Multi-agent Reinforcement Learning

Richard Hájek, Ladislav Maleček,
Matouš Melecký, Jaroslava Schovancová

#TeamLearn

Seminar on Artificial Intelligence 2
2021-04-13

NeurIPS 2020 Warm-Up: Completed
NeurIPS 2020 Round 1: Completed
NeurIPS 2020 Round 2: Completed
AMLD 2021 Warm-Up: Completed
#neurips
#reinforcement_learning

FLATLAND
AMLD 2021 Competition
Multi-Agent Reinforcement Learning on Trains

AMLD icaps

By SNCF & SBB & DB Deutsche Bahn

How to efficiently manage dense traffic on complex railway networks?

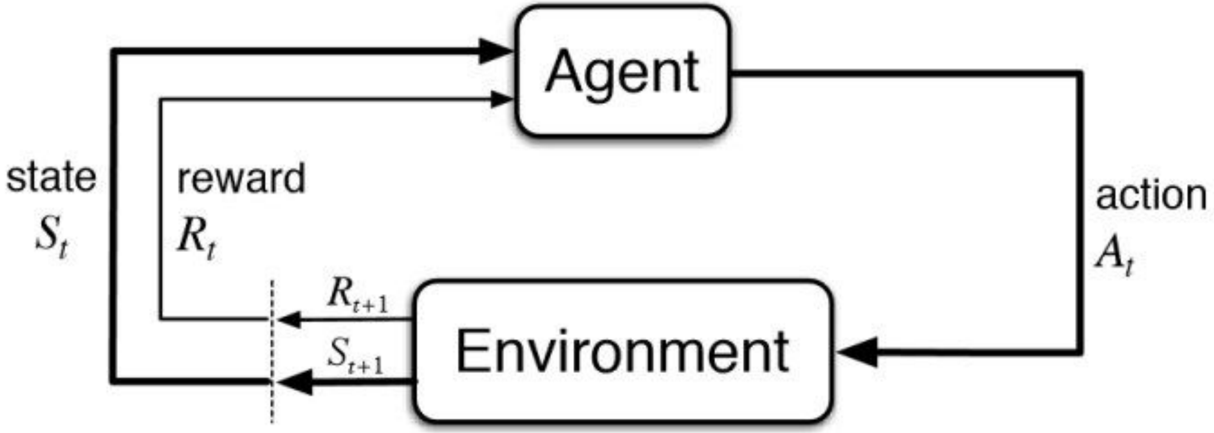
Outline

- Single-agent RL
- Multi-agent RL
- Past solutions of the Flatland challenge
- Code components & experiments
- Considerations for the next steps

Single-agent Reinforcement Learning

Agent perceives the **environment** and its **state** through **observations** and receives from it **reward** signals upon which it can **act**.

Time is usually discretized.



