Reinforcement Learning - 1

Recap

- **So far.**
  - Supervised Learning
  - Unsupervised Learning
  - Generative Modeling

- **Today.**
  - Reinforcement Learning

Reinforcement learning

- Different from other ML paradigms in several aspects:
  - No supervisor — only a reward
  - Delayed feedback — not instantaneous
  - Time matters — Sequential, non-i.i.d. data
  - Agent’s action affect the subsequent data it receives

Overview: A Maze Example

Problem

- Suppose that we want to solve maze
- **Task.** Given any maze, train a model that can find a path: Start -> End

$$f(\text{maze}) = \hat{\text{path}} = (\text{loc}_1, \text{loc}_2, \ldots)$$

- **Goal.**

(1) Path should solve the maze:

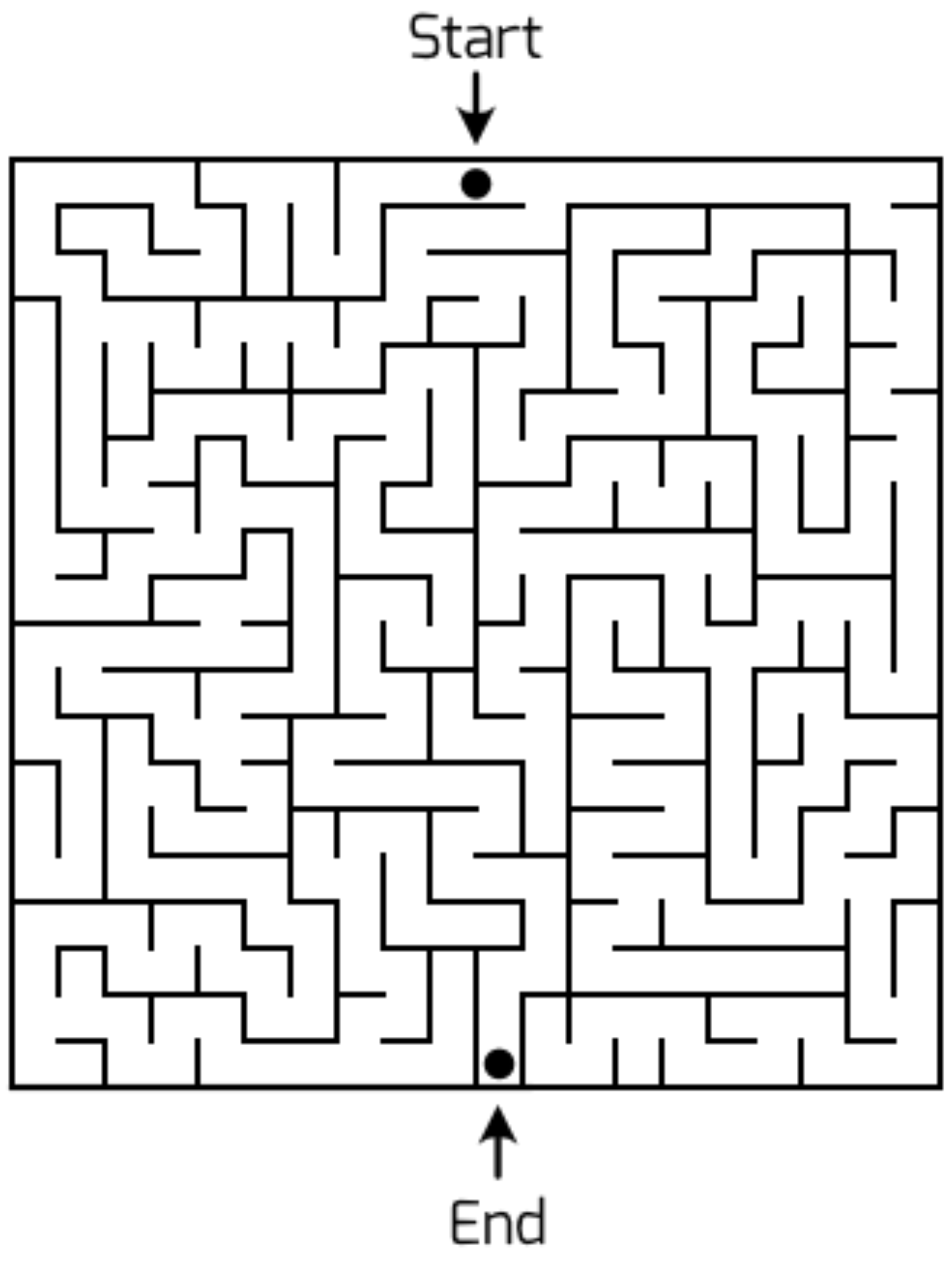
$$\hat{\text{path}}[0] = \text{start}, \ \hat{\text{path}}[-1] = \text{end}$$

(2) Path should be valid

$$\hat{\text{path}} \cup \text{wall} = \emptyset$$

(3) Make it shortest as possible

$$\min \mathbb{E}[\text{len}(\hat{\text{path}})]$$



Supervised Learning

- How do we solve this problem?
- **Naïve.** Try a supervised learning
  - Collect many samples of mazes

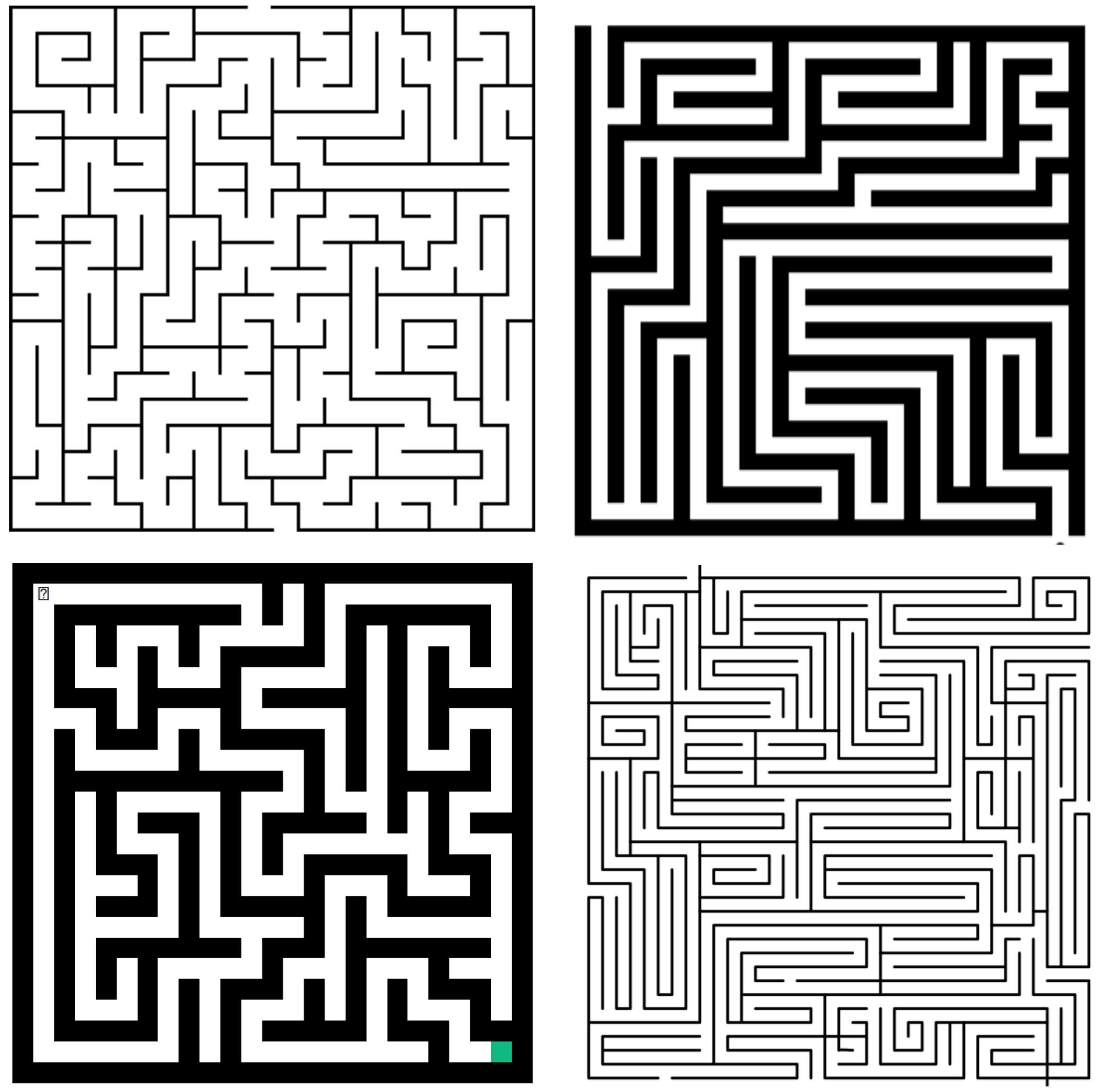
$$\text{maze}_1, \text{maze}_2, \ldots, \text{maze}_n$$

