Lecture 8

Cezar Ionescu
06/07/2019

Administrative

- Homework from 29/06/2019 due now!
- Please complete and hand in the declarations of authorship.

Questions?

Solution to homework from lecture 6

$$y_1(v_{11}, v_{21}) = f(x_1 * v_{11} + x_2 * v_{21}) = -2$$
$$y_2(v_{11}, v_{21}) = f(x_1 * v_{12} + x_2 * v_{22}) = 2$$
$$o(w_1, w_2, y_1, y_2) = f(y_1 * w_1 + y_2 * w_2) = 2$$
$$E(w_1, w_2, v_{11}, \ldots) = (t - o)^2$$

$$\begin{aligned}
\partial E / \partial v_{11} &= E'(o) * \partial o / \partial v_{11} \\
&= 2*(t - o)*(-1) * \partial o / \partial v_{11} \\
&= 2*(-2)*(-1)*f'(z)*\partial g / \partial v_{11} \\
&= 4*1*\partial (y_1*w_1 + y_2*w_2) / \partial v_{11} \\
&= 4 * \partial (y_1*w_1 + y_2*w_2) / \partial y_1 * \partial y_1 / \partial v_{11} \\
&= 4 * w_1 * f'(z) * \partial (x_1*v_{11} + x_2*v_{21}) / \partial v_{11} \\
&= 4 * x_1 = -4
\end{aligned}$$

Solutions to homework from lecture 7

> *A smart traffic control system can adjust the delays of traffic lights in order to alleviate bottlenecks in the flow of vehicles and ensure a smooth driving experience. In order to achieve this, the system must anticipate how the delays will influence the evolution of the traffic pattern. We want to train this system using an idealised version of the matchbox system.*

We need to determine what actions to take depending on the configuration of traffic. If a sequence of actions leads to delays and traffic jams, we can decrease the likelihood of choosing those actions in the future.

Solutions to homework from lecture 7 (continued)

What are the states? What are the actions (sugar drops)?

A state is a configuration of the traffic system: positions of cars on the grid, their speeds, the state of the traffic lights, etc.

The actions are traffic light delays, so the sugar drops would stand for the possible traffic light delays. We would need many as many colors as possible delays (e.g., a colour for each of `10`, `20`, `30`, `40`, `50` seconds).

Solutions to homework from lecture 7 (continued)

What is the equivalent of a matchbox for this system?

A matchbox would have as label a configuration of the traffic system.

What could be a long-term goal for this system? What could be a short-term goal of the system?

A long-term goal would be to ensure a smooth driving experience for all participants. A short-term goal would be to eliminate a current traffic jam.

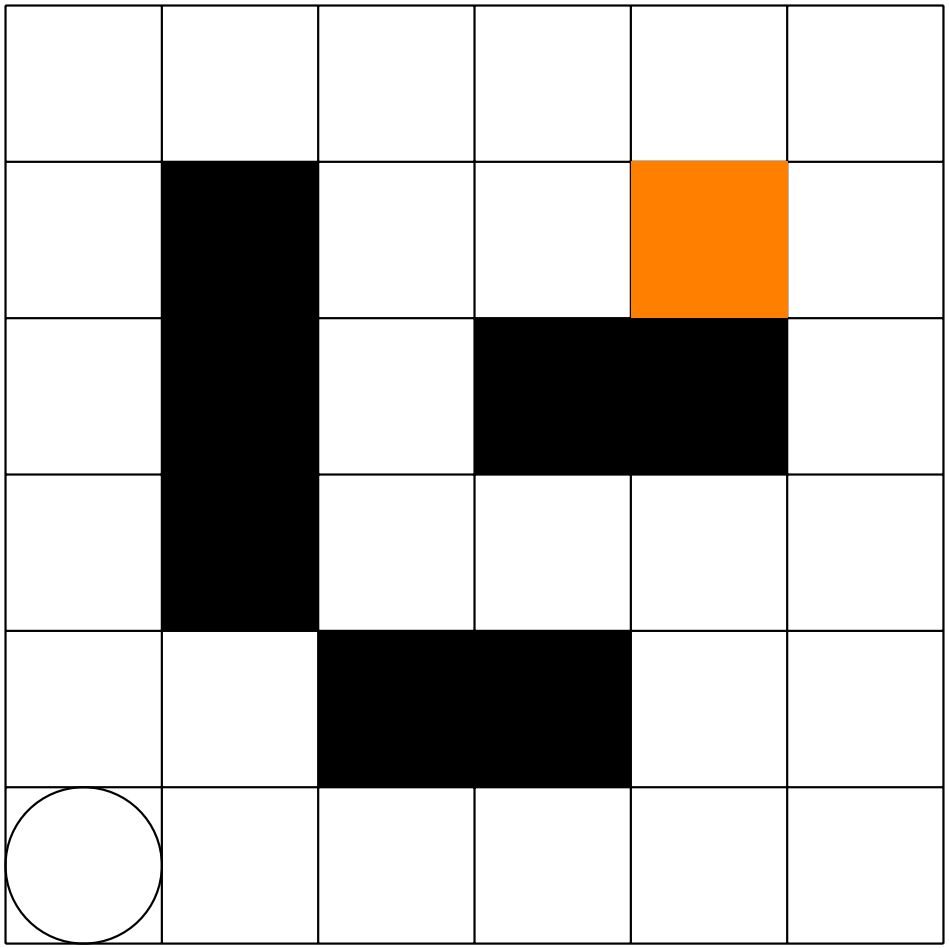
Solutions to homework from lecture 7 (continued)

How would the updating process work (how can bad actions be eliminated)?

We need a measure of the quality of traffic, for example a weighted sum that gives low scores to many stops, to time spent in traffic, build-up of traffic jams, etc. We can record the actions taking during a certain amount of time, such as an hour, or a day, and if the quality measure is not good enough, we eliminate the actions we have taken during this time.

