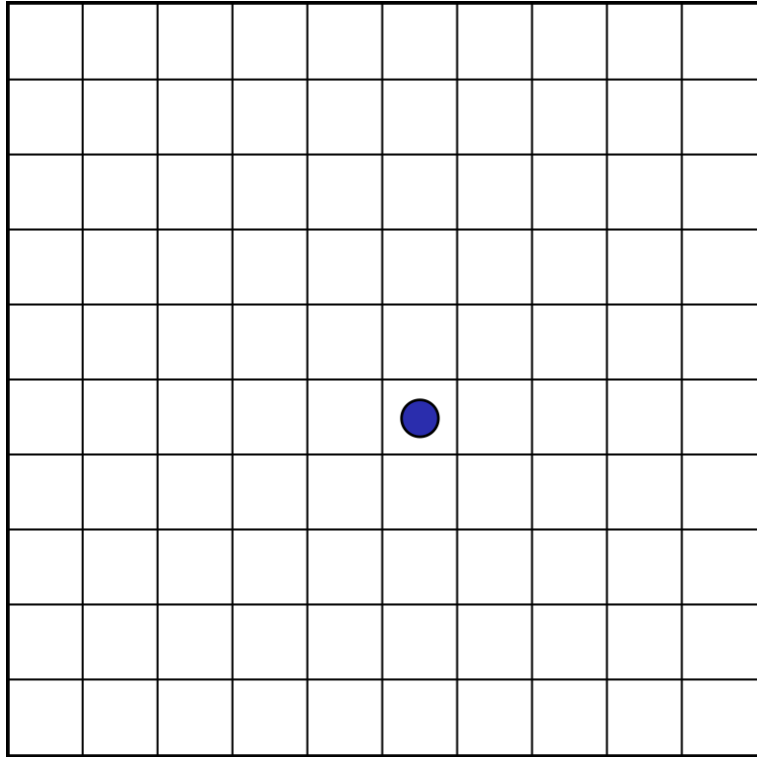


Gridworld with Q-learning

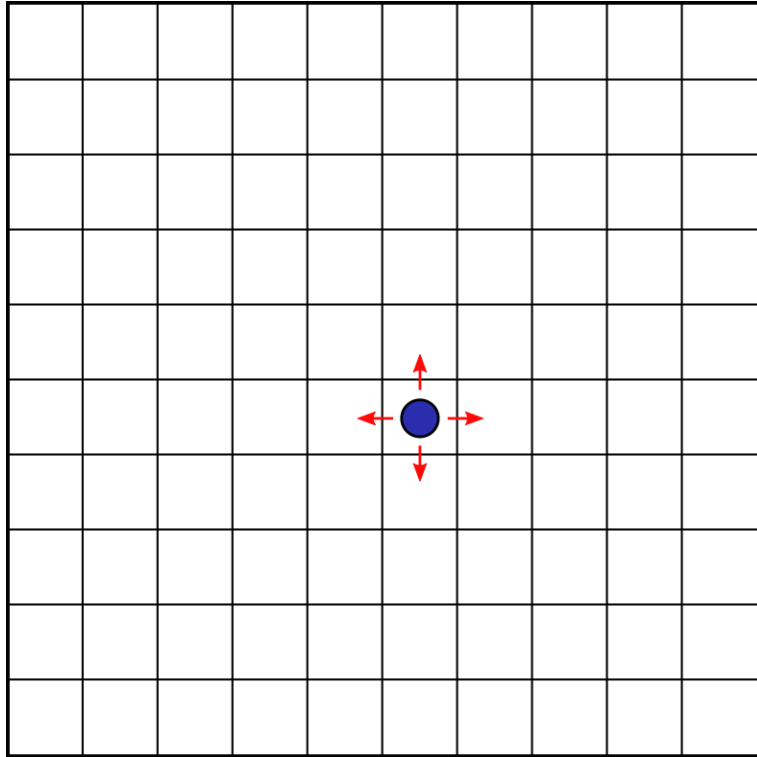
Andrea Mazzolini, HPC 2020

Gridworld



$d \times d$ 2-dimensional lattice.

