Introduction to Q-learning and Deep Q-networks
Homeproblem B: Playing Tetris using Reinforcement: Learning

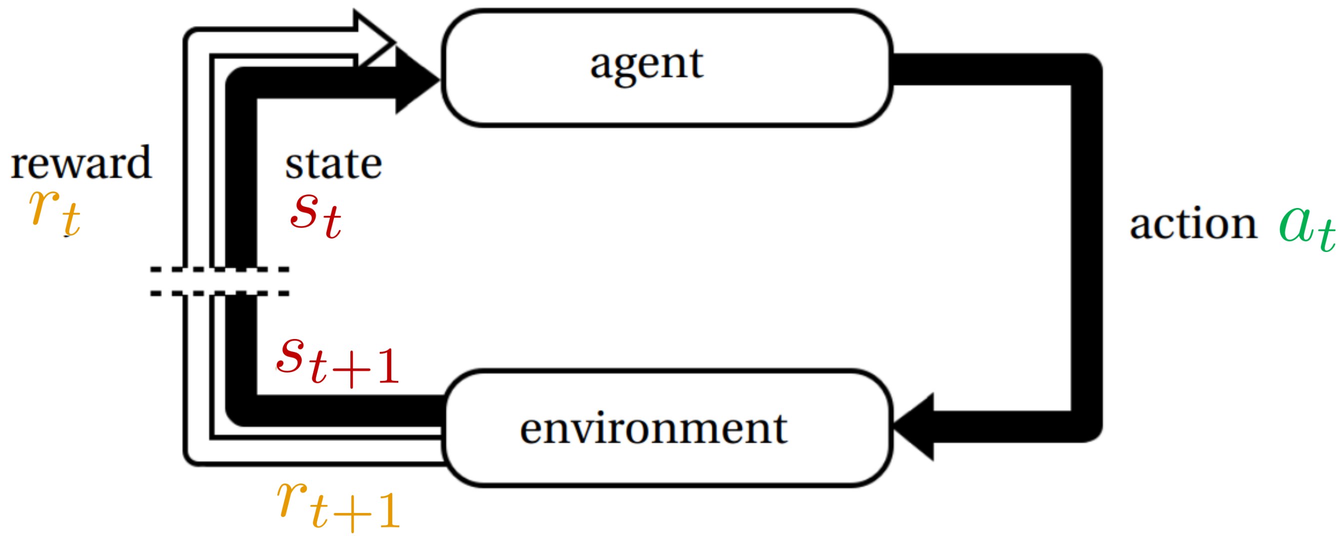
Advanced machine learning with neural networks

Kristian Gustavsson, Department of Physics, University of Gothenburg, Sweden
kristian.gustafsson@physics.gu.se

What is Reinforcement Learning?

Reinforcement Learning: An introduction, Sutton & Barto (2018)
Machine learning with neural networks, Mehlig (2021)

Scheme that uses trial and error to find optimal strategies to achieve a predetermined task.



Typical learning framework

Learning uses a reward signal only (no supervisor).

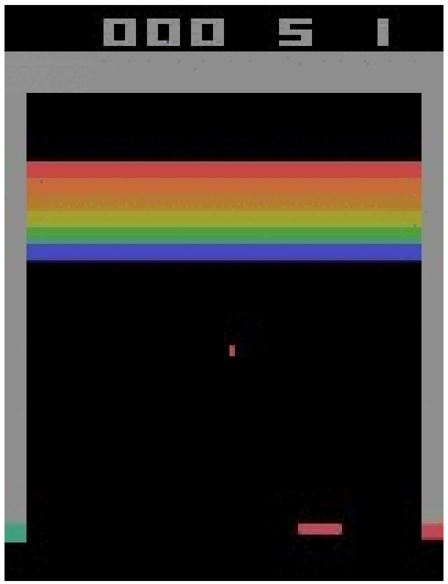
Example applications

- Gaming strategies
- Control theory
- Industrial optimizations
- Understanding nature

Minih et al, Nature **518** 529-533 (2015)

Application: Gaming strategies

before training

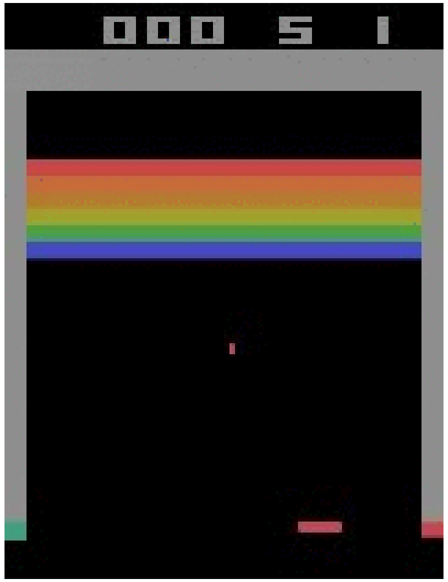


Learning to play 'Breakout'

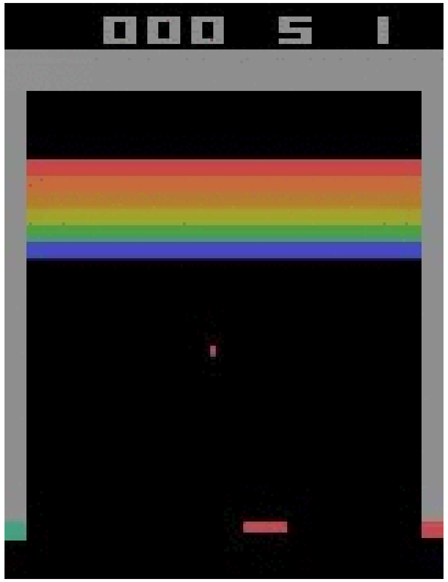
Minih et al, Nature **518** 529-533 (2015)

Application: Gaming strategies

before training



during training

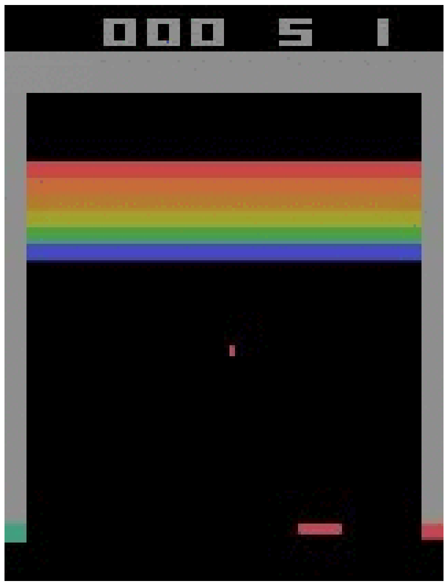


Learning to play 'Breakout'

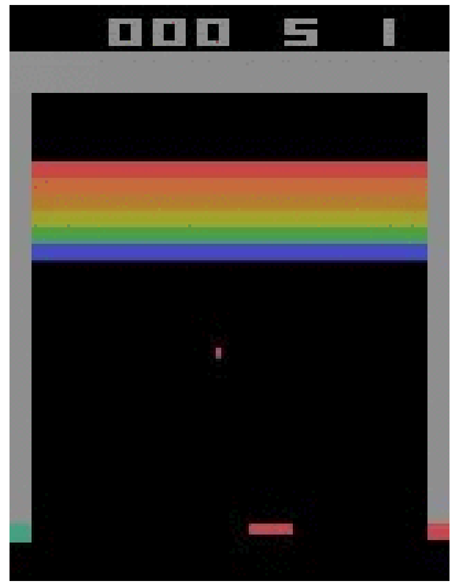
Minih et al, Nature **518** 529-533 (2015)

Application: Gaming strategies

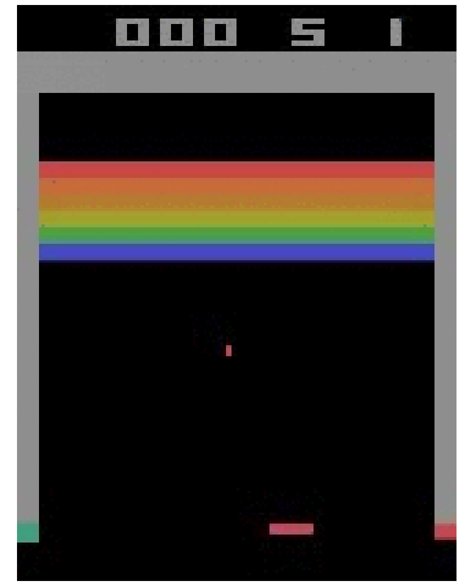
before training



during training



after training



Learning to play 'Breakout'

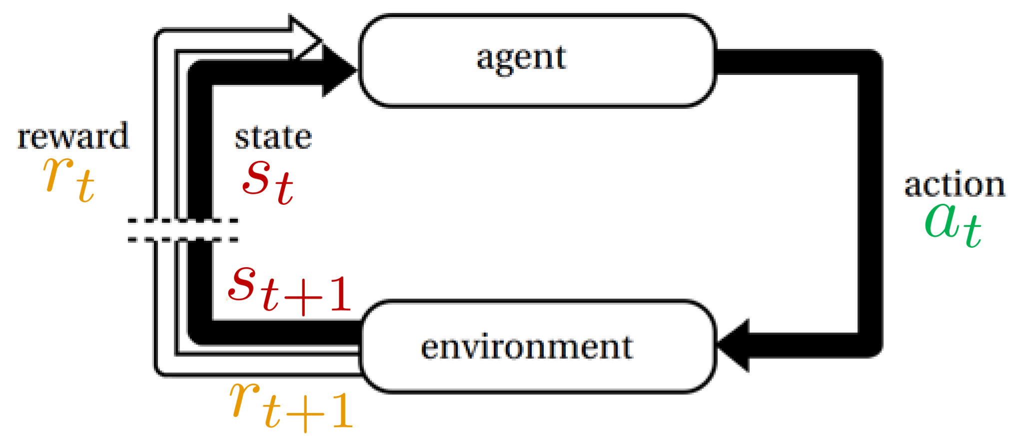
Agent The player

Environment The game (and player)

States Four most recent game images

Actions Joystick direction

Reward Increment in game score



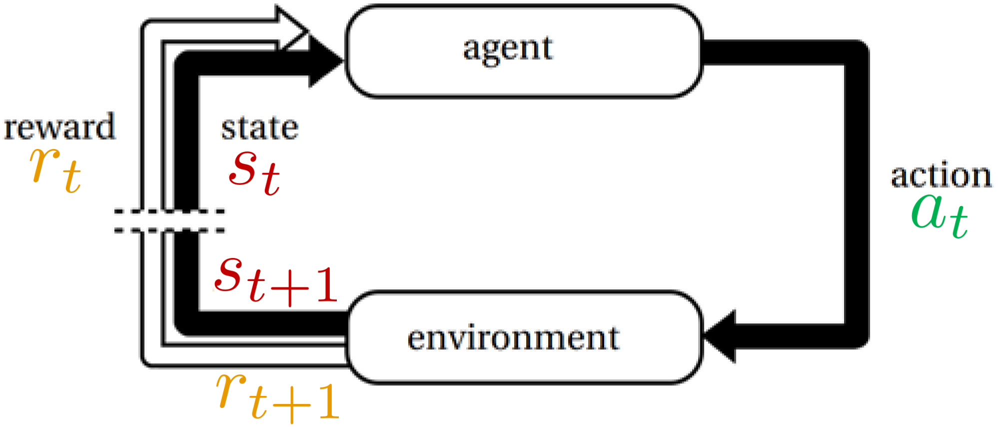
Application: Gaming strategies

Silver et al, Nature **529** 484-489 (2016)
Nature **550** 354-359 (2017)

Learning to play 'Go'

Top view

Agents Both players
Environment The game (and players)
States Positions of game markers
Actions Place new game marker
Reward + on win; – on lose



Application: Control theory

Abbeel, Coates & Ng, IJRR (2010)

Helicopter aerobatics

Agent The helicopter operator

Environment The helicopter and the surroundings

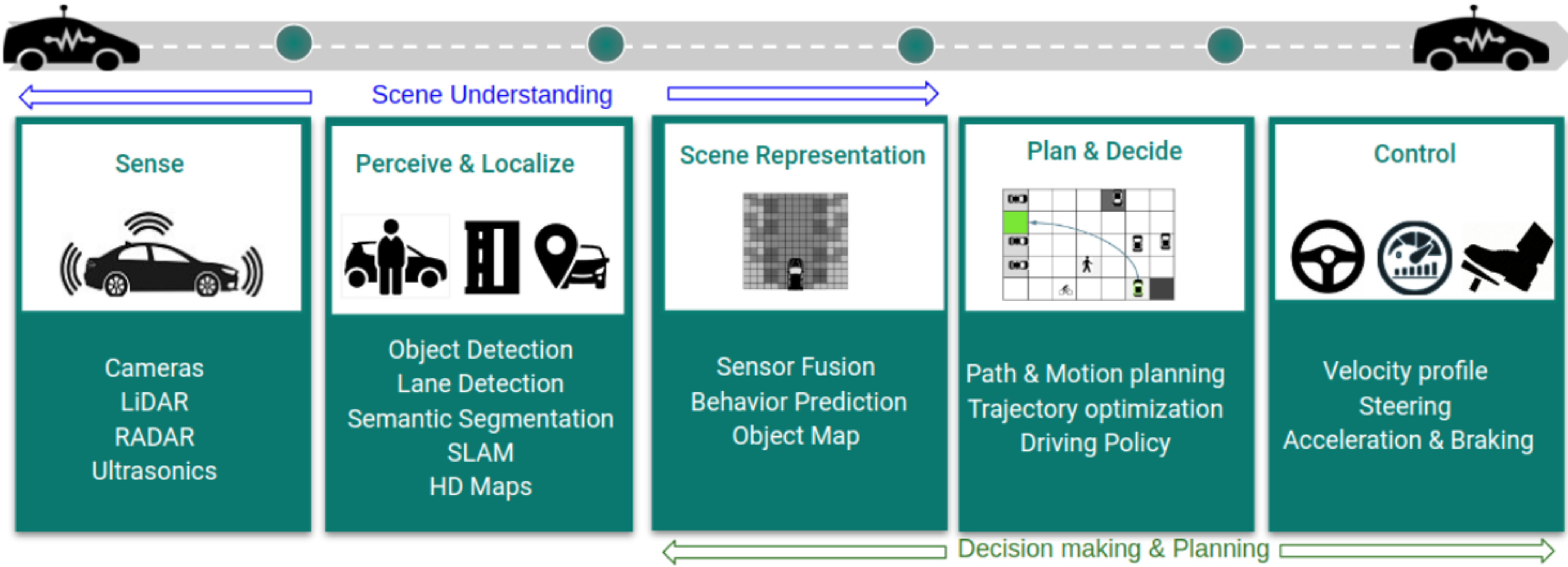
States Position, orientation, velocity, and angular velocity

Actions Change main rotor tilt or angle of attack, change tail rotor thrust

Reward Penalty for deviations from target trajectory

Kiran et al, IEEE Transactions on Intelligent Transportation Systems (2021)

Application: Control theory

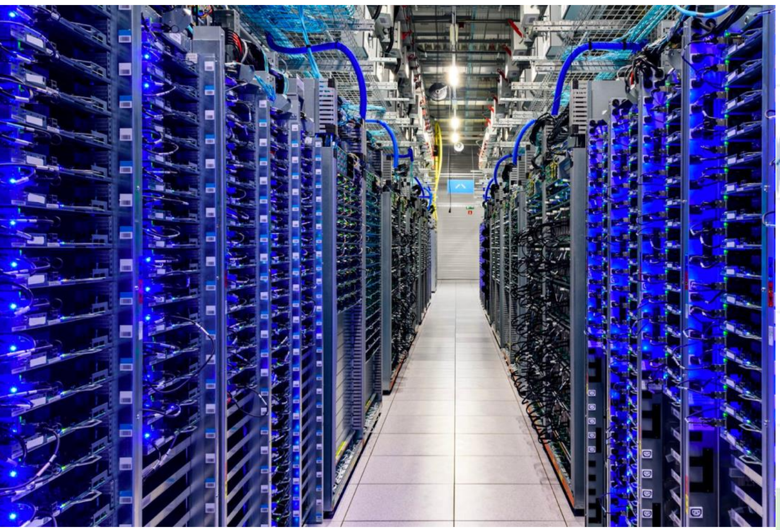


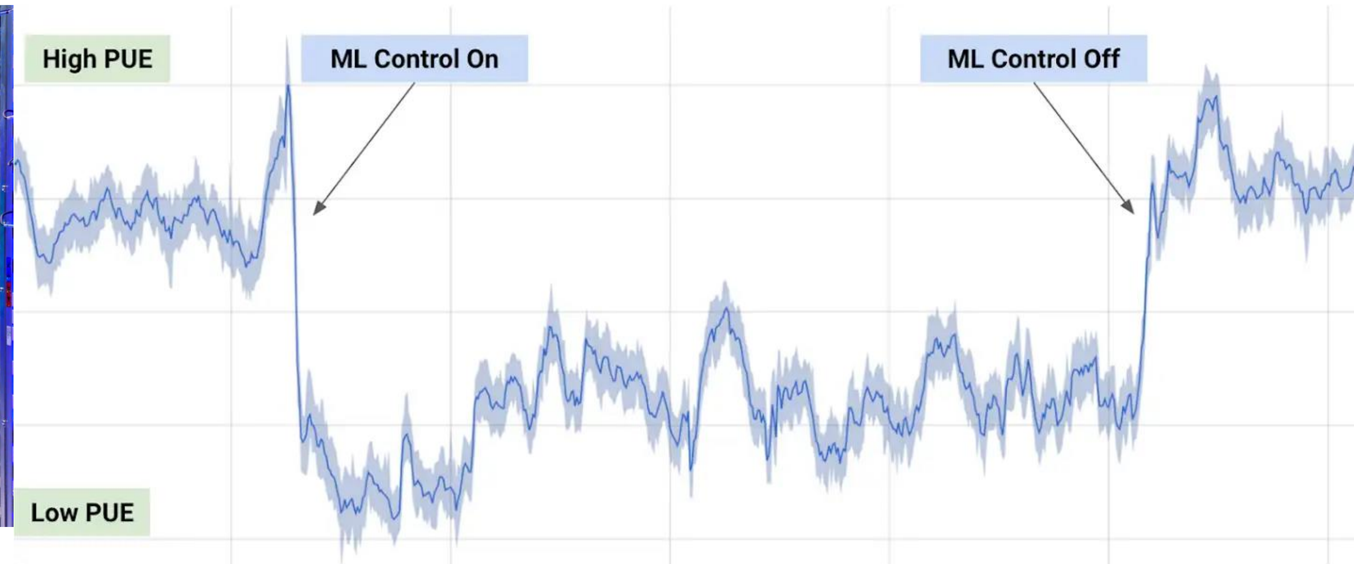
Decision making and planning in self-driving cars

Agent The driver
Environment The car and the surroundings
States Position and velocity. Scene representation.
Actions Steering, accelerating, braking. Control blinkers, lights, horn etc.
Reward + for following road and reaching target; Neg. for collision or crash

Li et al, IEEE Transactions on Cybernetics (2020)