Dynamic programming

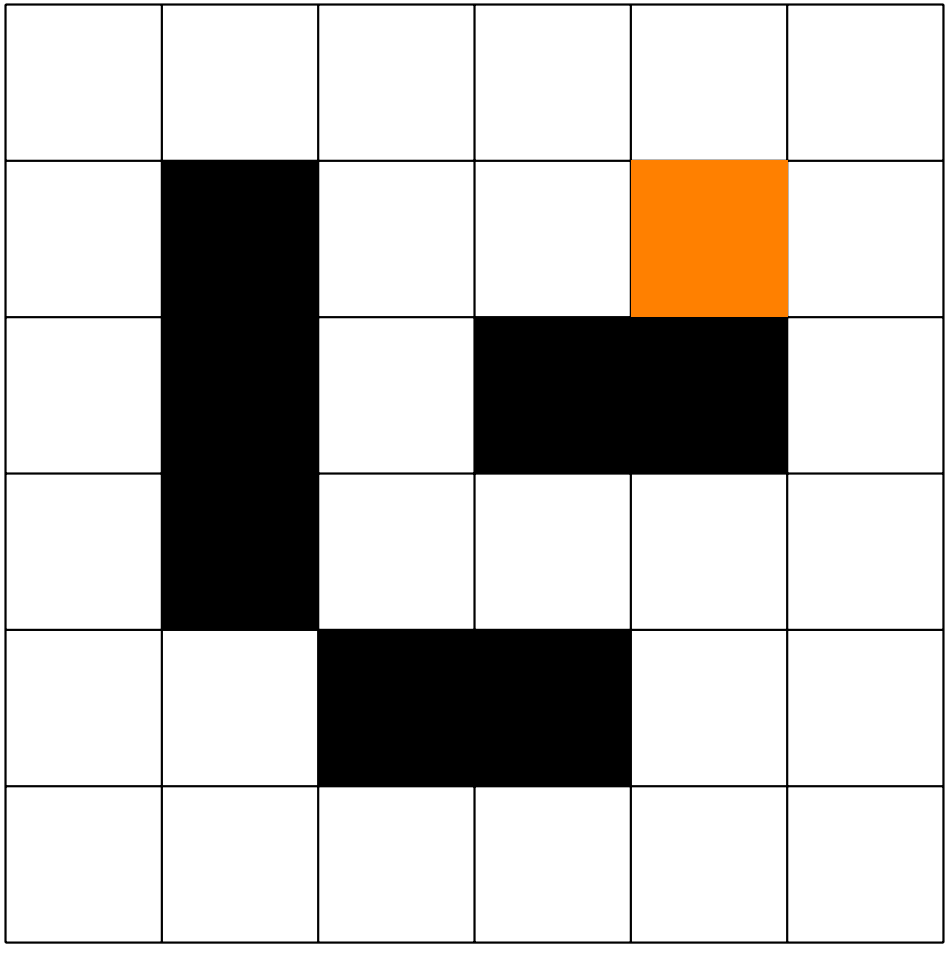
A maze



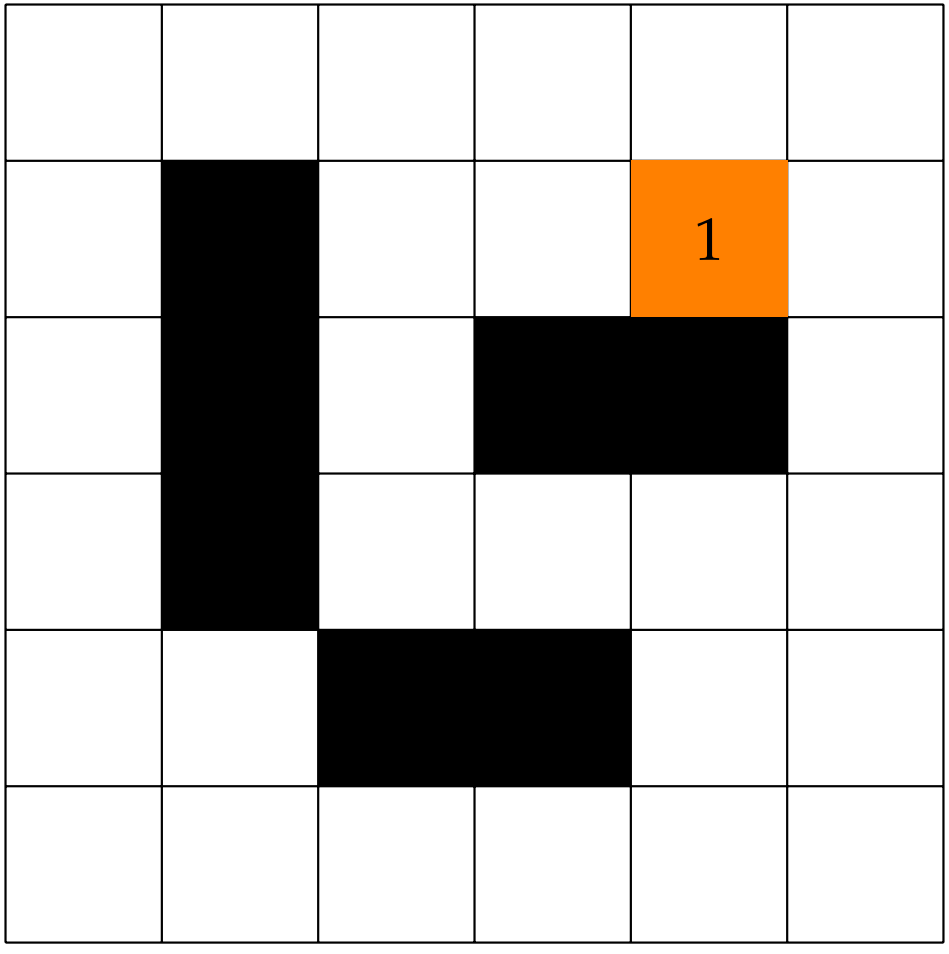
Value map

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

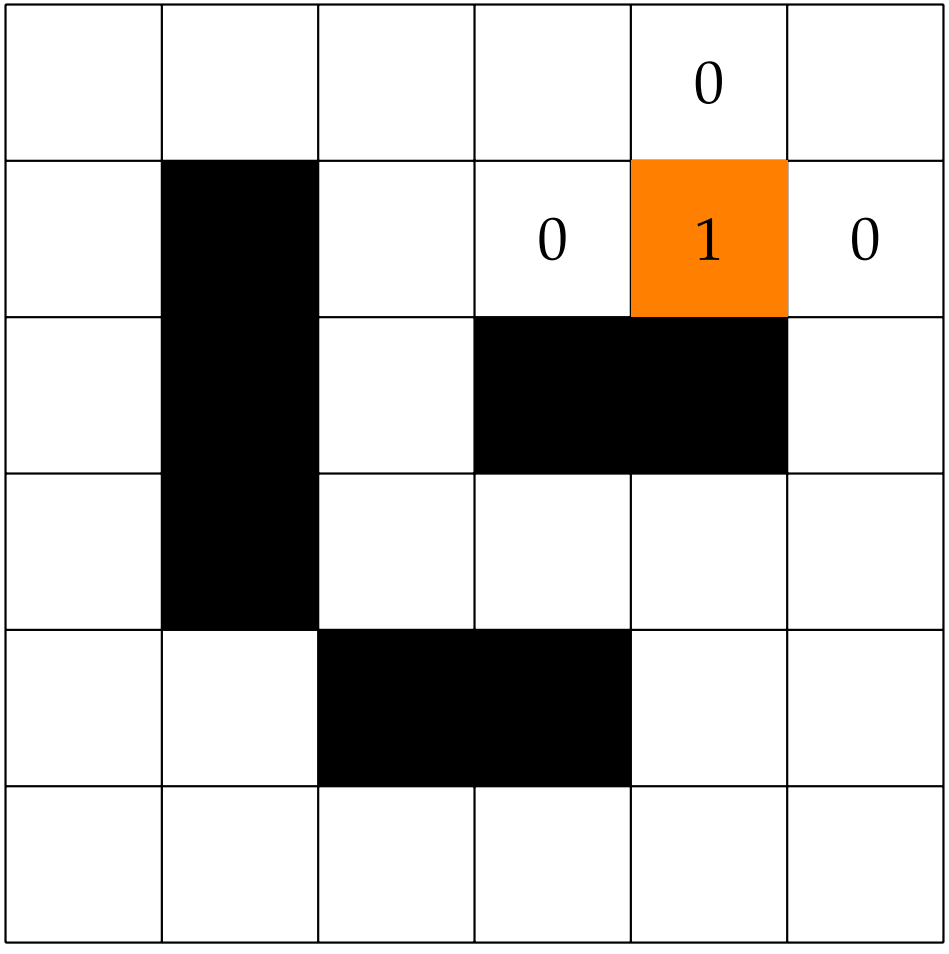
Dynamic programming



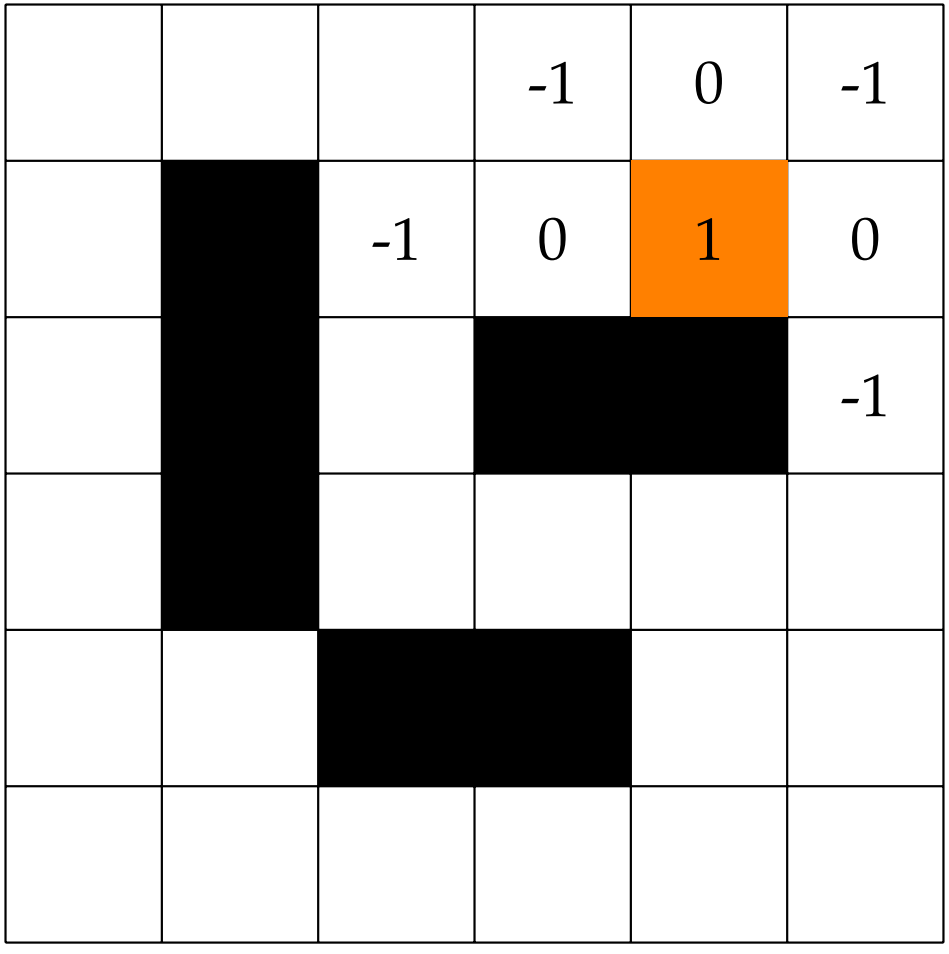
Dynamic programming



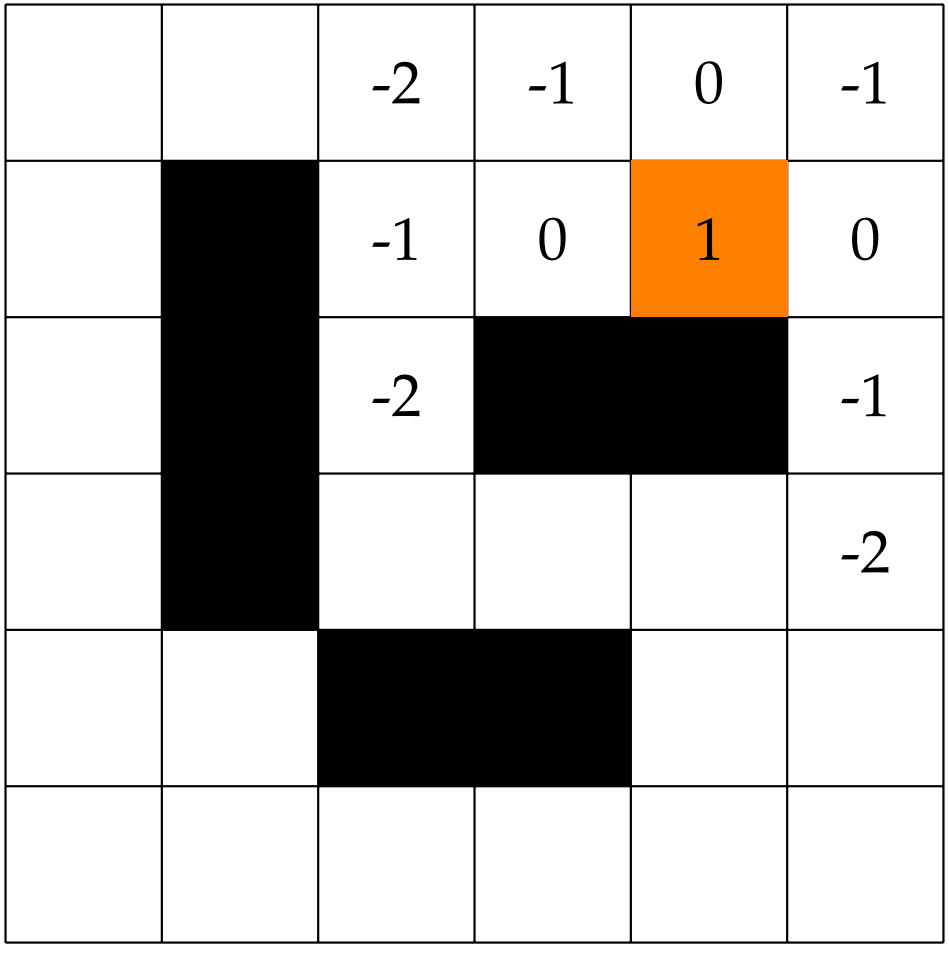
Dynamic programming



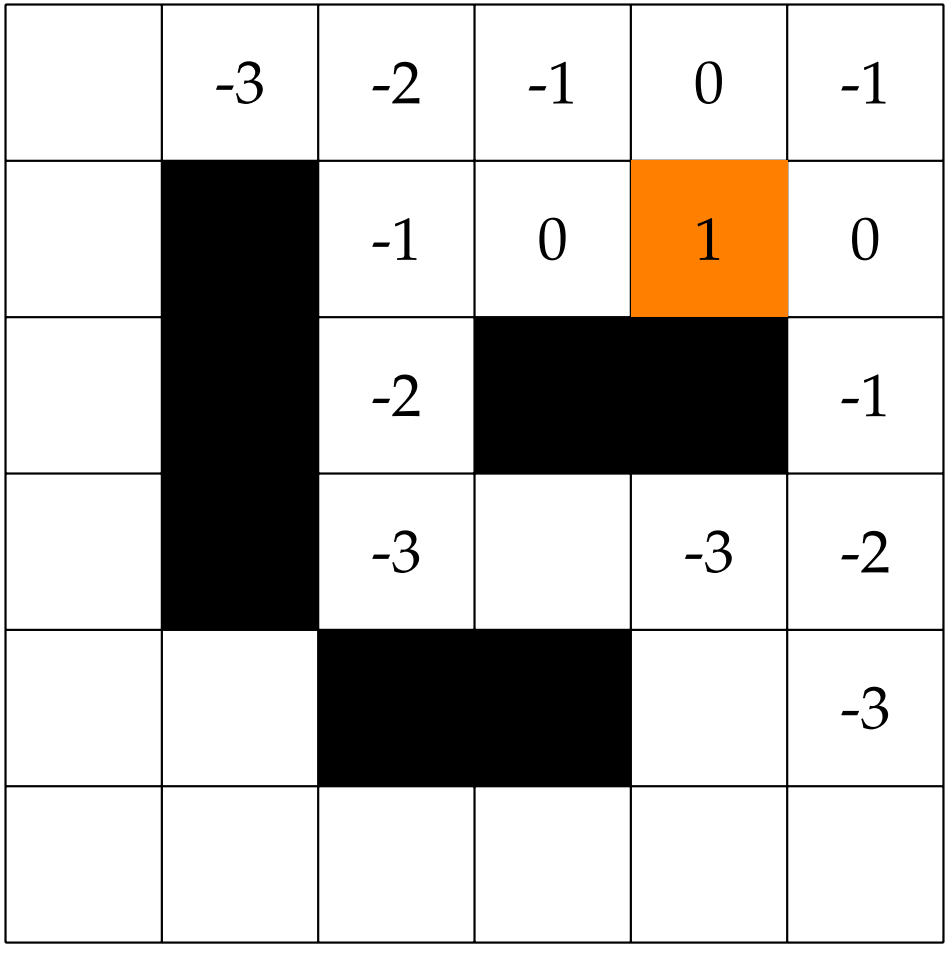
Dynamic programming



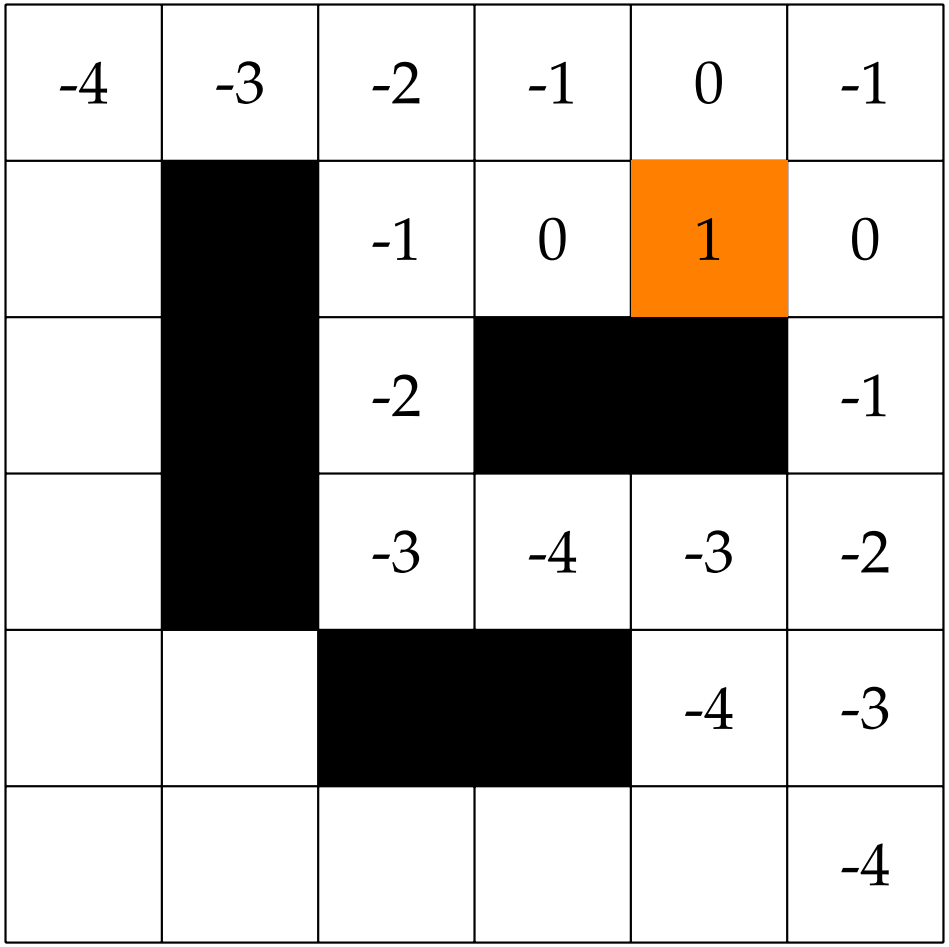
Dynamic programming



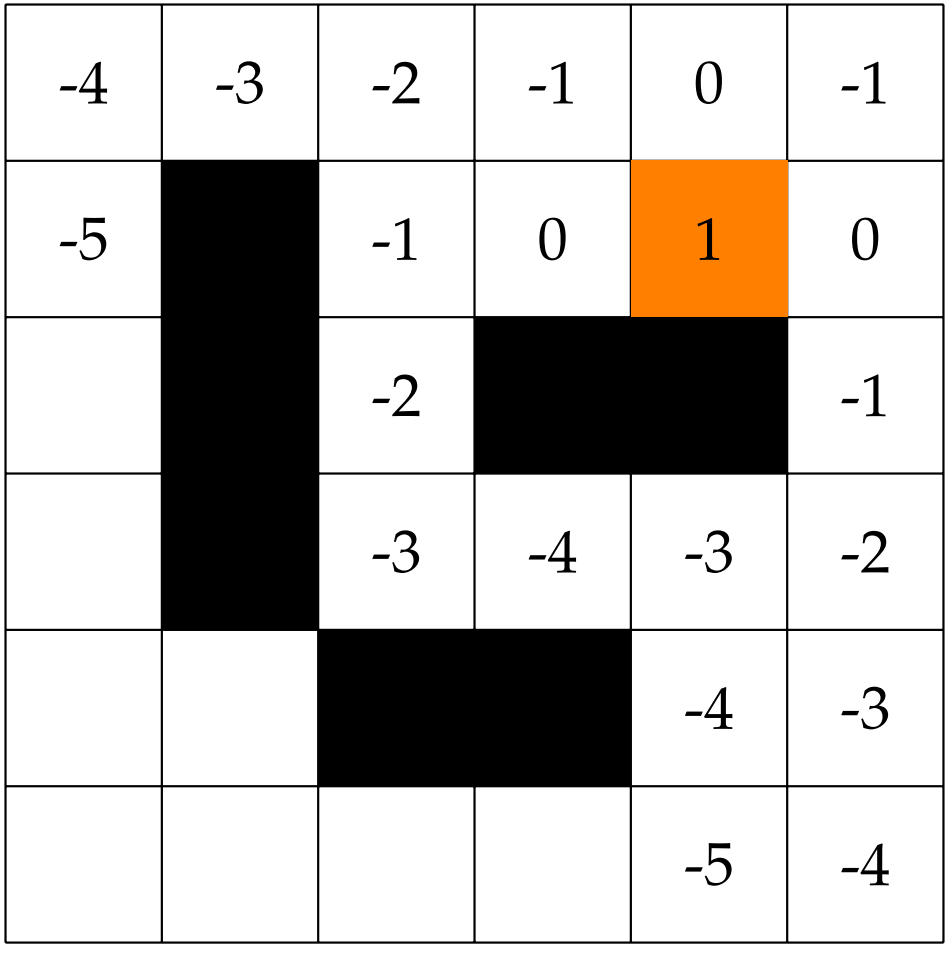
Dynamic programming



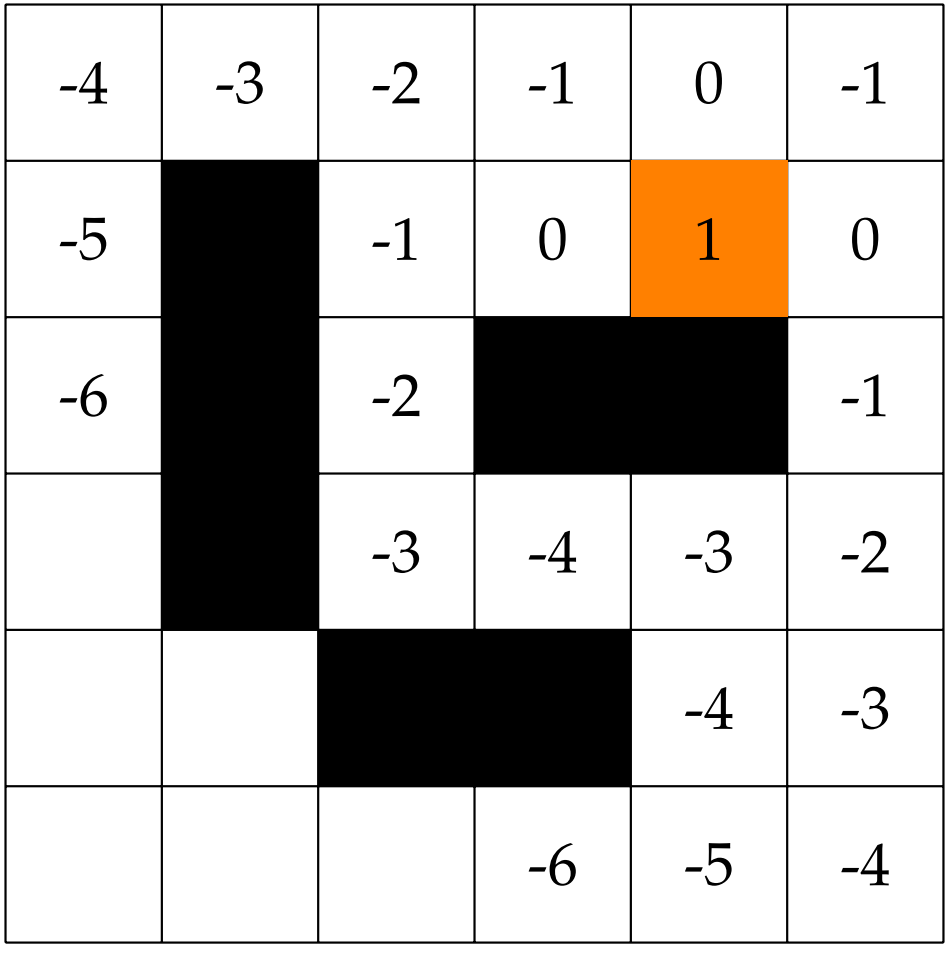
Dynamic programming