  - Label the samples (e.g., hire maze experts)

$$(\text{maze}_1, \text{path}_1), \ldots, (\text{maze}_n, \text{path}_n)$$

  - Train a predictor using some distance measure (e.g., MSE loss)

$$\min_f \frac{1}{n} \sum_{i=1}^{n} \text{diff}(\text{path}_i, f(\text{maze}_i)) \quad + \lambda \cdot \text{len}(f(\text{maze}_i))$$



Limitations

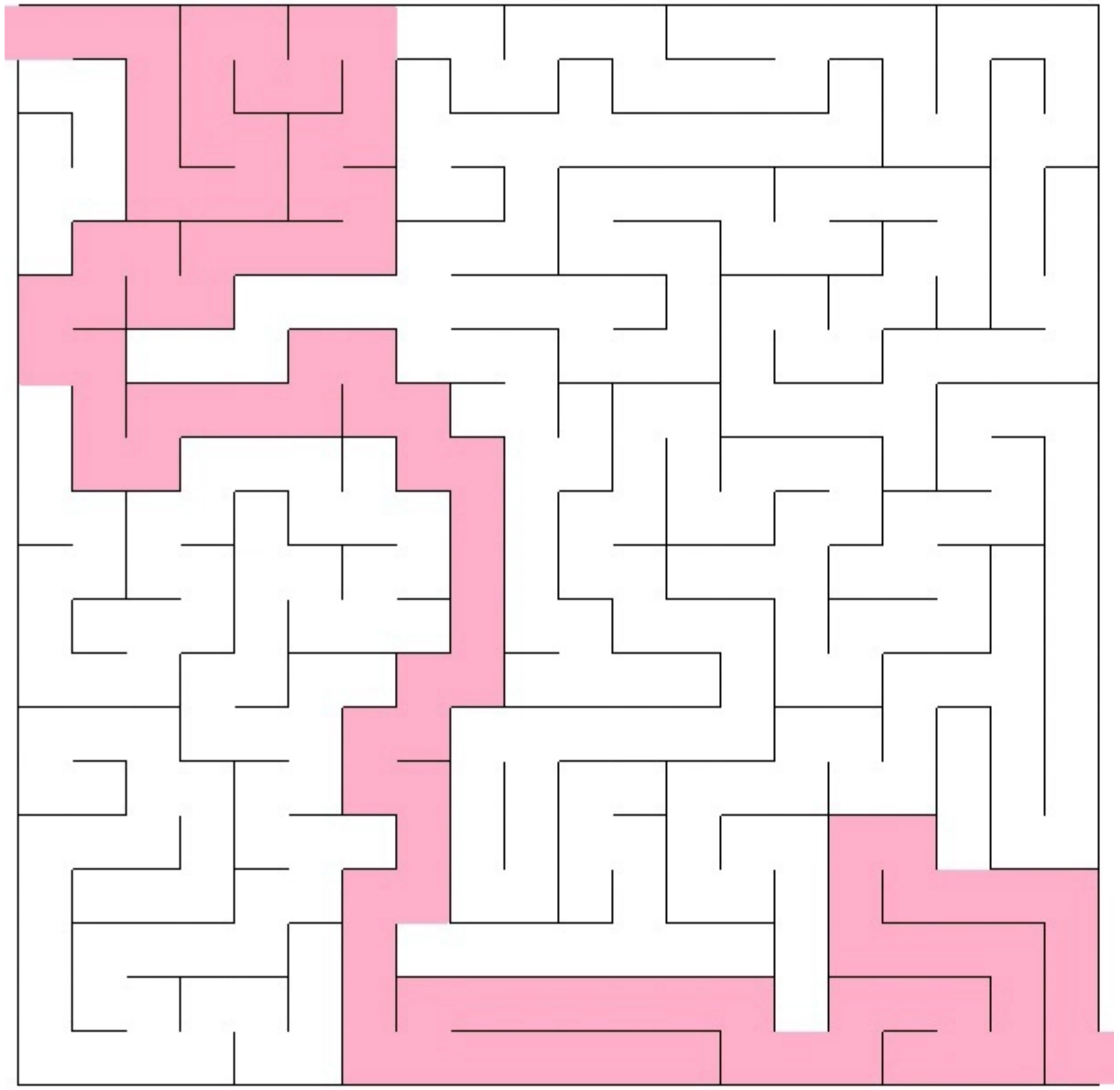
- This approach suffers from many limitations

1. Annotation. Costly to collect

- Expensive to hire expert maze solvers
- Some mazes are too difficult
- Multiple solutions can be true

