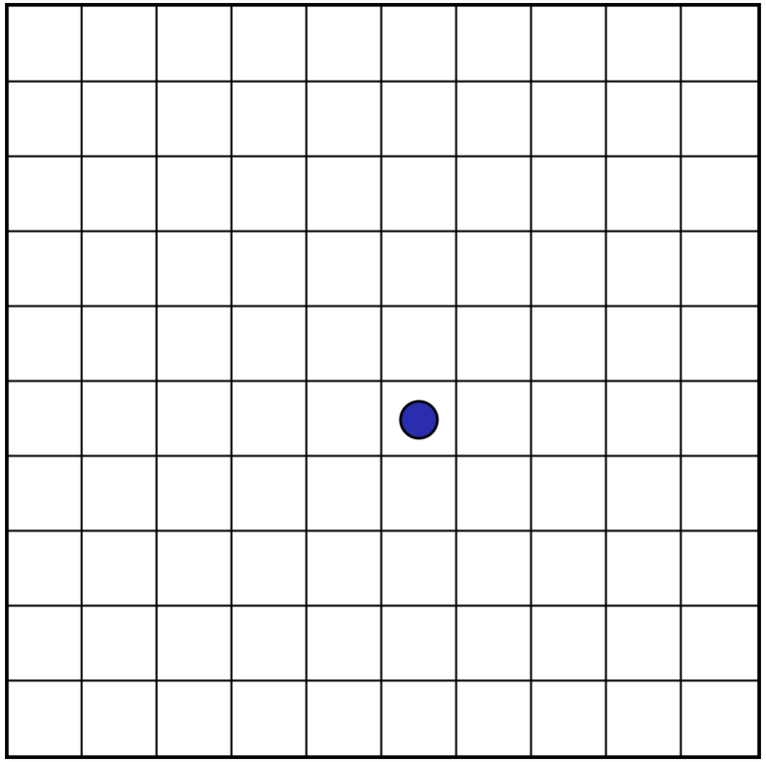
Gridworld with Q-learning

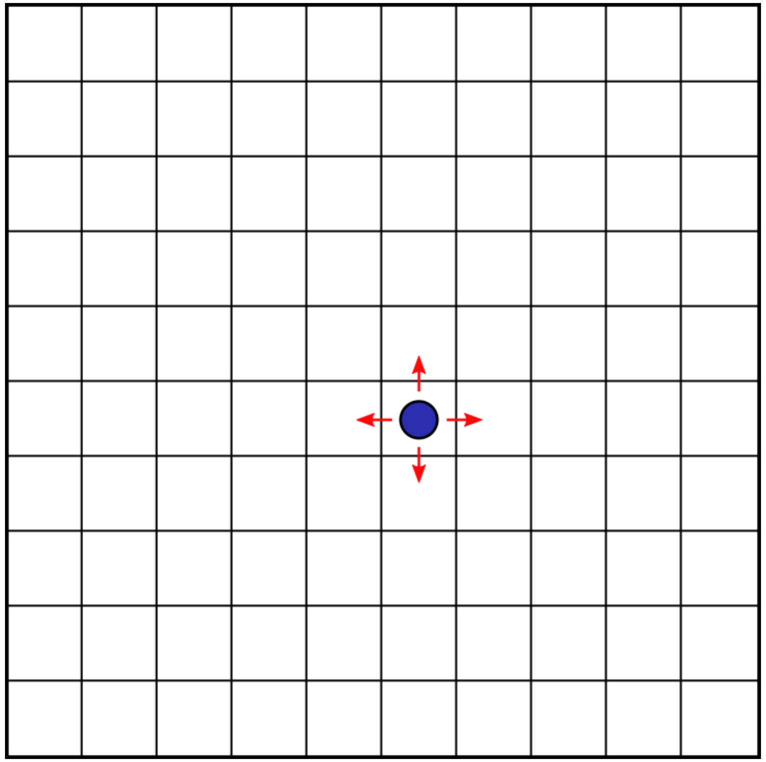
Andrea Mazzolini, HPC 2020

Gridworld



$d \times d$ 2-dimensional lattice.