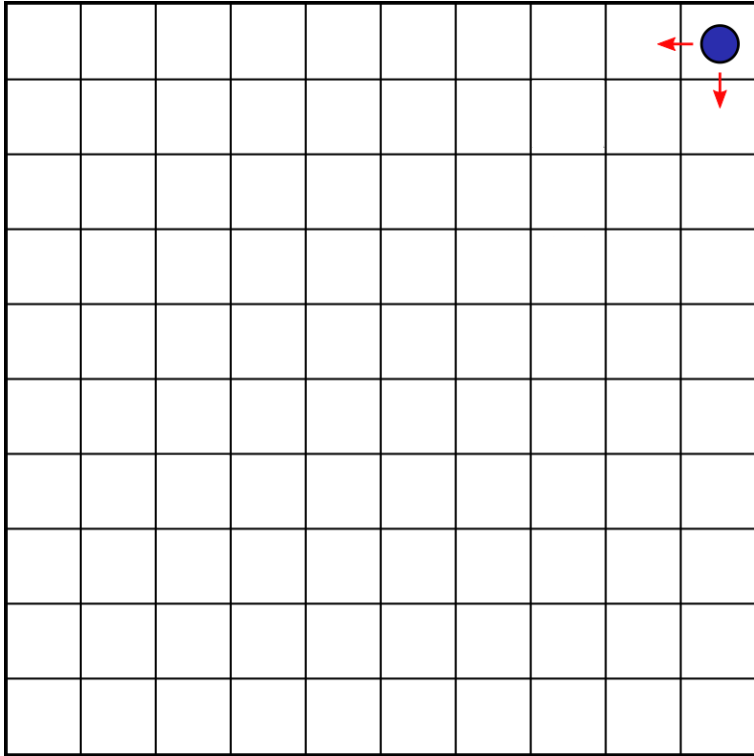
Gridworld



$d \times d$ 2-dimensional lattice.

At each time step, a player can move in the nearest neighbours.

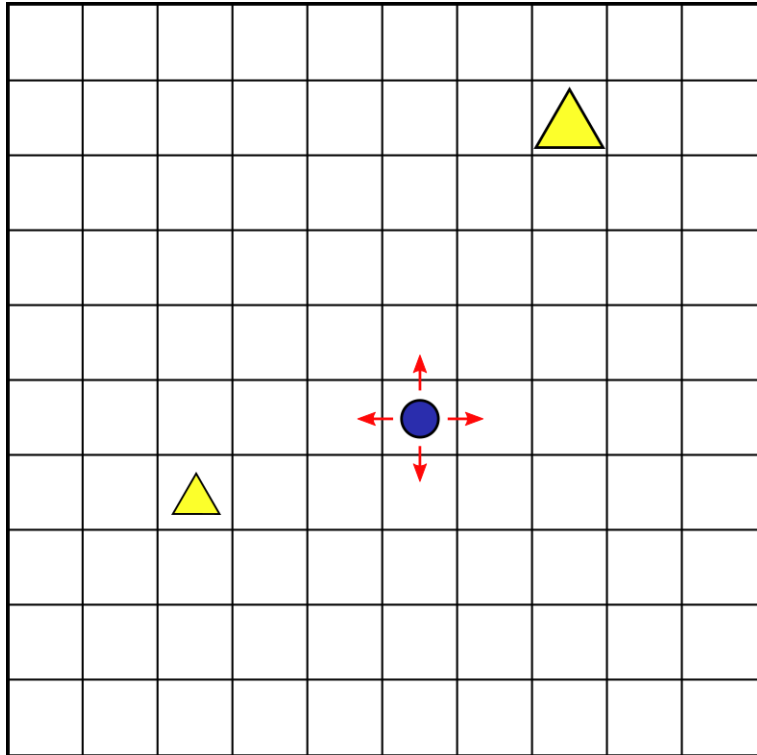
Gridworld



$d \times d$ 2-dimensional lattice.

At each time step, a player can move in the nearest neighbours (not outside the field).

Gridworld



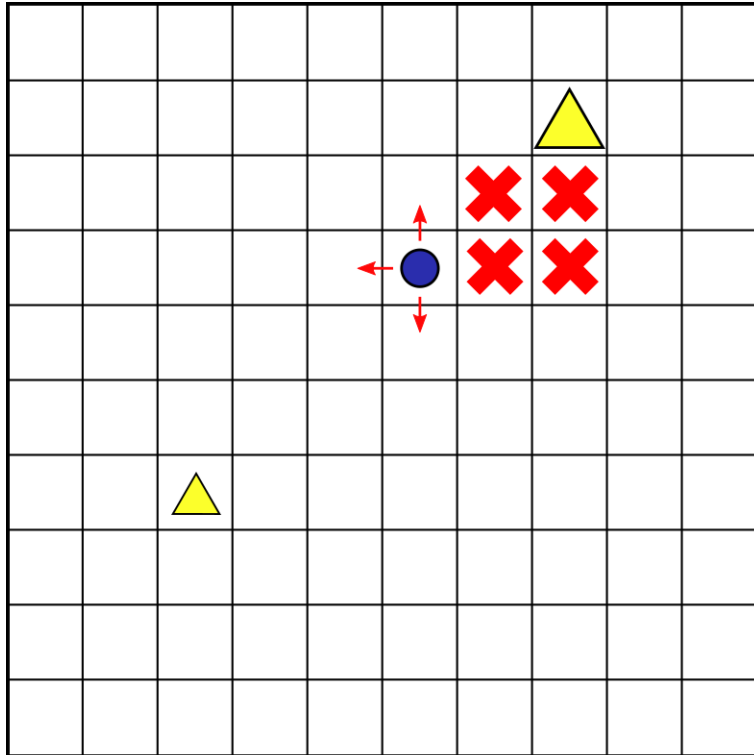
$d \times d$ 2-dimensional lattice.

At each time step, a player can move in the nearest neighbours.




The purpose of the player is to get the rewards that some cells contain.


Gridworld



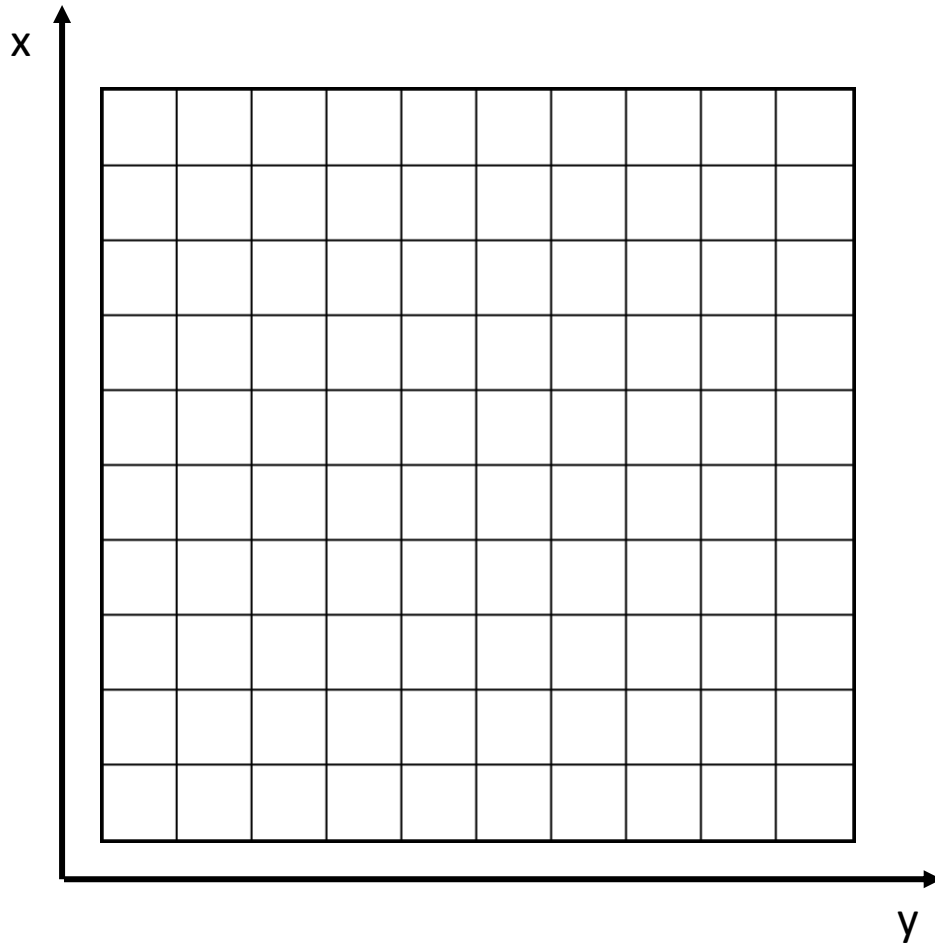
$d \times d$ 2-dimensional lattice.

At each time step, a player can move in the nearest neighbours.

 The purpose of the player is to get the rewards that some cells contain.

 The player cannot move over obstacles and outside the gridworld.

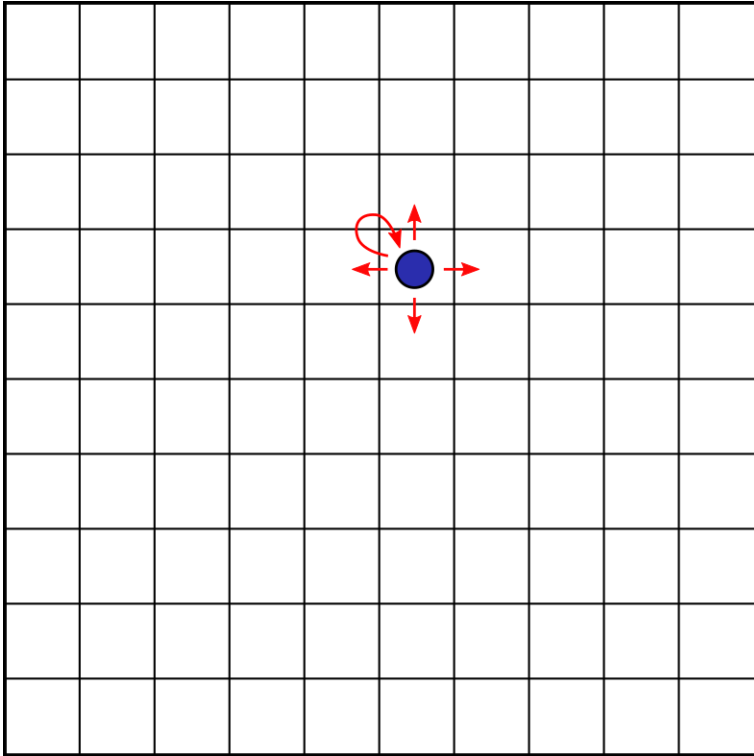
Gridworld as a MDP



- The states are the world coordinates:
 $s = (x, y)$



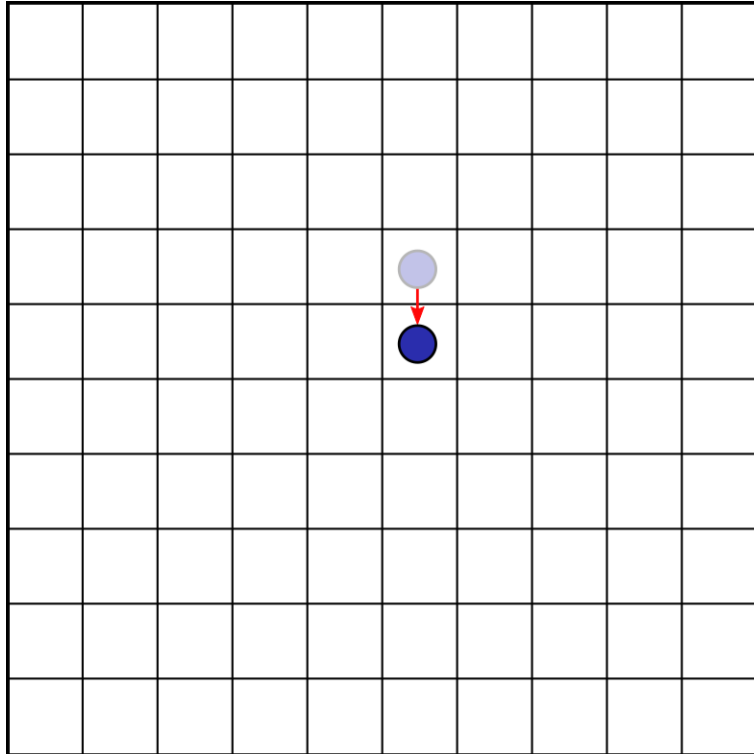
Gridworld as a MDP



- The states are the world coordinates:
 $s = (x, y)$
- The actions are moving on a nearest neighbour or staying still:
 $a \in \{(1,0), (-1,0), (0,1), (0,-1), (0,0)\}$
At the boundary or close to obstacles: restricted number of actions.



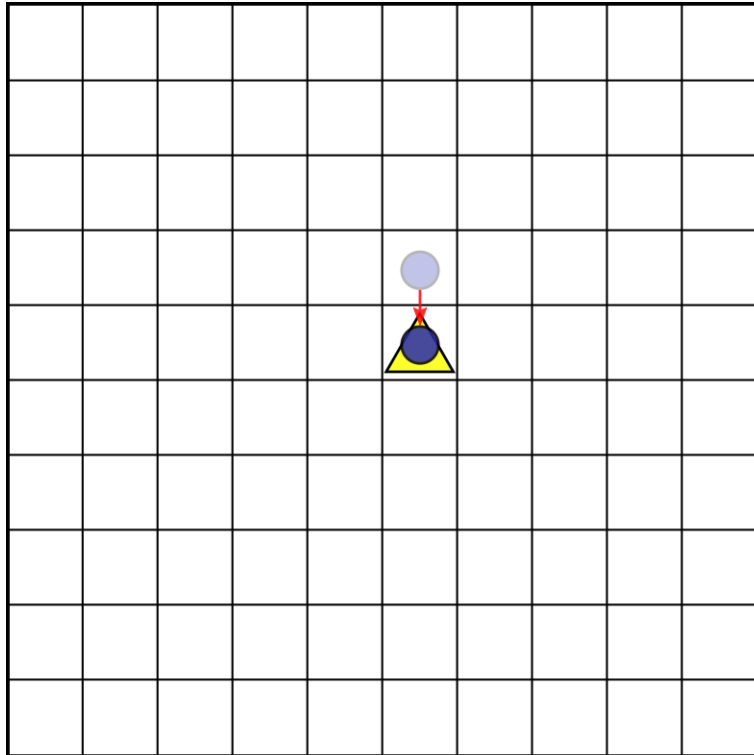
Gridworld as a MDP



- The states are the world coordinates:
 $s = (x, y)$
- The actions are moving on a nearest neighbour or staying still:
 $a \in \{(1,0), (-1,0), (0,1), (0,-1), (0,0)\}$
The states at the boundary have a restricted number of actions.
- Transition probabilities are deterministic:
 $p(s'|a, s) = \delta(s' = a + s)$



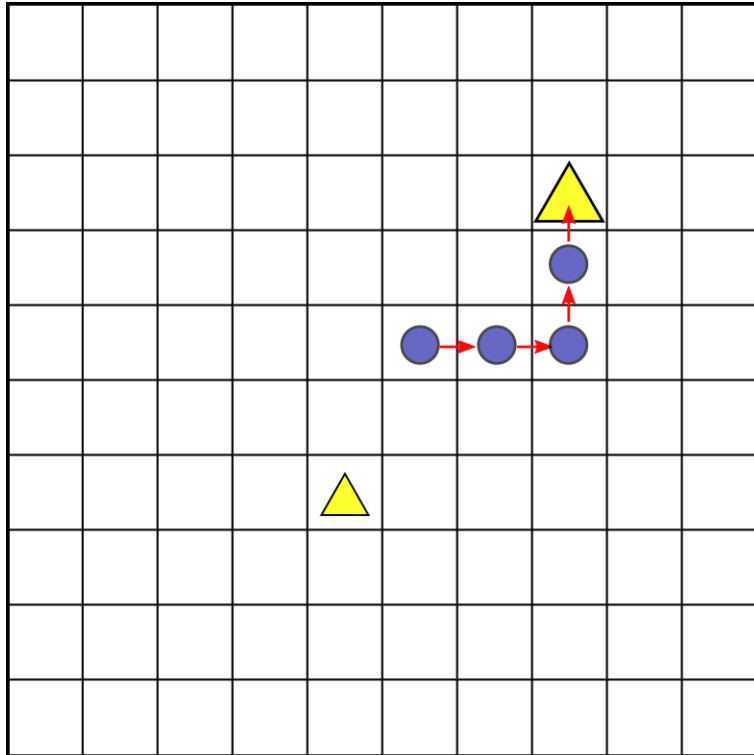
Gridworld as a MDP



- The states are the world coordinates:
 $s = (x, y)$
- The actions are moving on a nearest neighbour or staying still:
 $a \in \{(1,0), (-1,0), (0,1), (0,-1), (0,0)\}$
The states at the boundary have a restricted number of actions.
- Transition probabilities are deterministic:
 $p(s'|a, s) = \delta(s' = a + s)$
- Reward > 0 when the agent moves on or stay in a cell with a resource.



Gridworld as a MDP



Utility function:

$$V_{\pi}(x, y) = E_{\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = (x, y) \right]$$

Policy: which action to take

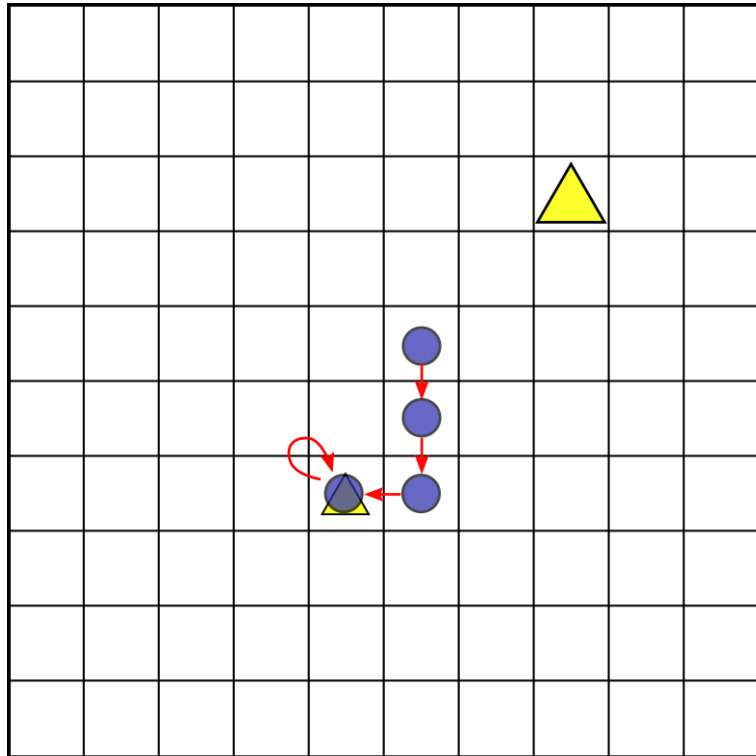
Discount factor

Reward

$$V_{(\rightarrow, \rightarrow, \uparrow, \uparrow)}(x = 5, y = 5) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot \triangle$$



Gridworld as a MDP



Utility function:

$$V_{\pi}(x, y) = E_{\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = (x, y) \right]$$

\downarrow
 Policy: which action to take

\swarrow
 Discount factor

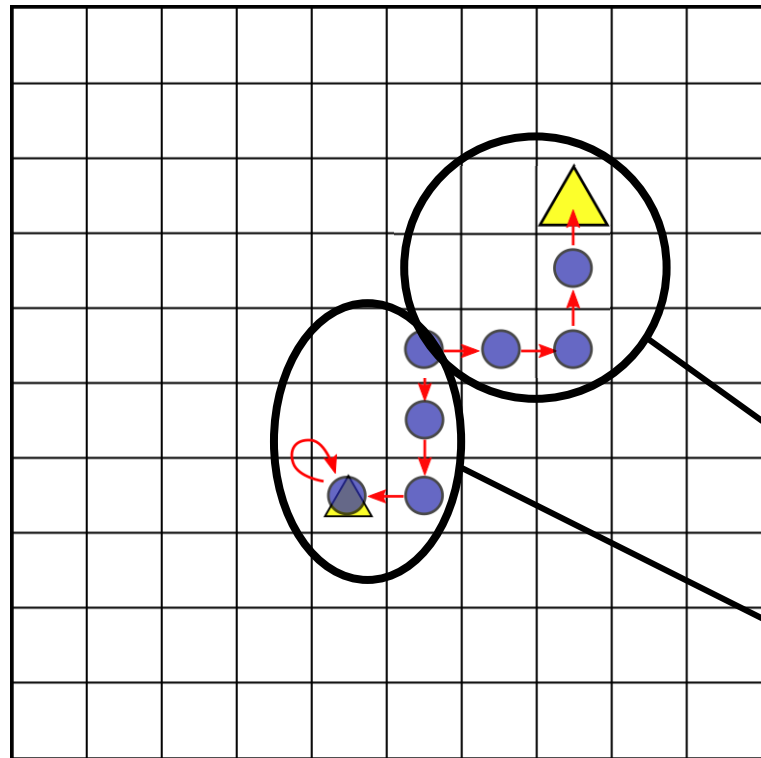
\searrow
 Reward

$$V_{(\rightarrow, \rightarrow, \uparrow, \uparrow)}(x = 5, y = 5) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot \triangle$$

$$V_{(\downarrow, \downarrow, \leftarrow, \cdot)}(x = 5, y = 5) = 0 + \gamma \cdot 0 + \gamma^2 \cdot \triangle + \gamma^3 \cdot \triangle$$



Gridworld as a MDP



Utility function:

$$V_{\pi}(x, y) = E_{\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = (x, y) \right]$$

\downarrow
 Policy: which action to take

\downarrow
 Discount factor

\downarrow
 Reward

$$V_{(\rightarrow, \rightarrow, \uparrow, \uparrow)}(x = 5, y = 5) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot \triangle$$





$$V_{(\downarrow, \downarrow, \leftarrow, \cdot)}(x = 5, y = 5) = 0 + \gamma \cdot 0 + \gamma^2 \cdot \triangle + \gamma^3 \cdot \triangle$$

The best strategy depends on the time horizon: $V_{(\rightarrow, \rightarrow, \uparrow, \uparrow)} > V_{(\downarrow, \downarrow, \leftarrow, \cdot)}$ if $\gamma > \frac{\triangle - \triangle}{\triangle}$

Q-learning

Q-learning is a model-free reinforcement-learning algorithm for any finite Markov decision processes.

Each state-action pair has a quality associated, $Q(s,a)$.

		States (gridworld position)					
		<div>000 100</div>	<div>000 010</div>	<div>000 001</div>	<div>100 000</div>	<div>010 000</div>	<div>001 000</div>
Actions		0.2	0.3	1.0	-0.22	-0.3	0.0
		-0.5	-0.4	-0.2	-0.04	-0.02	0.0
		0.21	0.4	-0.3	0.5	1.0	0.0
		-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Q-learning

Q-learning is a model-free reinforcement-learning algorithm for any finite Markov decision processes.

Each state-action pair has a quality associated, $Q(s,a)$.

States (gridworld position)		000 100	000 010	000 001	100 000	010 000	001 000
Actions	↑	0.2	0.3	1.0	-0.22	-0.3	0.0
	↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
	→	0.21	0.4	-0.3	0.5	1.0	0.0
	←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

At convergence the quality is the best return from a given state taking a given action:

$$Q(s, a) \rightarrow Q^*(s, a) = \max_{\pi} \left[\mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \right]$$

$$\pi^*(s) = \delta(a - \operatorname{argmax}_b Q(s, b))$$

Q-learning

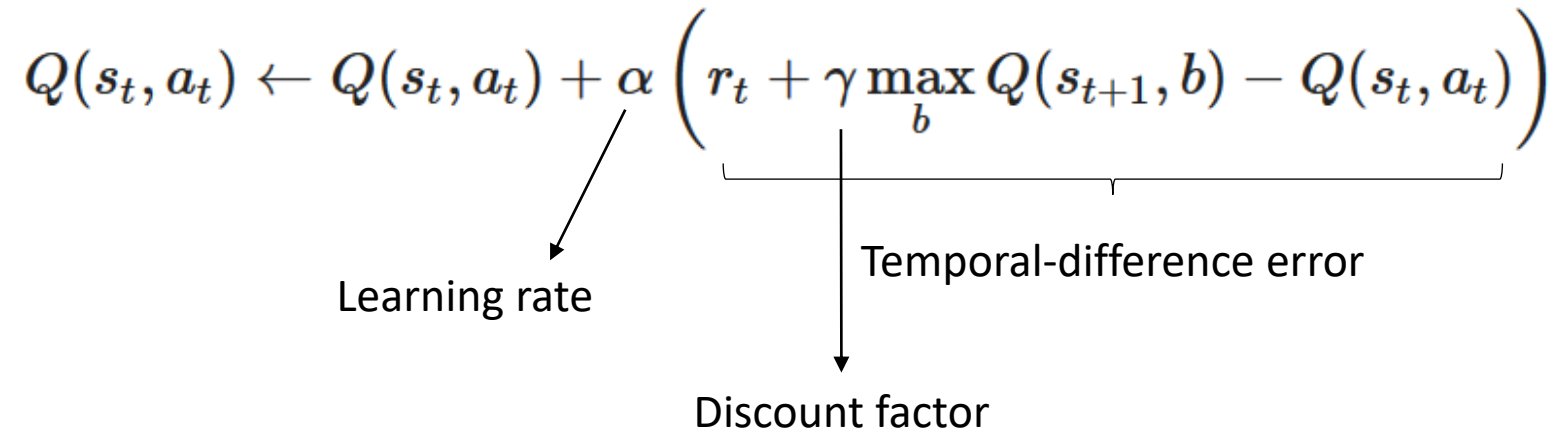
The core of the algorithm is to update the Quality table every game transition:

$$s_t, a_t \rightarrow s_{t+1}, r_t$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(\underbrace{r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)}_{\text{Temporal-difference error}} \right)$$

Learning rate

Discount factor

The diagram shows the Q-learning update equation. The term α is labeled 'Learning rate' with an arrow pointing to it. The term γ is labeled 'Discount factor' with an arrow pointing to it. The entire term in parentheses, $r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)$, is bracketed and labeled 'Temporal-difference error' with an arrow pointing to the bracket.

Q-learning

The core of the algorithm is to update the Quality table every game transition:

$$s_t, a_t \rightarrow s_{t+1}, r_t$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(\underbrace{r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)}_{\text{Temporal-difference error}} \right)$$

Learning rate

Temporal-difference error

Discount factor

Deviation from the Bellman equation in one sample:

$$Q(s_t, a_t)^* = \mathbb{E} \left[r_t + \gamma \max_b Q^*(s_{t+1}, b) \right]$$

Epsilon-greedy Q-learning

How to choose an action \mathbf{a}_t given a state s_t for the transition:

$$s_t, \mathbf{a}_t \rightarrow s_{t+1}, r_t \quad ?$$

- **Exploration** move: choose at random.
- **Exploitation** move: choose the action that maximizes the current quality function.

Epsilon greedy rule: choose exploration with probability epsilon, choose exploitation otherwise.

Pseudocode, first version

- Initialize the Q-matrix and choose the algorithm parameters γ, α, ϵ .
- Set the agent in the starting state s_0 .
- For $t = 1, \dots$ until convergence:
 - With probability ϵ choose a_t at random from the possible actions, otherwise choose the action that maximizes the Qualities $a_t = \operatorname{argmax}_b Q(s_t, b)$.
 - Play a step in the game and get the new state and the reward $s_t, a_t \rightarrow s_{t+1}, r_t$
 - Update the quality matrix using the obtained sample

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t) \right)$$

Pseudocode, second version

Two tricks to improve the performance:

- **Episodic training:** After T steps the game restarts from s_0 . Each one of these runs is called an episode.
- **Epsilon scheduling:** I need more exploration at the beginning to have a general idea of the qualities, and less at the end, in order to improve the estimates around the «best» trajectory. Epsilon decreases with time.

Pseudocode, second version

- Initialize the Q-matrix and choose the algorithm parameters $\gamma, \alpha, \epsilon_0, T_{episode}$.

For episodes $e = 1, \dots$ until convergence:

- Set the agent in the starting state s_0 .
- For steps in the episode $t = 1, \dots, T_{episode}$:
 - With probability ϵ_e choose a_t at random from the possible actions, otherwise choose the action that maximizes the Qualities $a_t = \operatorname{argmax}_b Q(s_t, b)$.
 - Play a step in the game and get the new state and the reward $s_t, a_t \rightarrow s_{t+1}, r_t$
 - Update the quality matrix using the obtained sample

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t) \right)$$

- Decrease the exploration rate ϵ_e .