CS3243: Introduction to Artificial Intelligence

Semester 2, 2019/2020

Reinforcement Learning

AIMA Chapter 21

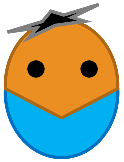
Based in part on slides from Zemel, Urtason and Fidler (2016), as well as Li, Johnson and Yeung (2017)

Outline

- Introduction to Learning Agents
- Reinforcement Learning Formulation
- Agent policy and optimal policies
- Learning an optimal policy
- Q-learning

Supervised Learning

Agent



=dog

=dog

=cat

=cat

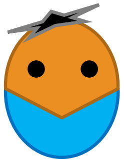
=cat

=dog

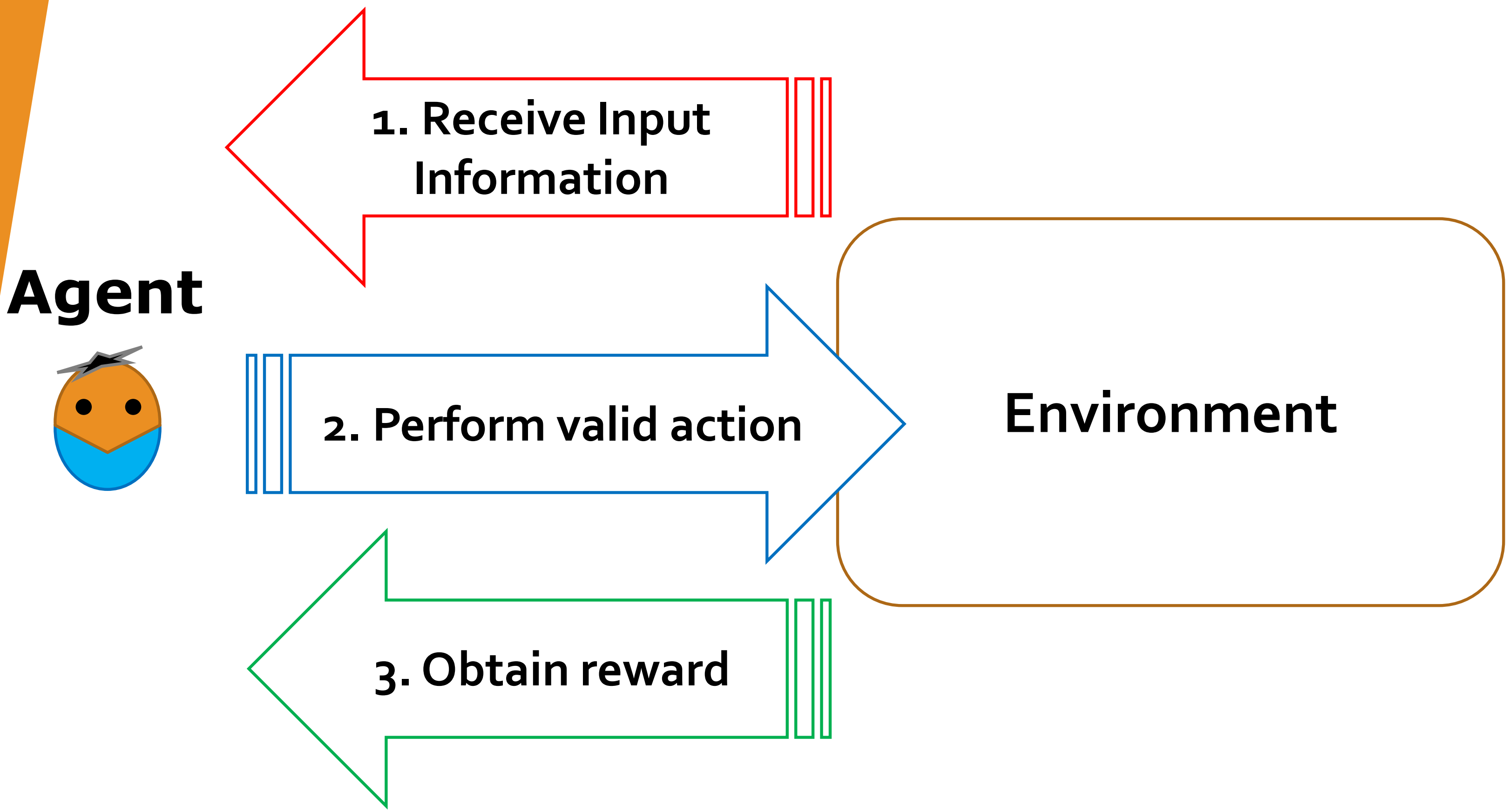
=?