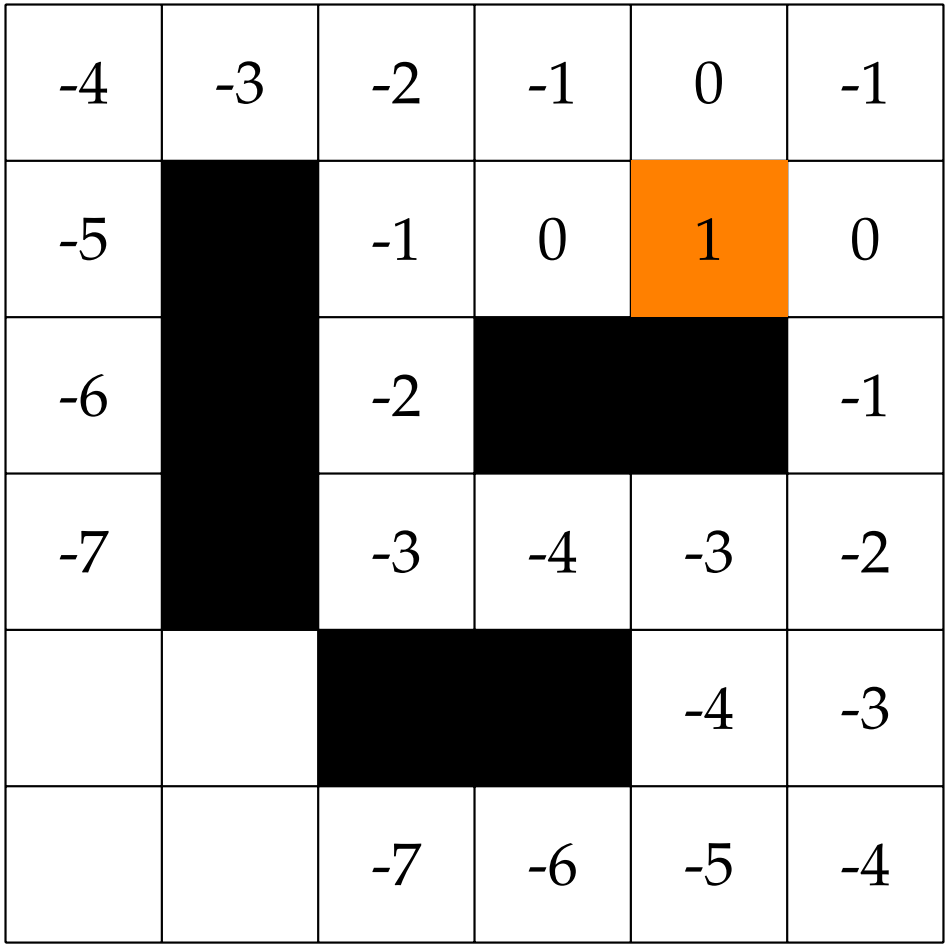
Dynamic programming



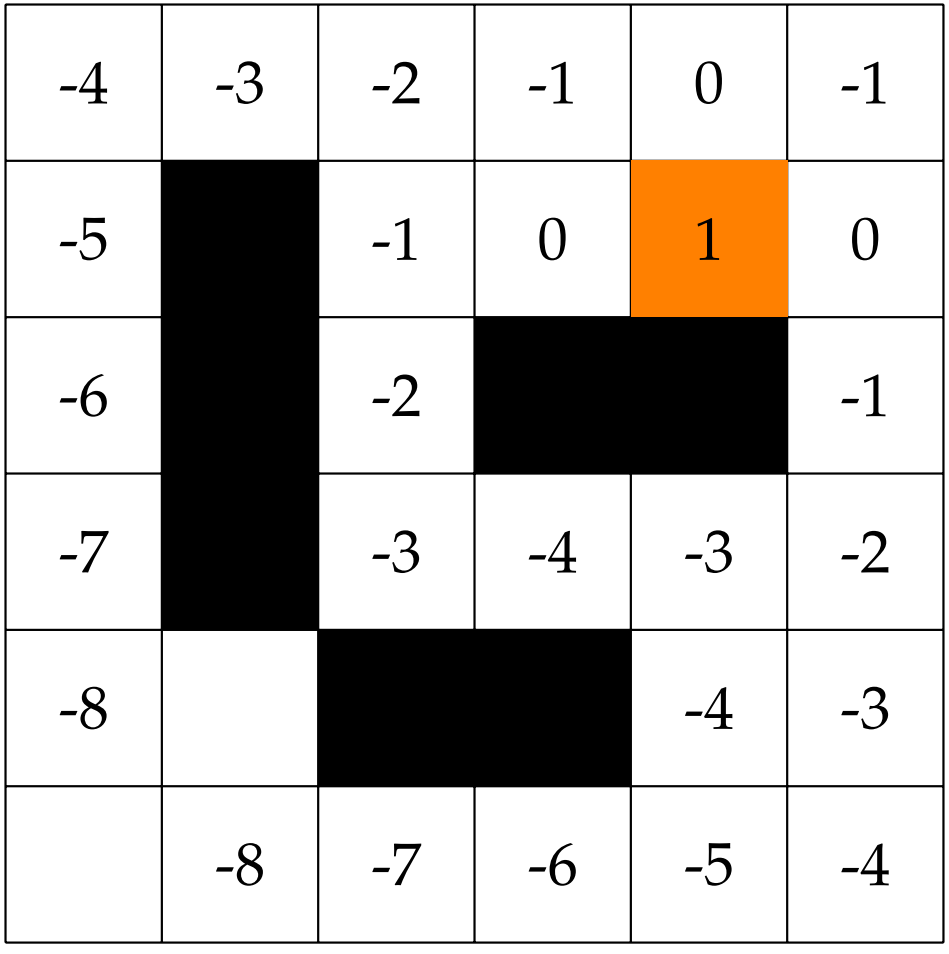
Dynamic programming



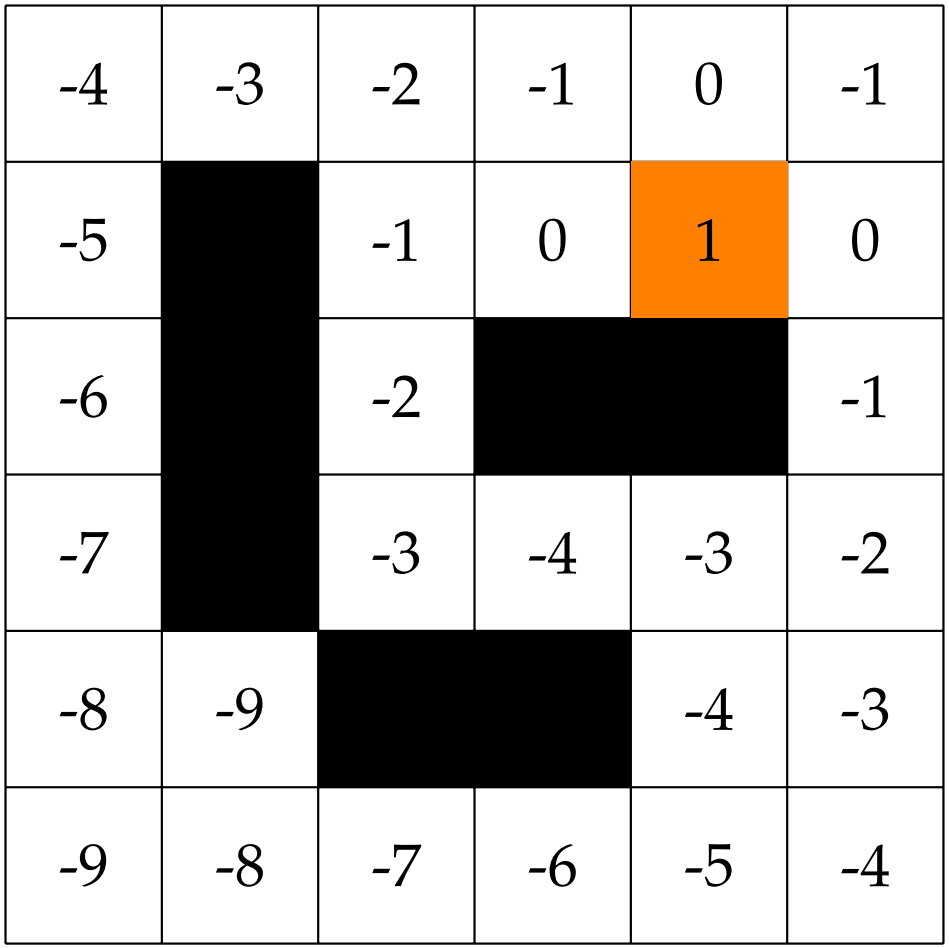
Dynamic programming



Dynamic programming



Dynamic programming



Optimal policy

If we have the optimal value map, defining the optimal policy is easy:

> *Choose the action that results in the greatest sum of reward now and value in the next state.*

Choose `pol(s)` that maximises

```
Reward(s, a) + Value(sys(s, a))
```

where `sys : (State, Action) ~> State` is the function that tells us what happens depending on the current state and chosen action.