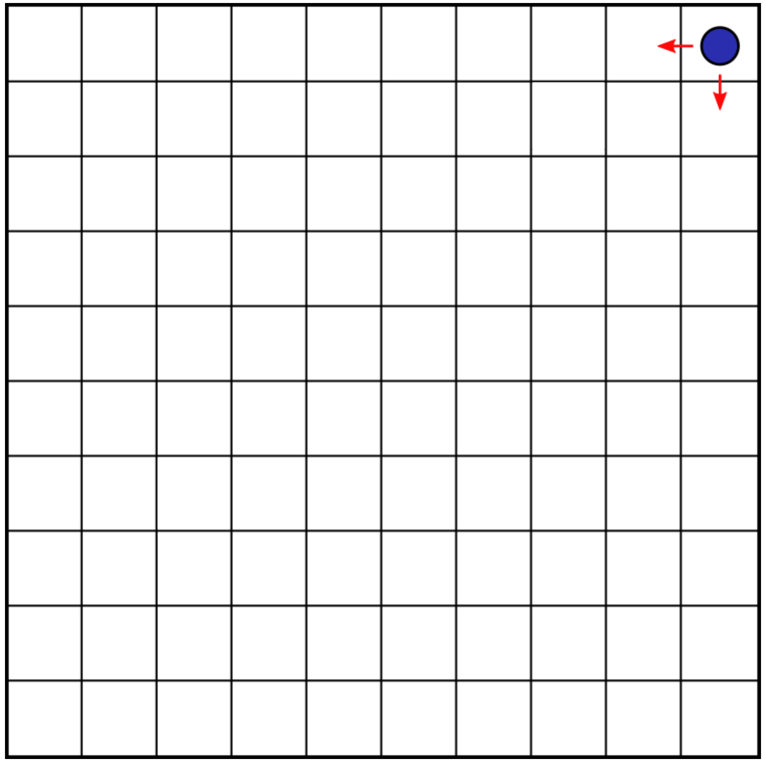
Gridworld



$d \times d$ 2-dimensional lattice.

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

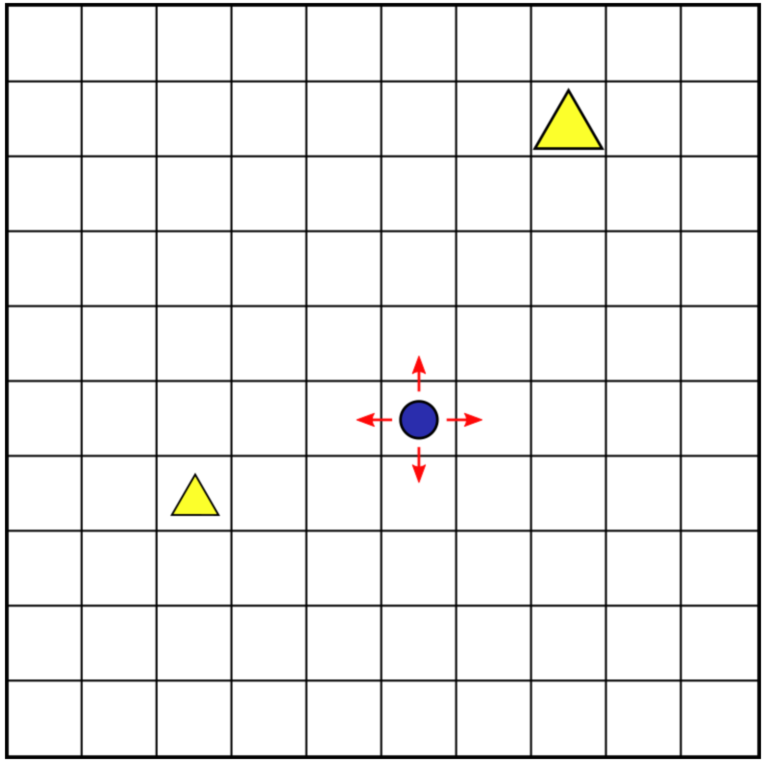
Gridworld



$d \times d$ 2-dimensional lattice.