https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

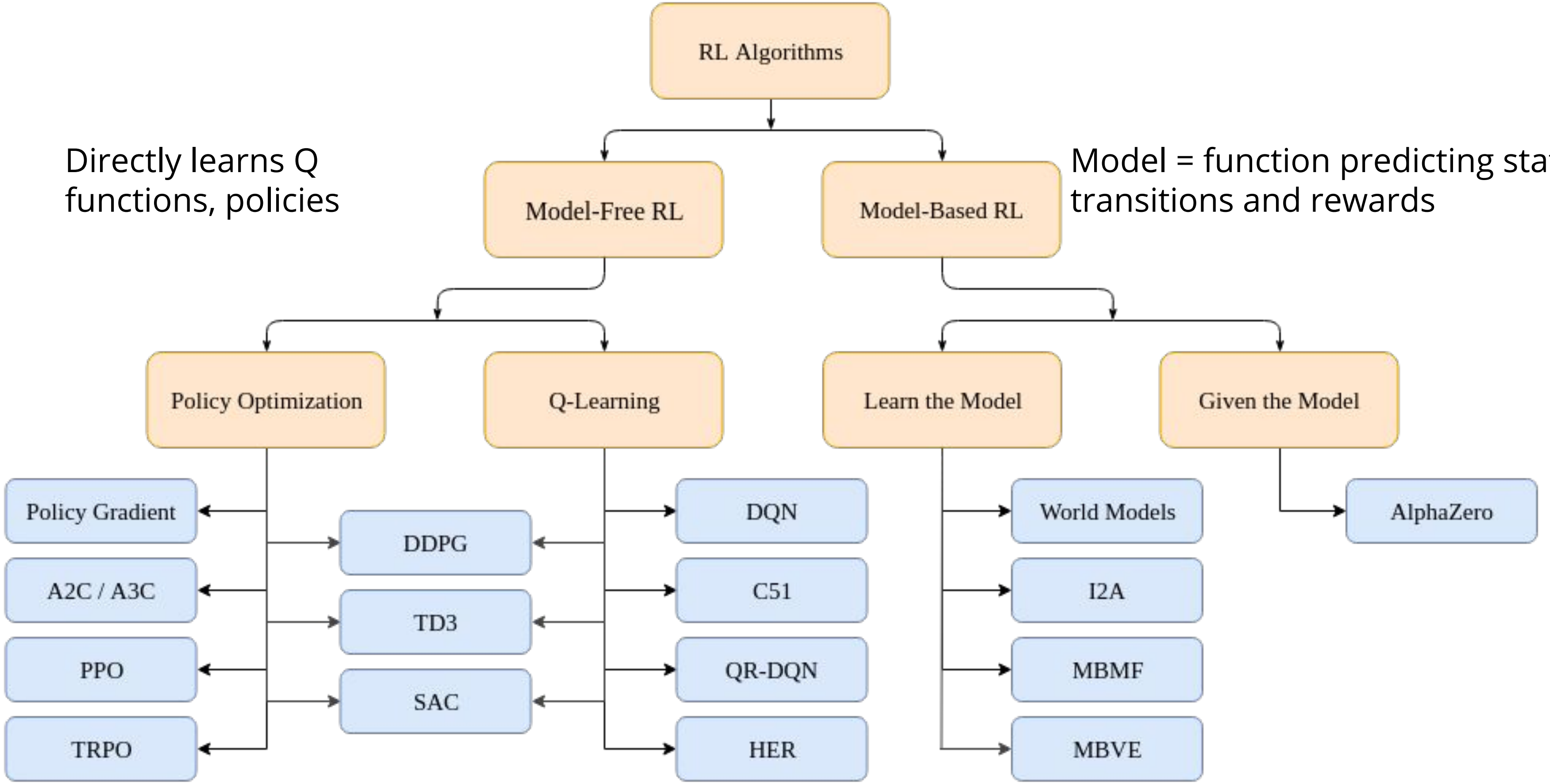
Basic Terminology

- Policy (π) - maps states to action
  - Stochastic X deterministic
- Reward function - reward given current state, action and resulting state
- Return - sum of rewards obtained during time
  - Discounted rewards
- On-policy value function (V) - given state and policy, computes the long-term return (expected return) of executing the policy from the state
- On-policy action-value function (Q) - similar to V, but additional parameter is the first action taken
- Optimal V, Q - maximum - following optimal policy
- Advantage function - A = Q - V

Deep Reinforcement Learning

- Deep neural networks are used as function approximators
- What can we approximate:
    - Policy
    - Q function
    - Advantage function

Directly learns Q functions, policies



Model = function predicting state transitions and rewards

https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

Q-Learning

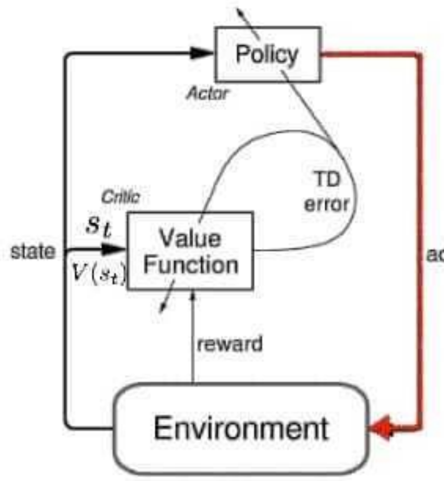
- Learn an approximator (NN) for the optimal action-value function (Q)
- Learning is based on the Bellman equation
  - Q function should match the Bellman equation computed from experience
- Uses experience replay - data can be reused
  - Training in batches - sampling past experience
  - Off-policy - the policy during exploration was different than the current one
- Training using epsilon-greedy policy
  - At first we explore more, but as the model gets better, we act upon its recommendations more often

Policy Optimization

- Directly approximates the policy function - without approximating the action-value function
- NN (policy) parameters are searched for using gradient ascent on the advantage function
- Performed **on-policy** - each update only uses data acquired while acting to the most recent policy
- Can estimate both deterministic and stochastic policies

Actor-Critic Approaches

- Combine policy optimization with Q-Learning
- **Actor** - learns the policy, controls how agent behaves
- **Critic** - evaluates the action by computing the value function
- Training of critic and actor is performed separately



- Actor: decides which action to take
- Critic: tells the actor how good its action was and how it should adjust

(Figure from Sutton & Barto, 1998)

Multi-agent Reinforcement Learning

Multiple agents interacting in the same environment

Can be divided based on the objective of the agents:

- **Fully cooperative**
- Fully competitive
- Mixed

Action spaces become joined action states - as of one dimension per agent.