Unsupervised Learning

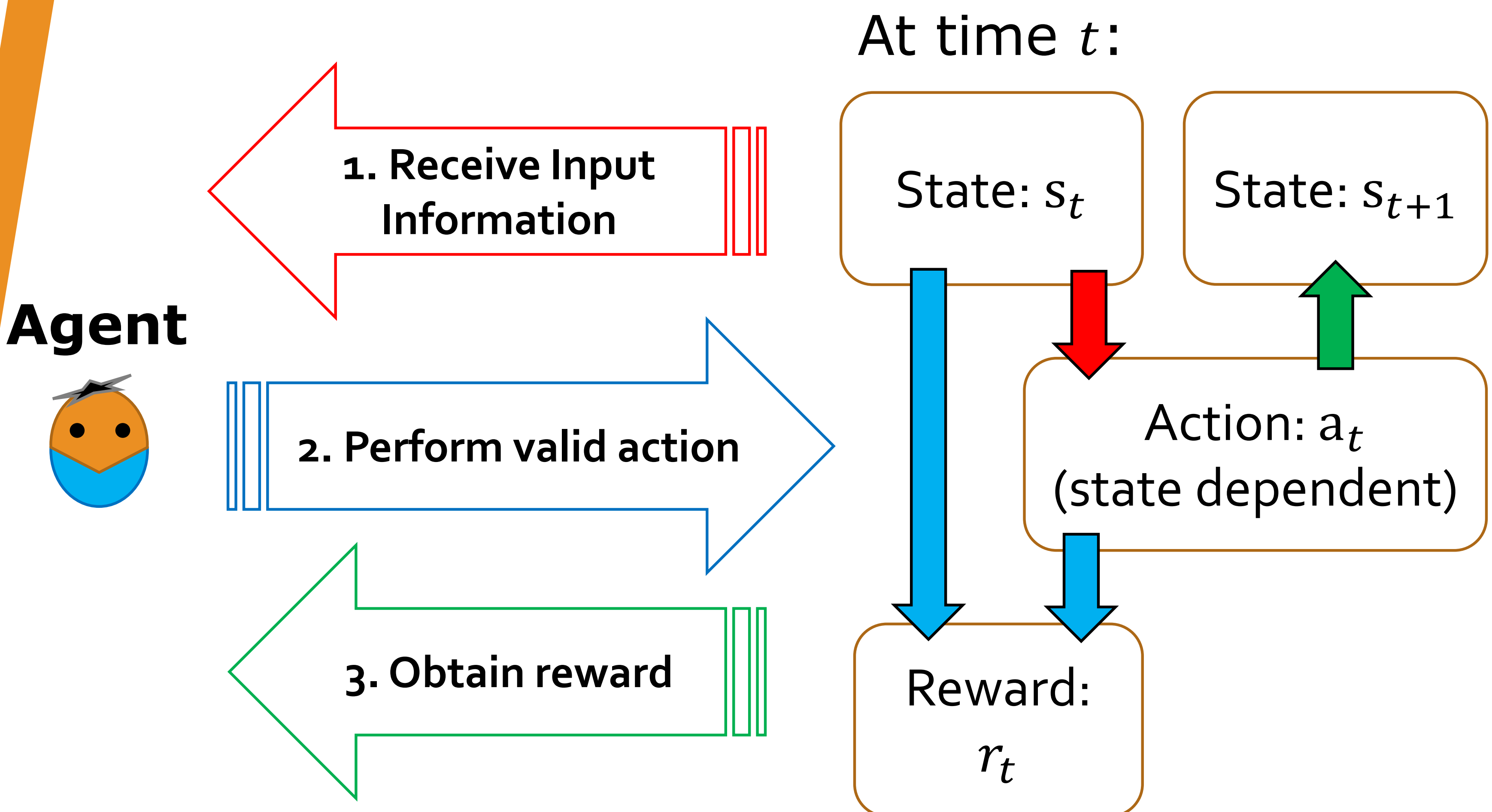
Agent



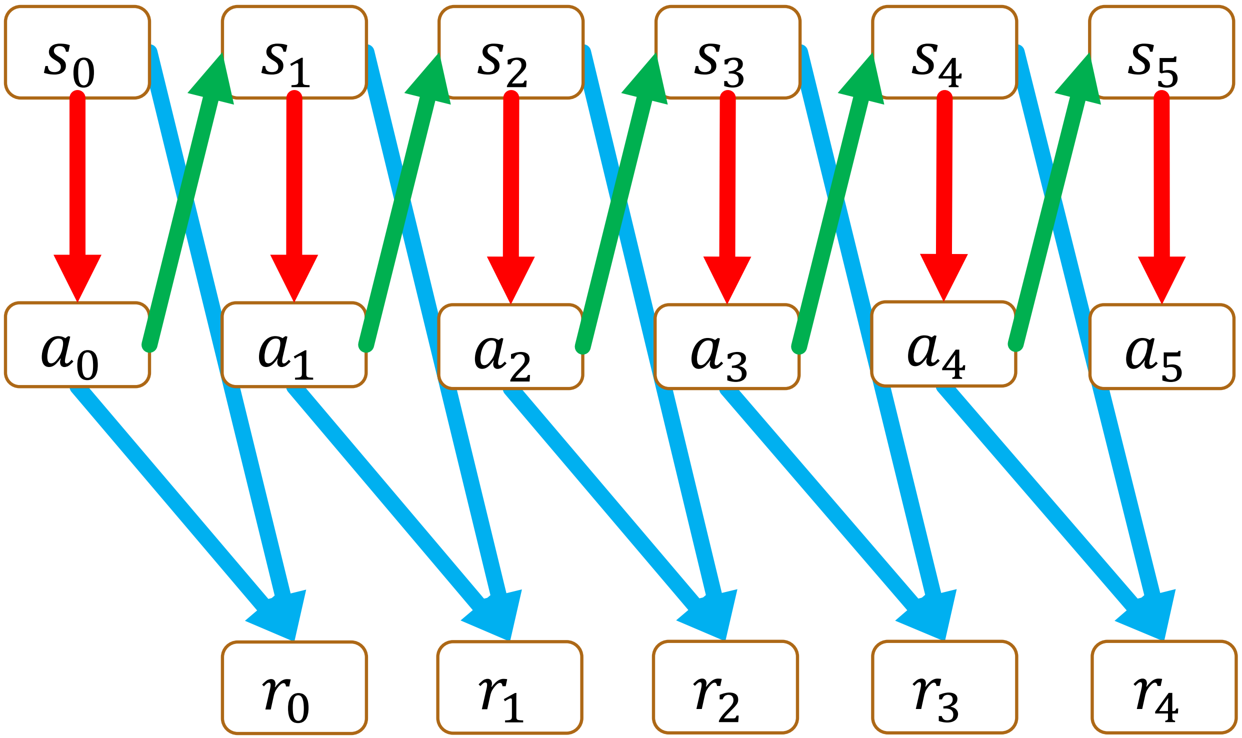
Reinforcement Learning



Reinforcement Learning

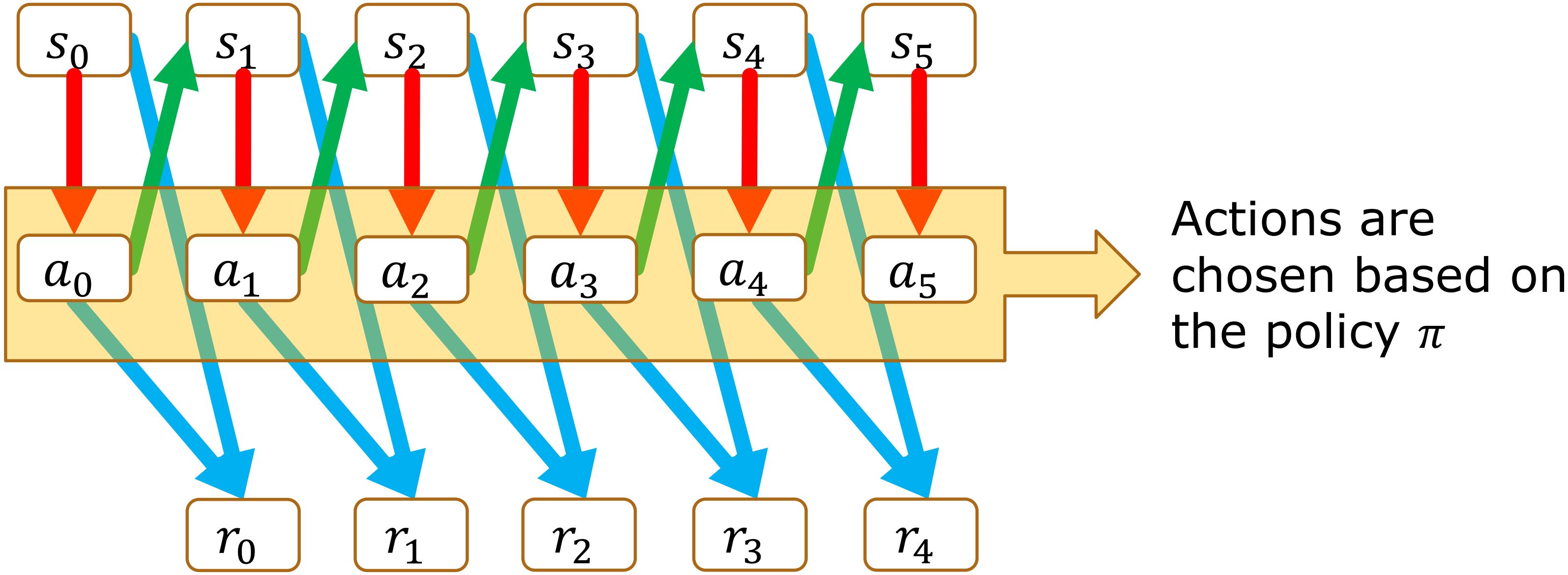


Reinforcement Learning



Markov property: next state is determined only based on current action and state, not entire sequence

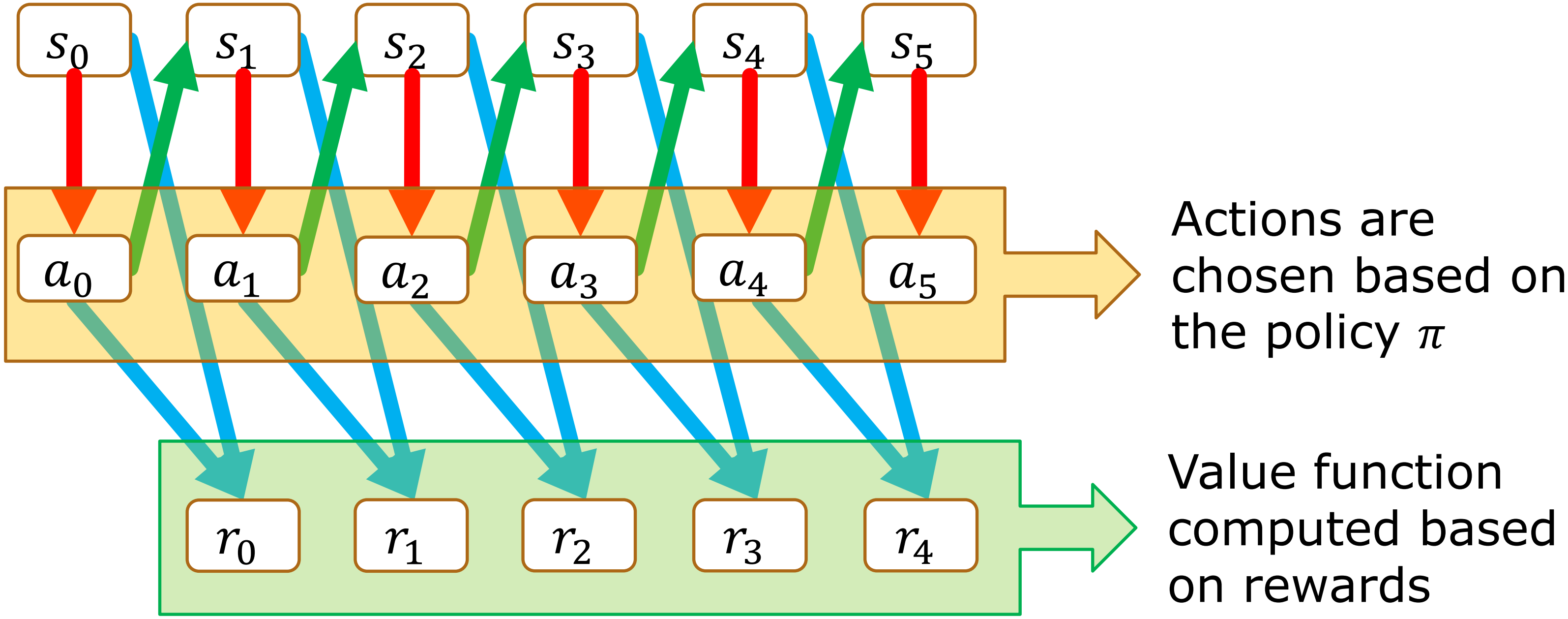
Policy



Policy π determines agent's behavior, i.e. actions

- Deterministic policy: $a_t = \pi(s_t)$
- Stochastic policy: $\pi(a \mid s_t) = \Pr[\, a_t = a \mid s_t \,]$

Value Function



Value function tries to predict how good a state is, given the rewards.

$$V^{\pi}(s_t) = r_t(a_t, s_t) + \gamma r_{t+1}(a_{t+1}, s_{t+1}) + \gamma^2 r_{t+2}(a_{t+2}, s_{t+2}) + \cdots$$

$$= \sum_{\ell=0}^{\infty} \gamma^{\ell} r_{t+\ell}(a_{t+\ell}, s_{t+\ell})$$

Value Function

Value function tries to predict how good a state is, given the rewards.

$$V^{\pi}(s_t) = r_t(a_t, s_t) + \gamma r_{t+1}(a_{t+1}, s_{t+1}) + \gamma^2 r_{t+2}(a_{t+2}, s_{t+2}) + \cdots$$
$$= \sum_{\ell=0}^{\infty} \gamma^{\ell} r_{t+\ell}(a_{t+\ell}, s_{t+\ell})$$

The value γ (a value between 0 and 1) is called the **discount rate**.

- High value of γ – long-sighted agent, cares about future rewards.
- Low value of γ – agent is greedy, who cares about the future?

Value Function – The Challenge

We want to choose a **value maximizing policy**.

However, we are missing two key bits of information:

The rewards of unobserved states.

What the next state will be when we take an action.