2. Complexity. Label space is too complicated

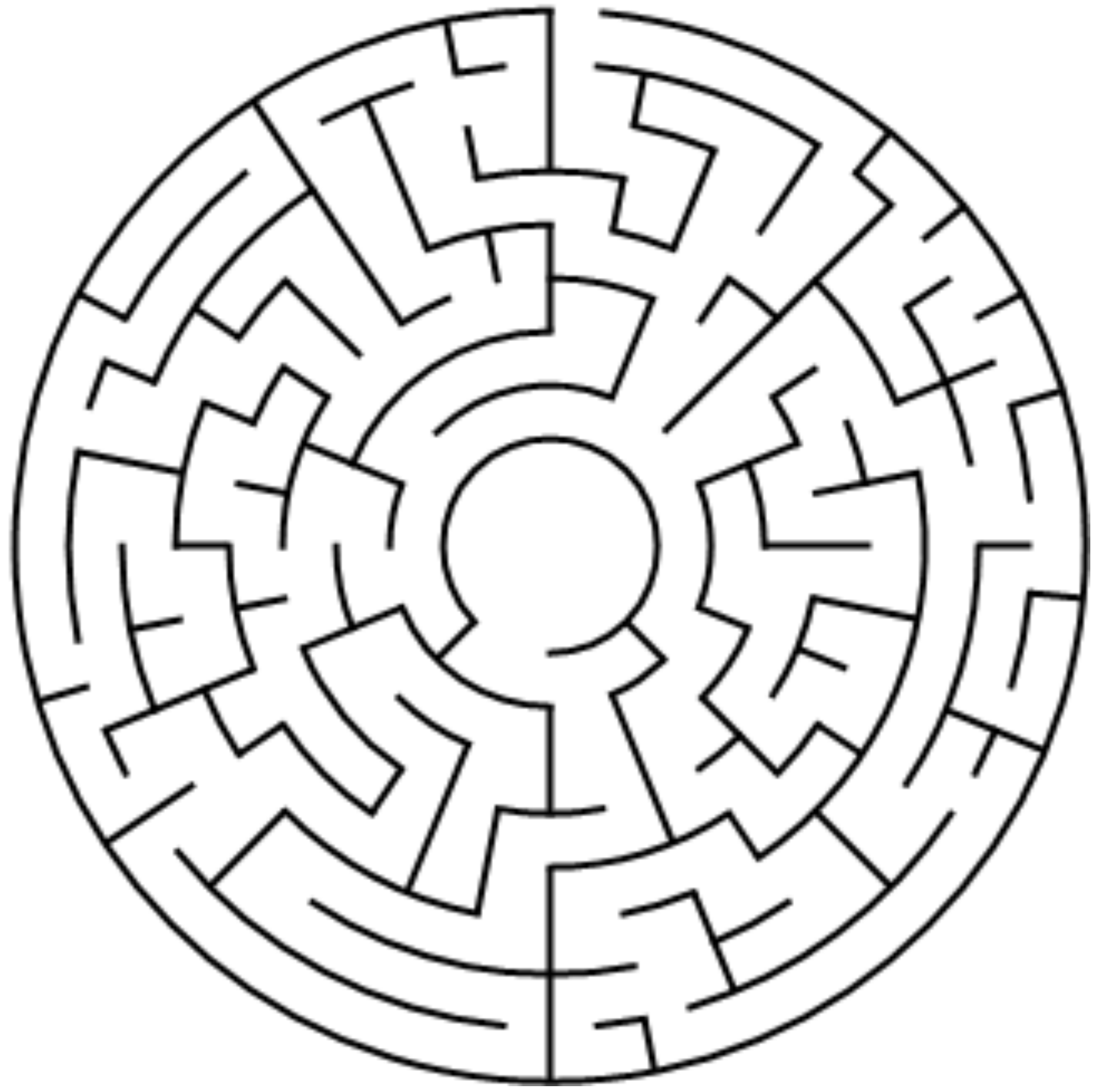
- Requires a lot of data

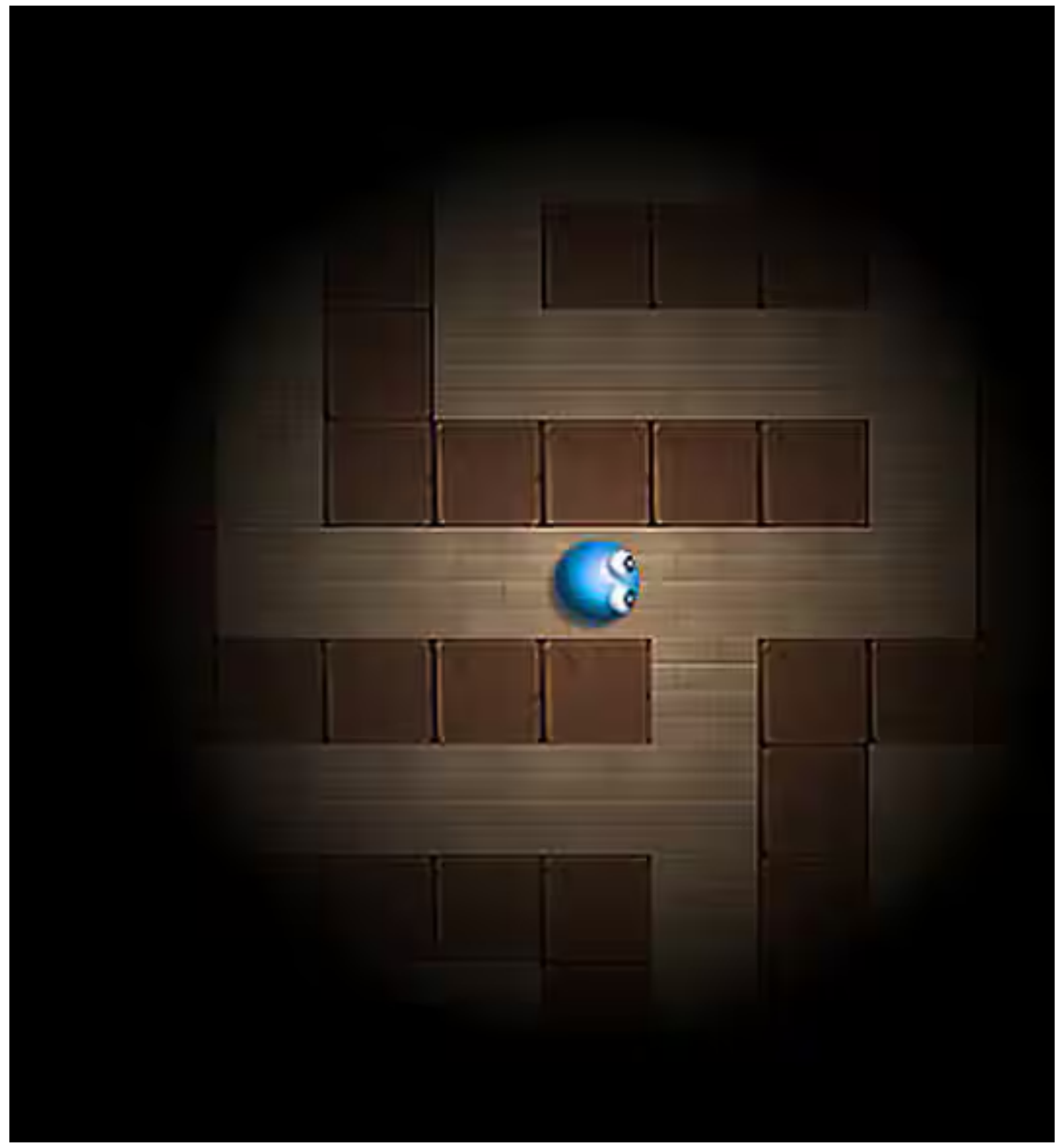


Limitations

3. Generalization. Often generalizes very poorly

- Unseen data types and styles
  - Circular maze
- Incomplete observables
  - Maze in the dark





Idea

- Formulate the problem as a sequential decision-making
  - Greatly simplifies the prediction task

- Given your current **state** (i.e., position), predict **action** (i.e., direction)

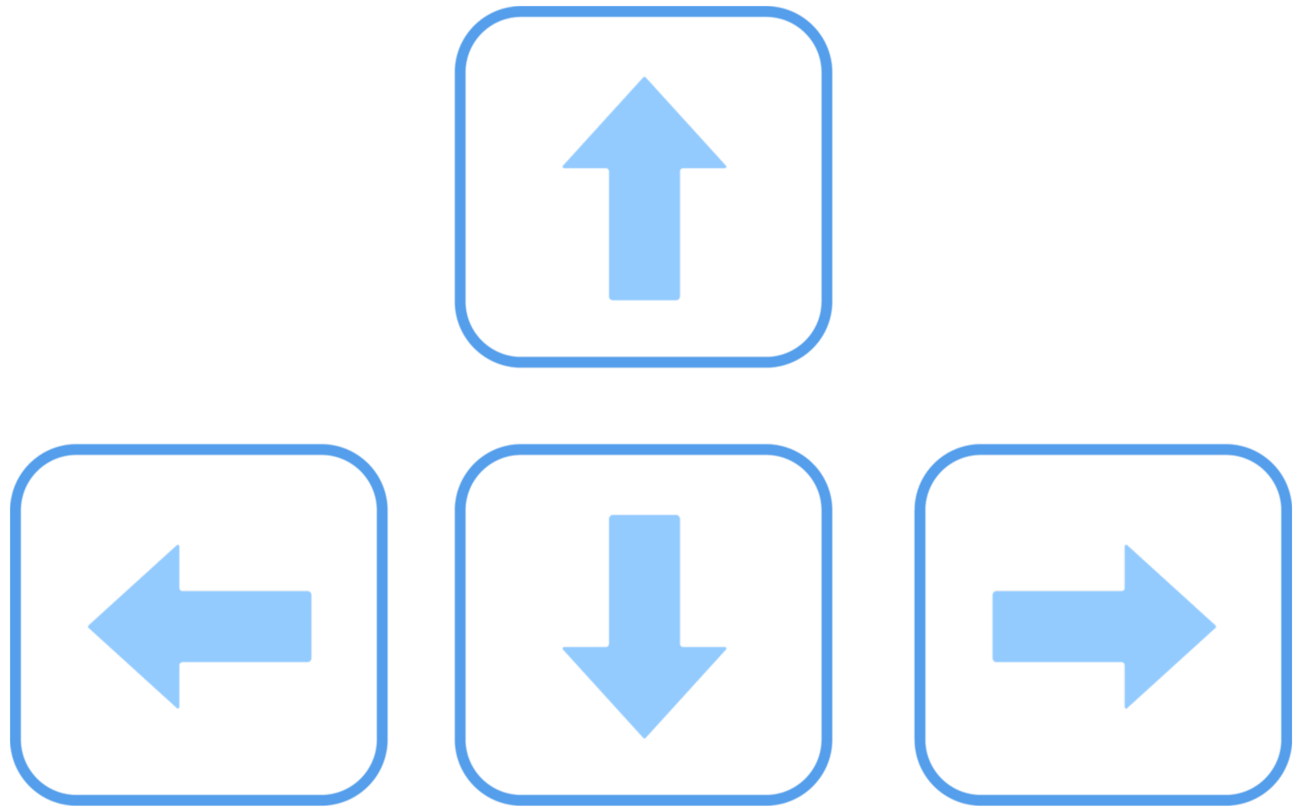
$$g(\text{loc}_i; \text{maze}) = \hat{\text{dir}}_i \in \{\text{up}, \text{down}, \text{left}, \text{right}\}$$