We are unable to use single agent RL in simple observable setting. Because that assumes that transition probabilities remain constant.

Decentralization vs centralization

What if we simply use single policy to manage all agents?

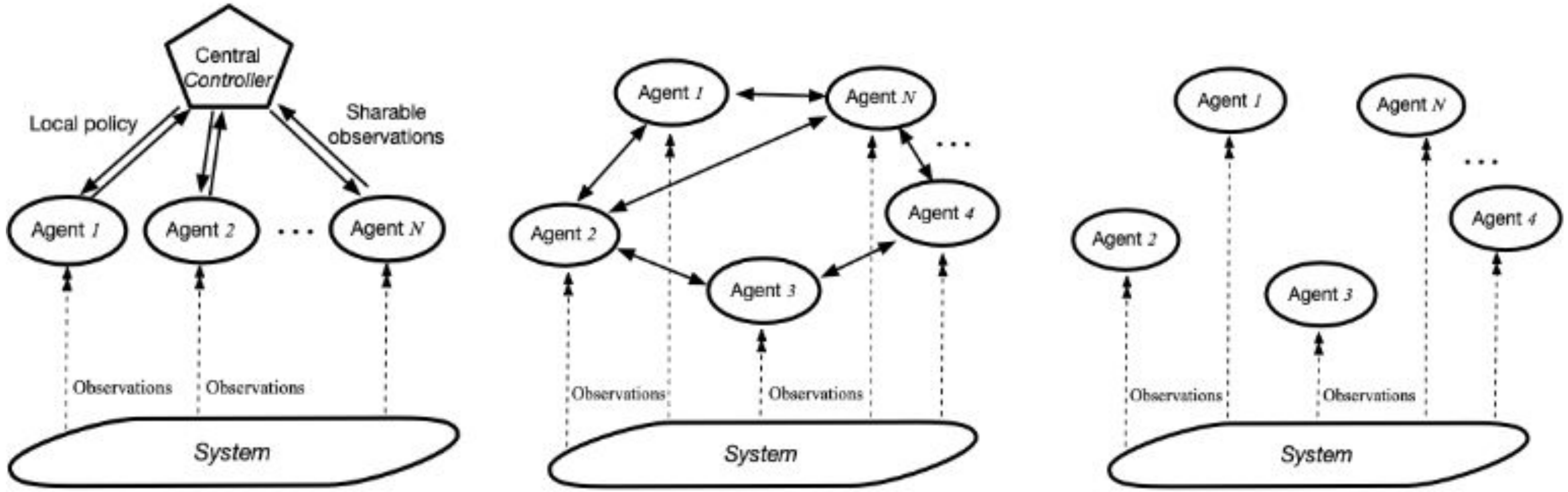
- Exponential growth of action space with respect to the #agents
- Curse of dimensionality when exploring

What if we use single agent alg. for each agent?

- Partial observability
- Becomes non-stationary

Centralization fights **scalability**, **decentralization** fights **non-stationarity**

Division based on centralization



(a) Centralized setting (b) Decentralized setting with networked agents (c) Fully decentralized setting