Review

Ingredients:

```
sys : (State, Action) -> State
rew : (State, Action) -> ℝ
pol : State -> Action
s₀ ∈ State -- given
```

Problem: Find `pol` that maximises Σ_0^n `rew(sᵢ, pol(sᵢ))`, where

$s_{i+1} =$ `sys(sᵢ, pol(sᵢ))`

Value function

$$\mathtt{Val} : \mathbb{N} \to \mathtt{State} \to \mathbb{R}$$

$$\mathtt{Val}_i(s) = \max_{pol} \Sigma_i^n \, \mathtt{rew}(s_i, \mathtt{pol}(s_i))$$

Note:

$\mathtt{Val}_0(s_0)$ is the maximal value for the entire problem.

Bellman equation

If $\texttt{pol(s)} = \texttt{a}^{\texttt{opt}}$ the optimal action in state s, then

$$\texttt{Val}_{\texttt{i}}\texttt{(s)} = \texttt{Rew(s, a}^{\texttt{opt}}\texttt{)} + \texttt{Val}_{\texttt{i+1}}\texttt{(sys(s, a}^{\texttt{opt}}\texttt{))}$$

Therefore

$$\texttt{Val}_{\texttt{i}}\texttt{(s)} = \texttt{max}_{\texttt{a}}\ \texttt{(Rew(s, a)} + \texttt{Val}_{\texttt{i+1}}\texttt{(sys(s, a)))}$$

This is called *Bellman's equation*.

Direct adaptive optimal control

Model of the problem

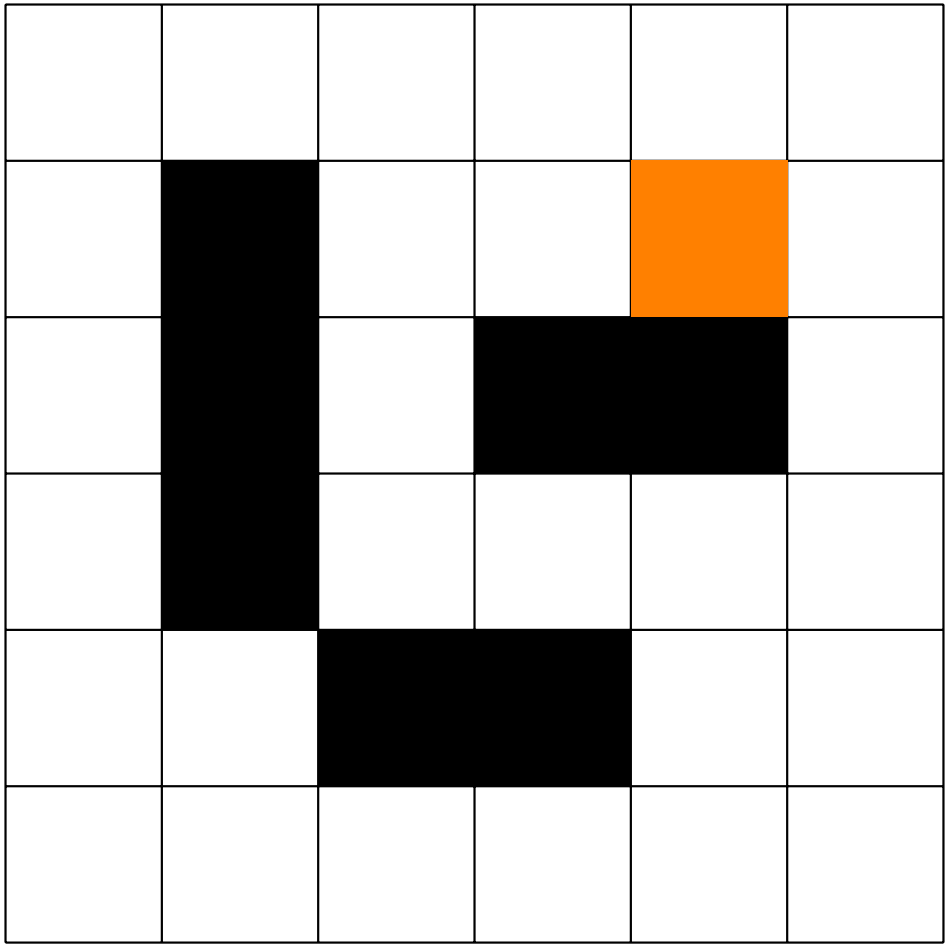
If we do not know the exact map of the maze, or the location of the goal, we cannot apply dynamic programming. We need an *adaptive method*. Alternatives:

- explore the maze and make a map of it, find the goal, and then apply dynamic programming (*indirect* method);
- learn the optimal value map directly! (*direct* method).

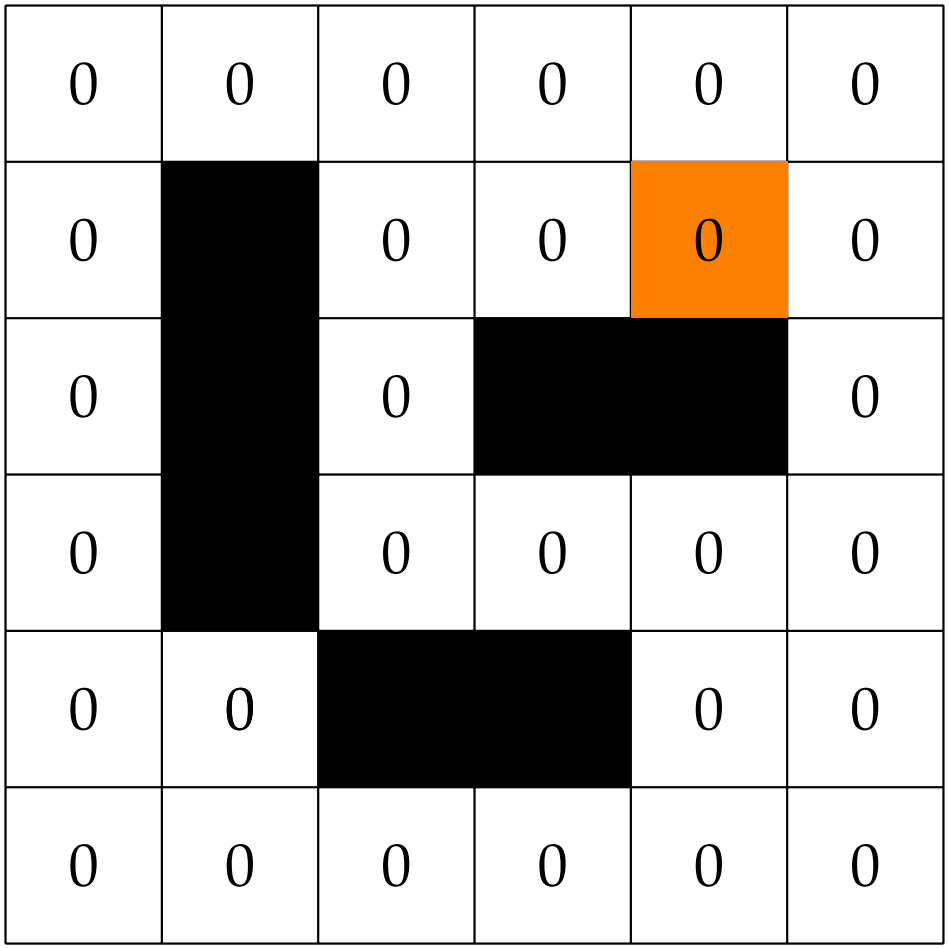
Reinforcement learning is *direct adaptive optimal control*.

Value iteration

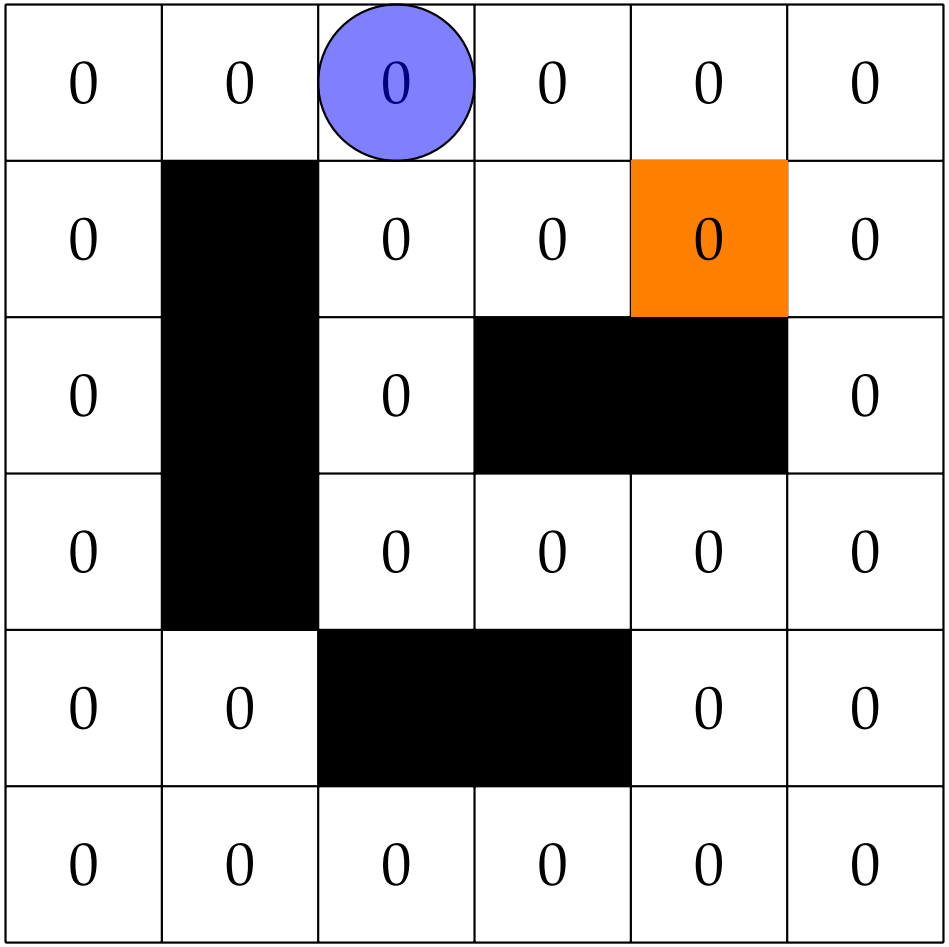
Start with a randomly generated value map



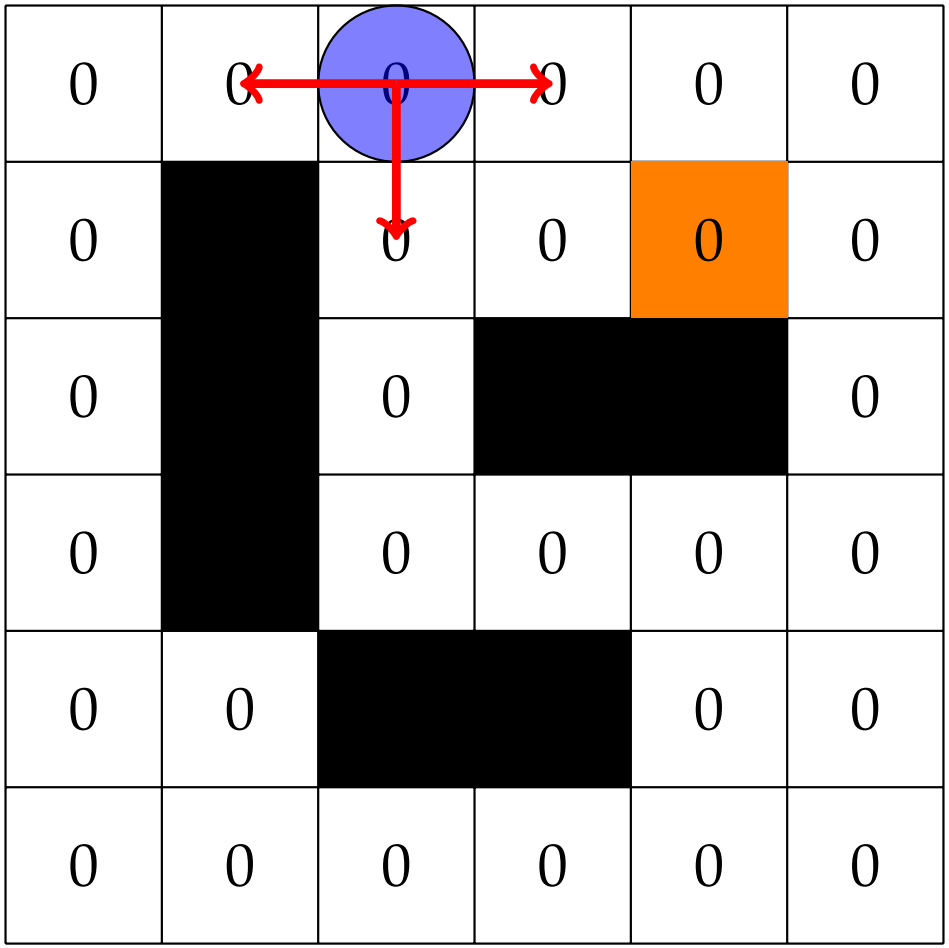
Start with a randomly generated value map



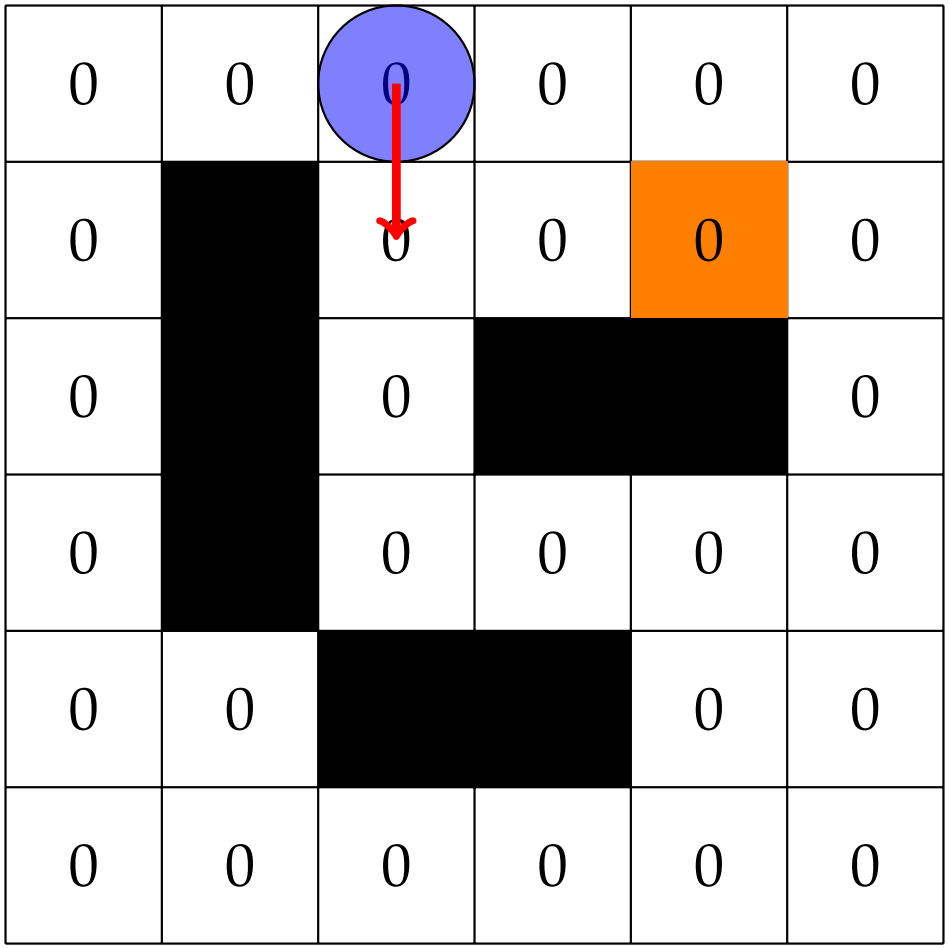
Alternatives



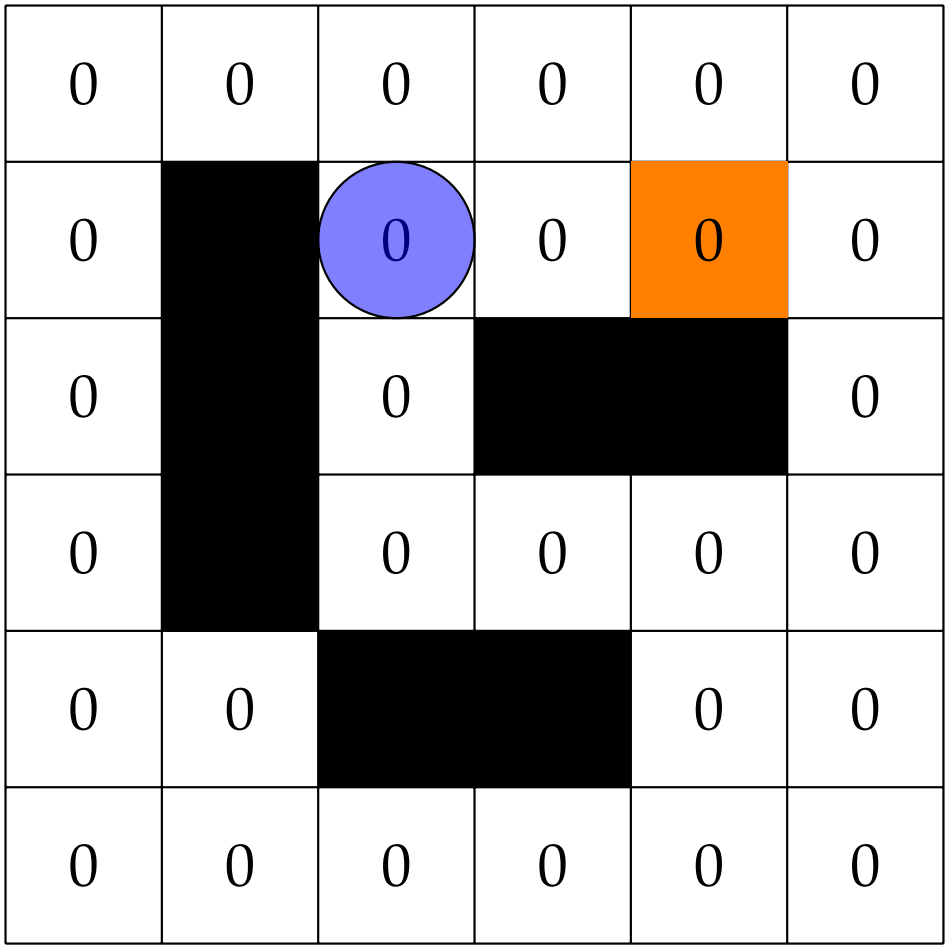
Alternatives



Choose an action



Make a move



Changing the current value function

If we had the optimal value function, then

```
Val((3,6)) = Rew((3,6), ↓) + Val((3,5))
```

But we have

```
Val((3, 6)) = 0,  Rew((3,6), ↓) = -1, Val((3,5)) = 0
```

Update rule

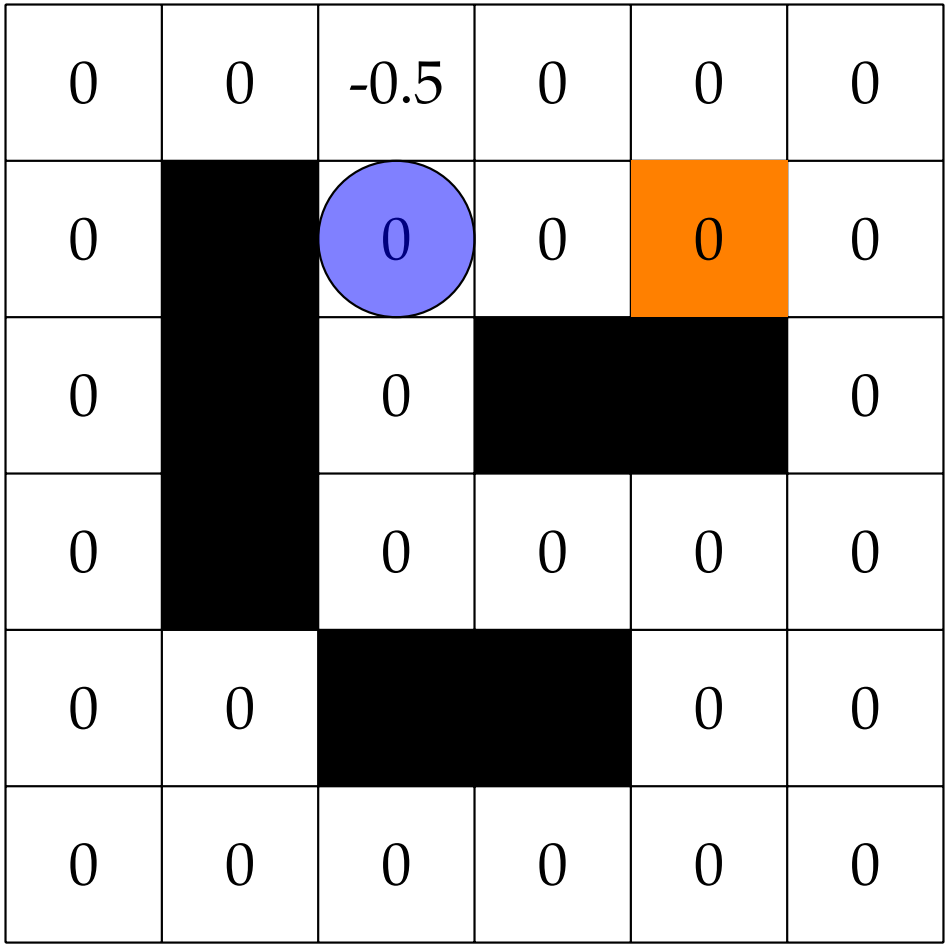
Idea from *supervised learning*:

```
Val((3,6)) <- Val((3,6)) +
              δ * (Rew((3,6), ↓) + Val((3,5)) - Val((3,6)))
```

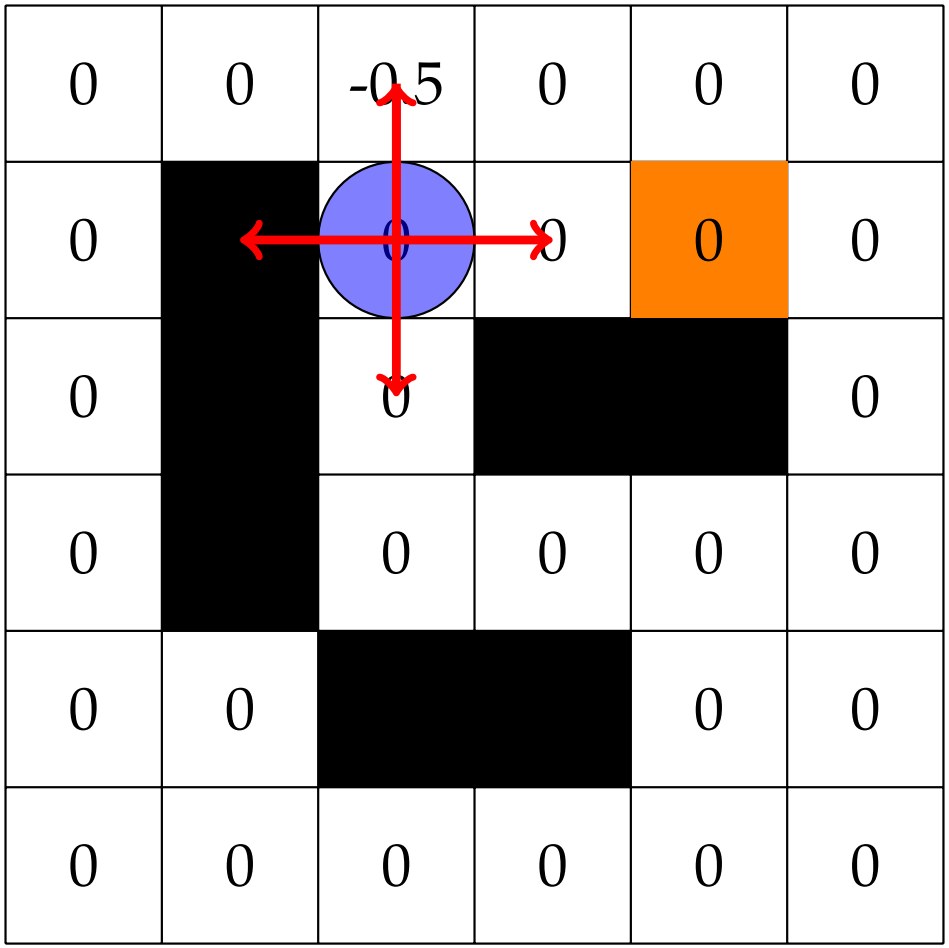
Therefore (say `δ = 0.5`)

```
Val(3,6) <- 0 + 0.5 * (-1 + 0 - 0) = -0.5
```

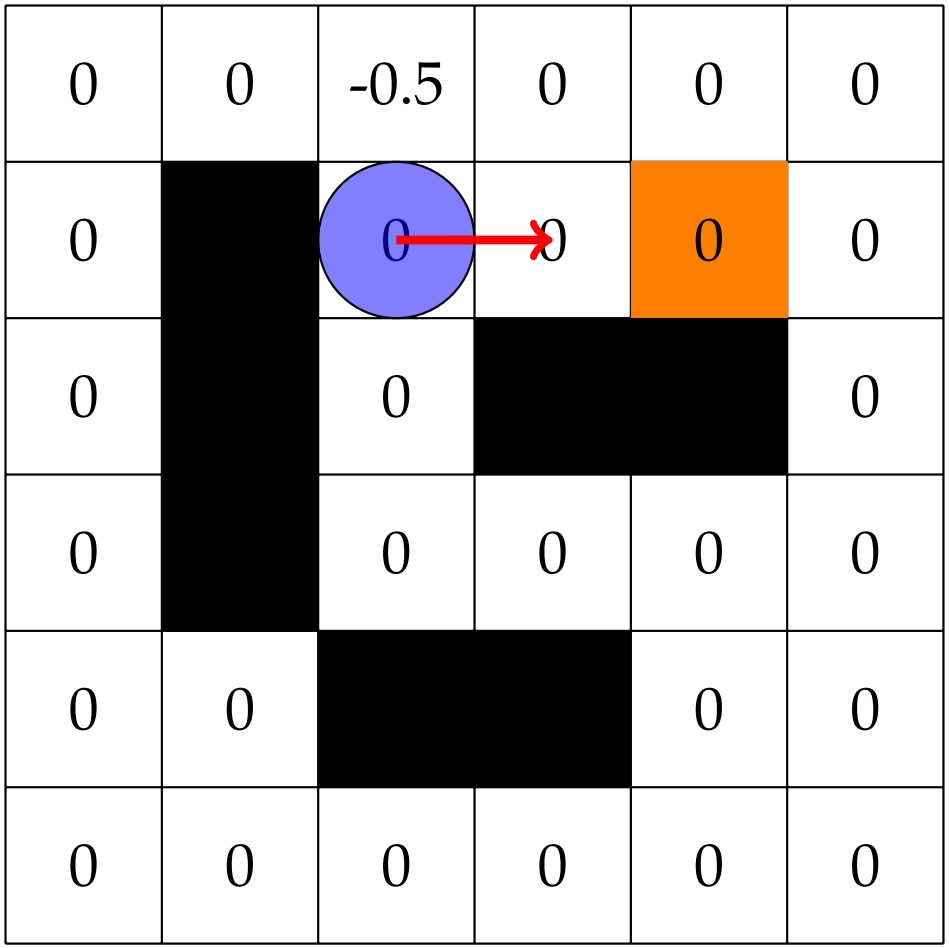
The new situation



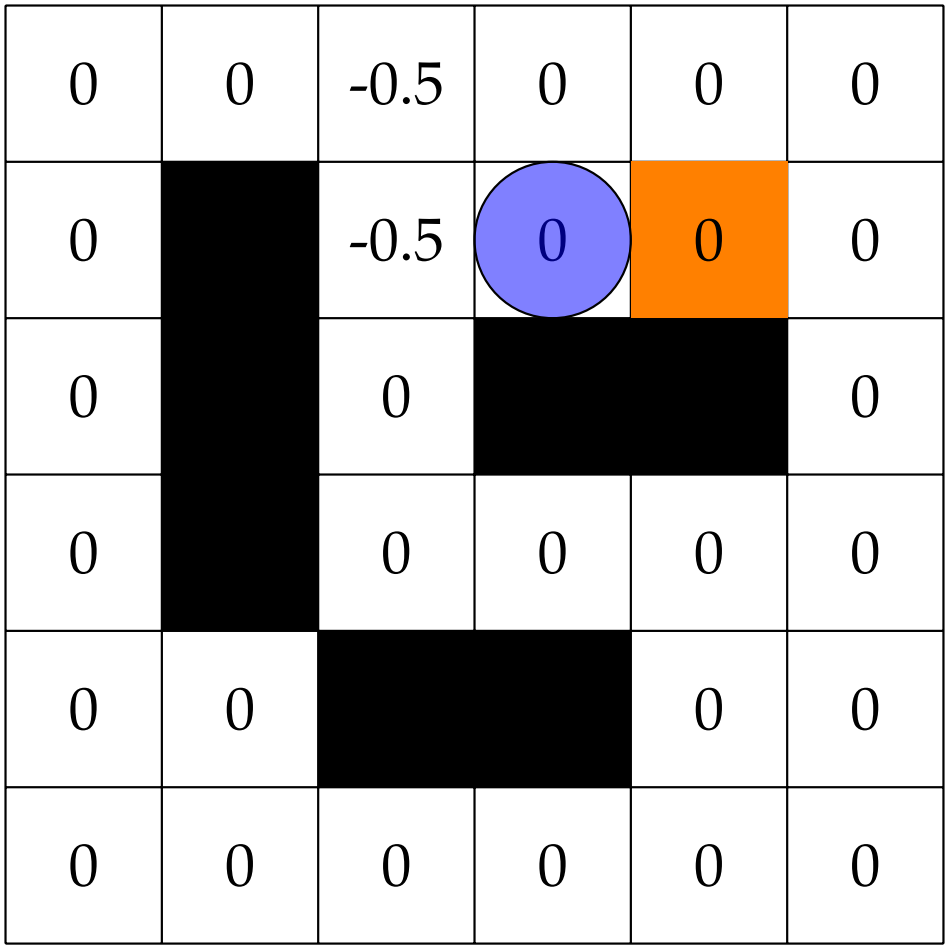
The new situation



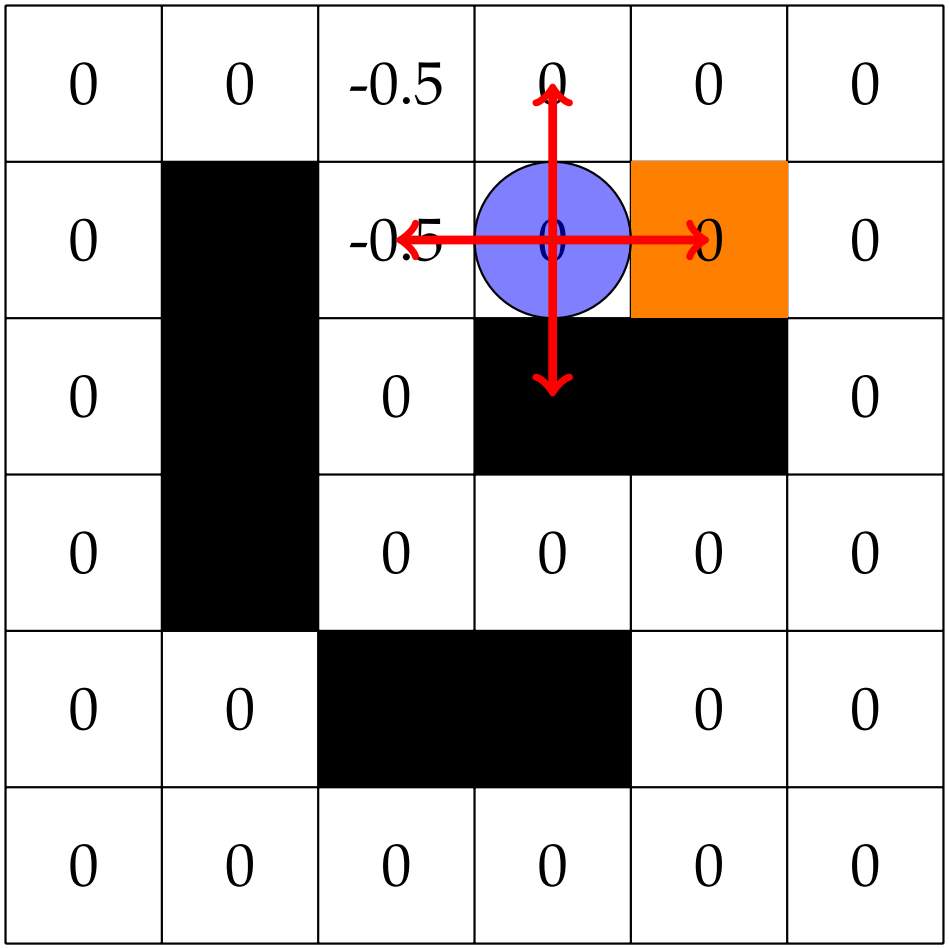
Pick a new direction



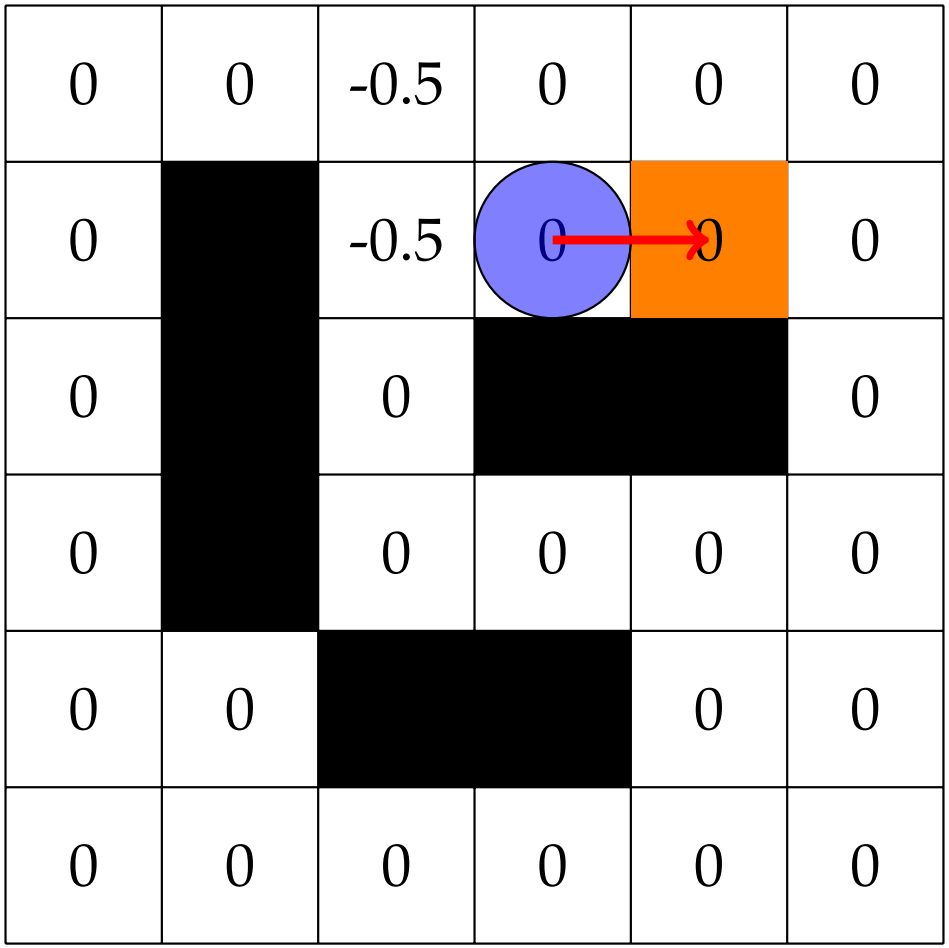
Move and update



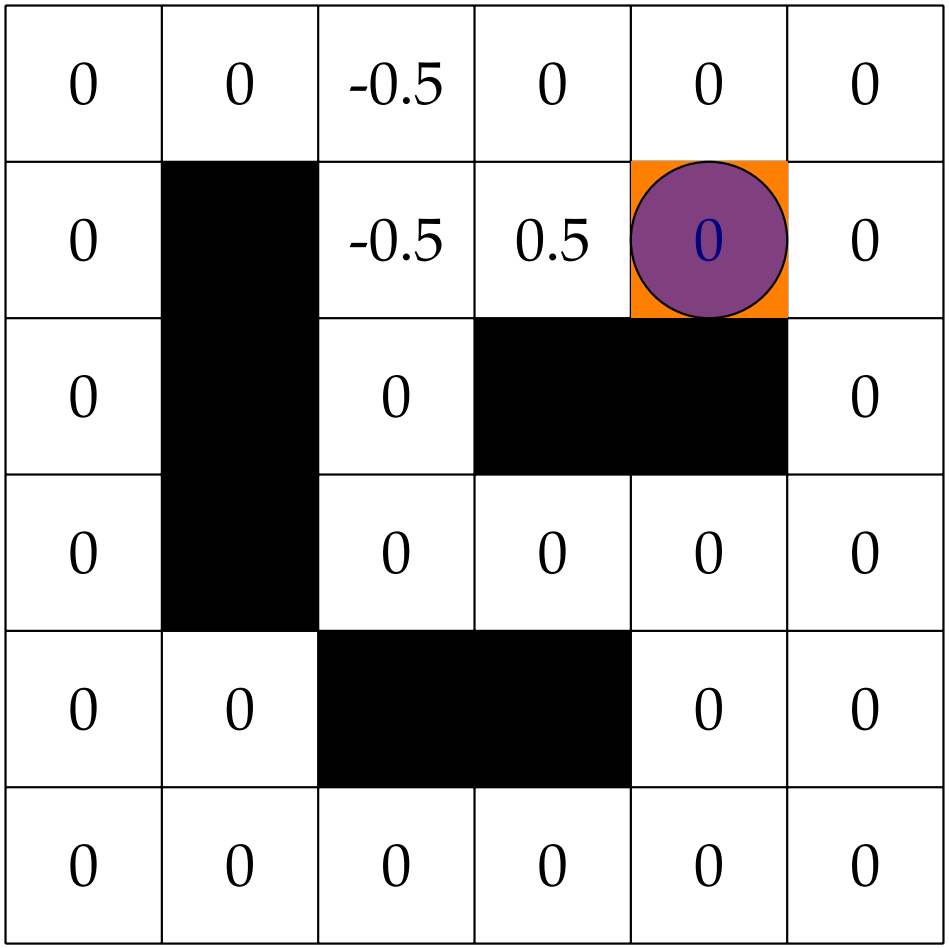
And so on…



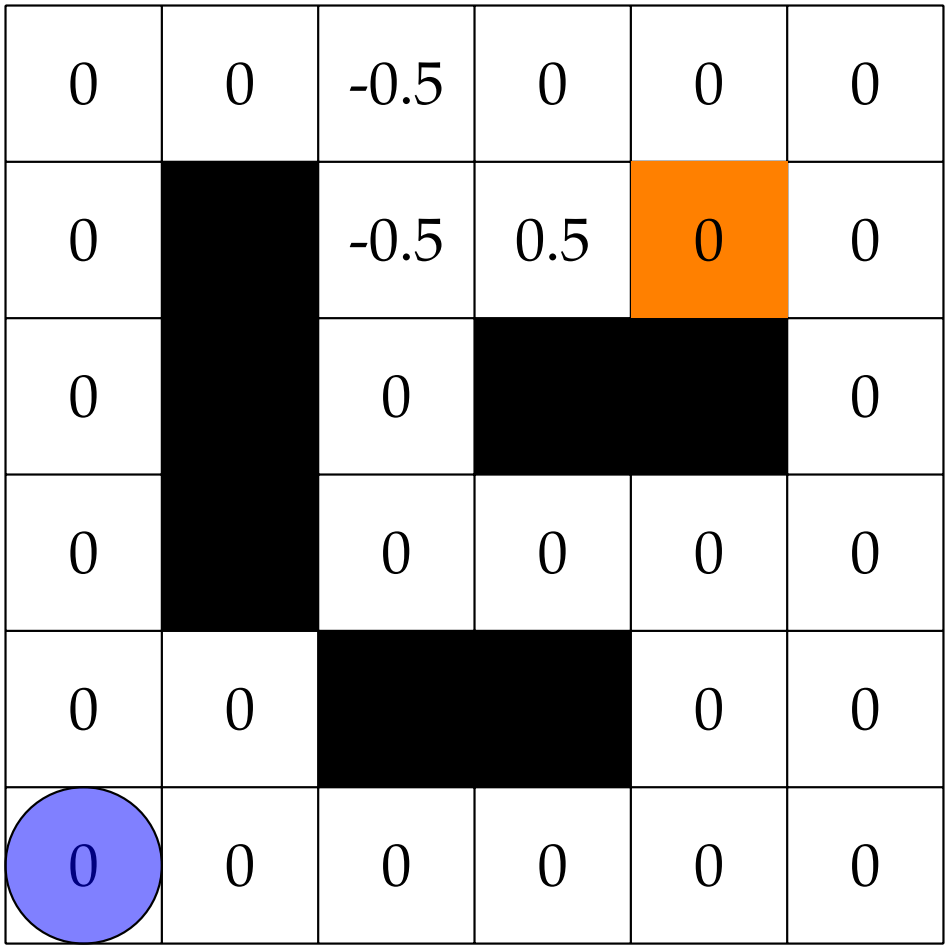
A good choice



A good choice



The game goes on



Discount factors

In this setting, we maximise the sum of rewards. Problem: if the number of steps is large, the values become difficult to estimate (huge absolute value). Moreover, in many situations we need to operate with *infinite horizons*.

To deal with this problem, we introduce a *discount factor*: $0 < \beta < 1$

Instead of maximising

$$Rew(s_0, pol(s_0)) + Rew(s_1, pol(s_1)) + Rew(s_2, pol(s_2)) + ...$$

we maximise

$$Rew(s_0, pol(s_0)) + \beta * Rew(s_1, pol(s_1)) + \beta^2 * Rew(s_2, pol(s_2)) + ...$$

where $s_{t+1} = sys(s_t, pol(s_t))$

Value function

If we have an infinite horizon, the value function is *stationary*:

$V_i(s) = \max_{pol} \Sigma_i^{\infty} \beta^t * rew(s_t, pol(s_t))$ where

$s_i = s$

$s_{t+1} = sys(s_t, pol(s_t))$

This is independent of i!

Bellman's equation

We choose `pol(s)` that maximises

```
Rew(s, a) + β * Val(sys(s, a))
```

Therefore

```
Val(s) = max_a (Rew(s, a) + β * Val(sys(s, a)))
```

Value iteration and function approximation

```
Val(s) = max_a (Rew(s, a) + β * Val(sys(s, a)))
```

Bellman's equation allows us to iteratively approximate the optimal value function.

```
Val(s) <- Val(s) +