Application: Industrial optimizations



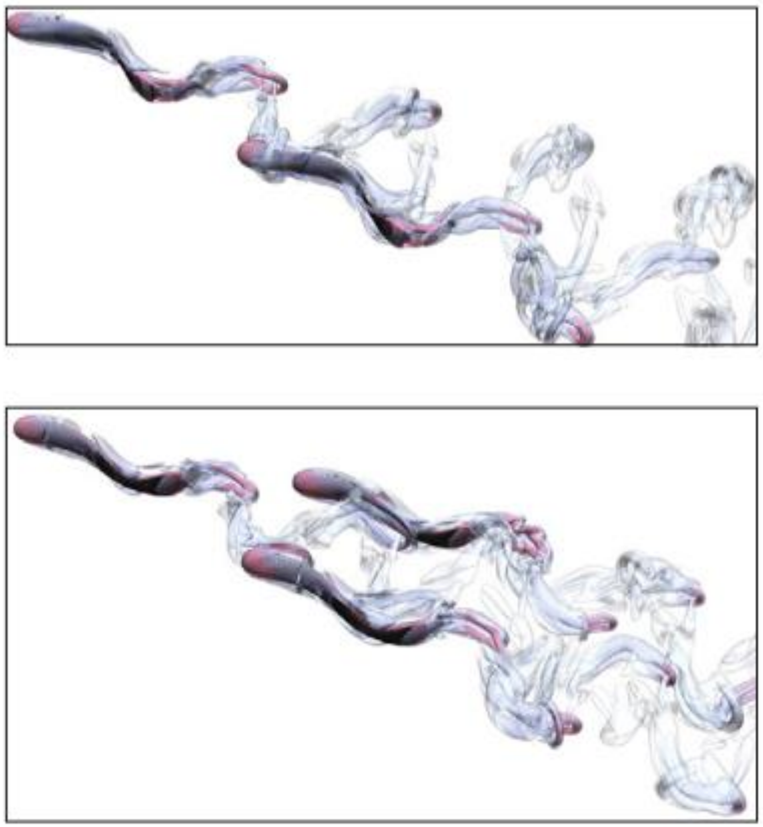


Reducing cooling energy consumption by 40% in Google servers

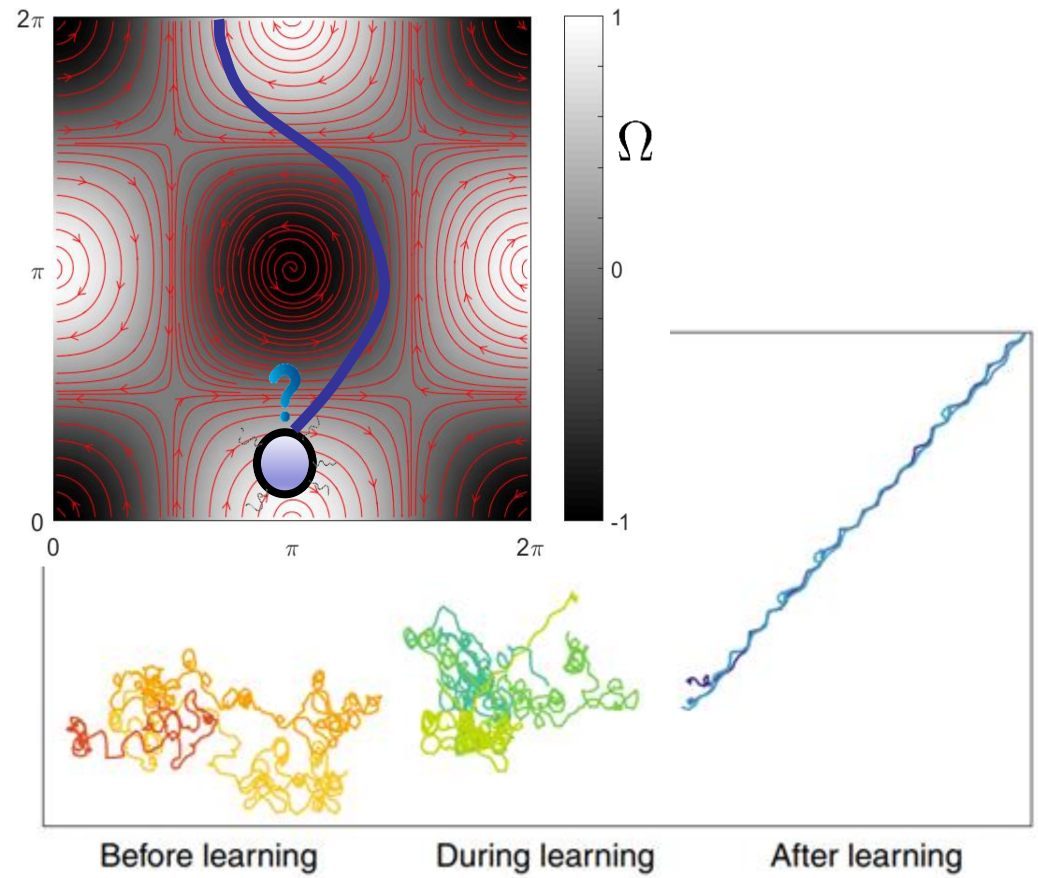
Agent Controller of cooling system
Environment Data center
States Workload and ambient temperature
Actions Control cooling system
Reward Neg. for power usage or outlet temperature outside specified range

Application: Understanding nature

Verma et al, PNAS **115**, 5849–5854 (2018)



Colabrese et al, PRL **118**, 158004 (2017)



Swimming strategies in turbulent flows

Agent Swimmer

Environment Turbulent flow

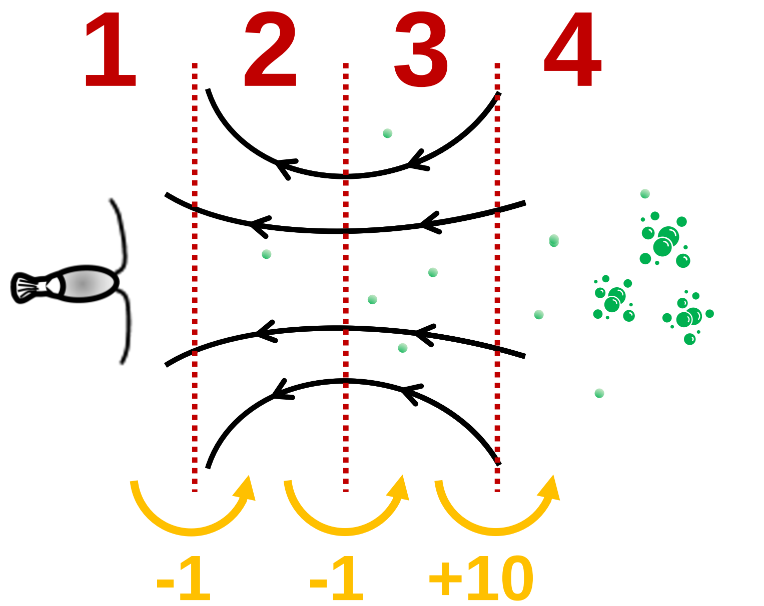
States Flow properties

Actions Swimming behavior

Reward Spend little energy, swimming in target direction

Example: Swimming against the flow

- **States** Flow regions **1**, **2**, **3** and 4
- **Actions**
- → Move one state right
- ← Move one state left
- **Rewards**
- **0** for moving one state left
- **-1** for state 1 to state 2
- **-1** for state 2 to state 3
- **+10** for state 3 to state 4
- **Q-table** (stores experience)

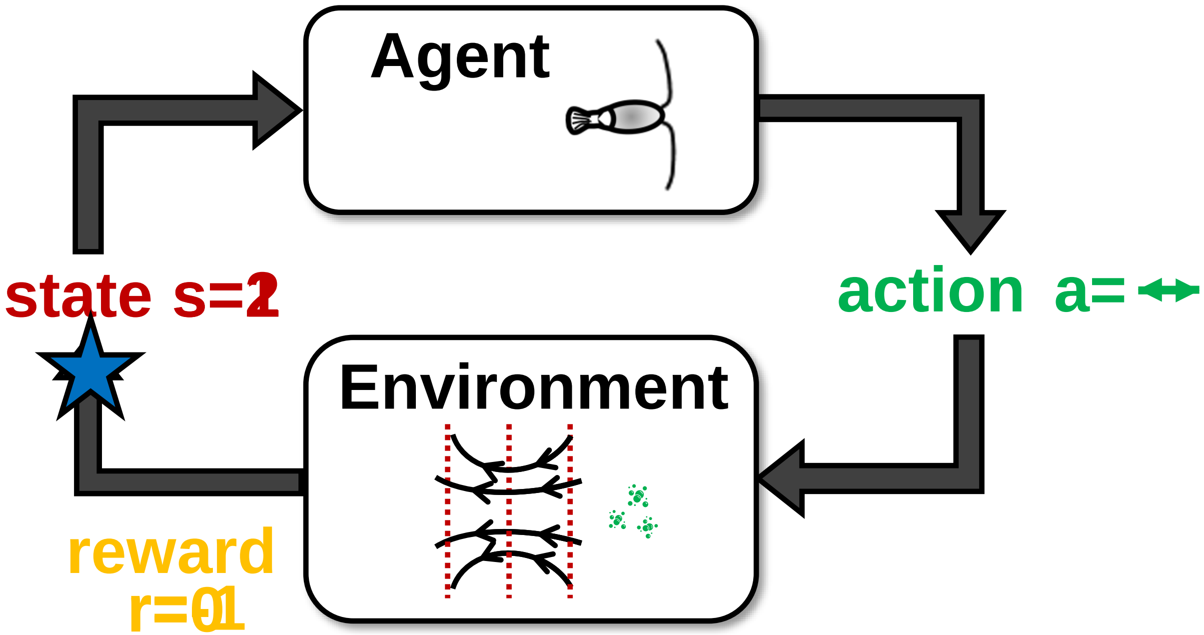


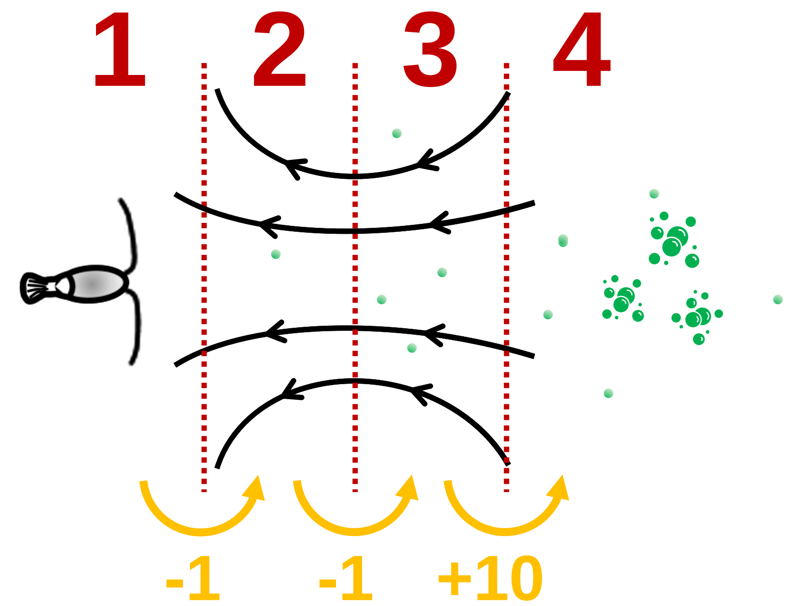
state

action	1	2	3	4
→	0	0	0	0
←	0	0	0	0

Q-table

Example: Swimming against the flow





Choose 'greedy' action

$$a = \mathrm{argmax}_{a'} Q(s, a')$$

Update Q-table

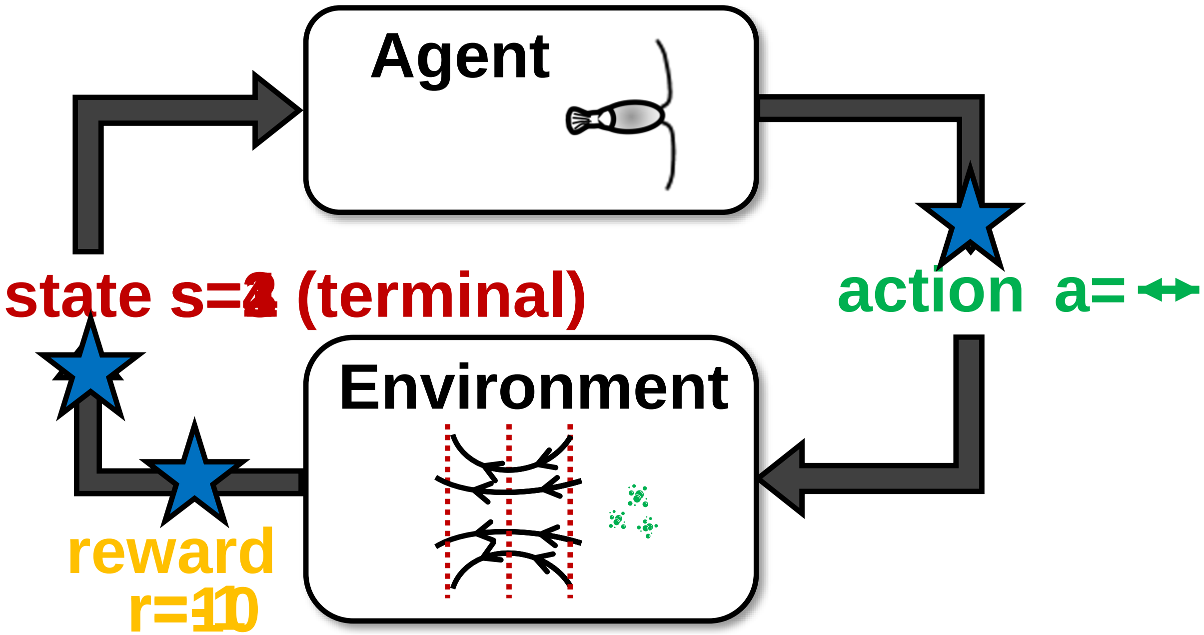
$$Q(s, a) = r$$

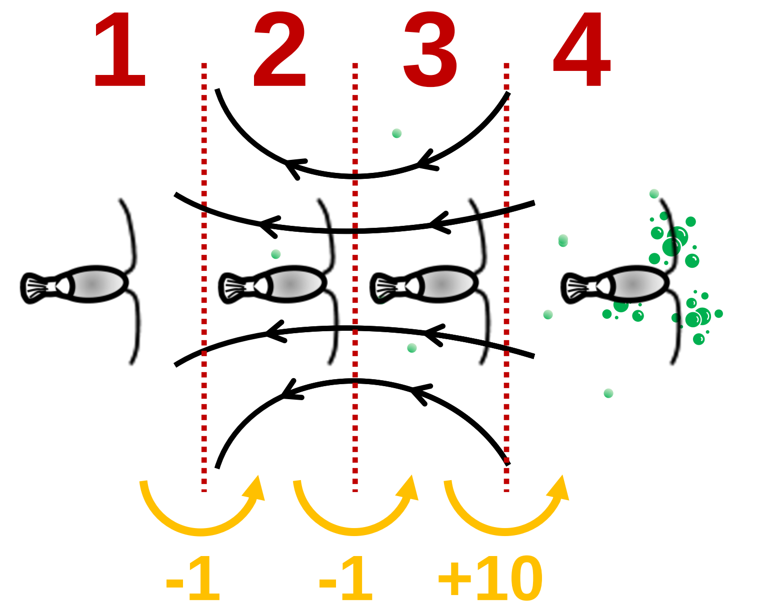
action \ state	1	2	3	4
→	-1	0	0	0
←	0	0	0	0

Q-table

> The scheme gets stuck at state **1**
> => Need to **add exploration**

Example: Swimming against the flow





Choose 'ε-greedy' action

probability ε — random a

probability $1-\varepsilon$ — $\text{argmax}_{a'} Q(s, a')$

Update Q-table

$$Q(s, a) = r$$

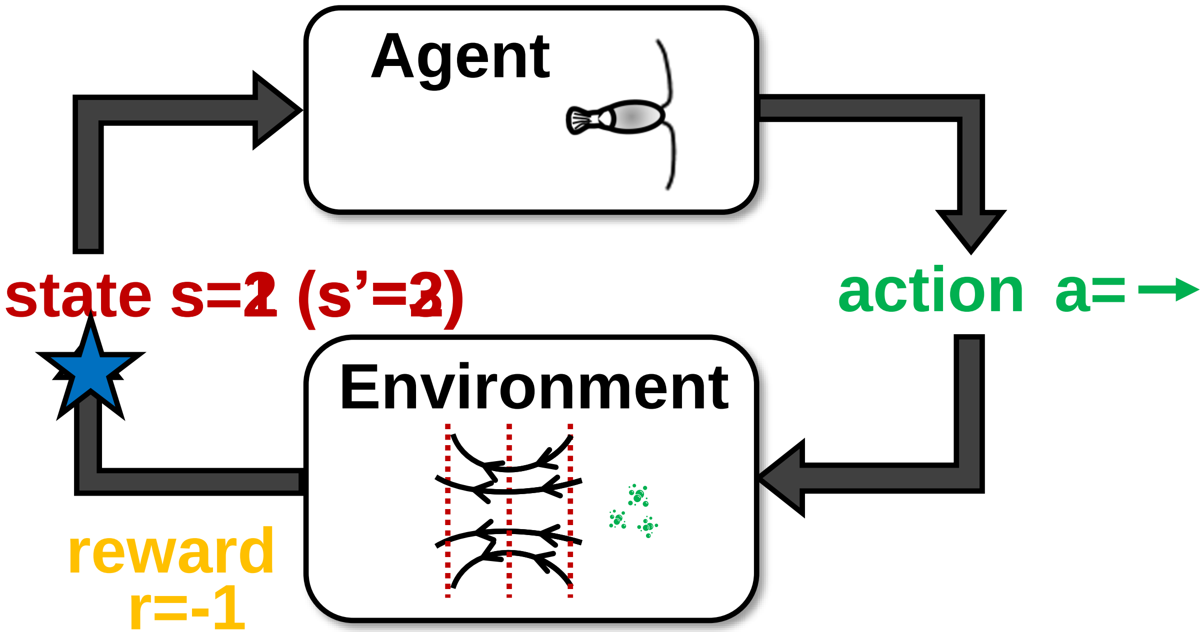
state

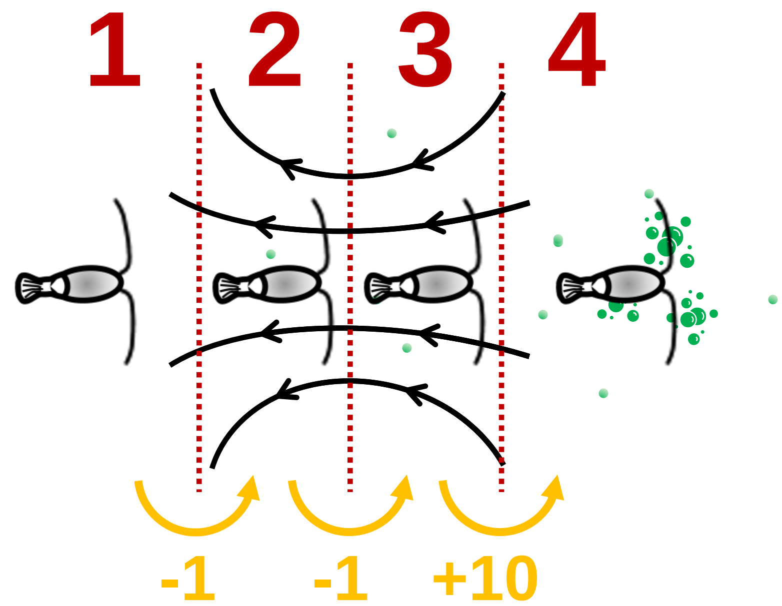
action	1	2	3	4
→	-1	-1	10	0
←	0	0	0	0

Q-table

Exploration => all states are reached.
But Q-table does not give a good strategy.
=> Need to improve update of Q

Example: Swimming against the flow





Choose 'ε-greedy' action

probability ε — random a

probability $1-\varepsilon$ — $\mathrm{argmax}_{a'} Q(s,a')$

action \ state	1	2	3	4
→	8	9	10	0
←	0	0	0	0

Q-table

Update Q-table (Q-learning)

$$Q(s,a) = r + \max_{a'} Q(s',a')$$

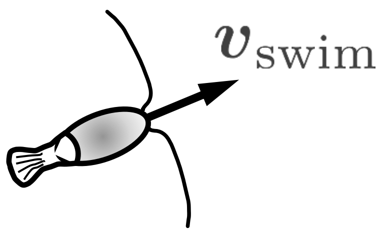
here s' is the next state.

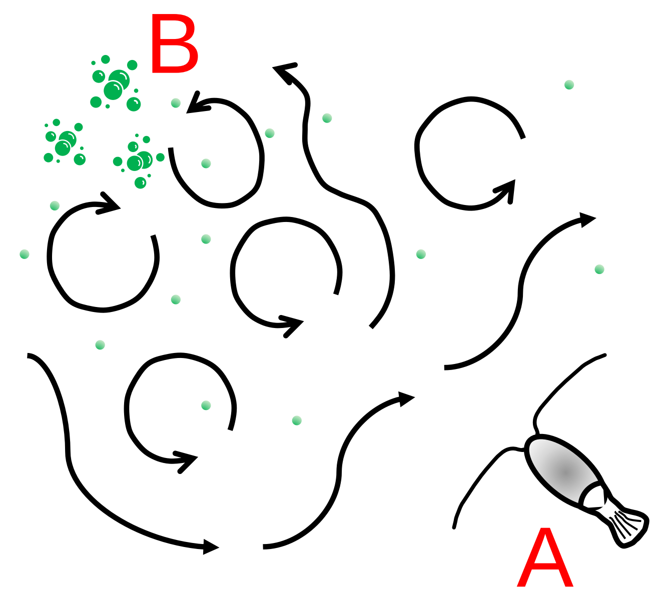
Q-table gives successful strategy

Zermelo, Z. Angew. Math. Mech. **11**, 114–124 (1931); Biferale et al, Chaos **29**, 103138 (2019)

Example: Zermelo's problem

- Navigation from point A to point B in a background flow
- Constant swimming speed v_{swim}



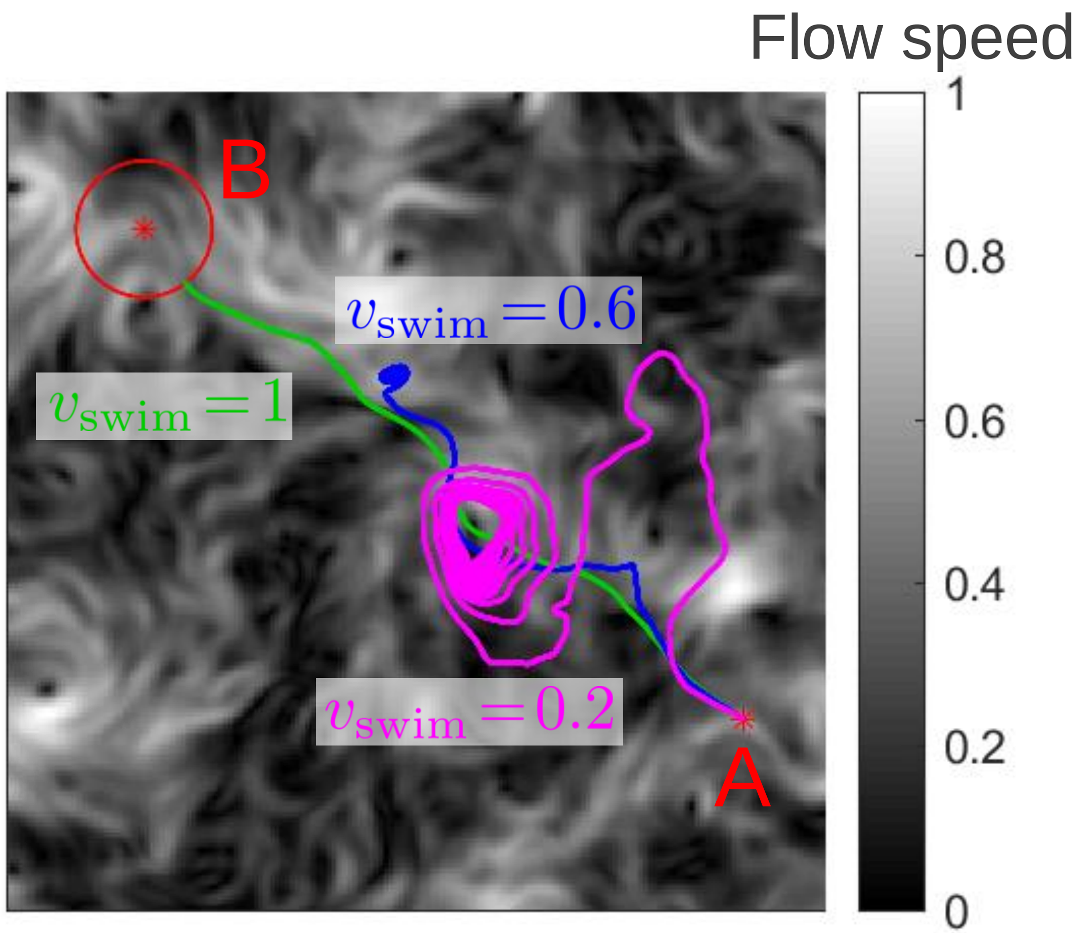


- How to choose swimming direction?

Zermelo, Z. Angew. Math. Mech. **11**, 114–124 (1931); Biferale et al, Chaos **29**, 103138 (2019)

Example: Zermelo's problem

- Naive approach: always swim towards the target

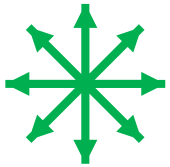


Zermelo, Z. Angew. Math. Mech. **11**, 114–124 (1931); Biferale et al, Chaos **29**, 103138 (2019)

Example: Zermelo's problem (Q-learning)

States
Flow grid

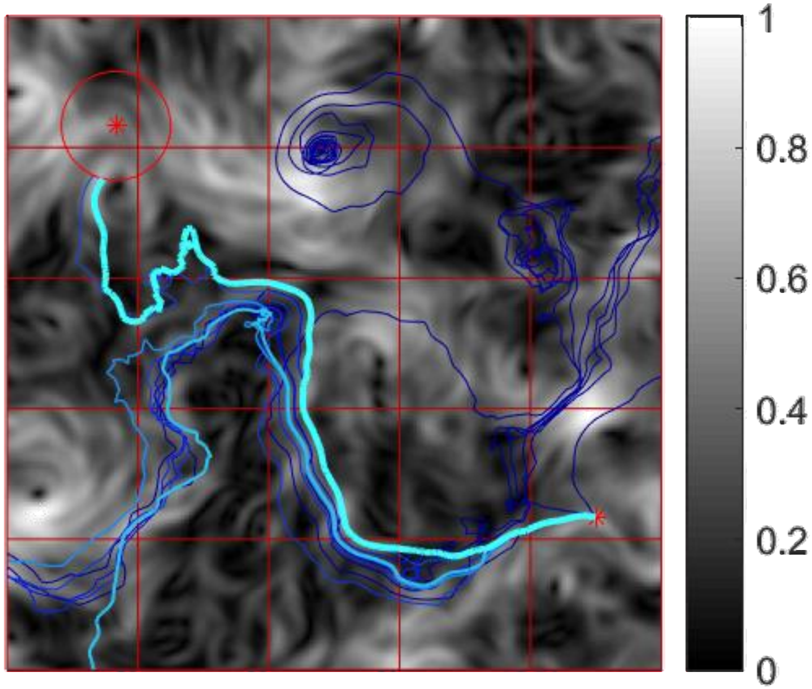
Actions
Swim direction



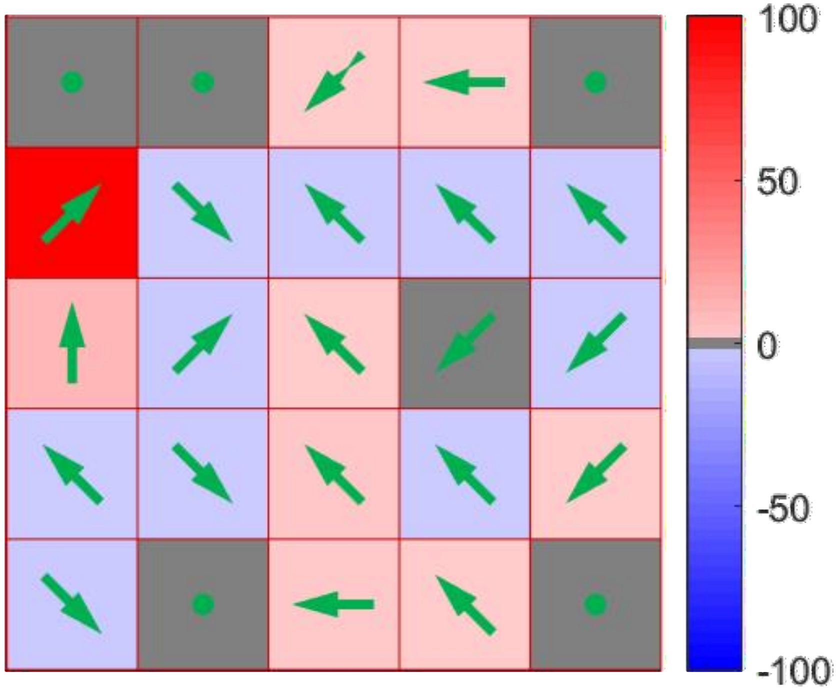
Rewards
- For state update/crossing bounds
+ For moving towards/reaching target

Learning rate α**:** $Q(s,a) = \alpha\,[r + \max_{a'} Q(s',a')] + (1-\alpha)Q(s,a)$

Flow speed



$\max_{a'} Q(s,a')$

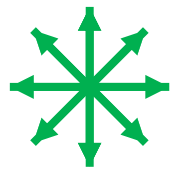


Zermelo, Z. Angew. Math. Mech. **11**, 114–124 (1931); Biferale et al, Chaos **29**, 103138 (2019)

Example: Zermelo's problem (Q-learning)

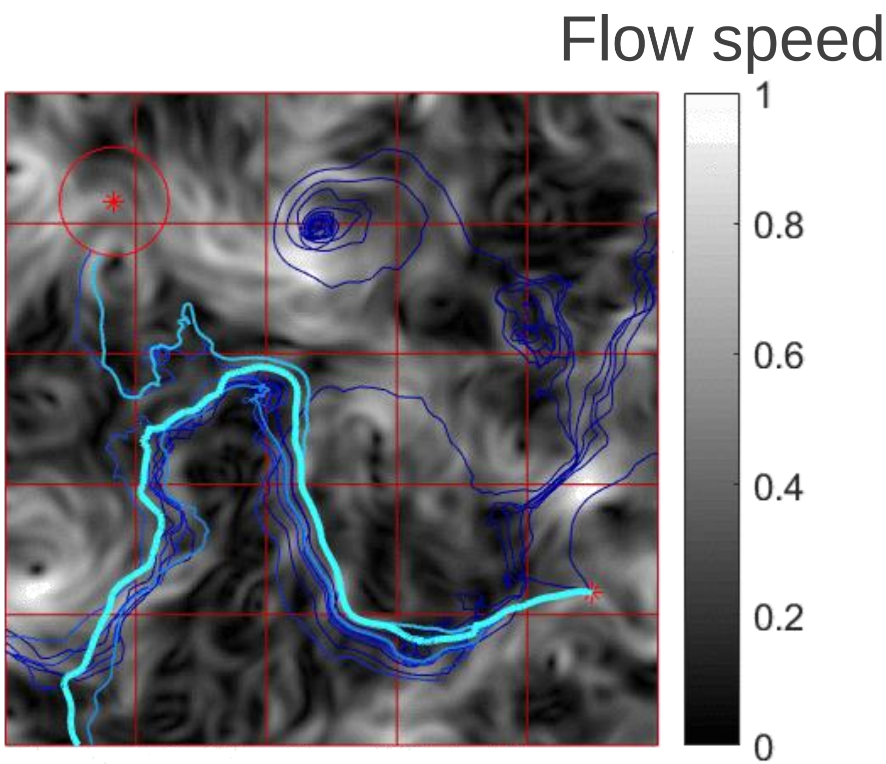
States
Flow grid

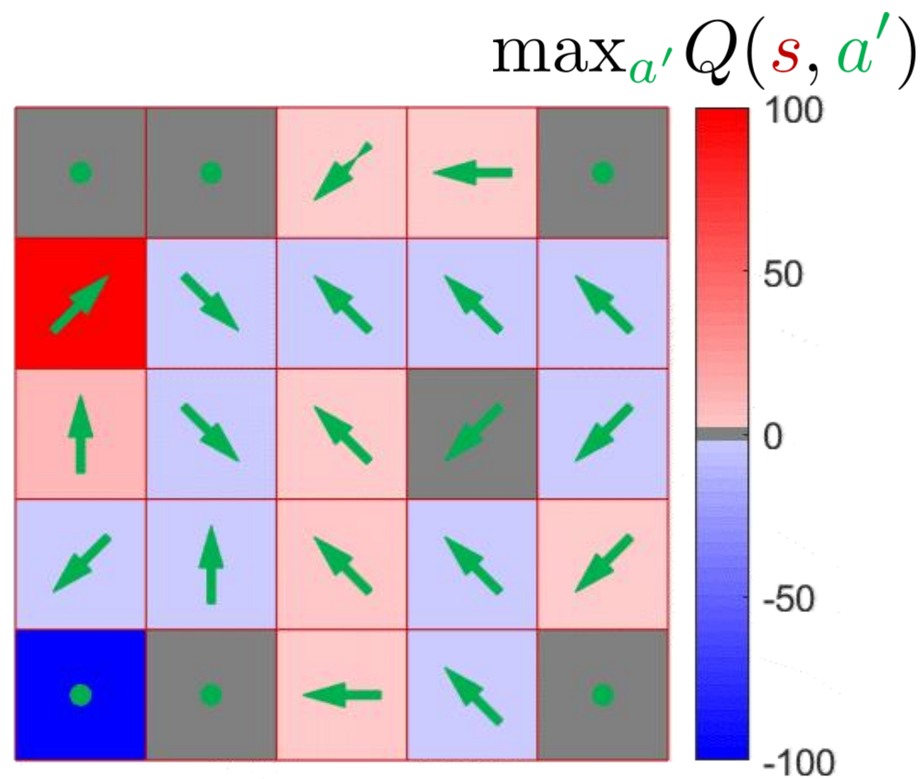
Actions
Swim direction



Rewards
- For state update/crossing bounds
+ For moving towards/reaching target

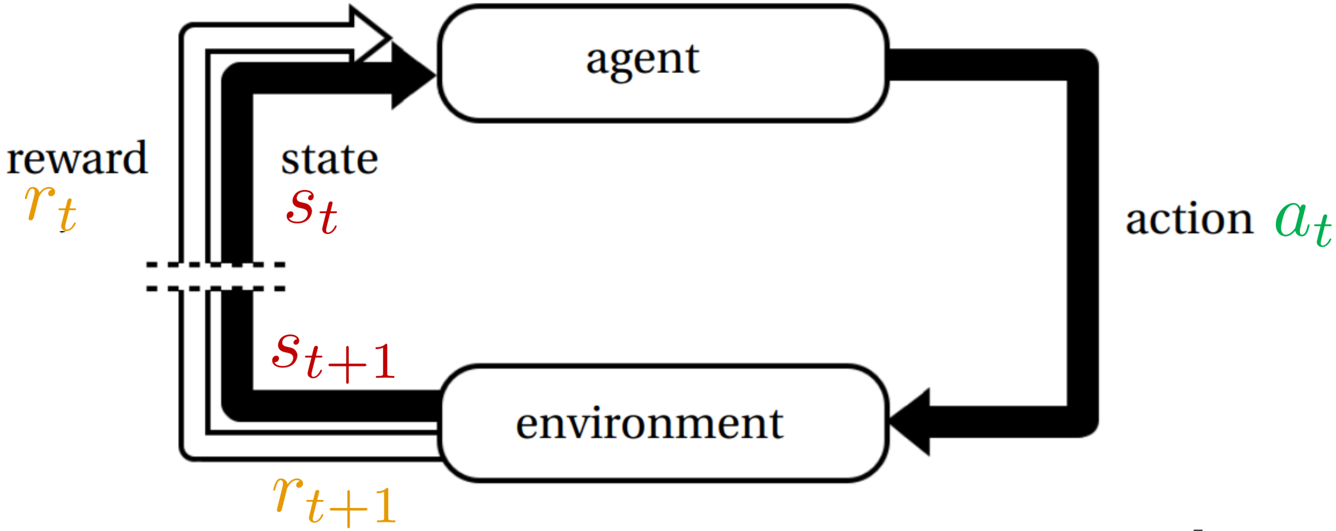
Learning rate α: $Q(s,a) = \alpha\,[r + \max_{a'} Q(s',a')] + (1-\alpha)Q(s,a)$





Summary of Q-learning

$0 \leq \varepsilon \leq 1$ **Exploration rate**
$0 < \alpha \leq 1$ **Learning rate**



ε-greedy policy

probability ε	probability $1-\varepsilon$
random a	$\text{argmax}_{a'} Q(s, a')$

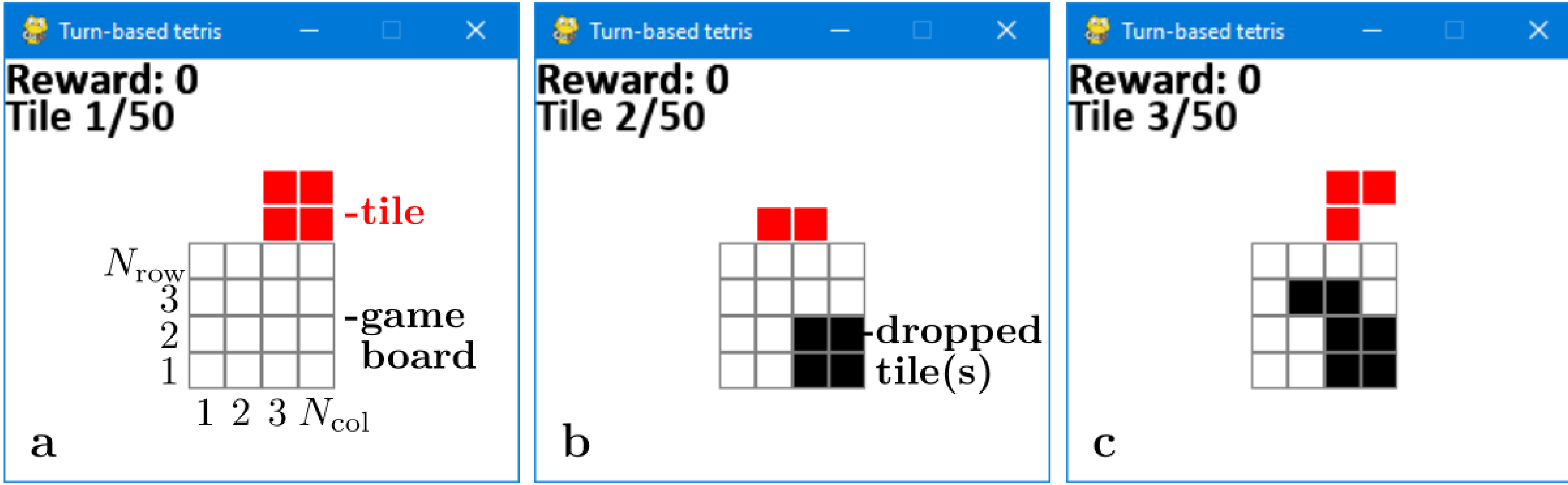
Use Q-table to store experience (expected future reward)

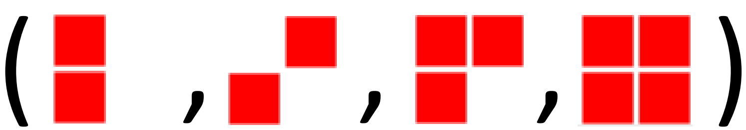
$$Q(s,a) = \alpha \left[r + \max_{a'} Q(s', a') \right] + (1-\alpha) Q(s,a)$$

Canvas/files/Homeworks/HW B Kristian/HomeworkB_RL.pdf

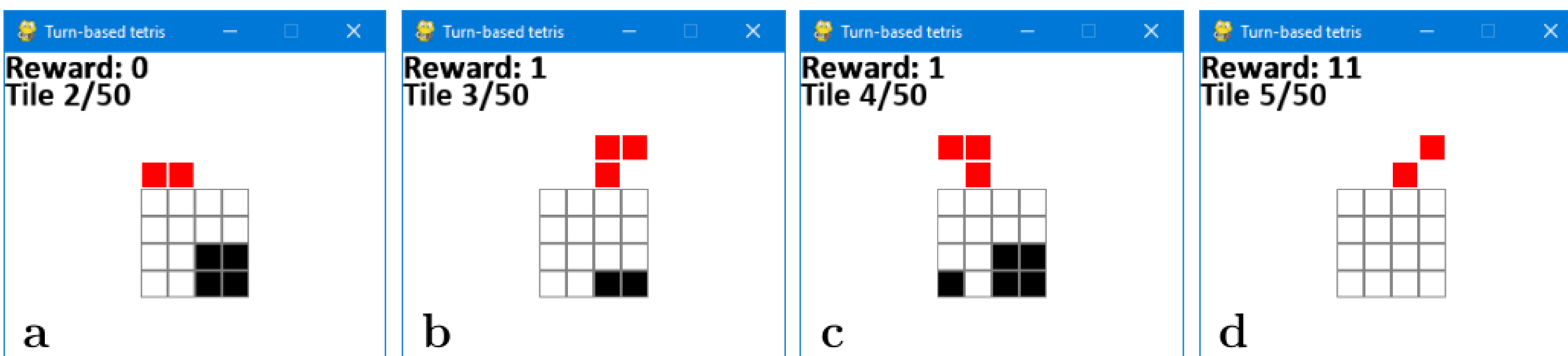
Home problem B: Introduction

Task To learn to play simplified Tetris



Tile set 

Reward is given for completing rows

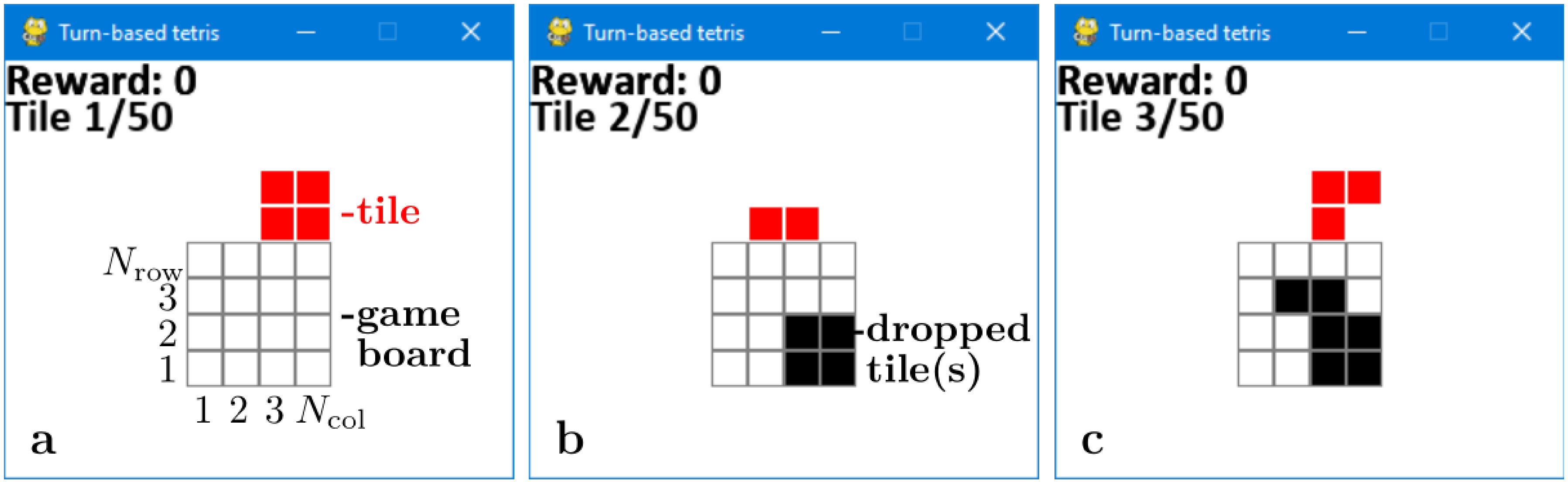


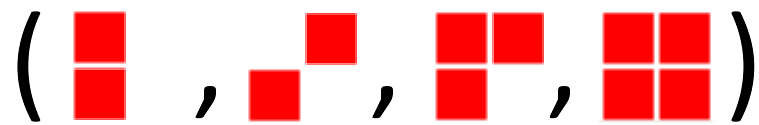
Canvas/files/Homeworks/HW B Kristian/HomeworkB_RL.pdf

Home problem B: Getting started

- **Download the following files**
  - Canvas/files/Homeworks/HW B Kristian/tetris.py
  - Canvas/files/Homeworks/HW B Kristian/gameboardClass.py
  - Canvas/files/Homeworks/HW B Kristian/agentClass.py
- **Make sure you have all required packages installed**
  - numpy
  - pygame
  - h5py
  - torch/tensorflow
- **Run tetris.py**
  - Use original parameters to test game
  - Set human_player=0 and param_set to one of PARAM_TASK1a, PARAM_TASK1b, PARAM_TASK1c, PARAM_TASK1d, PARAM_TASK2a (and optional PARAM_TASK2b) to address the corresponding task in HomeworkB_RL.pdf
- **Complete the code in agentClass.py to address the tasks in HomeworkB_RL.pdf**

Home problem B: Discussion



Tile set 

Think about/Discuss with your neighbors

What are possible choices of states and actions?

What is good/bad with different choices of states and actions?

Agent The player

Environment The game (and player)

States ???

Actions ???

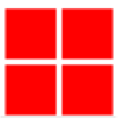
Reward + 10^{N-1} for completing N rows, -100 for game over

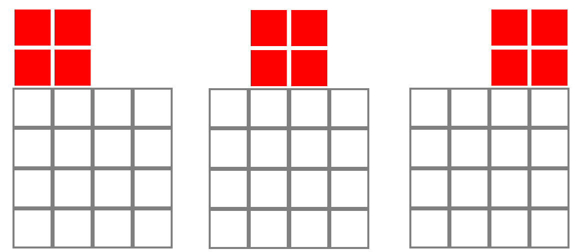
Canvas/files/Homeworks/HW B Kristian/HomeworkB_RL.pdf

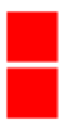
Home problem B: Actions

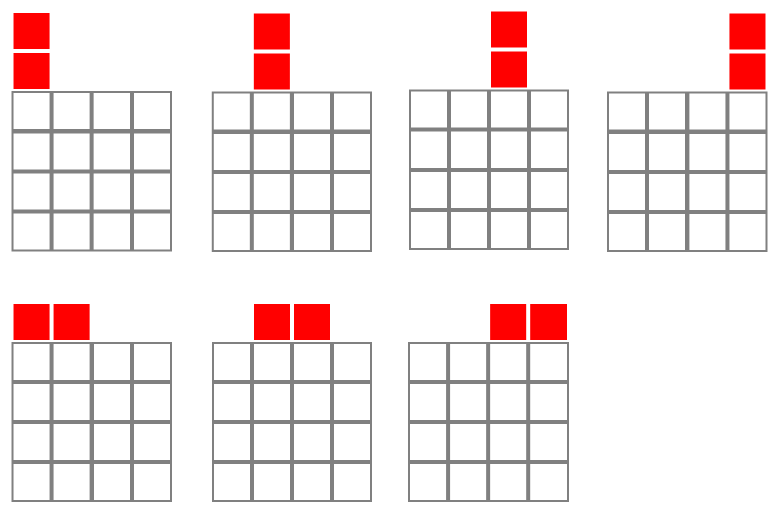
Use all allowed ways to place a tile as actions

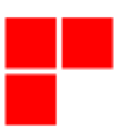
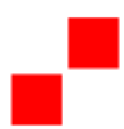
- Tile orientation
- Horizontal position of left tile edge

Example  has 3 actions:



Example  has 7 actions:



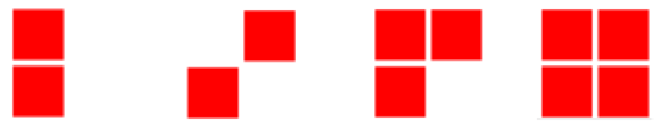
 has 12 actions and  has 6 actions

Canvas/files/Homeworks/HW B Kristian/HomeworkB_RL.pdf

Home problem B: States

States that should be used are a combination of the following

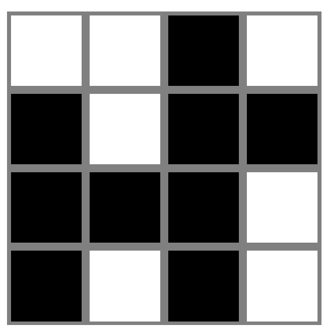
- Tile identifier of tile to be placed



Tile identifier 0 1 2 3

- Binary matrix representation of game board occupation

  Example



Game board

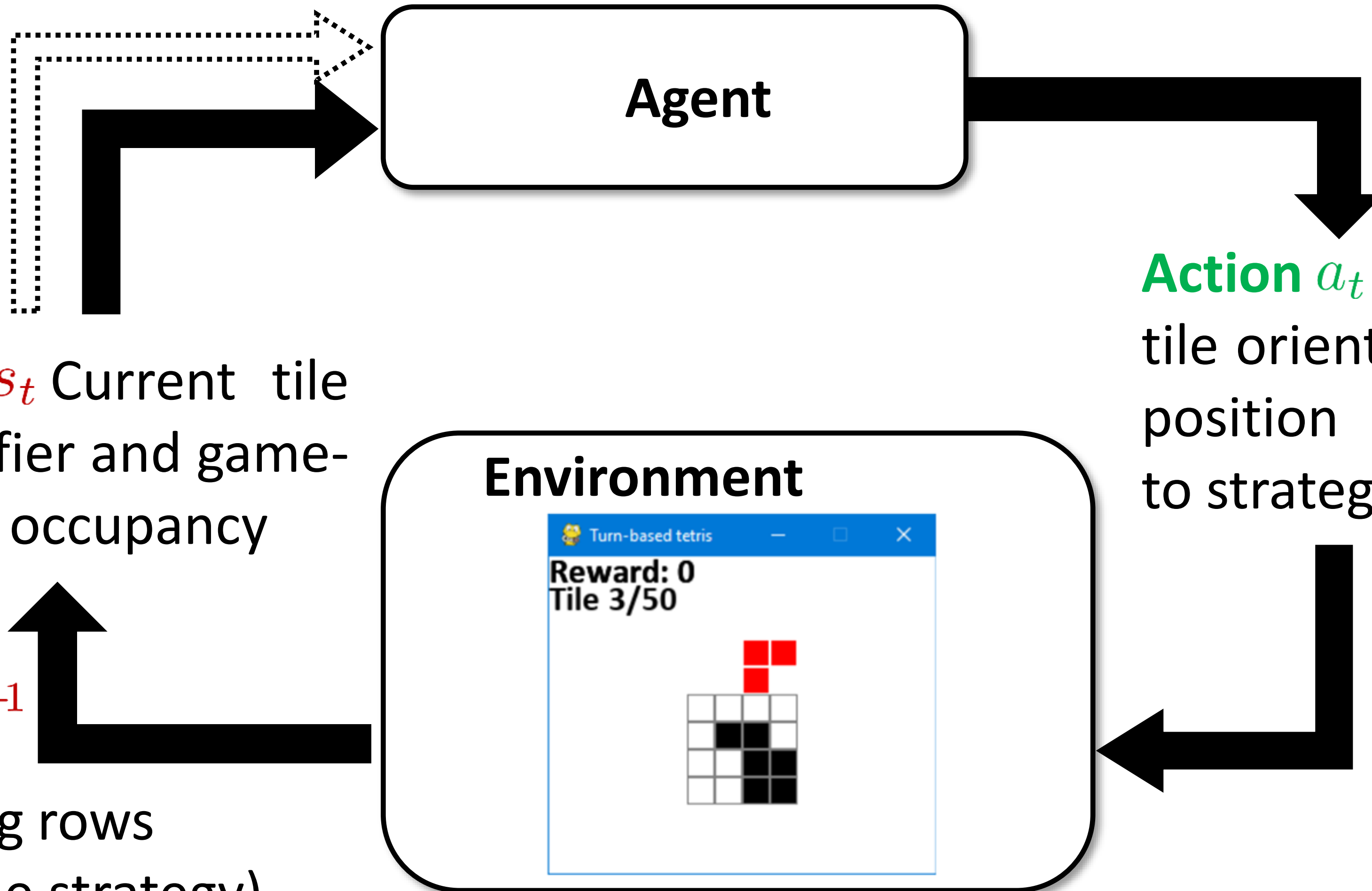
$$\begin{pmatrix} -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

Matrix representation

Sutton & Barto (2018), Mehlig (2021)

Home problem B: Reinforcement learning

Start here



Action a_t Choose tile orientation and position according to strategy (policy)

State s_t Current tile identifier and game-board occupancy

New state s_{t+1}
Reward r_{t+1}
By completing rows (used to refine strategy)

Repeat until game over or a predefined number of moves (episodic task)

$$s_0\,,\,a_0\,,\,r_1\,,\,s_1\,,\,a_1\,,\ldots,\,s_t\,,\,a_t\,,\,r_{t+1}\,,\,s_{t+1}\,,\,a_{t+1}\,,\ldots,\,r_T\,,\,s_T$$

Reinforcement learning algorithms aim to find strategy/policy (choice of action given state) that maximizes long term reward.

Sutton & Barto (2018), Mehlig (2021)

Optimal policy in Markovian systems

Policy Method of choosing a given s

Defined as probability $\pi(a|s)$

Examples

- Always choose the same action $a^{(1)}$: $\pi(a|s) = \delta_{a,a^{(1)}}$
- Choose action randomly: $\pi(a|s) = \frac{1}{N_a}$ with $a \in \{a^{(1)}, a^{(2)}, \ldots, a^{(N_a)}\}$
- Use the optimal policy: $\pi^*(a|s) = \delta_{a,a_*}$ where $a_* = \text{argmax}_{a'} Q^*(s, a')$ and $Q^*(s_t, a_t) = \langle r_{t+1} + r_{t+2} + \cdots + r_T \rangle$ is the expected future reward when following the optimal policy.

Goal Find $Q^*(s, a)$ => optimal policy $\pi^*(a|s)$

Brute force Find $Q^*(s, a)$ by evaluating $Q_\pi(s, a)$ of all policies π

$$\max_a Q^*(s, a) \geq \max_a Q_\pi(s, a)$$

Problem: ${N_a}^{N_s}$ different policies if N_s states and N_a actions

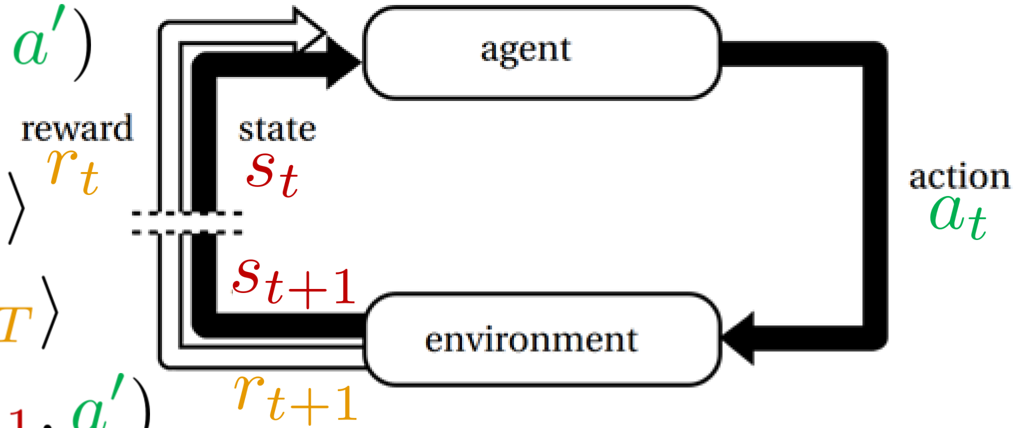
Sutton & Barto (2018), Mehlig (2021)

Optimal solution using Q-learning

Optimal policy $a_t = \mathrm{argmax}_{a'} Q^*(s_t, a')$

Expected future reward

$$Q^*(s_t, a_t) = \langle r_{t+1} + r_{t+2} + \ldots + r_T \rangle$$
$$= \langle r_{t+1} \rangle + \langle r_{t+2} + \ldots + r_T \rangle$$
$$= \langle r_{t+1} \rangle + \max_{a'} Q^*(s_{t+1}, a')$$



Q-learning algorithm (aims to find approximate $Q(s,a) \approx Q^*(s,a)$)

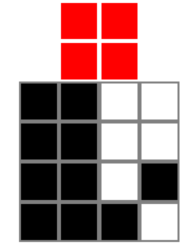
1. Start with arbitrary Q-table $Q_0(s,a)$ (size $N_s \times N_a$)
2. Evaluate initial state s_t (with $t = 0$)
3. Choose action a_t according to policy, e.g. ε-greedy
4. Apply a_t and evaluate s_{t+1} and r_{t+1}
5. Update $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \Delta Q$

$$\Delta Q = r_{t+1} + \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$

6. Repeat from 3. until episode is finished
7. Repeat from 2. (keeping Q) until convergence

$$\Delta Q = r_{t+1} + \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$

Q-learning Example I

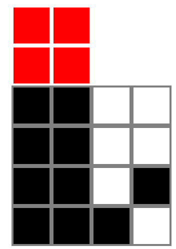


state $\hat{s}$

- First time state $\hat{s}$ is encountered

$$Q(\hat{s}, a) = (\boxed{0, 0, 0})$$

Choose random action on tie



Chosen action a_1

Game over

Reward $r = -100$

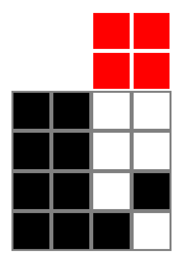
$$Q(\hat{s}, a_1) = -100\alpha = -20$$

(assume $\alpha = 0.2$)

- Second time $\hat{s}$ is encountered

$$Q(\hat{s}, a) = (-20, \boxed{0, 0})$$

Choose a_2 or a_3 randomly



Chosen action a_3

Two lines completed

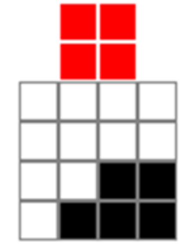
Reward $r = +10$

$$Q(\hat{s}, a_1) = 10\alpha = 2$$

- Third time $\hat{s}$ is encountered $Q(\hat{s}, a) = (-20, 0, \boxed{2})$ → action a_3

$$\Delta Q = r_{t+1} + \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)$$

Q-learning Example 2

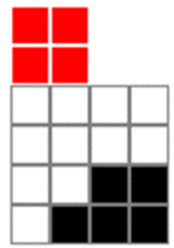


state $\hat{\hat{s}}$

- First time state $\hat{\hat{s}}$ is encountered

$$Q(\hat{\hat{s}}, a) = (\boxed{0, 0, 0})$$

Choose random action on tie



Chosen action a_1

One line is completed

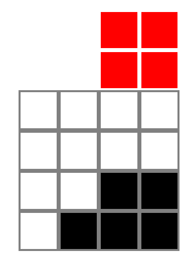
Reward $r = 1$

$$Q(\hat{\hat{s}}, a_1) = 1\alpha = 0.2$$

- Second time $\hat{\hat{s}}$ is encountered

$$Q(\hat{\hat{s}}, a) = (\boxed{0.2}, 0, 0) \longrightarrow \text{action } a_1$$

Problem action a_3 is better in the long run



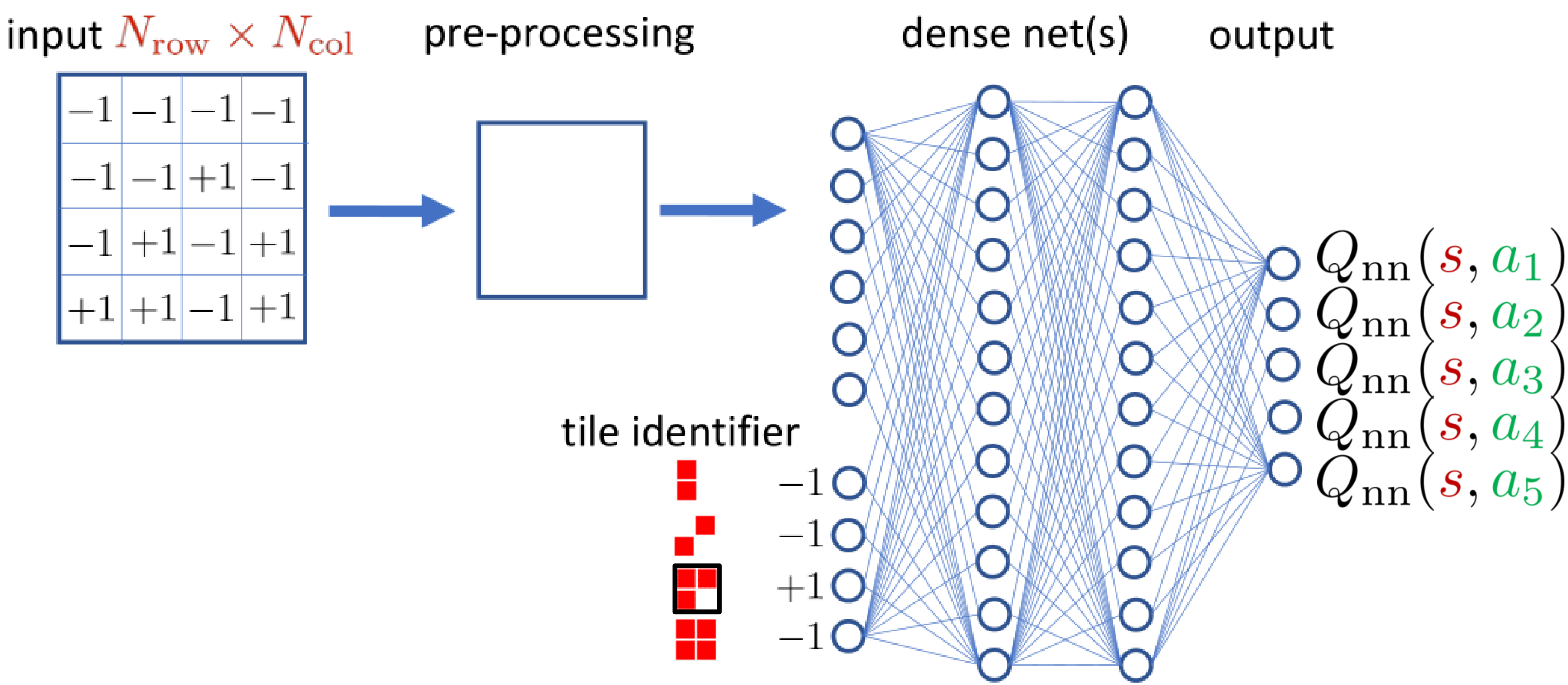
action a_3

Solution Add exploration, e.g. ε-greedy policy

Sutton & Barto (2018), Lapan (2020)

Optimal solution using Deep Q-networks

Idea Replace Q-table by deep neural network



Policy $a_t = \begin{cases} \text{random valid action} & \text{with prob. } \varepsilon_E \\ \operatorname{argmax}_{a'} Q_{\mathrm{nn}}(s_t, a') & \text{otherwise} \end{cases}$

$\varepsilon_E = \max(\varepsilon, 1 - E/E_0)$, episode E

Sutton & Barto (2018), Lapan (2020)

Optimal solution using Deep Q-networks

Minimize $|\Delta Q(T_t)| = |r_{t+1} + \max_{a'} Q_{\mathrm{nn}}(s_{t+1}, a') - Q_{\mathrm{nn}}(s_t, a_t)|$

for transitions $T_t = \{s_t, a_t, r_{t+1}, s_{t+1}\}$

Loss function $\sum_{\tau} \Delta Q(T_\tau)^2$

Problem 1

Data is not independent and identically distributed (needed for optimizing the loss function)

Solution

Use ‘experience replay buffer’ containing B last state transitions

T_t

T_{t-1}

$\vdots$

T_{t-B+1}

Choose b transitions randomly →

T_{i_1}

T_{i_2}

$\vdots$

T_{i_b}

Optimize the loss function for the mini batch data →

$Q_{\mathrm{nn},t+1}(s, a)$

Experience buffer | *Mini batch* | *Updated network*

Sutton & Barto (2018), Lapan (2020)

Optimal solution using Deep Q-networks

Minimize $|\Delta Q(T_t)| = |r_{t+1} + \max_{a'} Q_{\mathrm{nn}}(s_{t+1}, a') - Q_{\mathrm{nn}}(s_t, a_t)|$

for transitions $T_t = \{s_t, a_t, r_{t+1}, s_{t+1}\}$

Loss function $\sum_{\tau} \Delta Q(T_\tau)^2$

Problem 2

Potential instability due to 'bootstrapping': we use Q_{nn} to estimate the expected future reward when updating Q_{nn}.

Solution

Introduce 'target network' $\hat{Q}_{\mathrm{nn}}(s, a)$ when estimating future reward

$$\Delta Q(T_t) = r_{t+1} + \max_{a'} \hat{Q}_{\mathrm{nn}}(s_{t+1}, a') - Q_{\mathrm{nn}}(s_t, a_t)$$

$\hat{Q}_{\mathrm{nn}}(s, a)$ is a copy of Q_{nn} , but only occasionally updated.

Sutton & Barto (2018), Lapan (2020)

Optimal solution using Deep Q-networks

Deep Q-network algorithm

1. Start with arbitrary and identical Q-networks $Q_{\text{nn}}(s, a)$ and $\hat{Q}_{\text{nn}}(s, a)$
2. Evaluate initial state
3. Choose action based on $Q_{\text{nn}}(s, a)$ and ε_E
4. Apply a_t and evaluate s_{t+1} and r_{t+1}
5. Store transition $T_t = \{s_t, a_t, r_{t+1}, s_{t+1}\}$ in replay buffer
6. Sample mini-batch of transitions from the replay buffer
7. For each transition in the mini-batch, use $\hat{Q}_{\text{nn}}(s, a)$ to calculate target value: $y = r$ if s_{t+1} is terminal,
   $y = r + \max_{a'} \hat{Q}_{\text{nn}}(s_{t+1}, a')$ otherwise.
8. Calculate loss $\mathcal{L} = (y - Q_{\text{nn}}(s, a))^2$
9. Update $Q_{\text{nn}}(s, a)$ using for example 'Adam' (built-in) to minimize loss
10. Every 100 episode copy weights from $Q_{\text{nn}}(s, a)$ to $\hat{Q}_{\text{nn}}(s, a)$
11. Repeat from 3. until convergence

Assignments

Complete the classes for Q-learning and Deep Q-networks in agentClass.py

Address the following tasks

- 1.a) [3p] Use Q-learning to train an artificial player on deterministic tile sequence. Use greedy policy.
- 1.b) [1p] Same as 1.a) with ε- greedy policy.
- 1.c) [1p] Same as 1.b) with random tile sequence.
- 1.d) [1p] Discuss possibility to scale up method to a larger game board

- 2.a) [4p] Solve task 1.c) using deep Q-networks.
- 2.b) [Optional task, 2p] Solve the problem on larger game board using deep Q-networks