Past years approaches

- Based on the experiences of 2 best RL teams of the 2020 challenge

🥇 **JBR_HSE** - 214.15 points

🥈 **ai-team-flatland** - 181.497 points

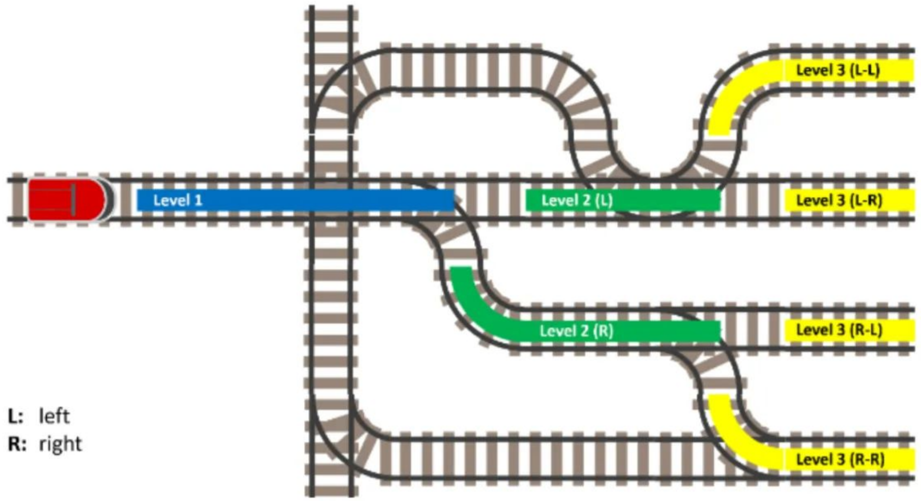
- OP solutions performed better - scored over 290 points
- That does not mean we cannot do better, or combine the two approaches
- Focus is put on the representation of the problem, that is subsequently fed into the neural network
- Usually, some local observations are needed, because maps can get big

JBR_HSE

- Reward function: 0.01 * Δd - 5*is_deadlocked + 10*succeeded
  - Δd – change of distance to goal since the decision has been made
- RL algorithm: PPO (Proximal Policy Optimization)
  - Actor - critic approach
- **Domain-specific simplifications** of the task greatly improve the results
  - Agent cannot make decision when not at an intersection
  - Agent’s episode ends after it reached the goal or got itself into a deadlock
    - Default: continuation

JBR_HSE: Agent's Observations

- Local tree
    - Fixed depth
    - Each edge and the vertex to which it leads has a vector of features calculated for a given section of the path
        - Length of the path
        - The distance from the end of the edge to the destination
- Simple observations
    - Minimum distance to destination
    - Whether the agent is at the intersection
    - Deadlock



JBR_HSE: Communication Mechanism

- Agents generate messages and send them to all neighbouring trains
- Received messages are averaged into a single message
- Self-attention mechanism determines the weights with which the messages will be averaged
- *“With such a structure of the communication process, there is no need to somehow modify the learning process, because it is sufficient to simply allow the gradients to pass through the messages during the backpropagation of the error.”*

JBR_HSE: Training

- Happens on a relatively small map
- Due to the locality of the observation, an assumption is made that the learned strategies will generalize well to bigger maps

JBR_HSE: Deadlock Prevention by Scheduling

- Most deadlocks occurred because trains started at the same time
- Only some agents are picked to start their journey, when some finishes, next agent is allowed
- At training time: pick agents closest to their goal - performed well for small maps
- Application time: classifier is trained to determine probability of reaching the goal
    - Agents wait until the probability becomes sufficiently large

ai-team-flatland: round 1

- RL algorithm: **DQN** - provided by the organizers
- Used a similar tree observation (*I think? I am not sure now..*)
- More information about future deadlocks, and penalization for getting into one
- Penalty for stopping and invalid actions
- Valid actions included into the observation
- Information about **tracks behind** an agent - to avoid agent blocking other agents behind it
- **Voting system**
  - Several models trained, that worked in different states and environments
  - Next action was the most frequent action proposed by models
  - Basically an ensemble system

ai-team-flatland: round 2

- Round 2:
  - Sparser, bigger network with more trains
  - Different speeds and malfunctions introduced
- Problems identified:
  - Easy to get into deadlock - rare positive reward
  - Deadlocks more costly (train blocked the only route)
  - Local observations limiting - agent got information about other agents too late