The challenge:

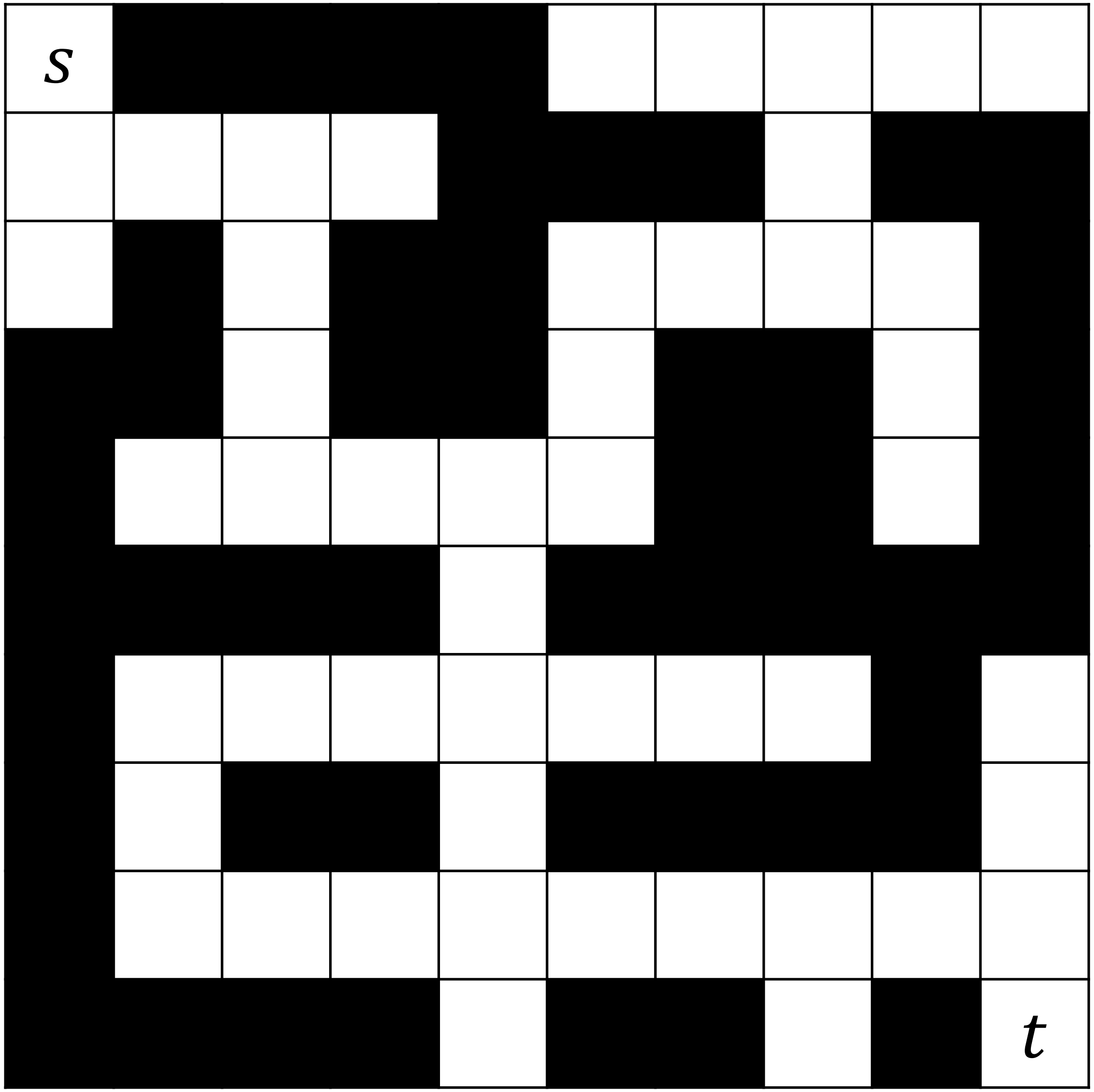
Infer rewards/state transitions as we go along

...while maximizing revenue.

State:
(x, y) position

Actions:
Up/Down/Left/Right

Reward:
-1 per time step
$(+10$ for reaching goal)

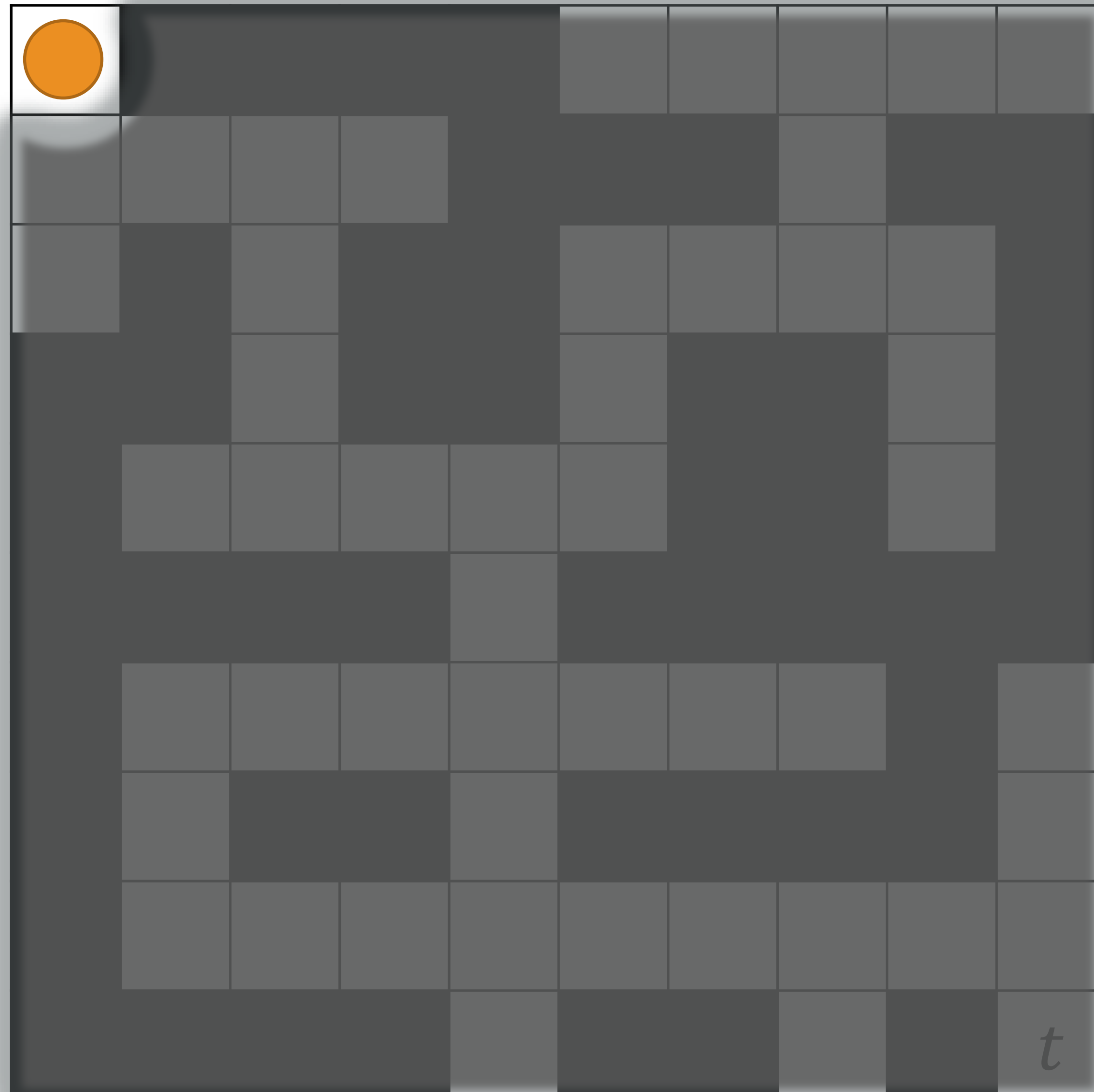


Policy:
Direction to go from each position (can be randomized).

Value Function:
Total reward of policy execution from given state.

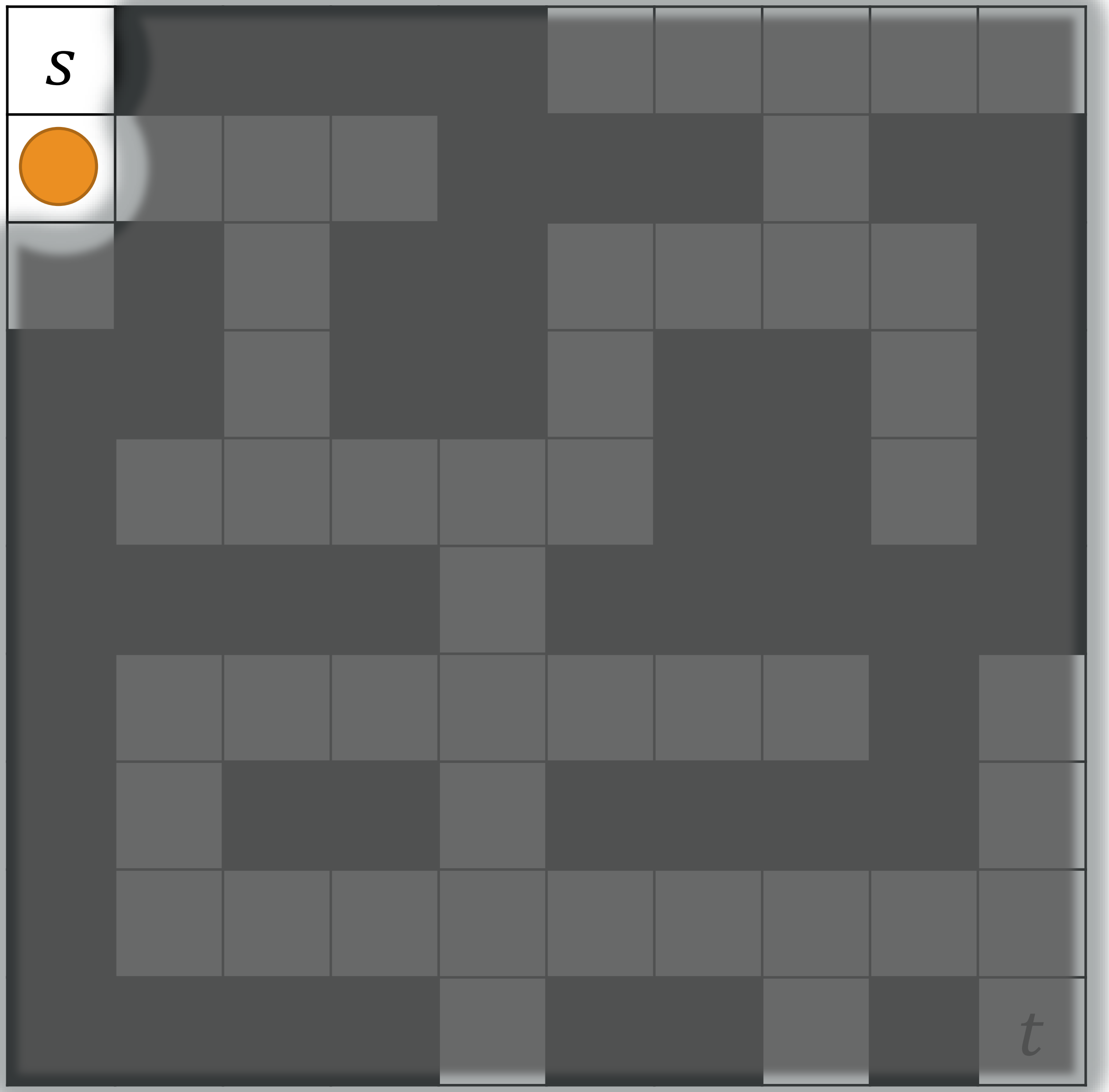
State:
(1,1) position

Actions:



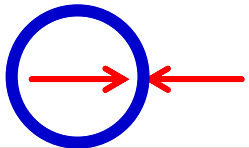
State:
(2,1) position

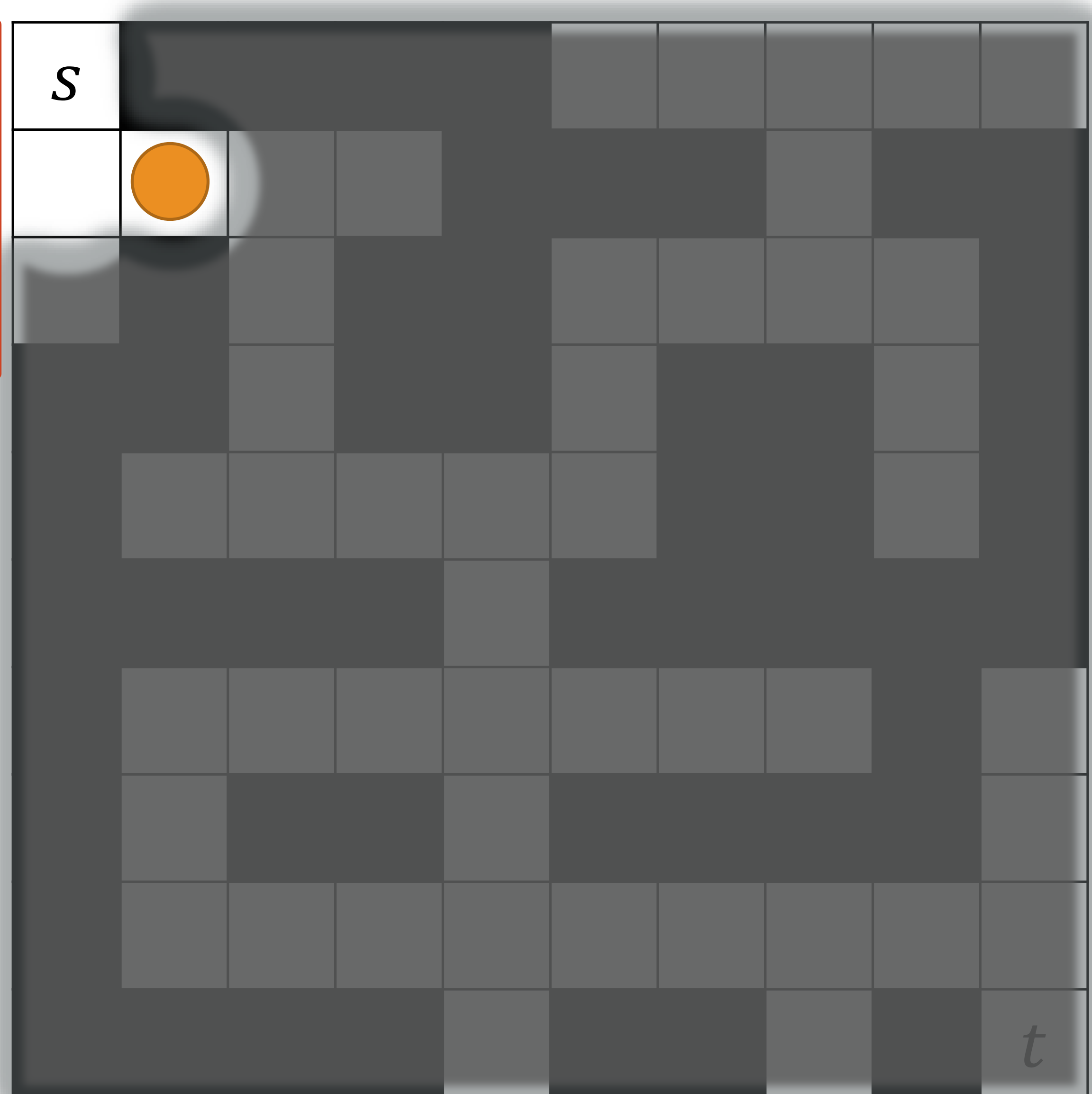
Actions:
↑ ↓ →



State:
(2,2) position

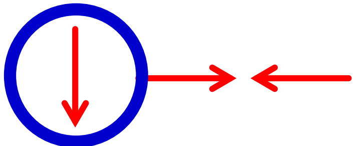
Actions:

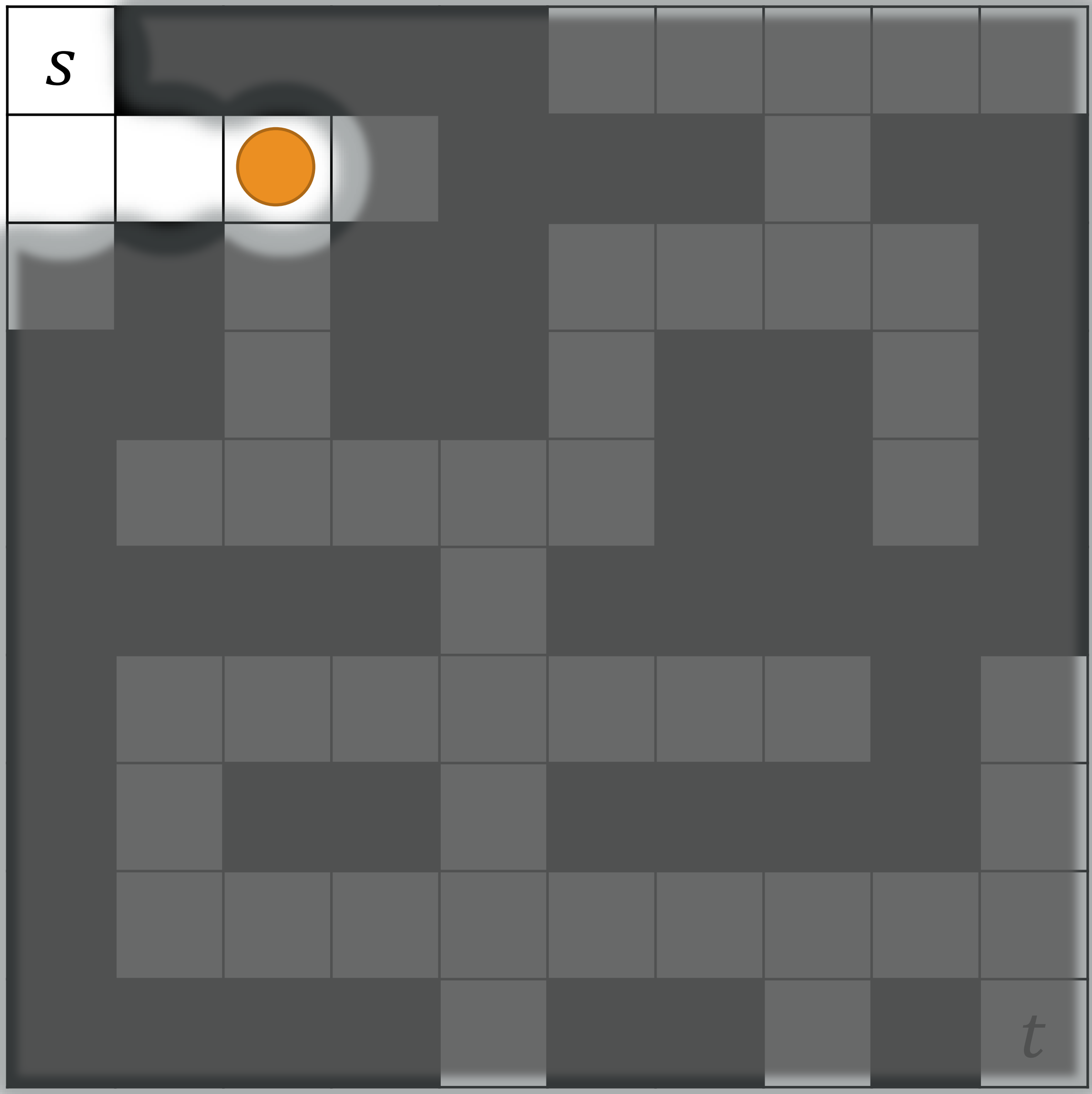




State:
$(2,3)$ position

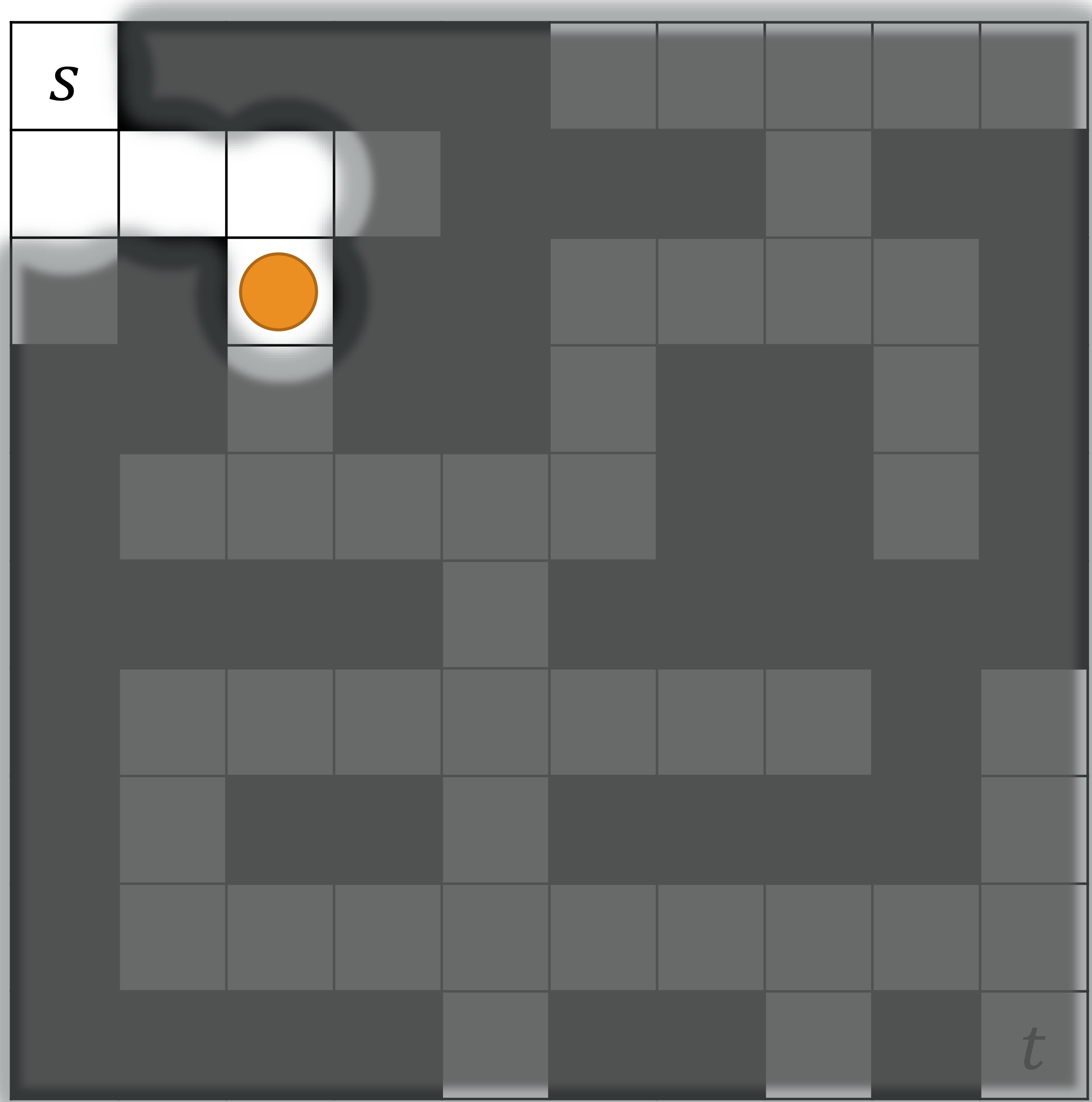
Actions:





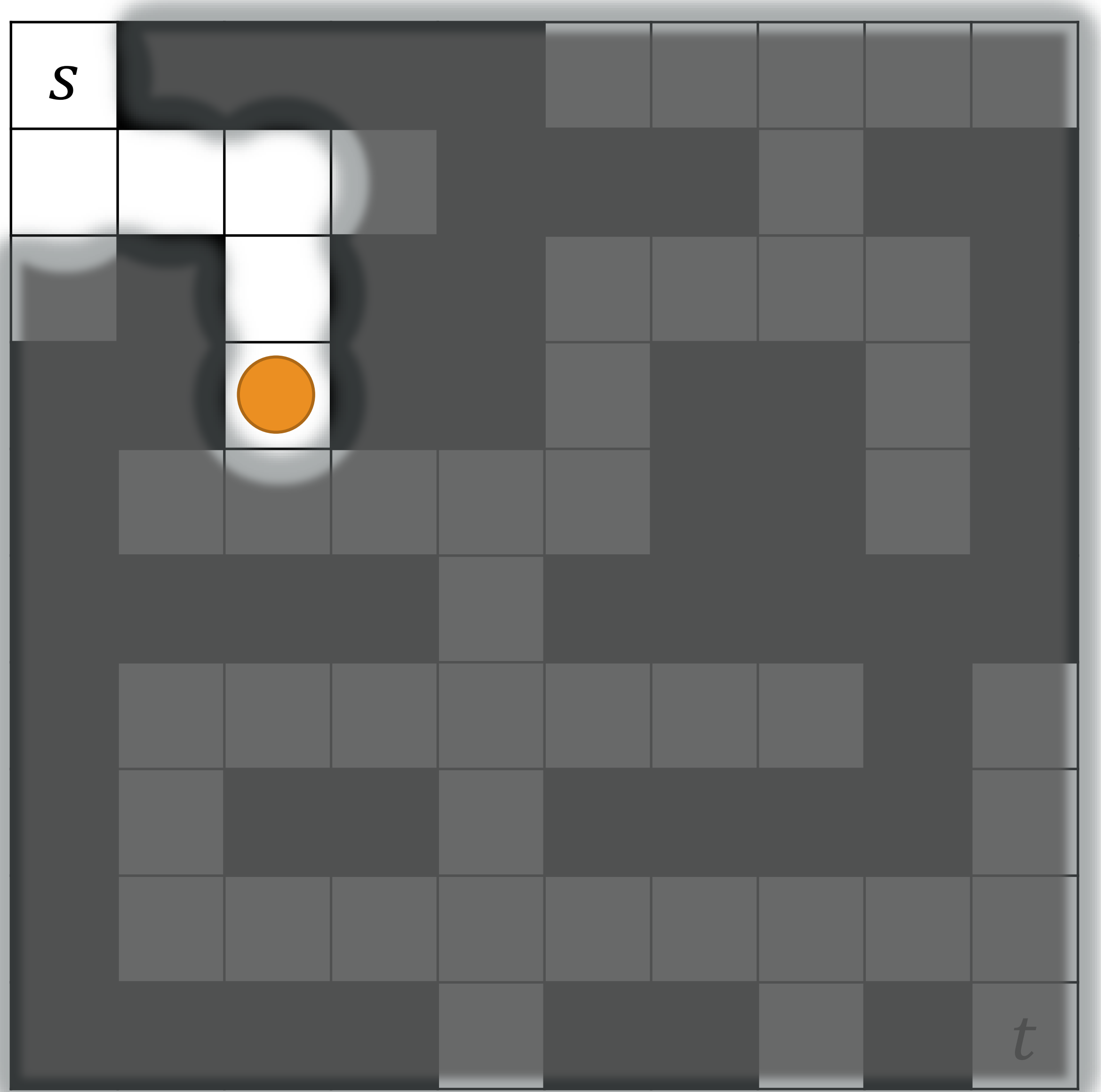
State:
(3,3) position

Actions:



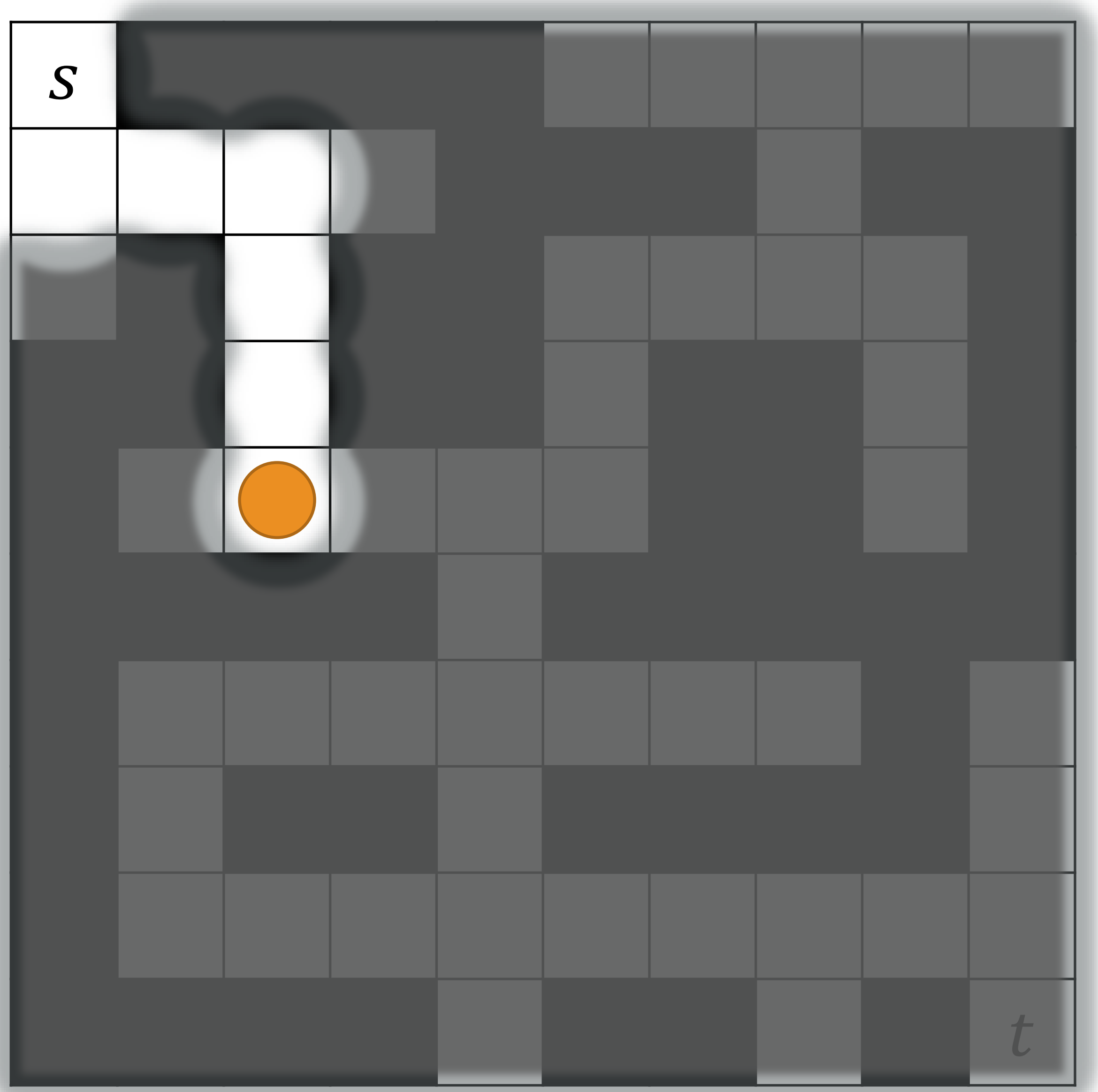
State:
$(4,3)$ position

Actions:
↑ ↓



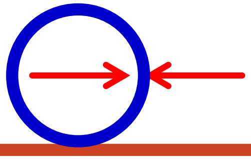
State:
(5,3) position

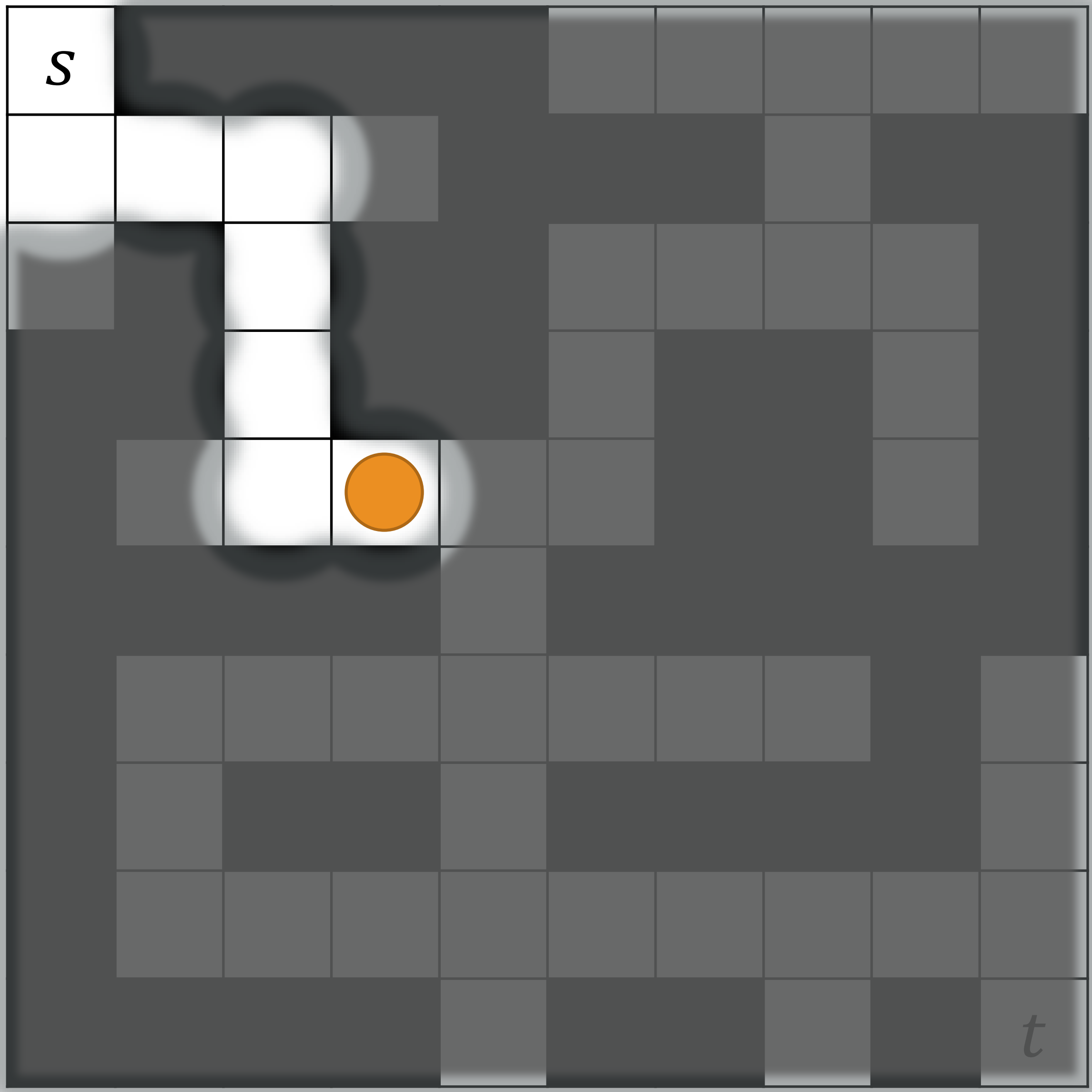
Actions:



State:
(5,4) position

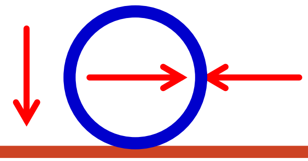
Actions:

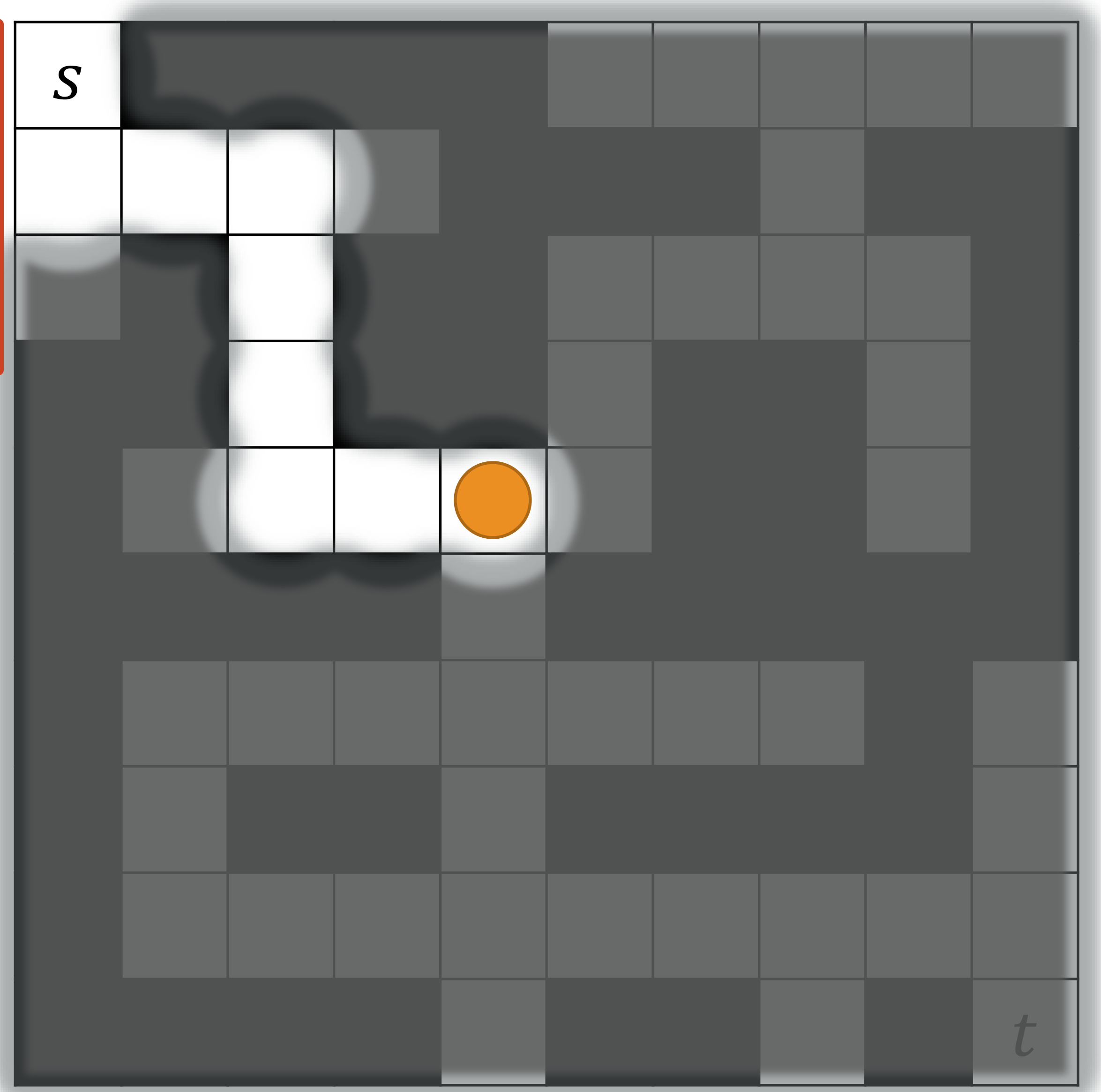




State:
(5,5) position

Actions:

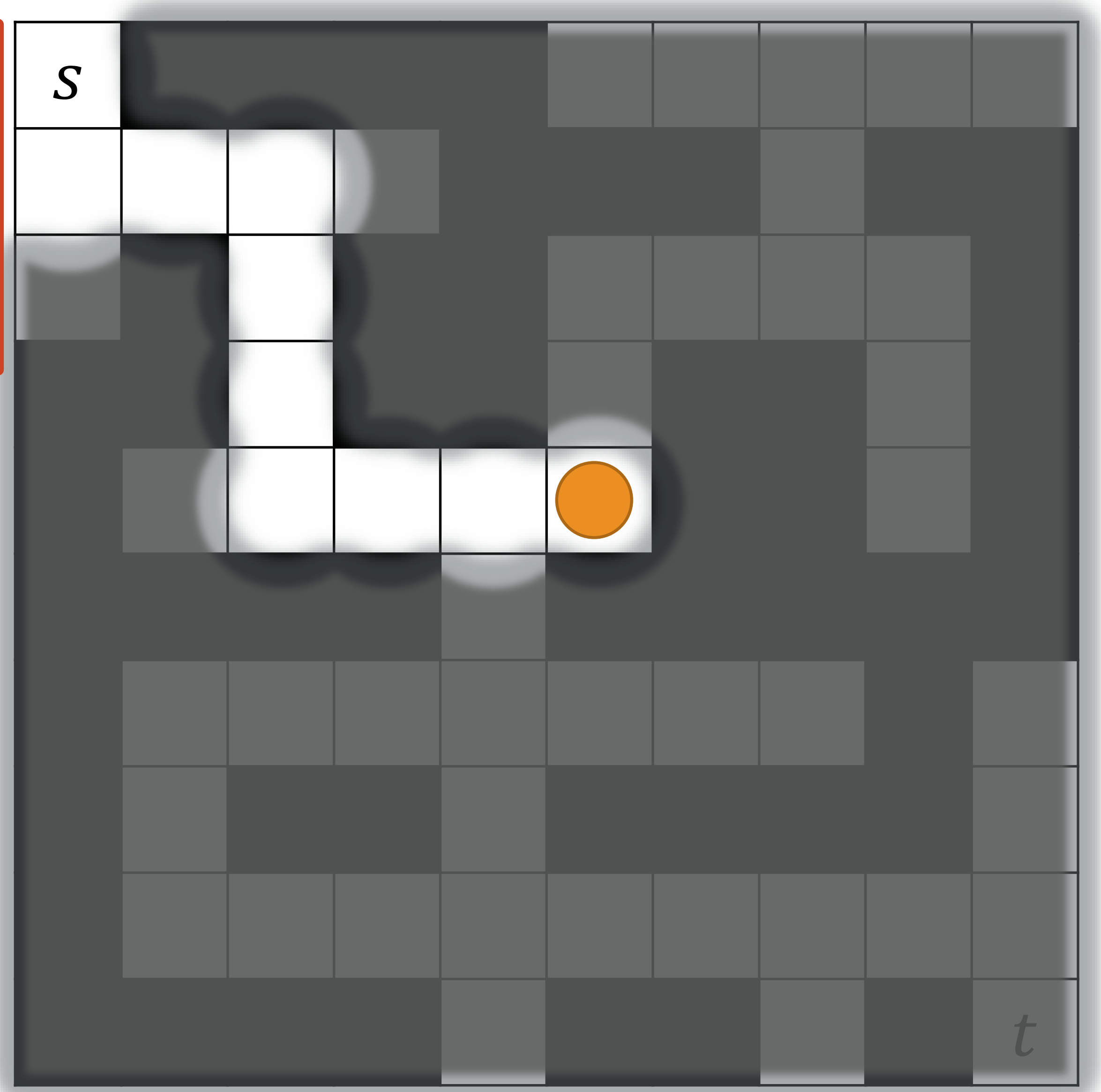




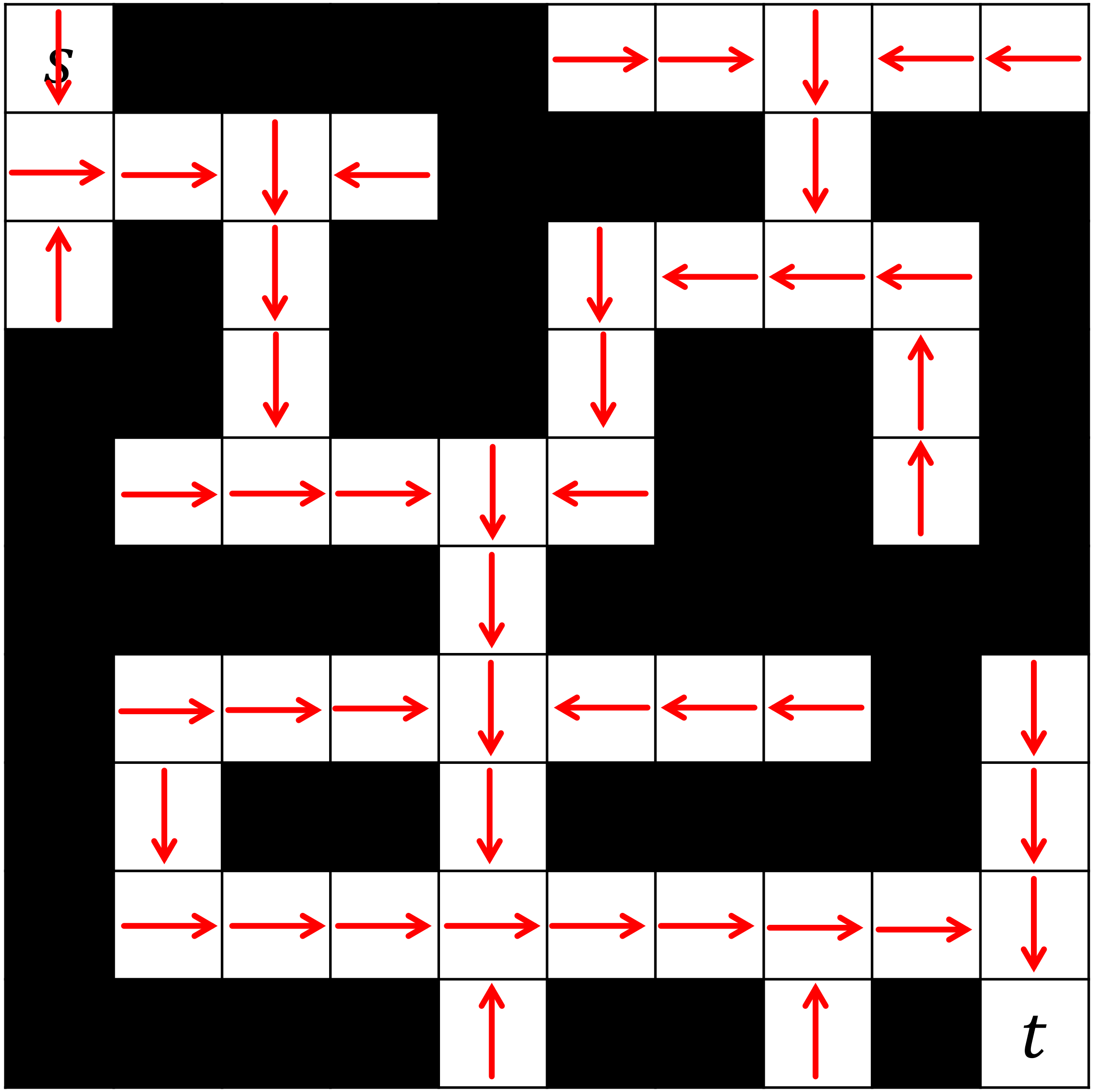
State:
(5,6) position

Actions:

↑ ←



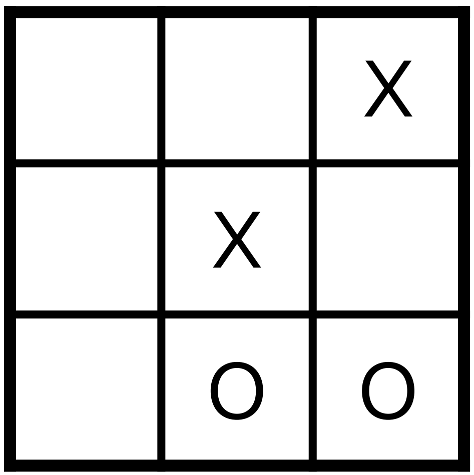
Optimal policy: spend as little time in the maze as possible, get to the goal.



Value Function: Discounted reward if starting from this state.

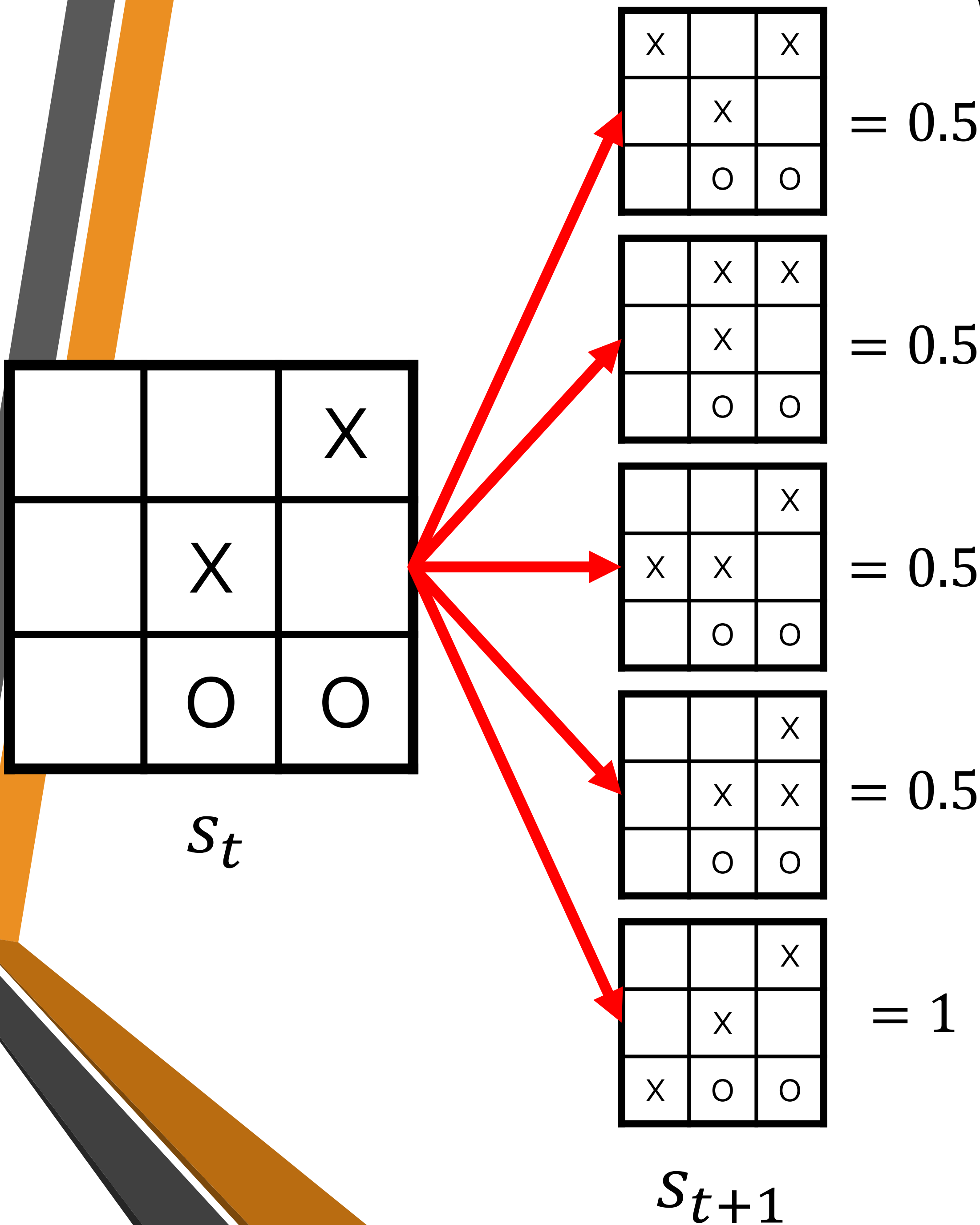
Numbers shown for discount factor $\gamma = 1$

-8					-9	-8	-7	-8	-9
-7	-6	-5	-6				-6		
-8		-4			-3	-4	-5	-6	
		-3			-2			-7	
	-3	-2	-1	0	-1			-8	
				1					
	-1	0	1	2	1	0	-1		7
	0			3					8
	1	2	3	4	5	6	7	8	9
				3			6		10



Reward:
$-1, 0, +1$ lose/tie/win (seen only on final move)
State:
Current positions of X's and O's on board
Policy:
What moves to make in given position?
Value function:
predict future reward given state.

Win Prob.



- All values are 0/0.5/1 initially
- At each turn choose move with highest win prob.
- Update table entries based on the game outcome.
- Value function will eventually represent true win probabilities

Alternatively:
- pick with probability proportional to win prob
- make a random choice.

Update strategy is critical:
- Necessary for convergence to optimal strategy.
- Some will work better than others.

Markov Decision Problem

Completely specified by a distribution:

$$\Pr[\, s_{t+1} = s; r_{t+1} = r \mid s_t, a_t \,]$$

"What is the next state and reward given current state and action?"

Planning: given an MDP, compute optimal policy

Learning: don't know the MDP, learn a strategy.

Markov Decision Problem

Given complete knowledge of MDP:

$$\Pr[\, s_{t+1} = s; r_{t+1} = r \mid s_t, a_t \,]$$

The optimal policy is deterministic - select optimal action in each state.

But… agent doesn't know the underlying MDP.

Needs to perform trial-and-error, interact with environment.

… and not lose too much reward along the way.

Learning Optimal Policies

We want to maximize (discounted) revenue.

Note that:

$$V^{\pi}(s_t) = r_{t+1} + \gamma V^{\pi}(s_{t+1})$$

Value now

Reward now

Value later

Optimal policy: $\pi(s)$ is an action in

$$\mathrm{argmax}_{\mathrm{a}}\{r(s,a) + \gamma V(\delta(s,a)\}$$

$\delta(s,a)$: next state given current state and action

Reward now

Value later

Learning Optimal Policies

Optimal policy: $\pi^*(s)$ is an action in

$$\text{argmax}_{\mathrm{a}}\{r(s,a) + \gamma V^*(\delta(s,a)\}$$

Its value is:

$$V^*(s_t) = r_{t+1} + \gamma V^*(s_{t+1})$$

We could identify optimal policy $\pi^*(s)$ if we knew $r(s,a)$ and $\delta(s,a)$.

We don't. Cannot choose optimal actions.

Q-Learning

Define:

$$Q^{\pi}(s,a) = r(s,a) + \gamma V^{\pi}\big(\delta(s,a)\big)$$

$Q^{\pi}(s,a)$ is the utility we obtain if we take action a at state s, and then follow the policy π from then on.

Define:

$$Q^{*}(s,a) = r(s,a) + \gamma V^{*}\big(\delta(s,a)\big)$$

$Q^{*}(s,a)$ is the utility we obtain if we take action a at state s, and then follow the optimal policy from then on.

Q-Learning

Q^* and V^* are very similar:

$$V^*(s) = \max_a Q(s,a)$$

Therefore:

$$\begin{aligned} Q^*(s_t,a_t) &= r(s_t,a_t) + \gamma V^*\big(\underbrace{\delta(s_t,a_t)}_{= s_{t+1}}\big) \\ &= r(s_t,a_t) + \gamma \max_{\mathrm{a}} Q(s_{t+1},a) \end{aligned}$$

Q-Learning – Value Iteration Algorithm

1. Initialize $\hat{Q}(s,a) \leftarrow 0$ for all s,a.
2. Start at s_0
3. For $t = 0, \ldots, \infty$:
    1. For every a: $\hat{Q}(s_t,a) \leftarrow r(s_t,a) + \gamma \max_{a'} \hat{Q}_i(\delta(s_t,a),a')$
    2. Pick action a_t maximizing $\hat{Q}(s_t,a)$
    3. Set $s_{t+1} \leftarrow \delta(s_t,a_t)$

Key observation: when $r(s,a) \geq 0$ and $\hat{Q} = 0$, $\hat{Q}(s,a) \leq Q^*(s,a)$ always.

In other words – we always underestimate the optimal Q values.

They always increase at every iteration, thus we converge to Q^*… and in particular to an optimal policy!

Q-Learning – Value Iteration Algorithm

1. Initialize $\hat{Q}(s, a) \leftarrow 0$ for all s, a.
2. Start at s_0
3. For $t = 0, \dots, \infty$:
   1. For every a: $\hat{Q}(s_t, a) \leftarrow r(s_t, a) + \gamma \max_{a'} \hat{Q}_i(\delta(s_t, a), a')$
   2. Pick action a_t maximizing $\hat{Q}(s_t, a)$
   3. Set $s_{t+1} \leftarrow \delta(s_t, a_t)$

Problem:

We ignore current $\hat{Q}(s_t, a)$ value in computation.

$r(s, a)$ can be stochastic, as is $\delta(s, a)$.

One bad experience can result in bad underestimate of $Q^*(s, a)$.

Q-Learning – Value Iteration Algorithm

1. Initialize $\hat{Q}(s, a) \leftarrow 0$ for all s, a.
2. Start at s_0
3. For $t = 0, \dots, \infty$:
   1. For every a: $\hat{Q}(s_t, a) \leftarrow r(s_t, a) + \gamma \max_{a'} \hat{Q}_i(\delta(s_t, a), a')$
   2. Pick action a_t maximizing $\hat{Q}(s_t, a)$
   3. Set $s_{t+1} \leftarrow \delta(s_t, a_t)$

Solution:

We need to maintain value of $\hat{Q}_i$ stable as we observe it more.

Change update rule:

$$\hat{Q}(s_t, a) \leftarrow (1 - \alpha_t)\hat{Q}(s_t, a) + \alpha_t \left(r(s_t, a) + \gamma \max_{a'} \hat{Q}_i(\delta(s_t, a), a') \right)$$

Where

$$\alpha_t = \frac{1}{1 + N[s_t, a]}$$

# of times action a taken at state s_t.

Q-Learning – Value Iteration Algorithm

1. Initialize $\hat{Q}(s, a) \leftarrow 0$ for all s, a.
2. Start at s_0
3. For $t = 0, \dots, \infty$:
   1. For every a: $\hat{Q}(s_t, a) \leftarrow r(s_t, a) + \gamma \max_{a'} \hat{Q}_i(\delta(s_t, a), a')$
   2. Pick action a_t maximizing $\hat{Q}(s_t, a)$
   3. Set $s_{t+1} \leftarrow \delta(s_t, a_t)$

Problem:

We greedily, deterministically, pick action a_t maximizing $\hat{Q}(s_t, a)$.

Deterministic algorithms can be 'fooled' by stochastic (or adversarial) inputs (more of this next lecture).

Q-Learning – Value Iteration Algorithm

1. Initialize $\hat{Q}(s,a) \leftarrow 0$ for all s, a.
2. Start at s_0
3. For $t = 0, \dots, \infty$:
    1. For every a: $\hat{Q}(s_t, a) \leftarrow r(s_t, a) + \gamma \max_{a'} \hat{Q}_i(\delta(s_t, a), a')$
    2. Pick action a_t maximizing $\hat{Q}(s_t, a)$
    3. Set $s_{t+1} \leftarrow \delta(s_t, a_t)$

Solution:

Pick action a_t **randomly**.

1. Totally randomly?
2. Randomly amongst current best actions?
3. Set $\Pr[\text{choosing action } a \mid s] = \frac{e^{\varepsilon \hat{Q}(s,a)}}{\sum_{a'} e^{\varepsilon \hat{Q}(s,a')}}$

$\Pr[\, a \mid s \,] \sim e^{\varepsilon \hat{Q}(s,a)}$