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

Gridworld

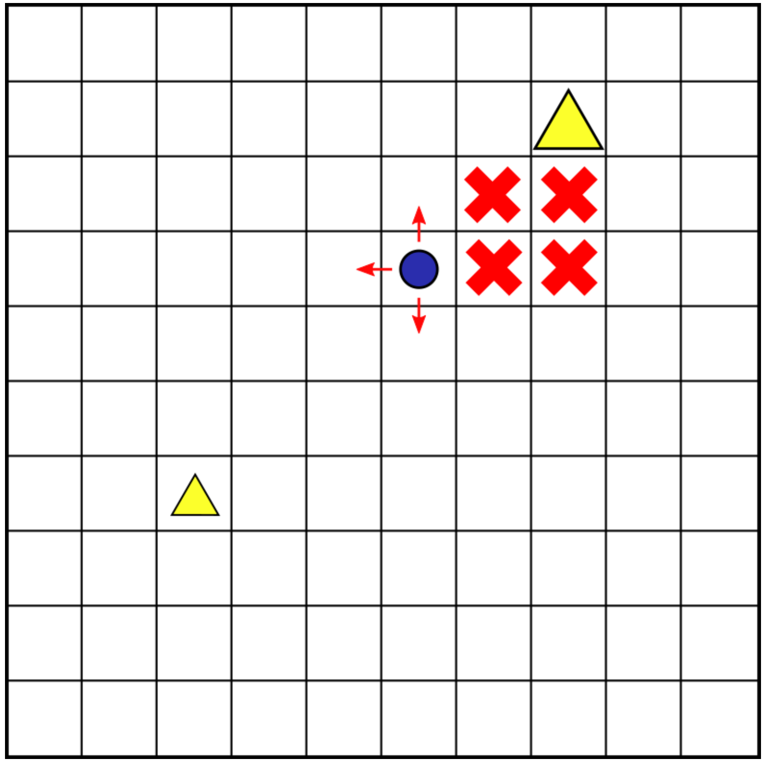


$d \times d$ 2-dimensional lattice.

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



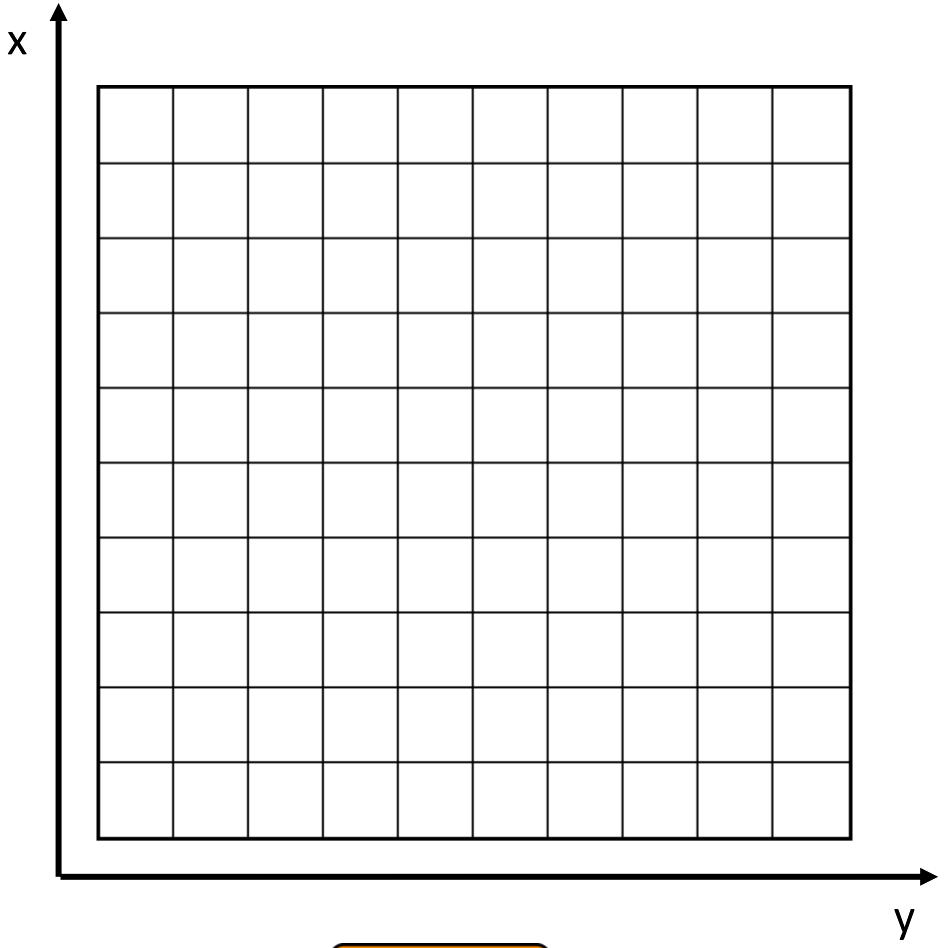
$d \times d$ 2-dimensional lattice.

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.

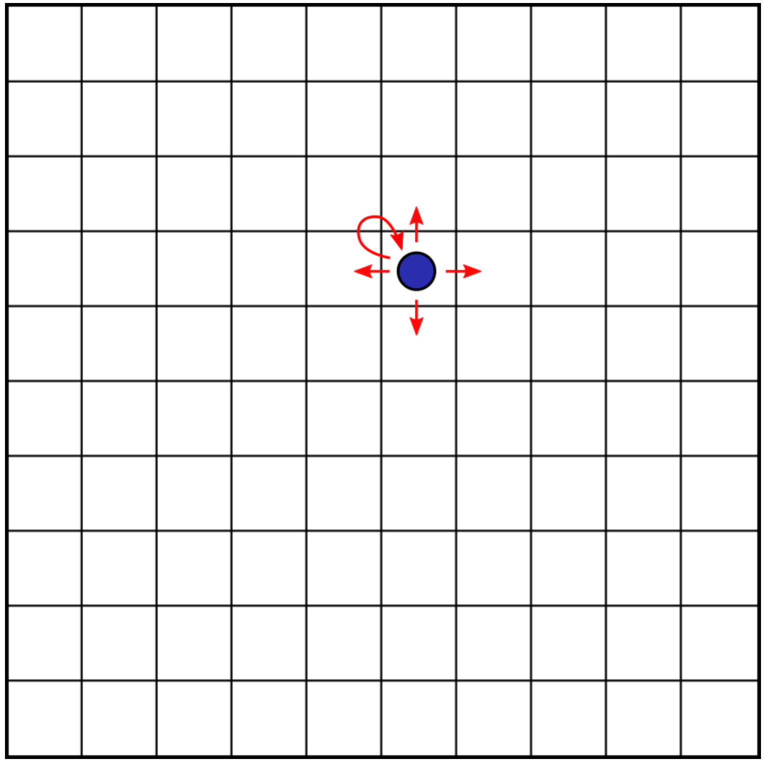
The player cannot move over obstacles and outside the girdworld.

Gridworld as a MDP



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

Gridworld as a MDP



- The states are the world coordinates:

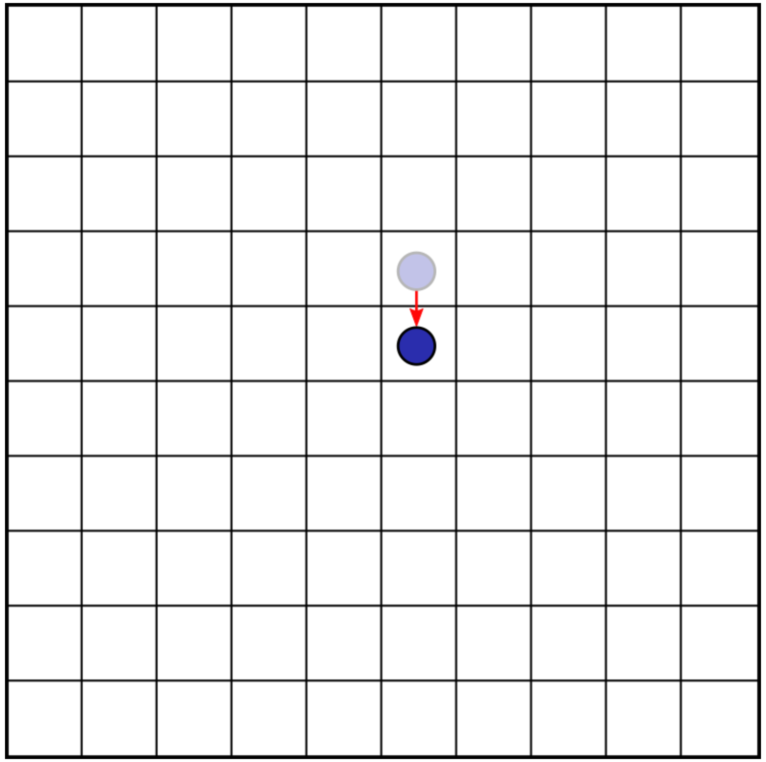
$$s = (x, y)$$

- The actions are moving on a nearest neighbour or staying still:

$$a \in \{(1{,}0), (-1{,}0), (0{,}1), (0, -1), (00)\}$$

At the boundary or close to obstacles: restricted number of actions.

Gridworld as a MDP

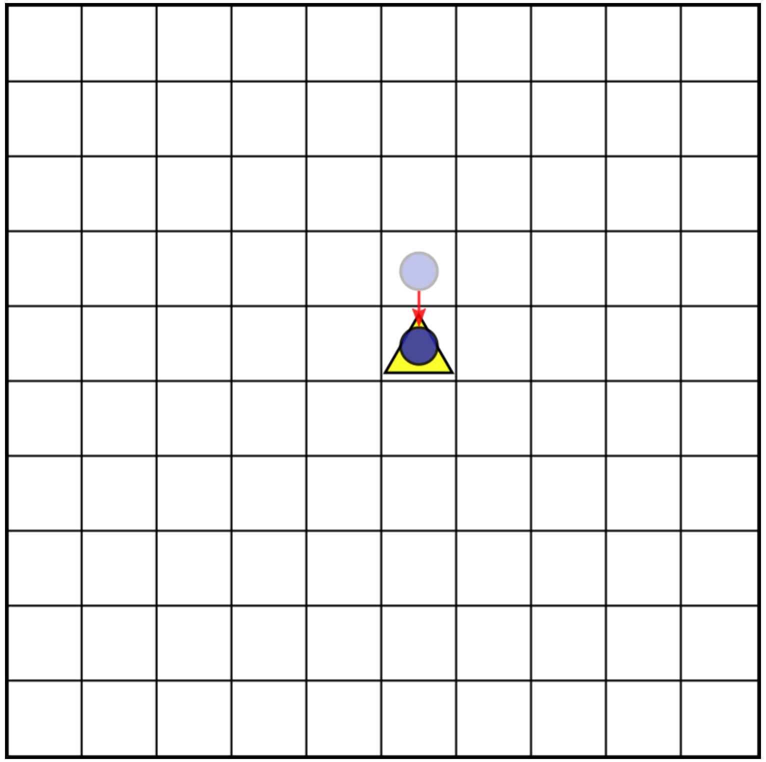


- The states are the world coordinates:
$$s = (x, y)$$

- The actions are moving on a nearest neighbour or staying still:
$$a \in \{(1,0), (-1,0), (0,1), (0,-1), (00)\}$$
At the boundary or close to obstacles: restricted number of actions.

- Transition probabilities are deterministic:
$$p(s'|a, s) = \delta(s' = a + s)$$

Gridworld as a MDP



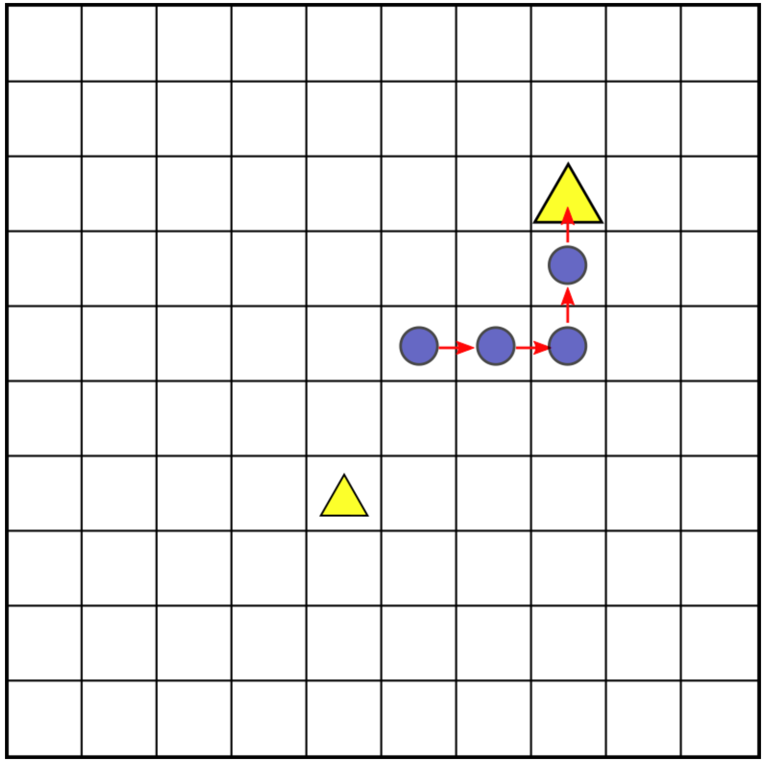
- The states are the world coordinates:
$$s = (x, y)$$

- The actions are moving on a nearest neighbour or staying still:
$$a \in \{(1,0), (-1,0), (0,1), (0,-1), (00)\}$$
At the boundary or close to obstacles: 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:

$$V_\pi(x,y) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|\, s_0 = (x,y)\right]$$

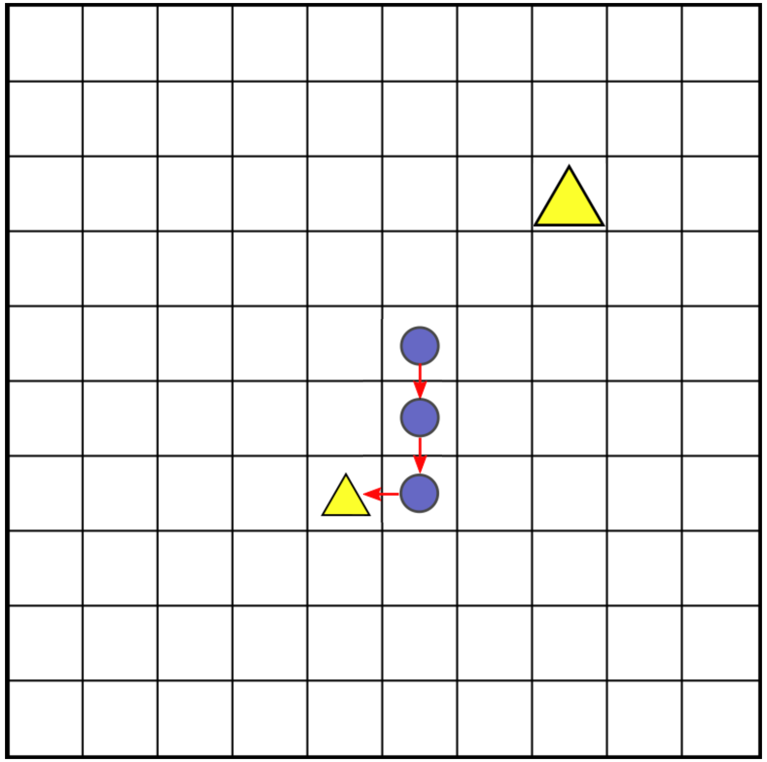
Policy: which action to take

Discount factor

Reward

$$V_{(\rightarrow,\rightarrow,\uparrow,\uparrow,\,alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \frac{\gamma^3}{1-\gamma} \cdot \triangle$$

Gridworld as a MDP



Utility function:

$$V_\pi(x,y) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t \,|\, s_0 = (x,y)\right]$$

Policy: which action to take

Discount factor

Reward

$$V_{(\rightarrow,\rightarrow,\uparrow,\uparrow,\, alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \frac{\gamma^3}{1-\gamma} \cdot \triangle_{\text{large}}$$

$$V_{(\downarrow,\downarrow,\leftarrow,\, alw\ stay)}(x,y) = 0 + \gamma \cdot 0 + \frac{\gamma^2}{1-\gamma} \cdot \triangle_{\text{small}}$$

The best strategy depends on the **discount factor**: $V_{(\rightarrow,\rightarrow,\uparrow,\uparrow,\, alw\ stay)} > V_{(\downarrow,\downarrow,\leftarrow,\, alw\ stay)}$ if $\gamma > \dfrac{\triangle_{\text{small}}}{\triangle_{\text{large}}}$

Discount factor and time horizon

$$V_\pi(s) = E_{p,\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s\right]$$

Discounted value function as defined before

$$V'_\pi(s) = E_{p,\pi,\boldsymbol{\gamma}}\left[\sum_{t=0}^{\infty} R_t | S_0 = s\right]$$

Undiscounted value function of a process that has probability $1 - \gamma$ of being stopped

Discount factor and time horizon

$$V_\pi(s) = E_{p,\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s\right]$$

Discounted value function as defined before

$$V'_\pi(s) = E_{p,\pi,\boldsymbol{\gamma}}\left[\sum_{t=0}^{\infty} R_t | S_0 = s\right]$$

Undiscounted value function of a process that has probability $1 - \gamma$ of being stopped

Equivalent problems

$$V'_\pi(s) = E_{p,\pi,\boldsymbol{\gamma}}\left[\sum_{t=0}^{\infty} R_t | S_0 = s\right] = E_{p,\pi}\left[\sum_{t=0}^{\infty} P(game\ lasts\ t\ steps) R_t | S_0 = s\right] = E_{p,\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s\right] = V_\pi(s)$$

Discount factor and time horizon

$$V'_\pi(s) = E_{p,\pi,\boldsymbol{\gamma}}\left[\sum_{t=0}^{\infty} R_t | S_0 = s\right]$$

Undiscounted value function of a process that has probability $1-\gamma$ of being stopped

Expected duration of the process?

$$\langle t\rangle_\gamma = \sum_{t=0}^{\infty} t\, P(\tau = t) = \sum_{t=0}^{\infty} t\,(1-\gamma)\gamma^{t-1} = (1-\gamma)\sum_{t=0}^{\infty}\frac{d\gamma^t}{d\gamma} = (1-\gamma)\frac{d}{d\gamma}\sum_{t=0}^{\infty}\gamma^t = \frac{1}{1-\gamma}$$

Discount factor and time horizon

$$V'_\pi(s) = E_{p,\pi,\boldsymbol{\gamma}}\left[\sum_{t=0}^{\infty} R_t | S_0 = s\right]$$

Undiscounted value function of a process that has probability $1-\gamma$ of being stopped

Expected duration of the process?

$$\langle t \rangle_\gamma = \sum_{t=0}^{\infty} t\, P(\tau = t) = \sum_{t=0}^{\infty} t\,(1-\gamma)\gamma^{t-1} = (1-\gamma)\sum_{t=0}^{\infty} \frac{d\gamma^t}{d\gamma} = (1-\gamma)\frac{d}{d\gamma}\sum_{t=0}^{\infty}\gamma^t = \frac{1}{1-\gamma}$$

$\frac{1}{1-\gamma}$ is the **time horizon** of the process

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)

Actions

	000 100	000 010	000 001	100 000	010 000	001 000
↑	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)

Actions

	000 100	000 010	000 001	100 000	010 000	001 000
⇧	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:

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

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

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

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

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 $\boldsymbol{a_t}$ given a state s_t for the transition:

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

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

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 $t = 1, \ldots$ until convergence:

> - With probability ϵ choose a_t at random from the possible actions, otherwise choose the action that maximizes the Qualities $a_t = \mathrm{argmax}_b Q(s_t, b)$.
> - 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 γ, α, ϵ_0, $T_{episode}$.

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

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