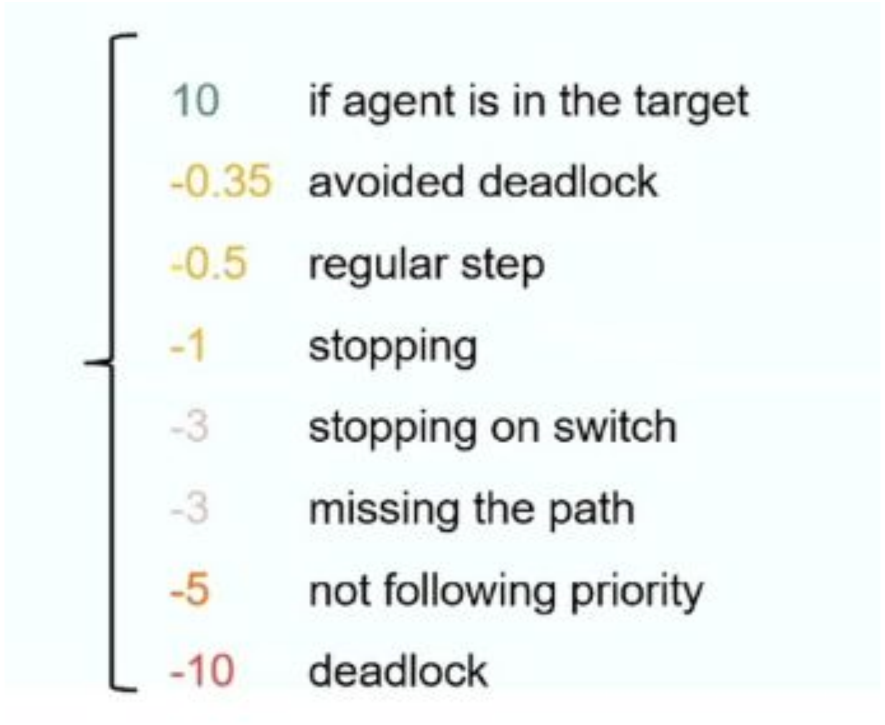
ai-team-flatland: Priority

- Solves the conflict between agents
- Conflicts for active agents:
  - Want to occupy the same cell
  - Want to surpass each other from different directions (swap conflict)
- Undirected conflict graph is built, edges represent conflicts
- Assigning priority (graph coloring problem)
  - Two agents in a conflict cannot have the same priority
- Heuristic
  - Give higher priority to agents with the highest degree (number of conflicts)
  - It is checked that blocked agents cannot have the highest priority
- For ready to depart agents:
  - Choose the fastest agents that have a free path

ai-team-flatland: Global Observation & Training

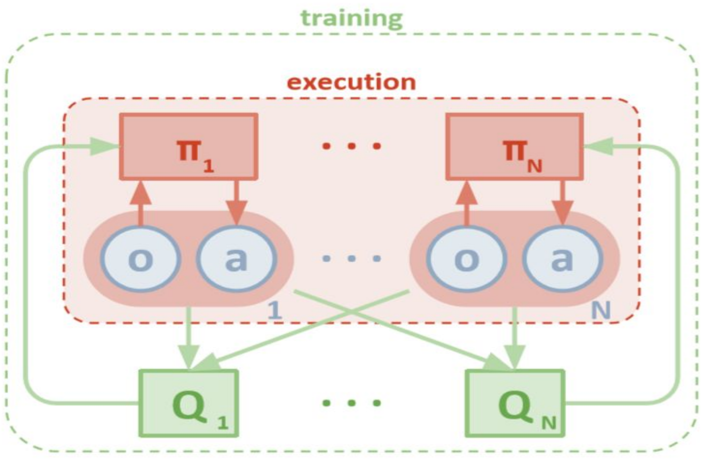
- Global observation
  - Information about whole paths to goal
  - Considers 2 shortest paths from the current switch
- Training
  - Gradually more complex
  - As agents became better on easier levels, the training progressed to next level
  - Example:
    - First level - learn to get to the goal without interruptions
    - Later levels would aim at avoiding deadlocks, when to take a slightly longer route etc.

ai-team-flatland: Reward Function



10	if agent is in the target
-0.35	avoided deadlock
-0.5	regular step
-1	stopping
-3	stopping on switch
-3	missing the path
-5	not following priority
-10	deadlock