          δ * (max_a (Rew(s, a) + β * Val(sys(s, a)))) - Val(s))
```

The optimal value function `Val` is unique (but there might be many policy functions that realise it).

Value iteration

```
Val(s) <- Val(s) +
          δ * (max_a (Rew(s, a) + β * Val(sys(s, a)))) - Val(s))
```

Problem: we require knowledge of the reward function.

We could use supervised learning to approximate the reward function.

But now we have two function approximations and an optimisation at every step!

Q-learning

Watkins' idea

The reward function only enters the picture when combined with `Val`, e.g.:

```
Val(s) = max_a (Rew(s, a) + β * Val(sys(s, a)))
```

Chris Watkins (1989): maybe it's simpler to learn the combination of `Rew` and `Val`!

```
Q(s, a) = Rew(s, a) + β * Val(sys(s, a))
```

Q-learning and Val

```
Q(s, a) = Rew(s, a) + β * Val(sys(s, a))
```

We have `Val(s) = max`$_a$ `Q(s, a)` therefore, knowing `Q` is sufficient for determining `Val`.

Q-learning and the optimal policy

```
Q(s, a) = Rew(s, a) + β * Val(sys(s, a))
```

The optimal policy is the one that maximises

```
Rew(s, a) + β * Val(sys(s, a))
```

Therefore, the optimal action in a state `s` is the one that maximises the `Q` function:

$$a^{opt} = \arg\max_a Q(s, a)$$

Recursive equation for Q-learning

$$Q(s, a) = Rew(s, a) + \beta * Val(sys(s, a))$$

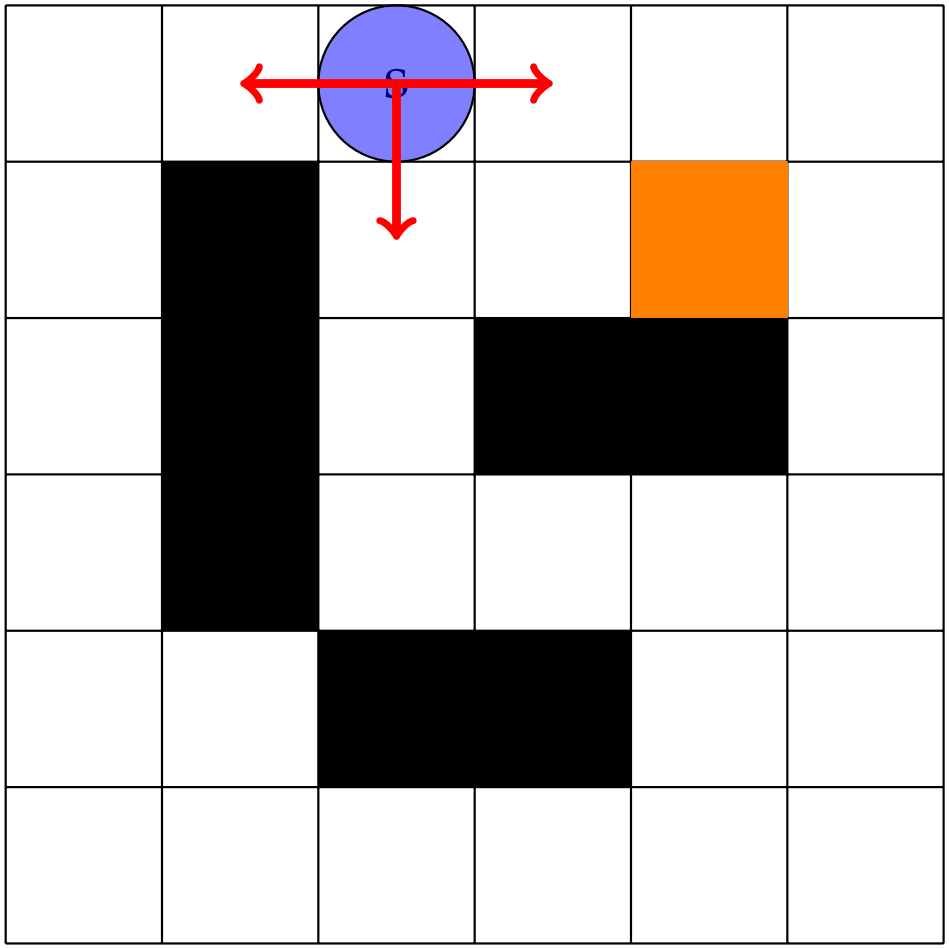
$$Val(s) = \max_a Q(s, a)$$

Therefore

$$Q(s, a) = \max_a (Rew(s, a) + \beta * \max_x Q(sys(s, a), x))$$

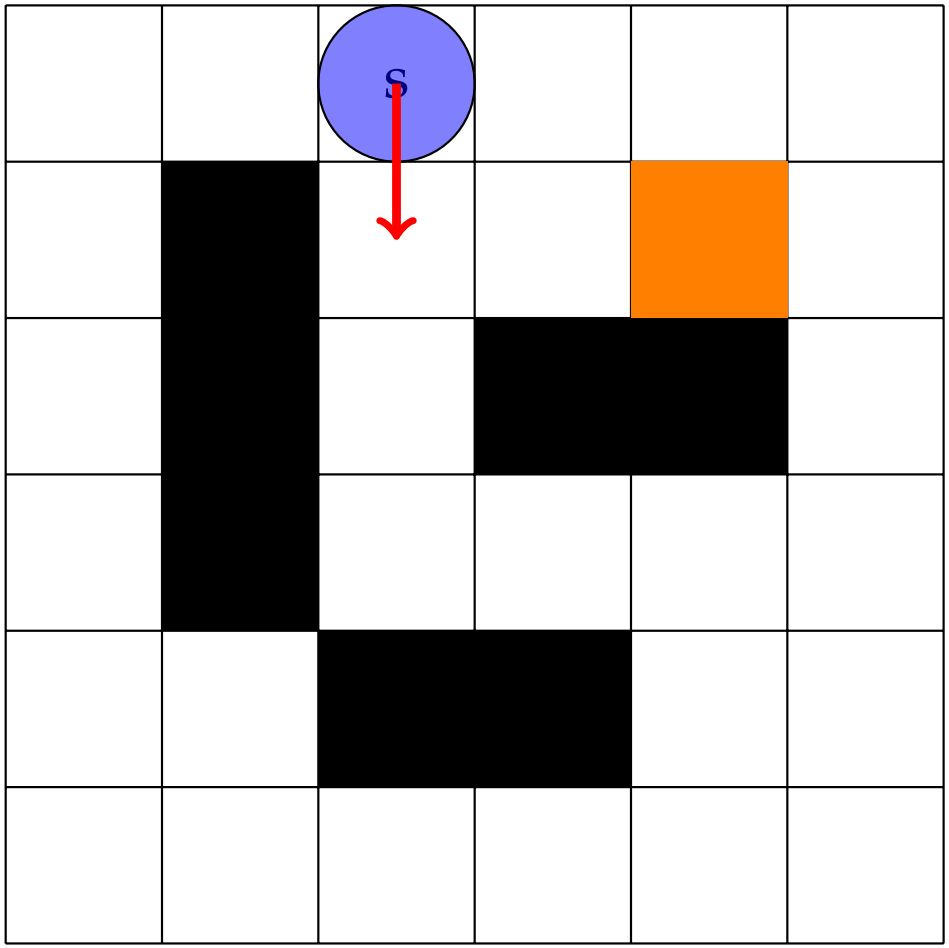
We have a recursive equation for the Q function.

Q-learning in action



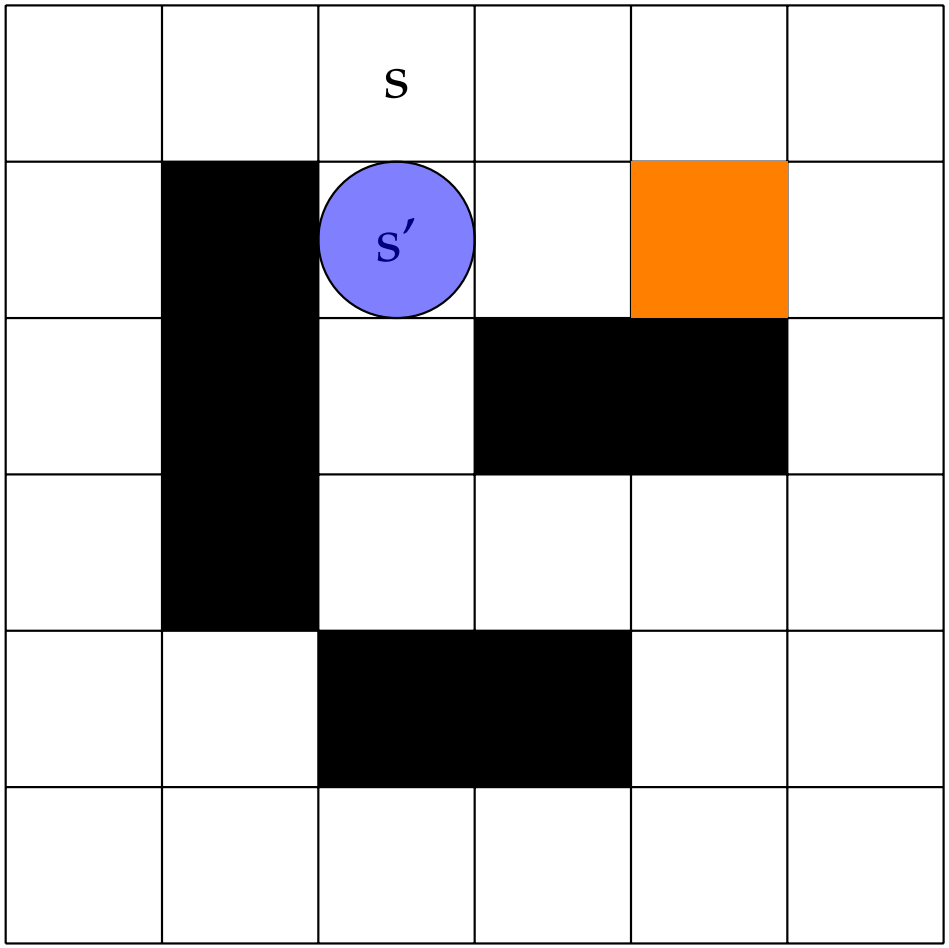
$$Q(s, \leftarrow) = -10$$
$$Q(s, \downarrow) = -8$$
$$Q(s, \rightarrow) = -9$$

Q-learning in action



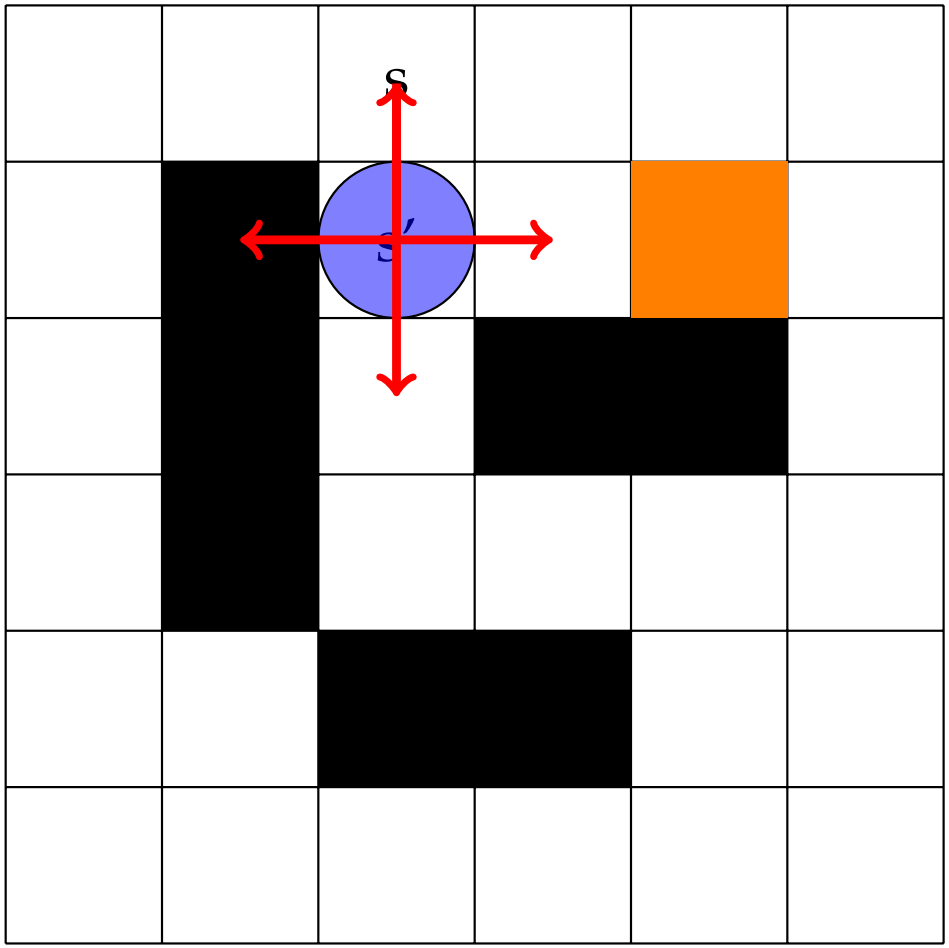
$$Q(s, \downarrow) = -8$$

Q-learning in action



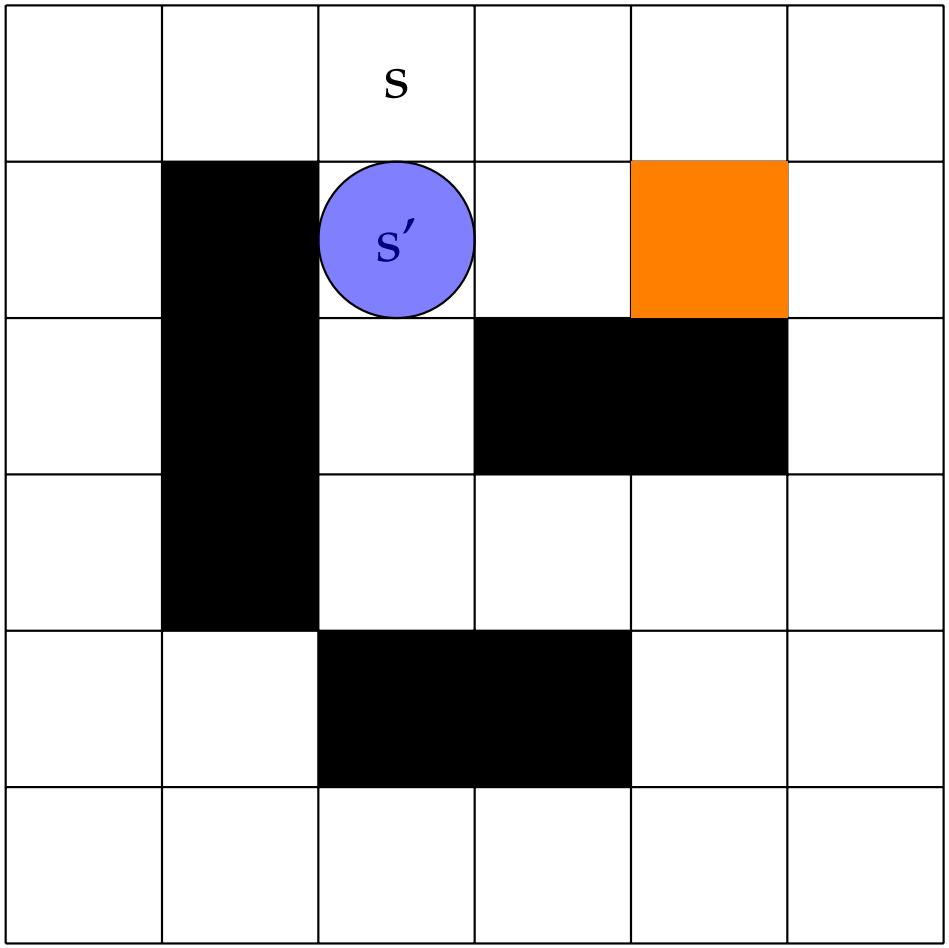
$$Q(s, \downarrow) = -8$$
$$Rew(s, \downarrow) = -1$$

Q-learning in action



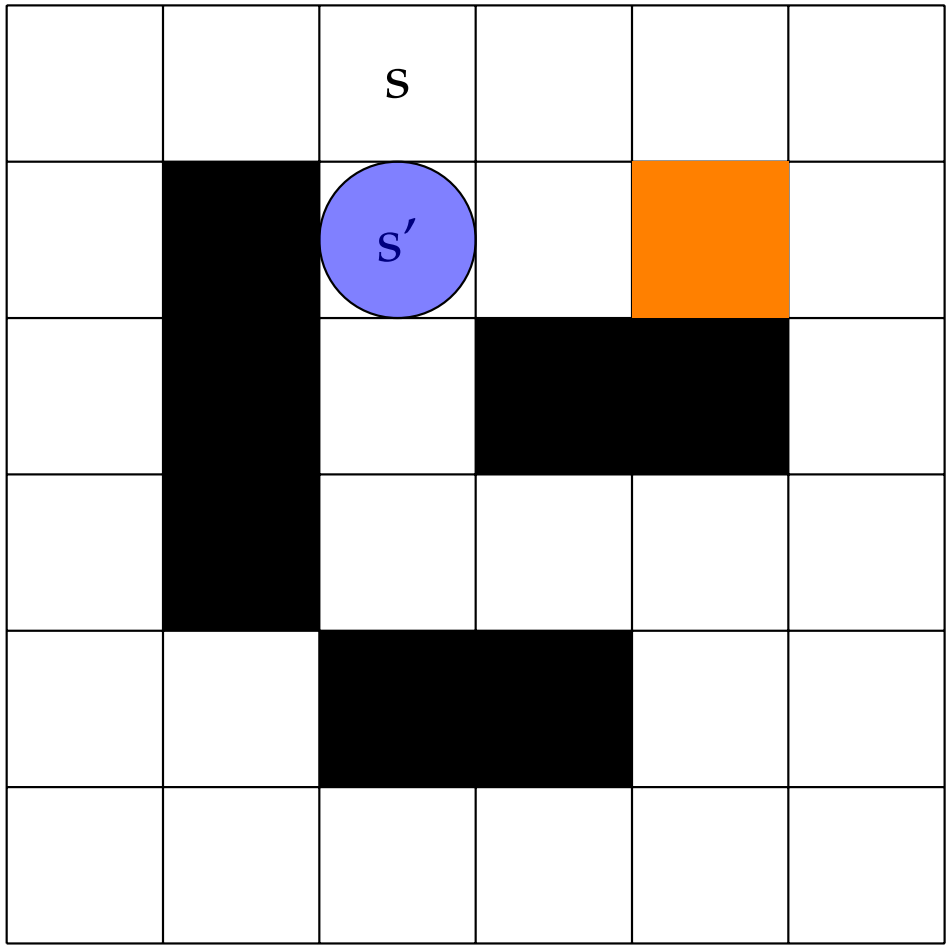
$$Q(s, \downarrow) = -8$$
$$Rew(s, \downarrow) = -1$$
$$Q(s', \leftarrow) = -\infty$$
$$Q(s', \downarrow) = -8$$
$$Q(s', \rightarrow) = -5$$
$$Q(s', \uparrow) = -10$$

Q-learning in action



$$Q(s,\downarrow) = -8$$
$$Rew(s,\downarrow) = -1$$
$$Val(s') =$$
$$max(-\infty, -8, -5, -10)$$
$$Rew(s,\downarrow) + \beta Val(s') =$$
$$-1 + 1 * (-5) = -6$$

Q-learning in action



$Q(s, \downarrow) = -8$
$Rew(s, \downarrow) + \beta Val(s') = -6$
$Q(s, \downarrow) \leftarrow -8 + \delta(-6 - (-8)) = -6$

Homework

Solve exercise 13.2, point a), from Mitchell (page 388). See next slide.

This homework is optional, you only need to hand it in if you haven't already passed four assignments.

Mitchell uses slightly different notations:

$V^*(s) = Val(s)$

$\gamma = \beta$

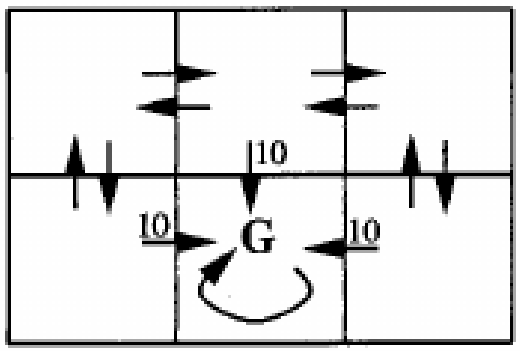
Homework

13.2. Consider the deterministic grid world shown below with the absorbing goal-state G. Here the immediate rewards are 10 for the labeled transitions and 0 for all unlabeled transitions.

(a) Give the V^* value for every state in this grid world. Give the $Q(s, a)$ value for every transition. Finally, show an optimal policy. Use $\gamma = 0.8$.

(b) Suggest a change to the reward function $r(s, a)$ that alters the $Q(s, a)$ values, but does not alter the optimal policy. Suggest a change to $r(s, a)$ that alters $Q(s, a)$ but does not alter $V^*(s, a)$.

(c) Now consider applying the Q learning algorithm to this grid world, assuming the table of $\hat{Q}$ values is initialized to zero. Assume the agent begins in the bottom left grid square and then travels clockwise around the perimeter of the grid until it reaches the absorbing goal state, completing the first training episode. Describe which $\hat{Q}$ values are modified as a result of this episode, and give their revised values. Answer the question again assuming the agent now performs a second identical episode. Answer it again for a third episode.



Stochastic systems

We have assumed a deterministic system, but in most cases this is unrealistic. E.g., the maze environment could take us (with a small probability) in a different direction than we chose. In that case, we need to maximise the *expected value* of the sum of rewards. Q-learning can be applied to these cases almost without any change!

Conclusions

Reinforcement learning applications: games, RoboCup Soccer, self-driving cars, etc. See also discussion in Week 1 (robot-catching arm, self-driving car, barriers on emissions…) Reinforcement learning seems to make the most out of very little: is it the road to Artificial General Intelligence? Chris Watkins (http://www.cs.rhul.ac.uk/~chrisw/):

> *I have long felt the standard model of RL is deceptively attractive, but limited.*