  - The next state is determined by both current state and action

$$\text{loc}_{i+1} = \text{loc}_i + \text{dir}_i$$

  - Repeat until you arrive (or reach max runtime)

$$\text{loc}_t = \text{end} \quad \text{or} \quad t \geq T$$



Idea

- **Objective.** Maximize the **reward** (i.e., success rate)

$$\mathbb{E}\left[\mathbf{1}[g^t(\text{start}; \text{maze}_i) = \text{end, for some } t]\right]$$

- **Data collection.** Deploy the model on sample mazes to collect many **episode** (i.e., path) unrolled by the model

$$\text{maze}_i, (\text{loc}_1, \text{loc}_2, \ldots, \text{loc}_T)$$

    - If succeeded, use it as a successful sample
    - If failed, use it as a wrong sample

- Mostly wrong samples at first, but will get better as training goes on

Idea

- **Note.** No human supervision at all!
  - Unlike image/text cases, the success depends on objective criterion
    - We have access to an **environment**, which gives us reward
  - If we have, we can also use human-labeled samples
    - Guide for early training

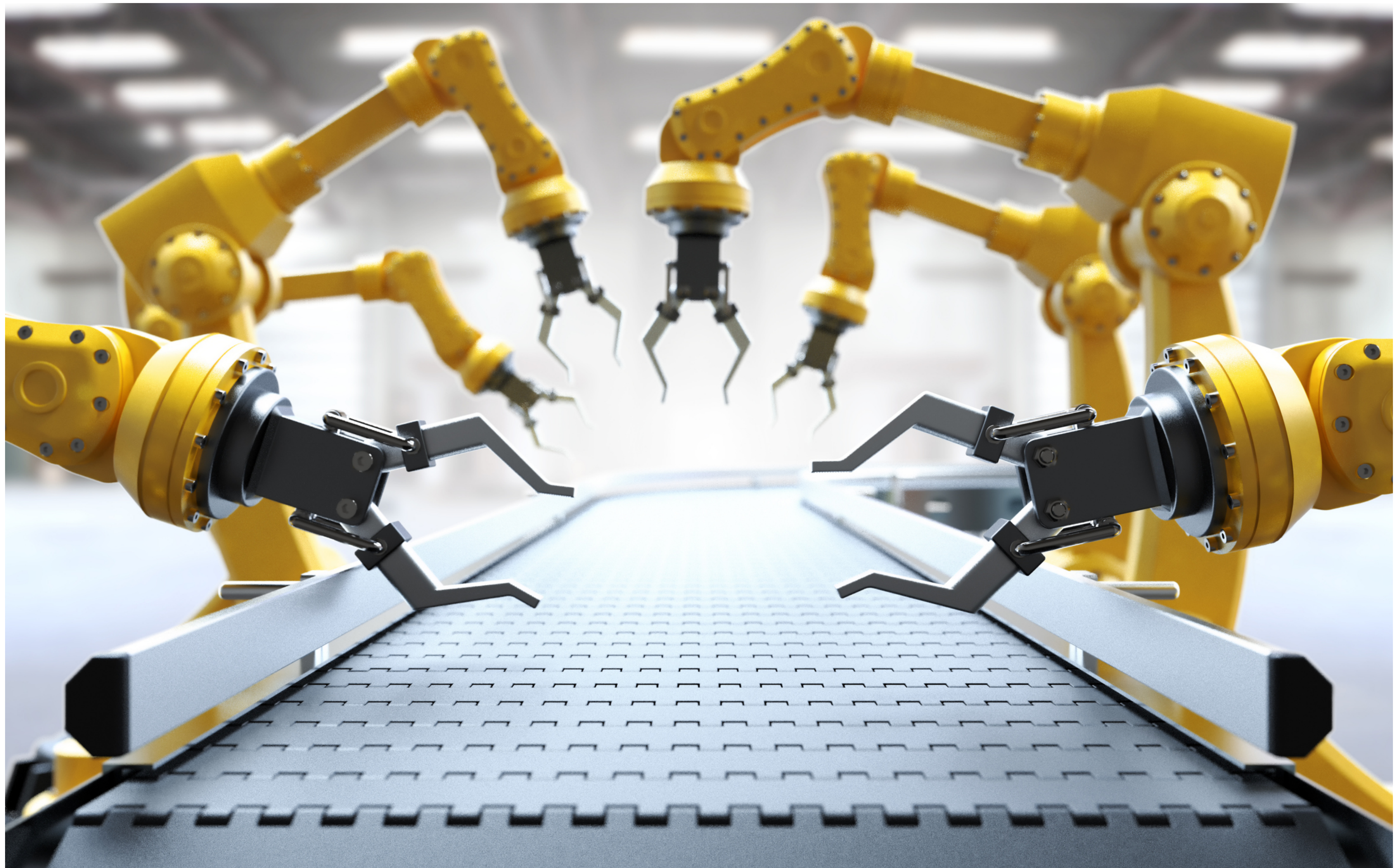
- **Question.** How do we encourage shorter path?
  - Select some discount factor $\gamma \in (0,1]$ and try to maximize

$$\mathbb{E}\left[\gamma^t \cdot \mathbf{1}[g^t(\text{start}; \text{maze}_i) = \text{end, for some } t]\right]$$

Applications

- Robotic control, especially when “human supervision” does not work

$$\text{movement}_i = g(\text{image}_i, \text{instruction}_i, \text{robot state}_i)$$



Applications

- Stock trading

$$\text{trading decision}_i = g(\text{market}_i, \text{portfolio}_i)$$

Applications

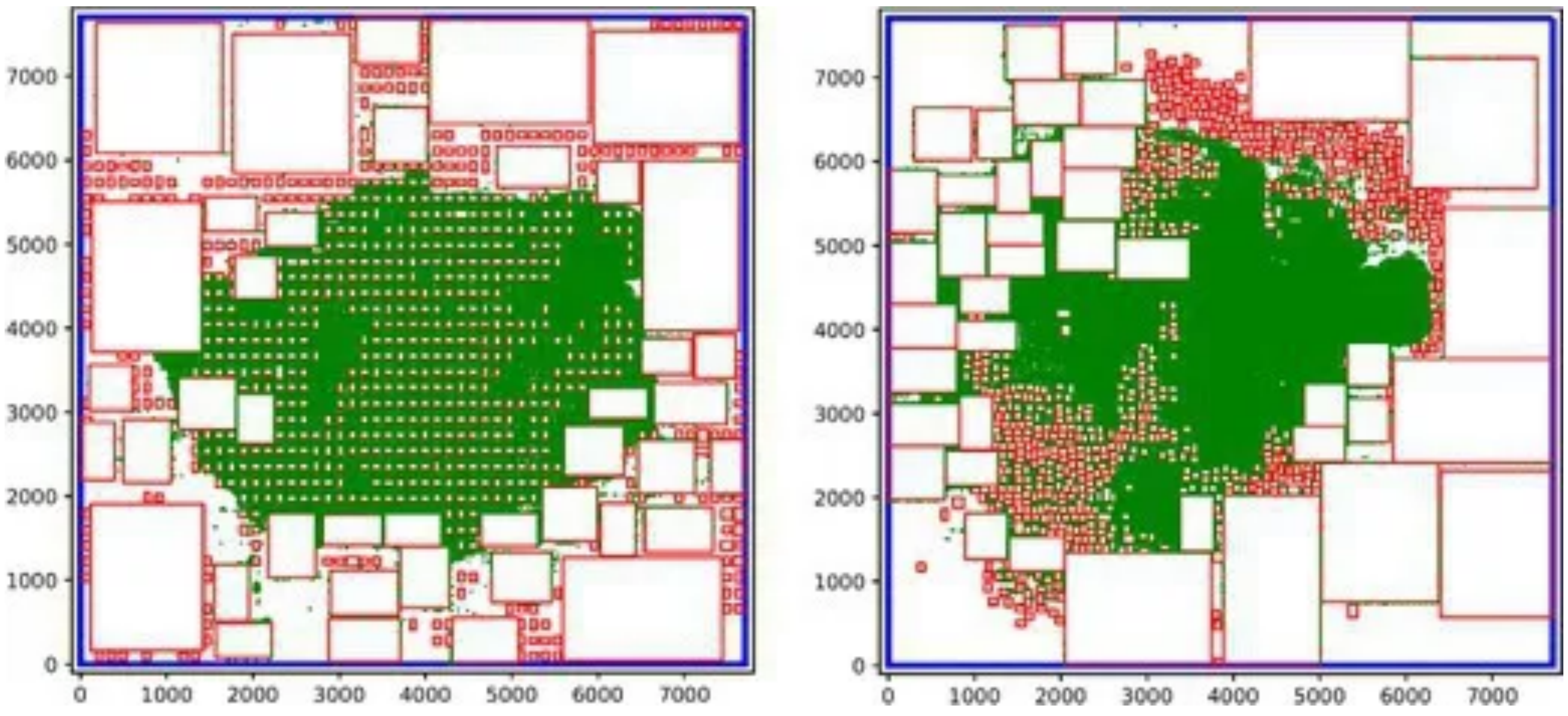
- Drug discovery
  - Next material to try can be generated by adding/subtracting atoms

$$\text{material to try}_i = g(\text{materials tried}_i)$$

Applications

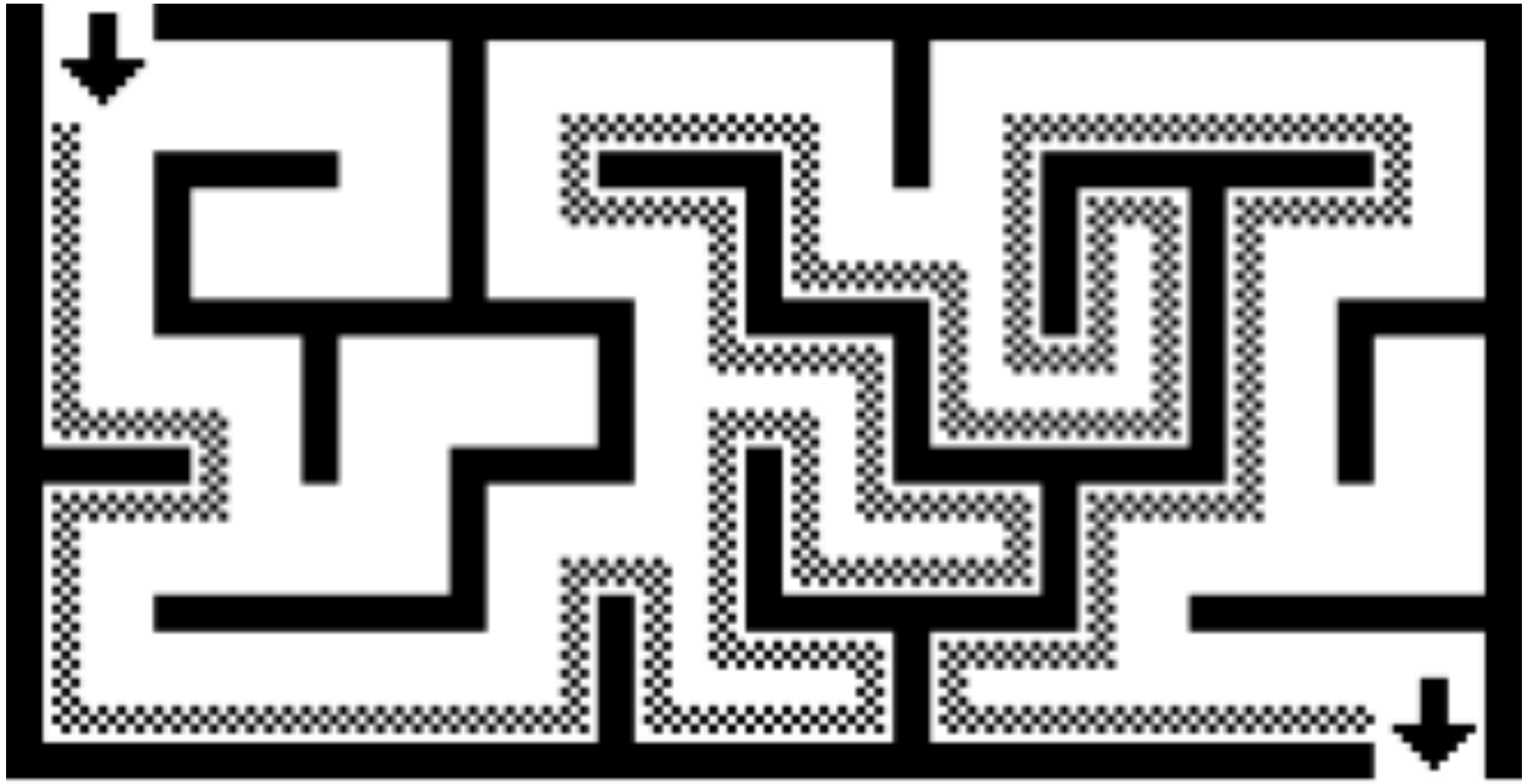
- Macro-level placement plans for chips

$$(\text{element}_i, \text{location}_i) = g(\text{current plan}_i)$$



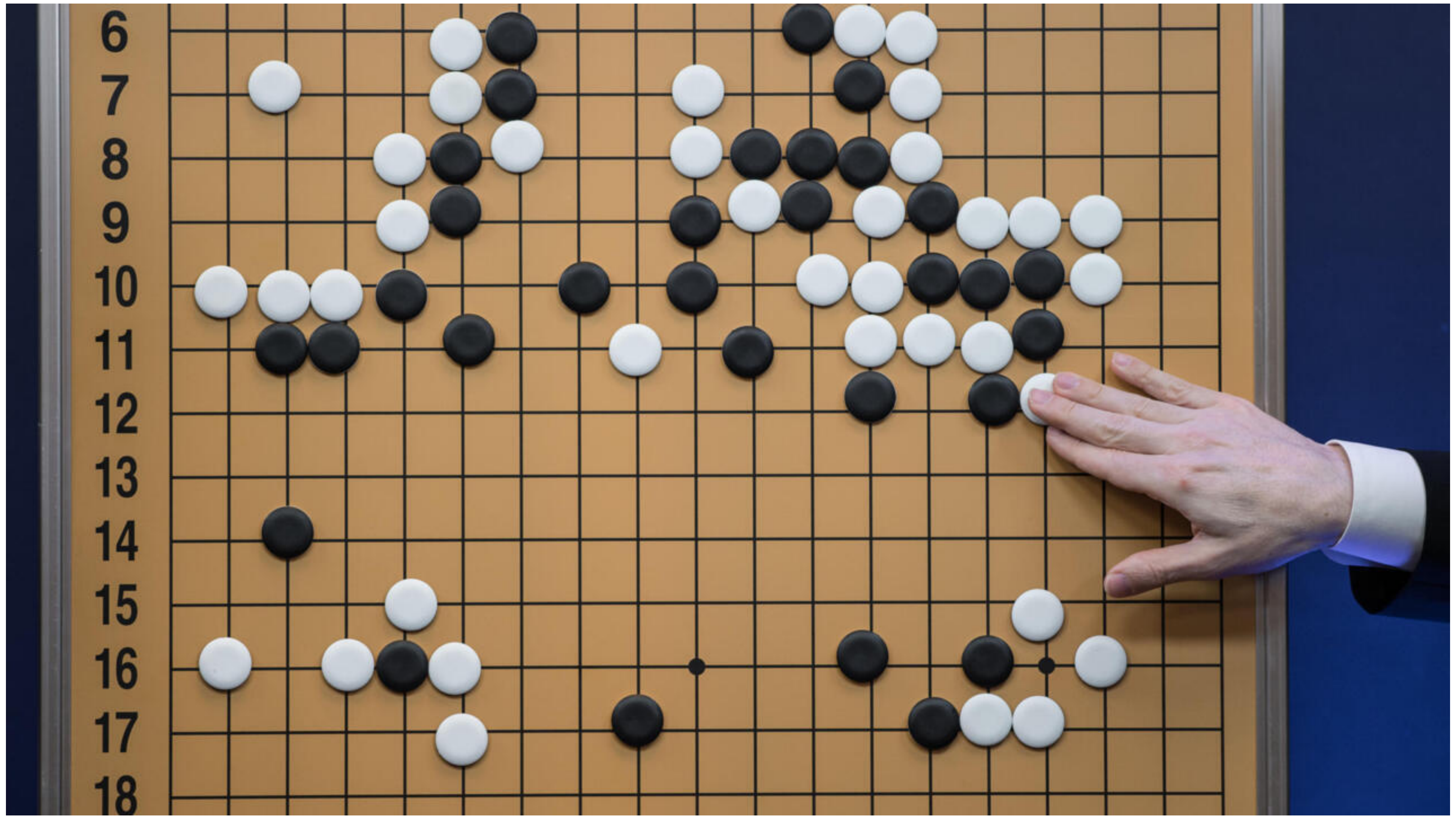
Key issues

- Often, we have an exploration-exploitation tradeoff
  - If we find a successful predictor —e.g., right-hand policy
  - Then we can either
    - **Explore.** Try a new policy, potentially finding a better one
    - **Exploit.** Stick to the policy, to maximize the success rate



Key issues

- Rewards are sparse in many cases
  - Difficult to know if your intermediate actions have been correct
  - Especially problematic for long-horizon tasks
    - Can fall into myopic actions



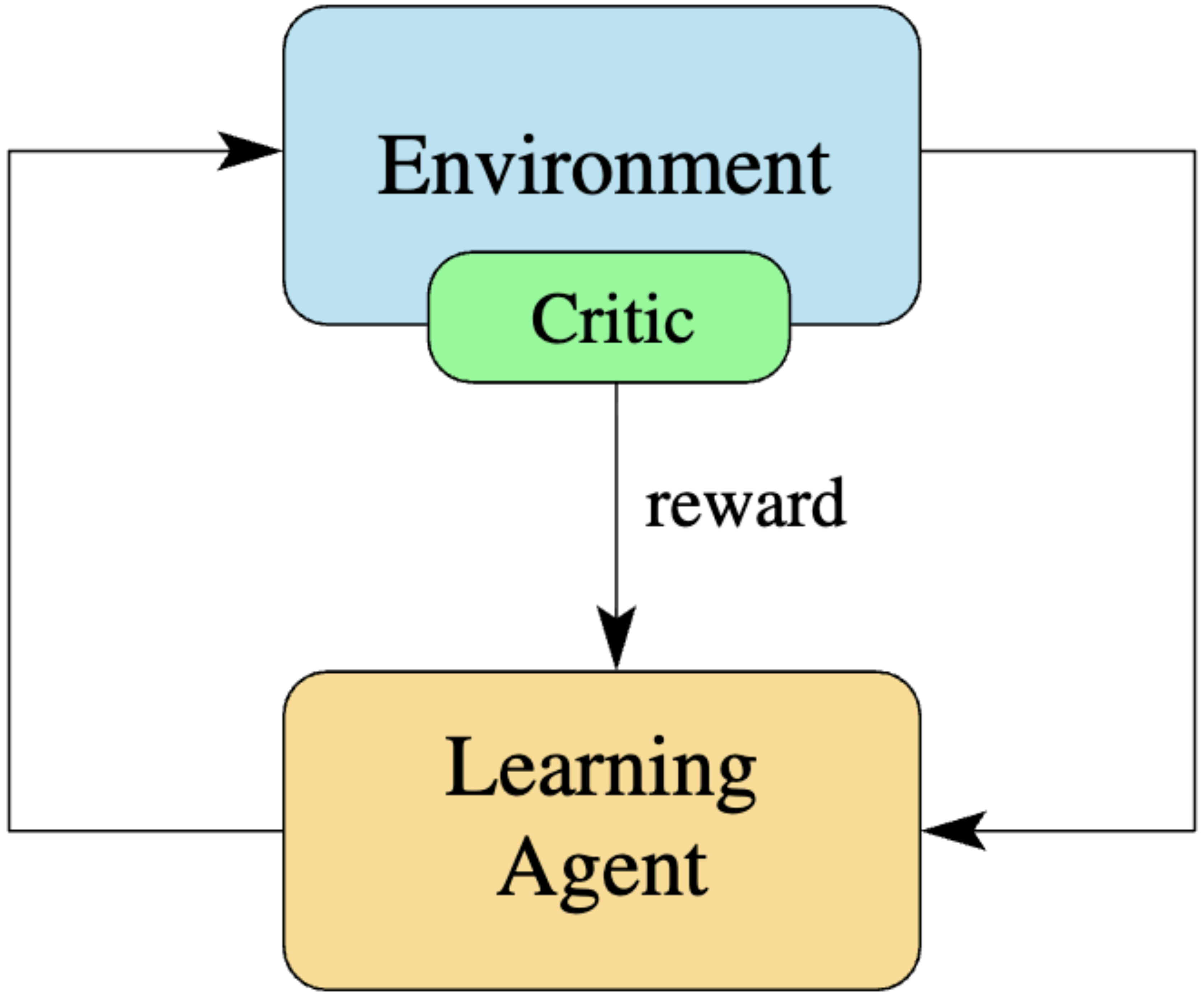
In what follows

- Formalisms
  - Markov Decision Process
  - Bellman Equation
- Algorithms
  - As time permits…

Formalism

Reinforcement learning

- Learning a behavior strategy (a policy),
  which maximizes the long term sum of rewards (delayed reward)
  by a direct interaction (trial-and-error)
  with an unknown and uncertain environments



Interaction protocol

- **Agent.** Generates action, given the current state (plus reward)