Proposal for our use case



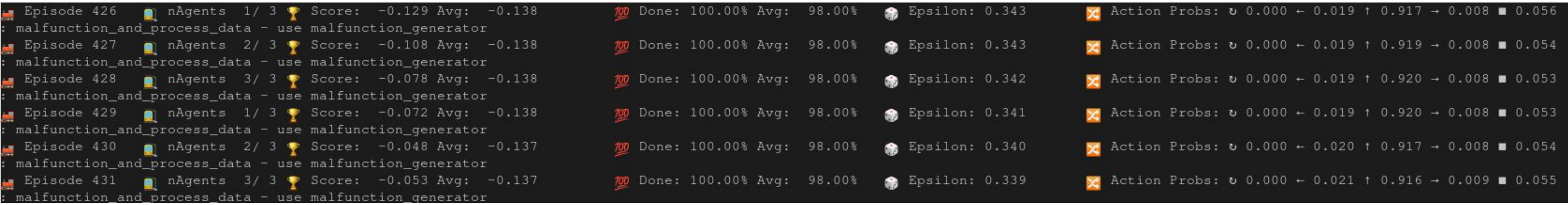
- **MADDPG**[7] with domain specific **propagation of data to the critic** for the RL part
- Simple and widely used
- As shown by the previous successful teams, preprocessing of the input is key. We could even use the **solvers from non-RL algorithms** to **feed the critic**. That is only done at the training phase.
- Possibility to use in between agents communication, MADDPG does not require the communication to be derivable. Steeply increases implementation difficulty.
- Usage of homogeneous agents
- Priority resolving as a parameter to for the agent
- Possible ensamblement of multiple agents (voting), and different agents for different types of trains (fast and slow)

Successful Observers

- 2020 1. Place - Team JBR_HSE - Hybrid Tree Observation combined with full observation
- 2020 - 2. Place - ai-team-flatland - Processed full observation

Experiments & first implementation steps

- Running starter kit went without a hitch
- It is unnecessary to check for action every step, only when an action is required
- Observations are extremely important



```
🚂 Episode 426    🚉 nAgents  1/ 3 🏆 Score:  -0.129 Avg:  -0.138          💯 Done: 100.00% Avg:  98.00%   🎲 Epsilon: 0.343        🔀 Action Probs: ↻ 0.000 ← 0.019 ↑ 0.917 → 0.008 ◼ 0.056
: malfunction_and_process_data - use malfunction_generator
🚂 Episode 427    🚉 nAgents  2/ 3 🏆 Score:  -0.108 Avg:  -0.138          💯 Done: 100.00% Avg:  98.00%   🎲 Epsilon: 0.343        🔀 Action Probs: ↻ 0.000 ← 0.019 ↑ 0.919 → 0.008 ◼ 0.054
: malfunction_and_process_data - use malfunction_generator
🚂 Episode 428    🚉 nAgents  3/ 3 🏆 Score:  -0.078 Avg:  -0.138          💯 Done: 100.00% Avg:  98.00%   🎲 Epsilon: 0.342        🔀 Action Probs: ↻ 0.000 ← 0.019 ↑ 0.920 → 0.008 ◼ 0.053
: malfunction_and_process_data - use malfunction_generator
🚂 Episode 429    🚉 nAgents  1/ 3 🏆 Score:  -0.072 Avg:  -0.138          💯 Done: 100.00% Avg:  98.00%   🎲 Epsilon: 0.341        🔀 Action Probs: ↻ 0.000 ← 0.019 ↑ 0.920 → 0.008 ◼ 0.053
: malfunction_and_process_data - use malfunction_generator
🚂 Episode 430    🚉 nAgents  2/ 3 🏆 Score:  -0.048 Avg:  -0.137          💯 Done: 100.00% Avg:  98.00%   🎲 Epsilon: 0.340        🔀 Action Probs: ↻ 0.000 ← 0.020 ↑ 0.917 → 0.008 ◼ 0.054
: malfunction_and_process_data - use malfunction_generator
🚂 Episode 431    🚉 nAgents  3/ 3 🏆 Score:  -0.053 Avg:  -0.137          💯 Done: 100.00% Avg:  98.00%   🎲 Epsilon: 0.339        🔀 Action Probs: ↻ 0.000 ← 0.021 ↑ 0.916 → 0.009 ◼ 0.055
: malfunction_and_process_data - use malfunction_generator
```

Summary & Next steps

- ✓ Performed a survey of SARL and MARL approaches
- ✓ Explored available SW packages & chose the tools to proceed with the study
- ✓ Identified the direction of the study

- Study: Multi-agent RL, Actor-critic, cooperative environment
  - Start small, iterative approach
  - Input pre-processing & critic feeding
  - Homogeneous agents, parametrized
  - Compare performance of different approaches, with different HW

References

1. https://spinningup.openai.com/en/latest/spinningup/rl_intro.html
2. https://torres.ai/deep-reinforcement-learning-explained-series/
3. https://theaisummer.com/Actor_critics/
4. https://ufal.mff.cuni.cz/courses/npfl122/2021-winter
5. https://medium.com/swlh/the-gist-multi-agent-reinforcement-learning-767b367b395f
6. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms
7. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments