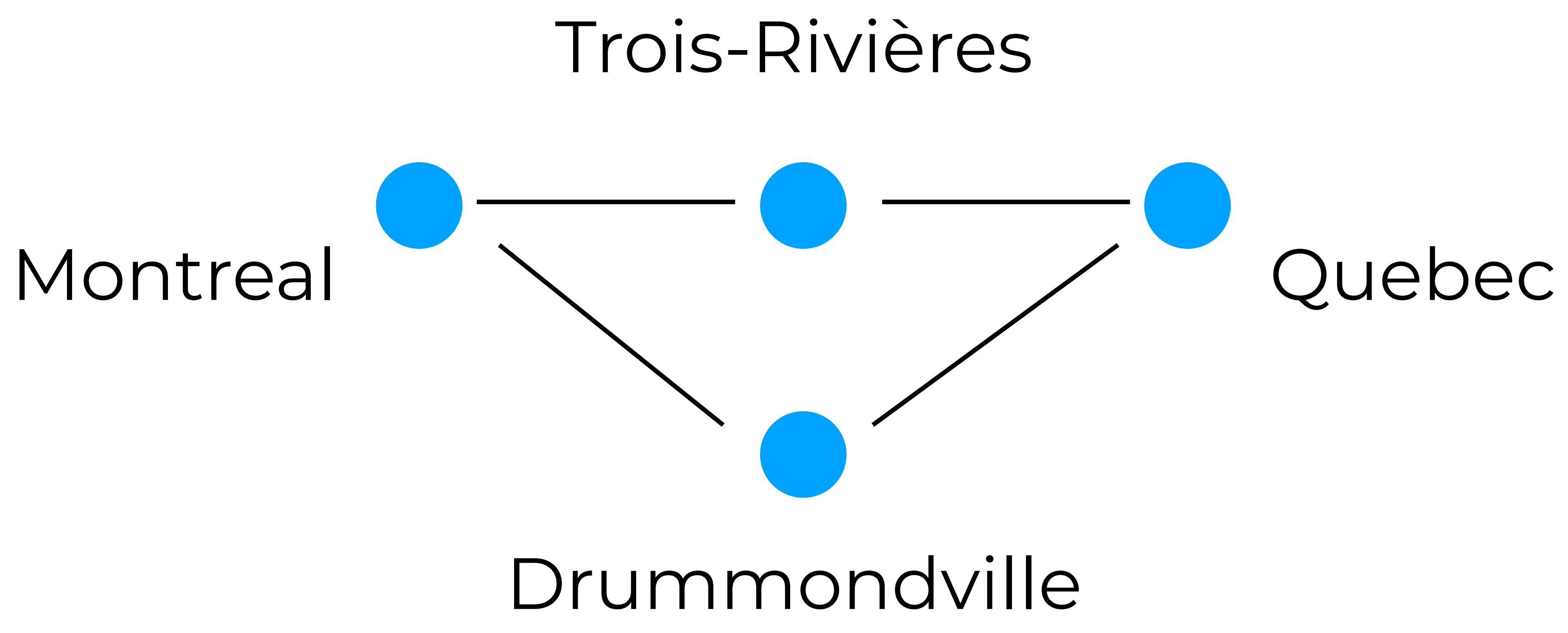
Dynamic Programming (in 1 slide)

- Solution technique that decomposes a problem into a set of subproblems
  - The solution to each subproblem is part of the solution of the original problem
  - E.g.,



Dynamic Programming (in 1 slide)

- Solution technique that decomposes a problem into a set of subproblems
  - The solution to each subproblem is part of the solution of the original problem
  - E.g.,



Value iteration (VI)

- Iteratively update V(s) for each state until convergence

Value iteration (VI)

- Iteratively update V(s) for each state until convergence
- (Initialize V(s) for every state)

Value iteration (VI)

- Iteratively update V(s) for each state until convergence
- (Initialize V(s) for every state)
- For i=1,2,3,...
  - For s=1,...,S

$$\mathbf{V}(\mathbf{s}) = \max_{\mathbf{a}} \left\{ \mathbf{R}(\boldsymbol{s}) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \mathbf{V}(\mathbf{s}') \right\}$$

Value iteration (VI)

- Iteratively update V(s) for each state until convergence
- (Initialize V(s) for every state)
- For i=1,2,3,...
  - For s=1,...,S

$$\mathbf{V}(\mathbf{s}) = \max_{\mathbf{a}} \left\{ \mathbf{R}(\boldsymbol{s}) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \mathbf{V}(\mathbf{s}') \right\}$$

- The policy is implicit
  - Once converged: $\boldsymbol{\pi}^*(\mathbf{s}) = \arg\max_{\mathbf{a}} \left\{ \mathbf{R}(\mathbf{s}) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \mathbf{V}^*(\mathbf{s}') \right\} \forall \mathbf{s}$

Policy Iteration (PI)

- **Improve policy explicitly.**

Policy Iteration (PI)

- Improve policy explicitly.

Start with any (e.g., random) policy π

Policy Iteration (PI)

- Improve policy explicitly.

Start with any (e.g., random) policy π

Iterate until convergence:

1. Given current policy get the value of each state

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^{\pi}(s') \qquad \forall s$$

Policy Iteration (PI)

- Improve policy explicitly.

Start with any (e.g., random) policy π

Iterate until convergence:

1. Given current policy get the value of each state

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^{\pi}(s') \qquad \forall s$$

2. Update the current policy

$$\pi'(s) = \arg\max_{a} \left\{ R(s) + \gamma \sum_{s'} P(s' \mid s, a) V^{\pi}(s') \right\} \quad \forall s$$

Policy Iteration (PI)

- Improve policy explicitly.

Start with any (e.g., random) policy π

Iterate until convergence:

Policy Evaluation

1. Given current policy get the value of each state

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^{\pi}(s') \qquad \forall s$$

Policy Update

2. Update the current policy

$$\pi'(s) = \arg\max_{a} \left\{ R(s) + \gamma \sum_{s'} P(s' \mid s, a) V^{\pi}(s') \right\} \quad \forall s$$

PI vs. VI

- **Value iteration is faster per iteration**
- **Policy iteration converges in fewer iterations**

- **Some of these slides were adapted from Pascal Poupart's slides (CS686 U.Waterloo)**