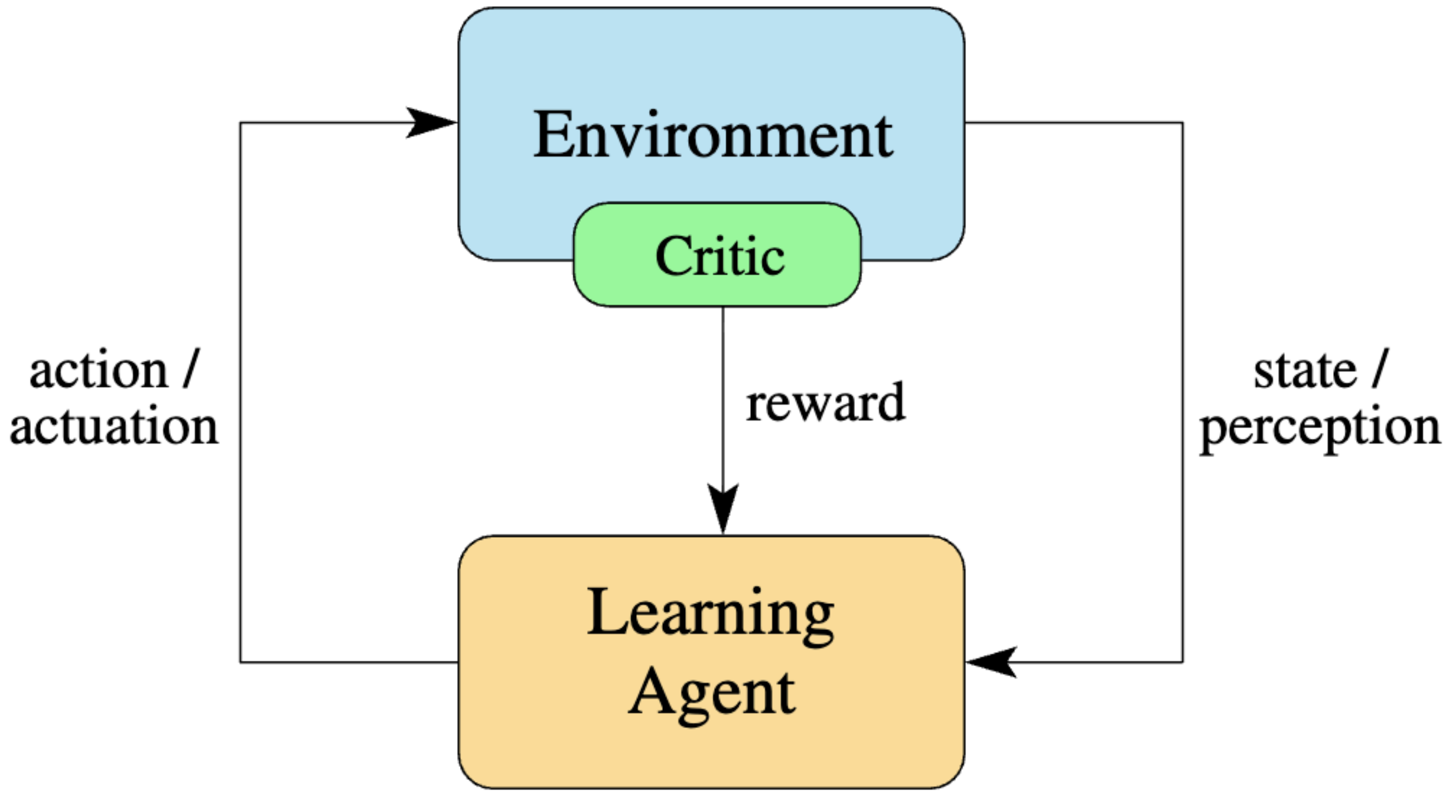
$$a_i = f_\theta(s_i)$$

- **Environment.** Generates next state, given the action and current state

$$s_{i+1} = g(s_i, a_i)$$

  - **Critic.** Generates reward

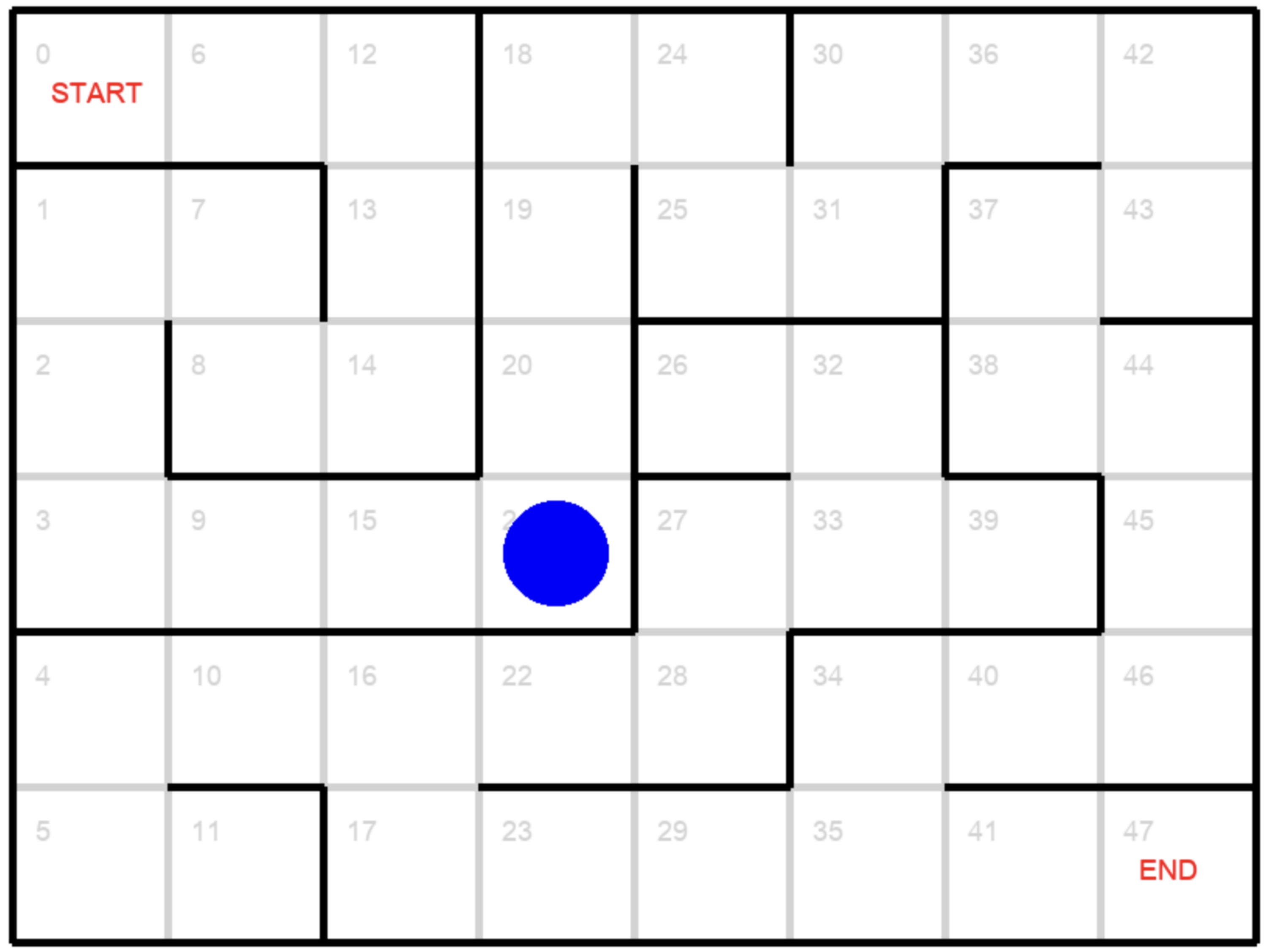
$$r_i = h(s_i, s_{i+1}, a_i)$$



for $t = 1, \ldots, n$ **do**
  The agent perceives state s_t
  The agent performs action a_t
  The environment evolves to s_{t+1}
  The agent receives reward r_t
end for

Markov chain

- A key property of this framework is that the future state s_{i+1} depends only on (s_i, a_i)
  - It does not depend on $(s_{i-1}, a_{i-1}), (s_{i-2}, a_{i-2}), \ldots$



  - All information about the past is contained inside s_i

Markov chain

- Let's make this argument a little more formal
  - Mathematical base for how to model "states"

Definition (Markov Chain).

Suppose that we have three random variables X, Y, Z. We say that they form a Markov chain $X - Y - Z$, whenever

$$P(Z \mid Y, X) = P(Z \mid Y)$$

That is, Z and X are conditionally independent, given Y

Markov chain

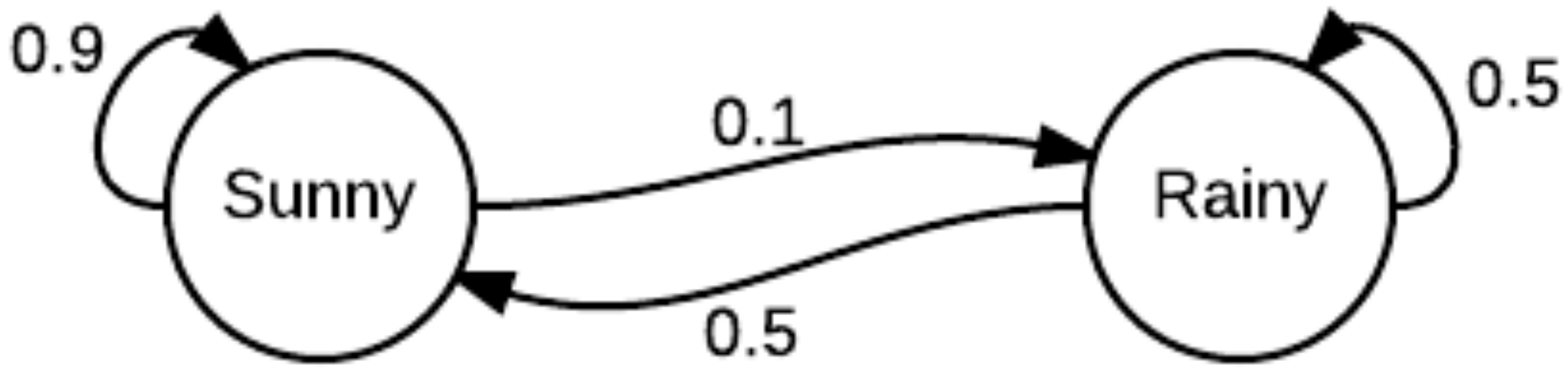
Example (Weather).

Suppose that in Pohang, it is only “sunny” ☀️ or “rainy” 🌧️

- If it was sunny yesterday, then it will stay sunny with 90% chance
- If it was rainy yesterday, then it will stay rainy with 50% chance

Then, letting

- X: Yesterday’s weather
- Y: Today’s weather
- Z: Tomorrow’s weather



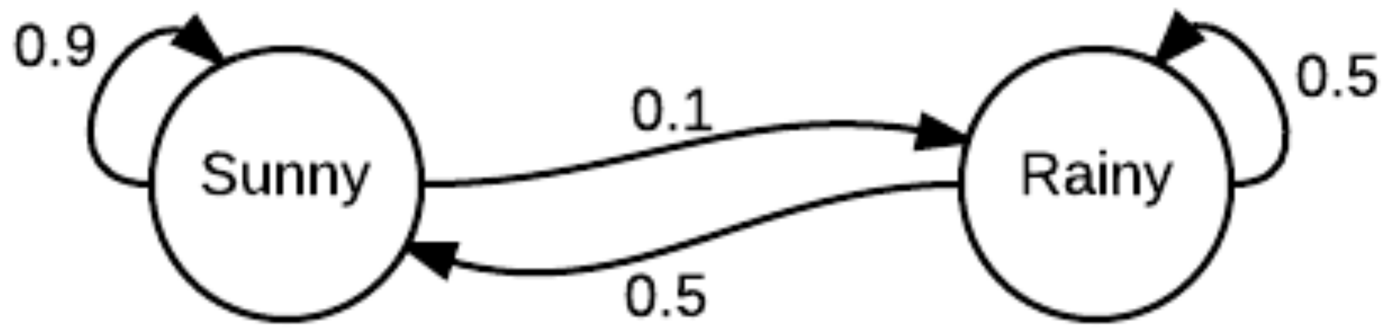
We have $X - Y - Z$ a Markov chain.

Markov chain

- For this weather model, let $p_1^{(i)}$ be the probability of being sunny on i-th day and $p_2^{(i)}$ be the probability of being rainy.
  - Then, we know that

State Transition Matrix

$$\begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \begin{bmatrix} p_1^{(i)} \\ p_2^{(i)} \end{bmatrix} = \begin{bmatrix} p_1^{(i+1)} \\ p_2^{(i+1)} \end{bmatrix}$$



  - If there is some p_1, p_2 such that

$$\begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

then the weather distribution is stationary — stays same everyday

Markov decision process

- The evolution of environment can be formalized as follows:

Definition (Markov Decision Process; MDP).

A Markov decision process is defined as a tuple $M = (S, A, p, r)$, where

- t: time clock
- S: state space
- A: action space
- $p(y \mid x, a)$: transition probability, with

$$p(y \mid x, a) = \mathbf{P}(s_{t+1} = y \mid s_t = x, a_t = a)$$

- $r(x, y, a)$: reward of transition (x, y, a)

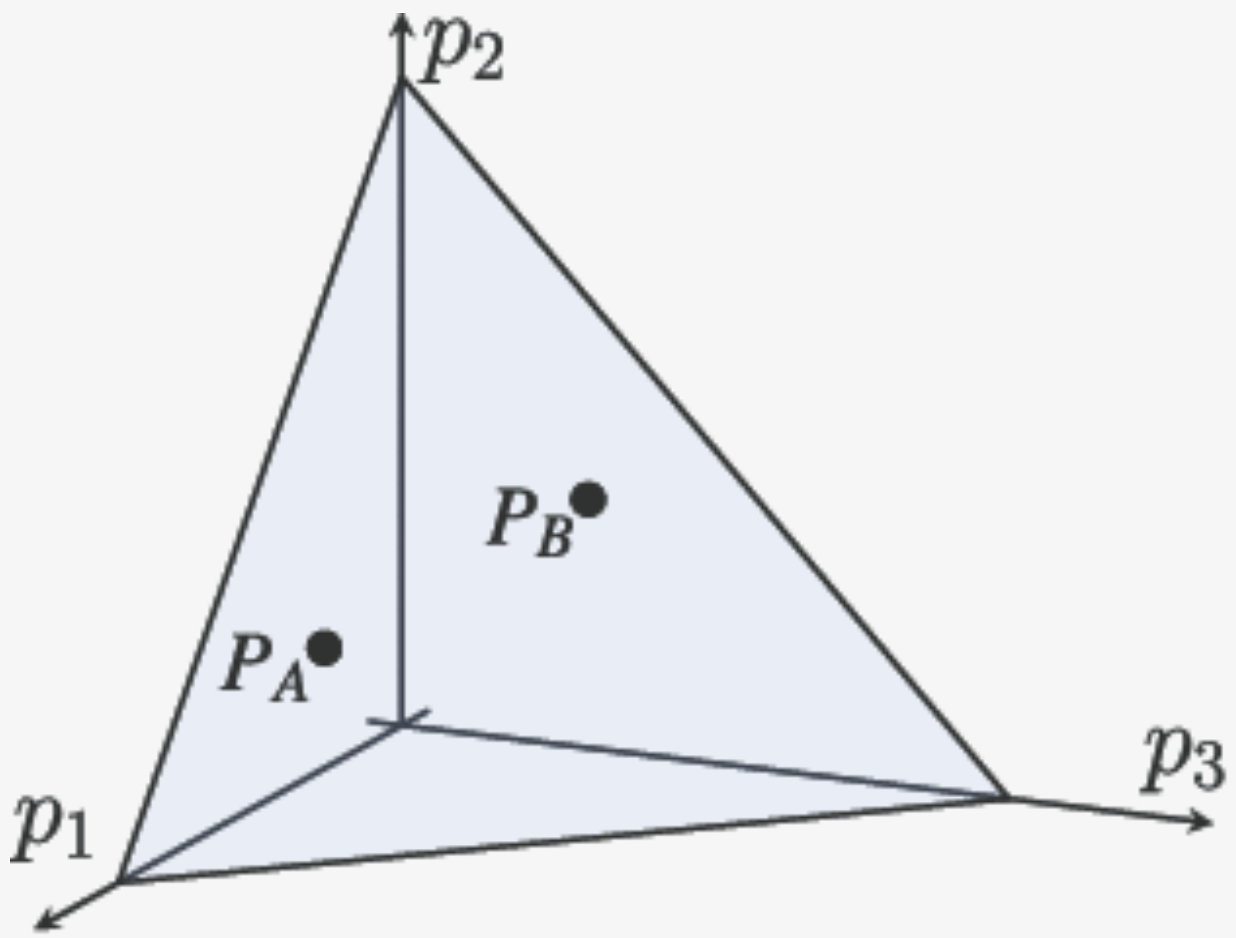
Policy

- The behavior of agents can be characterized by policy

Definition (Policy).

A decision rule π_t can be:

- Deterministic: $\pi_t : S \to A$
- Stochastic: $\pi_t : S \to \Delta(A)$



A policy can be:

- Non-stationary: $\pi = (\pi_0, \pi_1, \ldots)$
- Stationary: $\pi = (\pi, \pi, \ldots)$

MDP + Stationary Policy

- **Remark.** Suppose that …
  - Our state evolves as an MDP
  - Our policy is stationary

Then, we have a Markov chain of state S and transition probability

$$p(y \mid x) = p(y \mid x, \pi(x))$$

- This framework is powerful enough, capturing many sequential decision-making problems.

Interaction protocol

- The environment may differ by:
- **Controllability.** Fully (e.g., chess)
  Partially (e.g., portfolio optimization)
- **Uncertainty.** Deterministic (e.g., chess)
  Stochastic (e.g., dice-involving)
- **Reactive.** Adversarial (e.g., chess)
  Fixed (tetris)
- **Observability.** Full (e.g., chess)
  Partial (e.g., starcraft)
- **Availability.** Known (e.g., chess)
  Unknown (e.g., robotics)

Next class

- Markov reward process
- Bellman equation
- Basic algorithms

</lecture 22>