Value Decomposition, Population-based MARL, Emergent behaviours

Andrea Fanti

Sapienza Università di Roma

Value decomposition

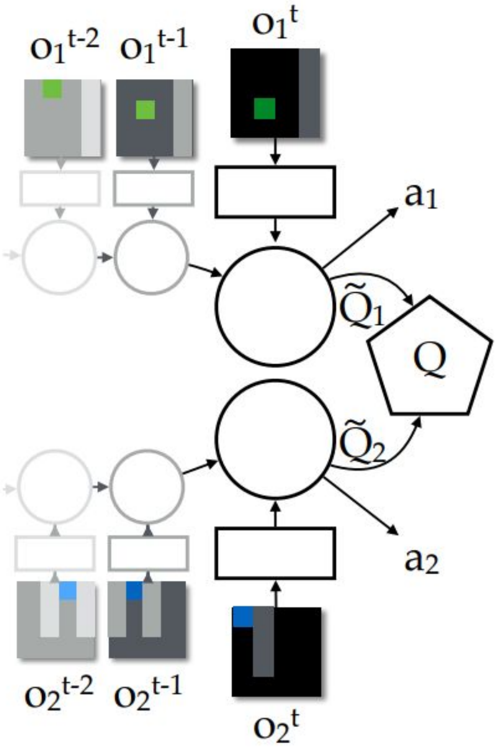
SAPIENZA
UNIVERSITÀ DI ROMA

Value Decomposition Networks

Sunehag, Peter, et al. "Value-decomposition networks for cooperative multi-agent learning." *arXiv preprint arXiv:1706.05296* (2017). - https://github.com/TonghanWang/NDQ

- Decomposes a team value function into an additive decomposition of the individual value functions
- Learns a joint action-value function $Q_{tot}(\boldsymbol{\tau},\boldsymbol{a})$ represented by the sum of individual value functions $Q_i(\tau_i,a_i;\theta_i)$

$$Q_{tot}(\boldsymbol{\tau},\boldsymbol{a}) = \sum_{i=1}^{N} Q_i(\tau^i, a^i; \theta^i)$$



QMIX

Rashid et al. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. https://arxiv.org/abs/1803.11485 (2018)

When decomposing Q_{tot}, we only actually care that:

$$\underset{\mathbf{u}}{\operatorname{argmax}}\, Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \operatorname{argmax}_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \operatorname{argmax}_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}$$

This also ensures that argmax Q_{tot} is trivial to compute from all the Q_i

QMIX

The previous constraint is satisfied if Q_{tot} is **monotonic** w.r.t. to each Q_i

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \ \forall a.$$

Can be easily satisfied with e.g. neural networks with positive weights

QMIX

Note: Q_i are NOT really “value functions” anymore

Why?

QMIX

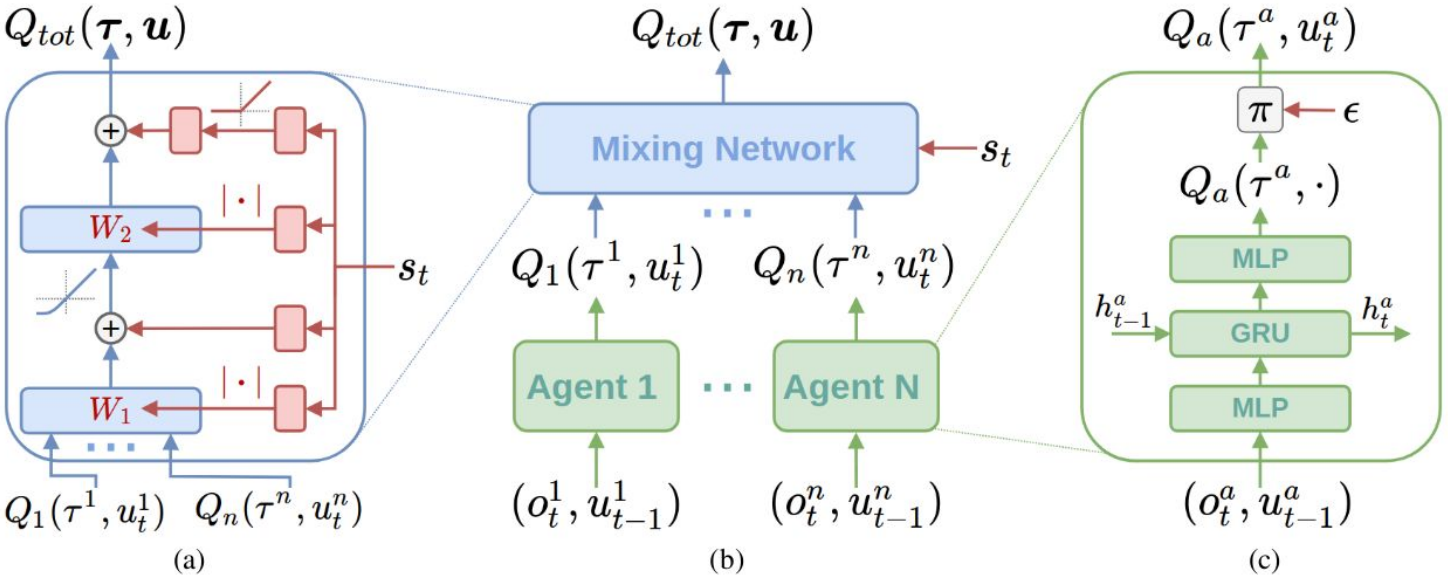
Note: Q_i are NOT truly “value functions” anymore

Why?

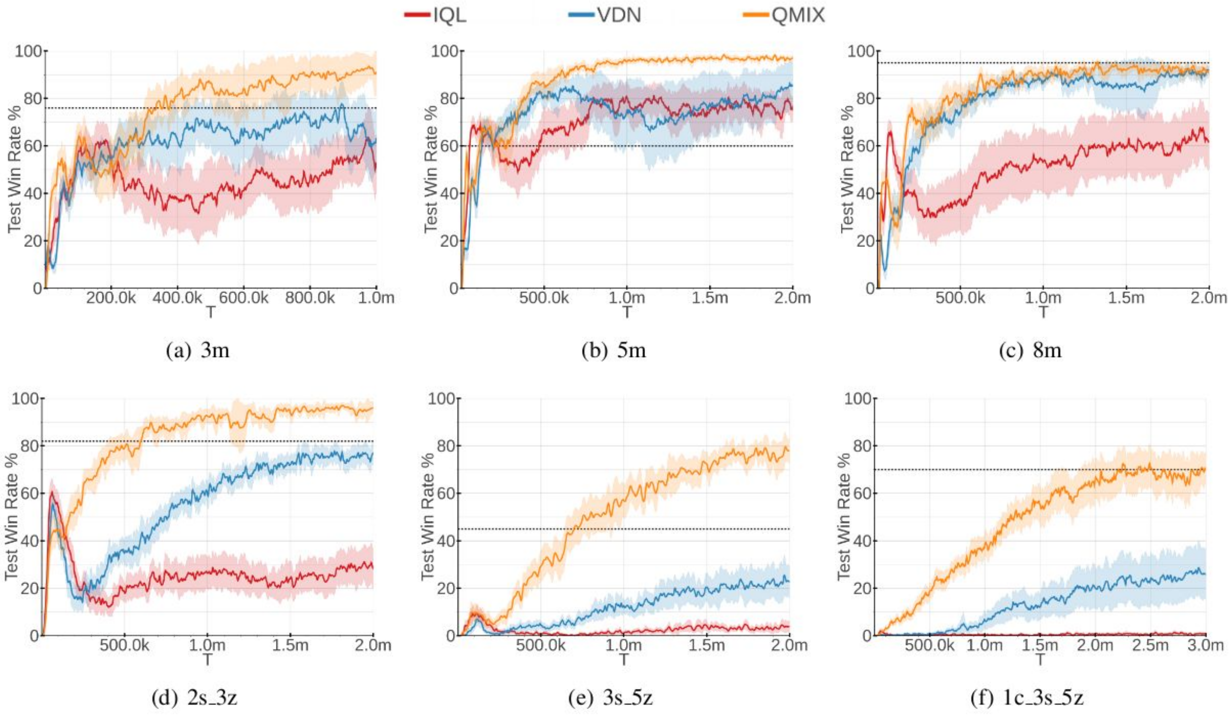
They are not expected returns

They are actually **utility functions**

QMIX - Architecture



QMIX - Results (Starcraft II)



SAPIENZA
UNIVERSITÀ DI ROMA

QMIX - Properties

Can represent a much **broader class** of Q_{tot} (**any monotonic** Q_{tot}) w.r.t. VDN (any additive Q_{tot})

Same scalability as VDN: **linear** in number of agents

QMIX - Limitations

Even if QMIX can represent any monotonic Q_{tot}, it can't approximate **any** joint value function (unlike COMA)

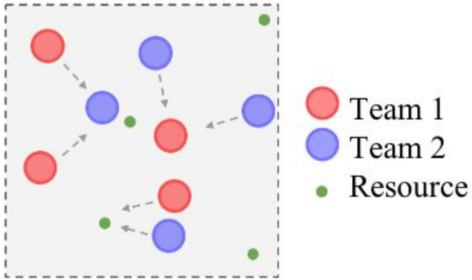
Example: the optimal action of one agent depends on the actions of other agents **at the same time step** (e.g. prisoner's dilemma)

Population-based MARL

SAPIENZA
UNIVERSITÀ DI ROMA

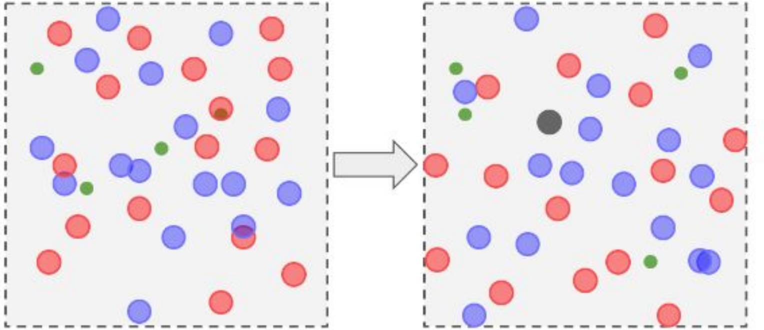
Scalability in MARL

Large number of agents is not only hard for computation
It also makes the **task much harder**



(b) *Adversarial battle*

Increase N of agents



(b) MADDPG

Evolutionary Population Curriculum (EPC)

Long et al. Evolutionary Population Curriculum for Scaling Multi-Agent Reinforcement Learning. https://arxiv.org/abs/2003.10423 (2020)

Start from training with **few agents**

Progressively increase the number of agents

This is a form of <u>curriculum learning</u>

EPC - Vanilla PC

(Vanilla) Population Curriculum (PC):

Given a target number of agents N^*:

1. train a small number of agents N_0 (e.g. MADDPG)
2. clone the population: $N_{k+1} = 2N_k$
3. train the population (e.g. MADDPG)
4. repeat 2-3 until $N >= N^*$

EPC - from vanilla PC to Evolutionary PC

Do **all policies** from stage k (with N_k) **transfer well** to stage k+1 (with $N_{k+1} = 2N_k$) ?

EPC - from vanilla PC to Evolutionary PC

Do **all policies** from stage k (with N_k) **transfer well** to stage k+1 (with $N_{k+1} = 2N_k$) ?

We **don't know** in general

EPC - from vanilla PC to Evolutionary PC

Do **all policies** from stage k (with N_k) **transfer well** to stage k+1 (with $N_{k+1} = 2N_k$) ?

We **don't know** in general

Solution: only **select** agents that transfer well

EPC - from vanilla PC to Evolutionary PC

Do **all policies** from stage k (with N_k) **transfer well** to stage k+1 (with $N_{k+1} = 2N_k$) ?

We **don't know** in general

Solution: only **select** agents that transfer well

- evolutionary algorithms

Evolutionary algorithms

Initialize the population $P_0 = \{x_1, \ldots, x_N\}$

For each generation i:

1. generate M offspring $O_i = \{y_1, \ldots, y_M\}$ with **crossover**
2. **mutate** O_i
3. P_{i+1} = best N individuals from P_i and O_i according to some performance metric (**fitness**)

Evolutionary Population Curriculum (EPC)

Initialize the population $P_0 = \{A_1, \ldots, A_K\}$, where each A_k is a group of N_0 agents

For each stage i:

1. **mix-and-match** (crossover): generate offspring $O_i = \{A'_1, \ldots, A'_C\}$, where A'_c is a group of $2N_i$ agents
2. **train** each group A'_c with MADDPG (mutation)
3. P_{k+1} = **best** K groups from O_k (selection)

EPC - Crossover (mix-and-match)

Concatenate each A_k with all the other $A_{k'}$ (k != k'):

- we obtain K(K+1)/2 groups of size $2N_i$

Sample C groups uniformly from all the K(K+1)/2 groups:

- we obtain the offspring $O_i = \{A'_1, \ldots, A'_C\}$, where A'_c is a group of $2N_i$ agents (C groups)

EPC - Mutation

Simply train each of the C groups with MADDPG, “mutating” their parameters

Can also use any other MARL (or even single-agent RL) algorithm

EPC - Selection

Select the K best groups from O_i according to average returns obtained after mutation

Note: this is the step that **improves over vanilla PC**: only the groups that **perform well** with the **increased population size** survive

EPC - Agent roles

EPC can also be applied when there is more than one agent "role"

"role" := agents that share the same observation space, action space, and reward function

EPC - Mix-and-match with multiple roles

We have K groups A_k^j for each role j

Concatenate each group A_k^j with all others of the **same role**

- we obtain K(K+1)/2 groups for each role j

If we have Ω roles, the possible combinations are:

$$C_{\max} = (K(K+1)/2)^{\Omega}$$

As before, we sample C combinations uniformly

EPC - Full algorithm

Algorithm 1: Evolutionary Population Curriculum

Data: environment $E(N, \{A_i\}_{1\leq i\leq\Omega})$ with N agents of Ω roles, desired population N_d, initial population N_0, evolution size K, mix-and-match size C

Result: a set of N_d best policies

$N \leftarrow N_0$;

initialize K parallel agent sets $A_i^{(1)}, \dots, A_i^{(K)}$ for each role $1 \leq i \leq \Omega$;

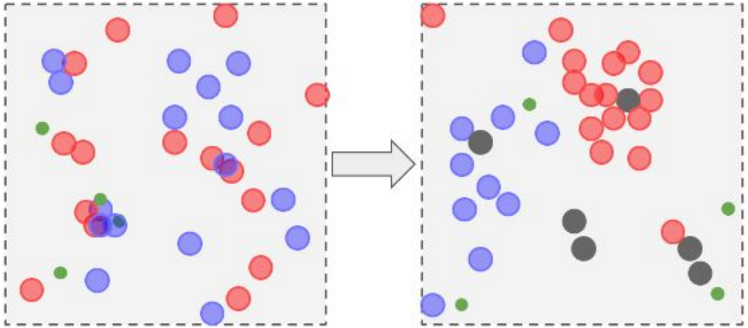
initial parallel MARL training on K games, $E(N, \{A_i^{(j)}\}_{1\leq i\leq\Omega})$ for $1 \leq j \leq K$;

while $N < N_d$ **do**

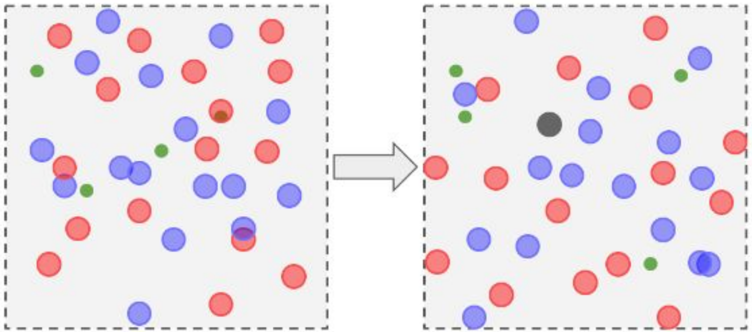
- $N \leftarrow 2 \times N$;
- **for** $1 \leq j \leq C$ **do**
  - for each role $1 \leq i \leq \Omega$: $j_1, j_2 \leftarrow \mathrm{unif}(1, K)$, $\tilde{A}_i^{(j)} \leftarrow A_i^{(j_1)} + A_i^{(j_2)}$ (*mix-and-match*);
- MARL training in parallel on $E(N, \{\tilde{A}_i^{(j)}\}_{1\leq i\leq\Omega})$ for $1 \leq j \leq C$ (*guided mutation*) ;
- **for** *role* $1 \leq i \leq \Omega$ **do**
  - **for** $1 \leq j \leq C$ **do**
    - $S_i^{(j)} \leftarrow \mathbb{E}_{k_{t\neq i}\sim[1,C]}\left[\text{avg. rewards on } E(N, \{\tilde{A}_1^{(k_1)}, \dots, \tilde{A}_i^{(j)}, \dots, \tilde{A}_\Omega^{(k_\Omega)}\})\right]$ (*fitness*);
  - $A_i^{(1)}, \dots, A_i^{(K)} \leftarrow$ top-K w.r.t. S_i from $\tilde{A}_i^{(1)}, \dots, \tilde{A}_i^{(C)}$ (*selection*);

return the best set of agents in each role, i.e., $\{A_i^{(k_i^\star)} | k_i^\star \in [1, K]\ \forall 1 \leq i \leq \Omega\}$;

EPC - Results



(a) EPC



(b) MADDPG

Malthusian Reinforcement Learning

Leibo et al. Malthusian Reinforcement Learning. https://arxiv.org/abs/1812.07019 (2019)

Malthusian Reinforcement Learning: drive the population growth based on performance

Why "Malthusian"? In 1798, Thomas Malthus argued that preindustrial income levels drove the subsequent population growth

Malthusian Reinforcement Learning

In Malthusian Reinforcement Learning, the population is divided into *species* that are spread across different *islands*

Islands := instances of the environment

Species := agents sharing the same policy

Malthusian Reinforcement Learning - Species

Each species l is composed of:

- a policy network π^l with parameters θ^l
- a categorical distribution μ^l over all islands i with weights $\mathbf{w}^l = (w_1{}^l, \ldots, w_I{}^l)$:

  $$\mu^l(i) = \mathrm{softmax}(w^l{}_i) = \exp(w_i{}^l)/\Sigma_j \exp(w_j{}^l)$$

In this way:

- each weight $w^l{}_i$ can be any real number
- $\mu^l(i)$ = probability of an agent of species l to be assigned to island i (and $\Sigma_i \mu^l(i) = 1$)

Malthusian Reinforcement Learning - Species

At each iteration e, for each species l we create the set of individuals $\Psi_{i,e}^{l}$ that are assigned to island i

The individuals are assigned to the islands by sampling M islands from each $\mu^{l}(i)$, so that each species has M individuals

In this algorithm, iterations are also called *ecological time steps* (that is why "e" is used for the iteration)

Malthusian Reinforcement Learning

The objective of Malthusian Reinforcement Learning is to maximize the fitness given by

$$\phi_{i,e}^{l} = \left(\sum_{k^{l} \in \Psi_{i,e}^{l}} \phi_{k^{l},e} \right) / |\Psi_{i,e}^{l}| \text{ and } 0 \text{ if } \Psi_{i,e}^{l} = \emptyset \,.$$

$\phi_{k^{l},e}$:= sum of rewards obtained by individual k^{l}

Malthusian Reinforcement Learning

The gradient update rule (with entropy regularization) is:

$$w_{e+1}^{l} = w_{e}^{l} + \alpha \left[\sum_{i \in \{1,\ldots,N_I\}} \nabla_{w^l} \mu^l(i)(\phi_{i,e}^{l} - \eta \log \mu^l(i)) \right]$$

Entropy regulation: don't overcommit to a specific island

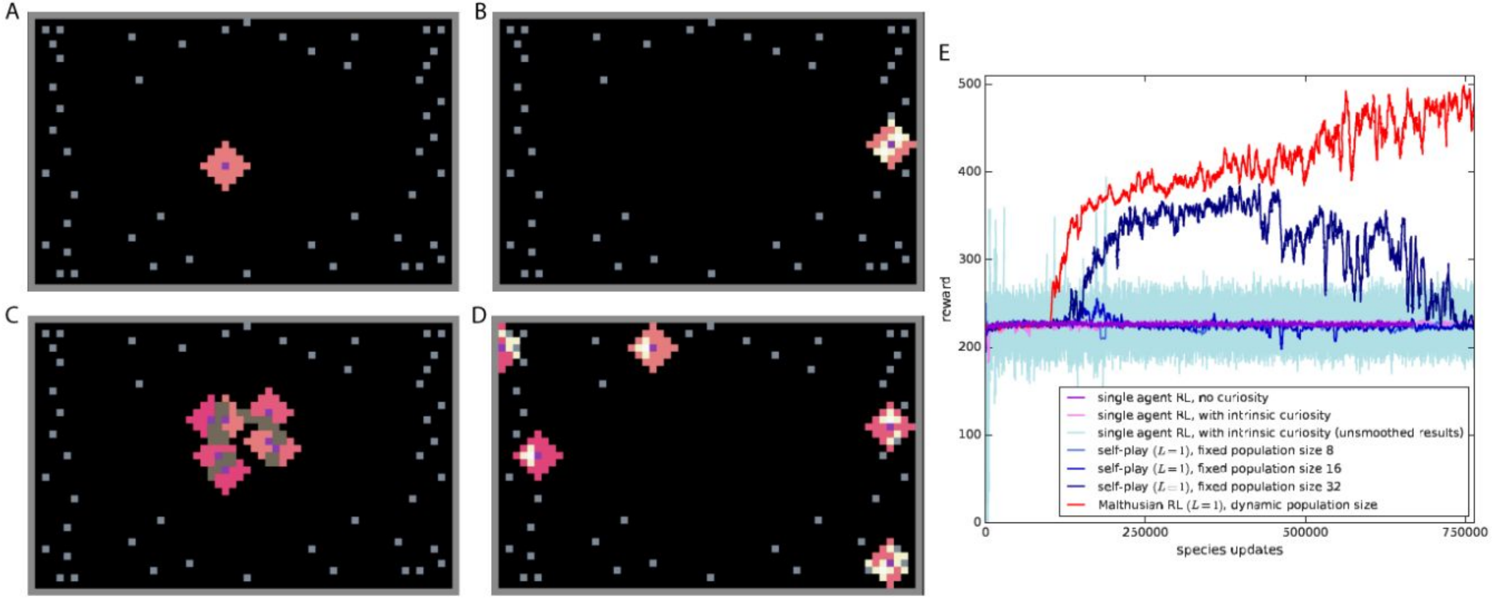
Malthusian Reinforcement Learning

Full algorithm:

1. Random initialization of policies π^l and distributions $\mu^l(i)$

2. For each iteration e:
   a. assign agents to islands using the $\mu^l(i)$
   b. collect one episode of experience from the islands
   c. update the policies π^l using any RL algorithm (V-Trace in this case)
   d. compute fitnesses
   e. update the species distribution weights $\mathbf{w}^l$

Malthusian Reinforcement Learning - Results



Large-population MARL: Capture the flag (Quake III)

Jaderberg et al. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. https://www.science.org/doi/10.1126/science.aau6249 (2019)

Large-population MARL: Capture the flag (Quake III)

(Some) challenges:

- partial observability
- credit assignment: reward is sparse
- generalization: opponents, teammates and maps can vary greatly
- several time-scales: strategy, navigation control, …
- computational complexity

Large-population MARL: Capture the flag (Quake III)

(Some) challenges:

- partial observability
- credit assignment: reward is sparse
- generalization: opponents, teammates and maps can vary greatly
- several time-scales: strategy, navigation control, …
- computational complexity

Large-population MARL: Capture the flag (Quake III)

Credit assignment problem: which actions contribute to the (sparse) reward?

Solution: learn an internal, **dense** reward signal **conditioned on game events** (e.g. flag captures)

Large-population MARL: Capture the flag (Quake III)

Generalization: how to adapt to different teammates, opponents, and maps?

Self-play? no:

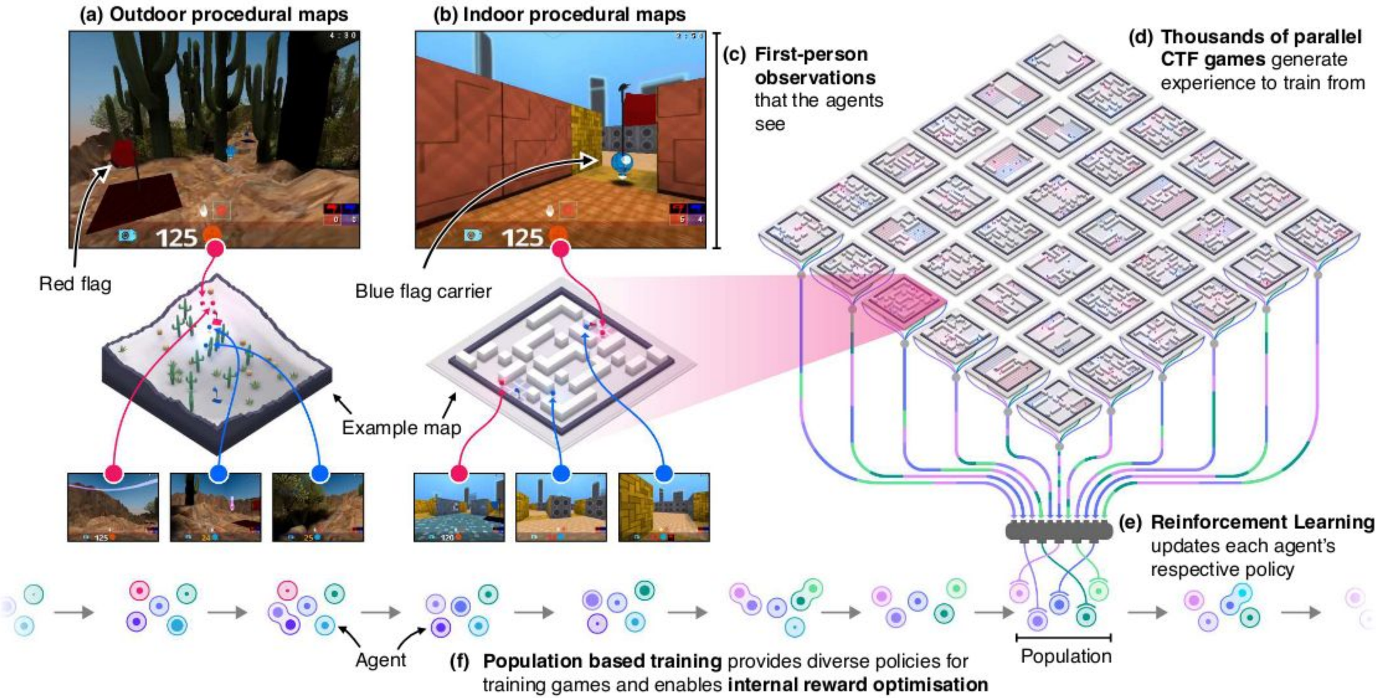
- does not guarantee behavioural variety of opponents/teammates
- not designed to be parallel (scalability problem)

Large-population MARL: Capture the flag (Quake III)

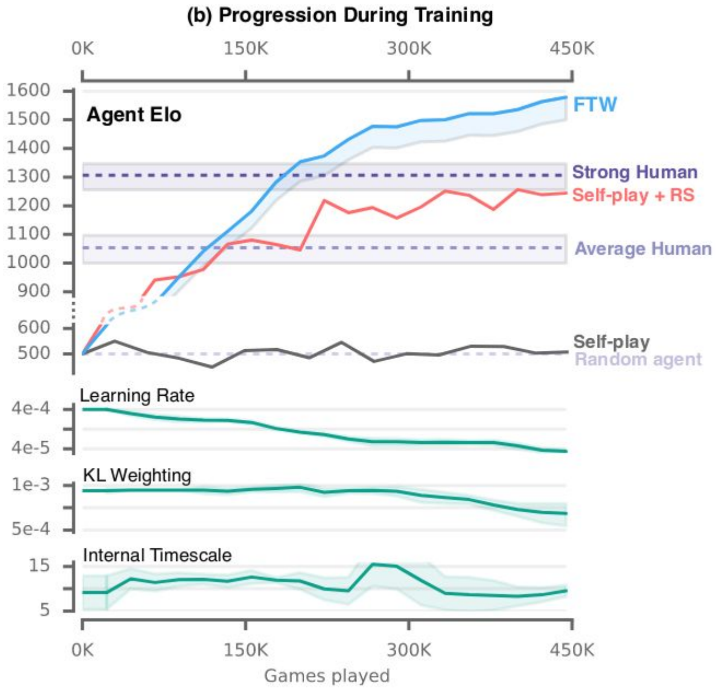
Large-population training:

- create N parallel games by sampling from the population
- use stochastic matchmaking to match agents of similar skill level (ELO-based)
- bad news: still requires a lot more compute! (good news: it's fully parallel)

Large-population MARL: Capture the flag (Quake III)

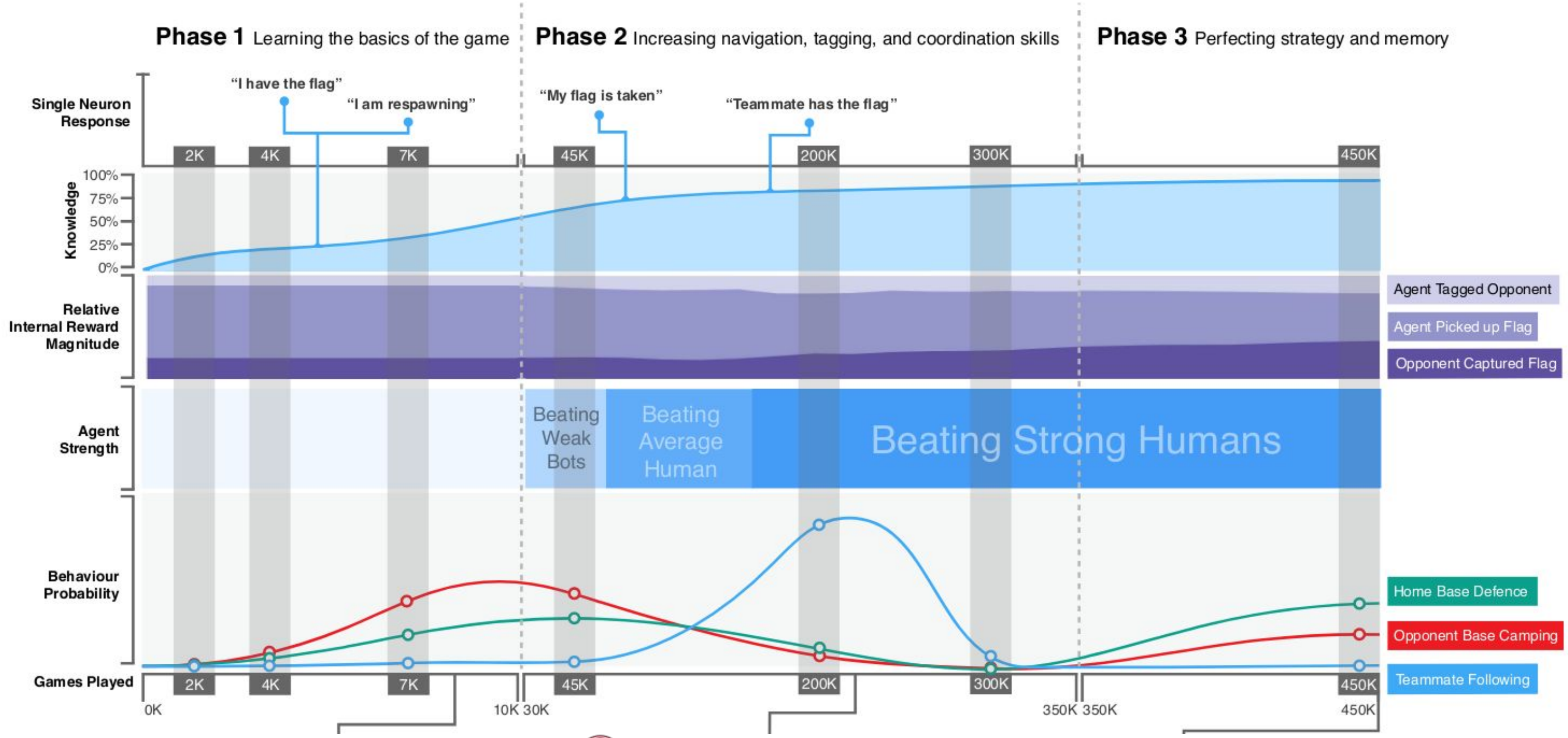


Large-population MARL: Capture the flag (Quake III)

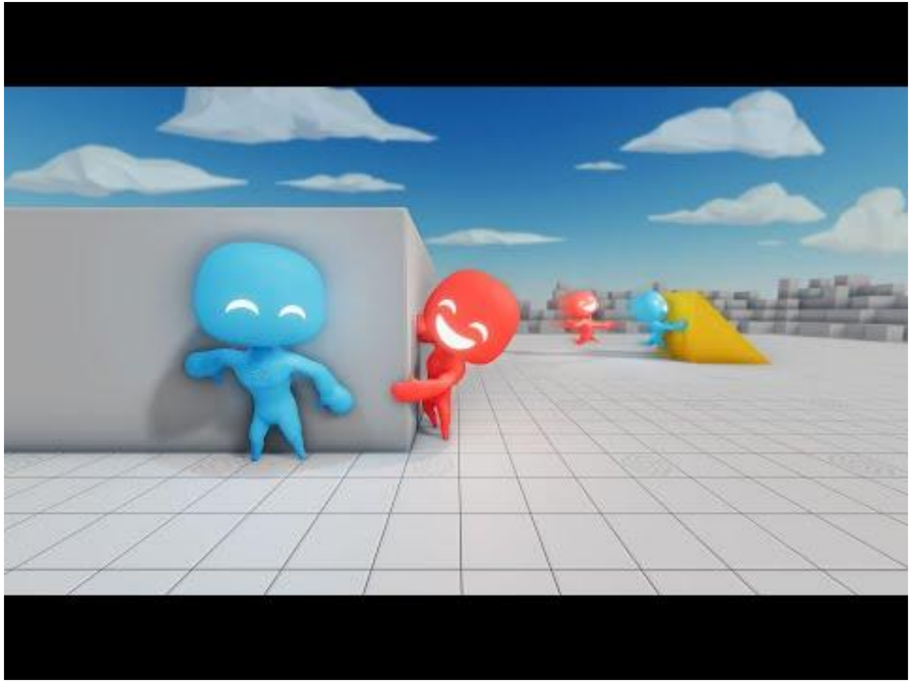


SAPIENZA
UNIVERSITÀ DI ROMA

Large-population MARL: Capture the flag (Quake III)



Emergent behaviours - Hide & Seek



Baker et al. Emergent Tool Use From Multi-Agent Autocurricula. https://arxiv.org/abs/1909.07528 (2019)

Emergent Behaviours - Motivations

What if non-stationarity (NS) was actually good?

Co-adaptation and competition can lead to “arms races” that continually produce **emergent skills** (automatic curriculum learning)

This is also similar to how biological life evolved on earth

Emergent Behaviours

To continue indefinitely, this process needs the right conditions:

At any given time, the challenge(s) posed to each agent by <u>(1) the other agents</u> and <u>(2) the environment</u> must **not be too easy or too difficult** to overcome (w.r.t. to the agent’s adaptation mechanism, e.g. any RL algorithm)

Emergent Behaviours - Hide & Seek

Emergent behavioural stages:

1. Running and chasing
2. Shelter construction
3. Ramp use
4. Ramp defense
5. Box surfing
6. Box defense

After these, **hiders become dominant** (no further emergence observed)

Emergent Behaviours - Evaluation

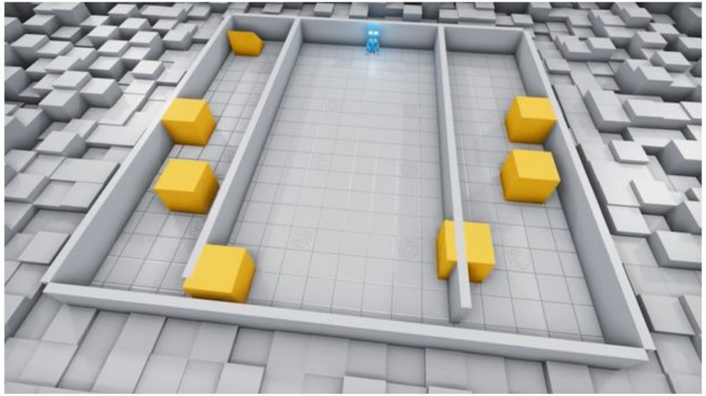
How to **evaluate** emergent behaviours?

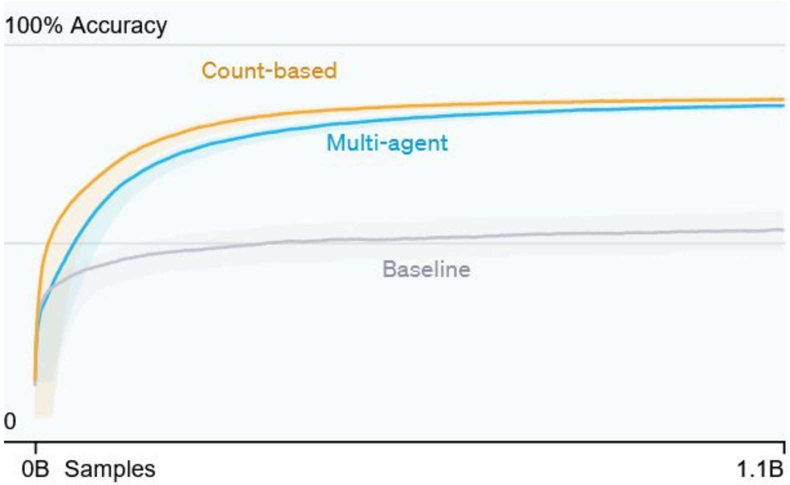
Test **transfer performance** on general skills:

- motor skills (navigation, manipulation)
- cognitive skills (memory)

Emergent behaviours - Object Permanence

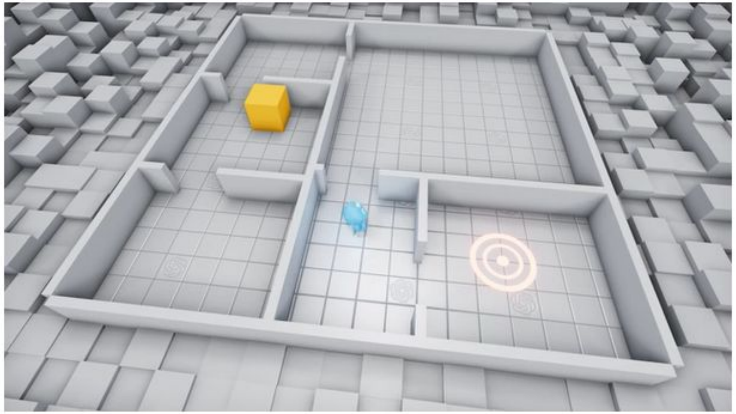
Object counting The agent is pinned in place and asked to predict how many objects have gone right or left, testing the agent's memory and sense of object permanence.

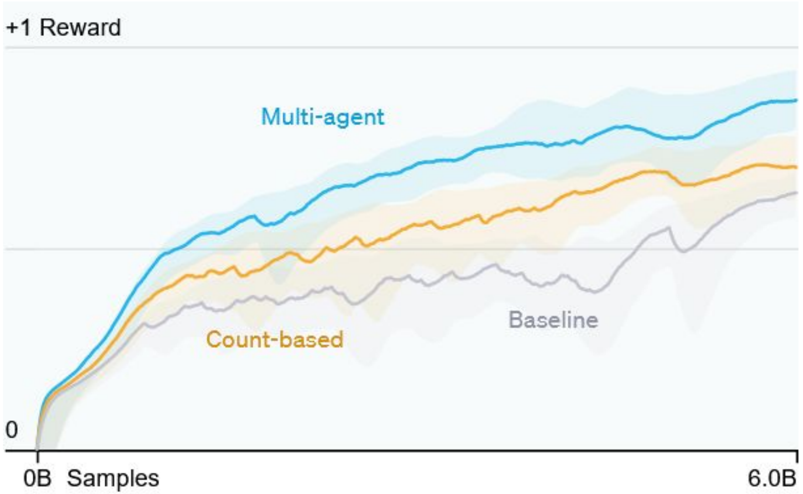




Emergent behaviours - Long term memory

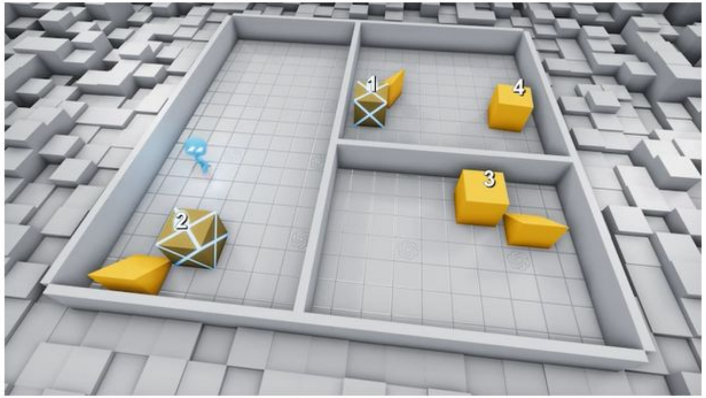
Lock and return The agent must find the box, lock it, and return to its original position, which tests the agent's long term memory of its location.

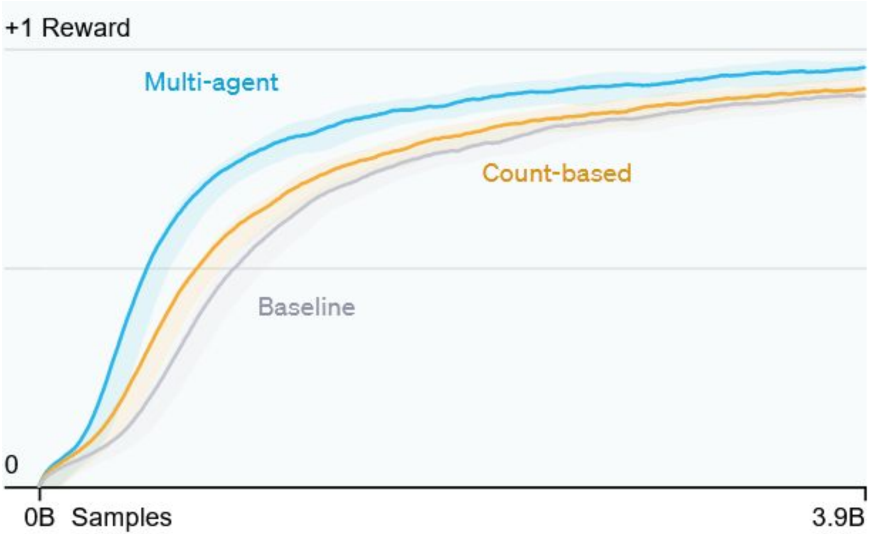




Emergent behaviours - Long term memory

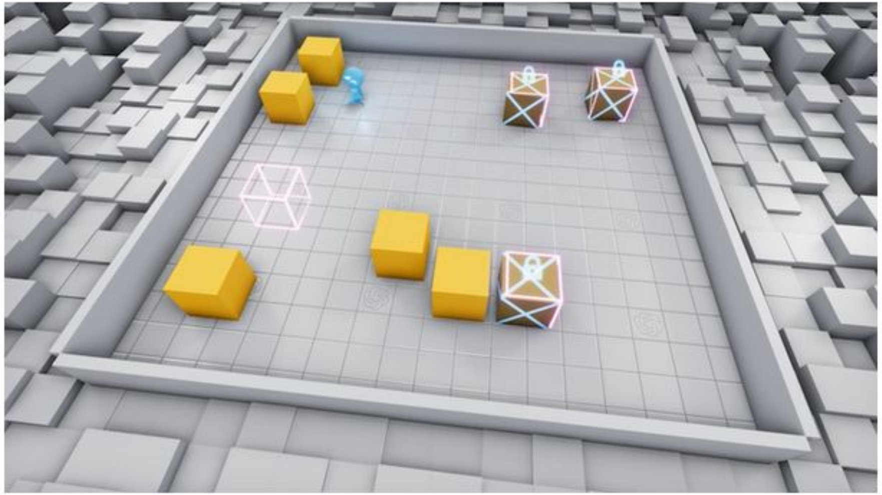
Sequential lock The agent must lock boxes in an order unobserved to the agent. Boxes can only be locked in the correct order, so the agent must remember the status of boxes it has seen.

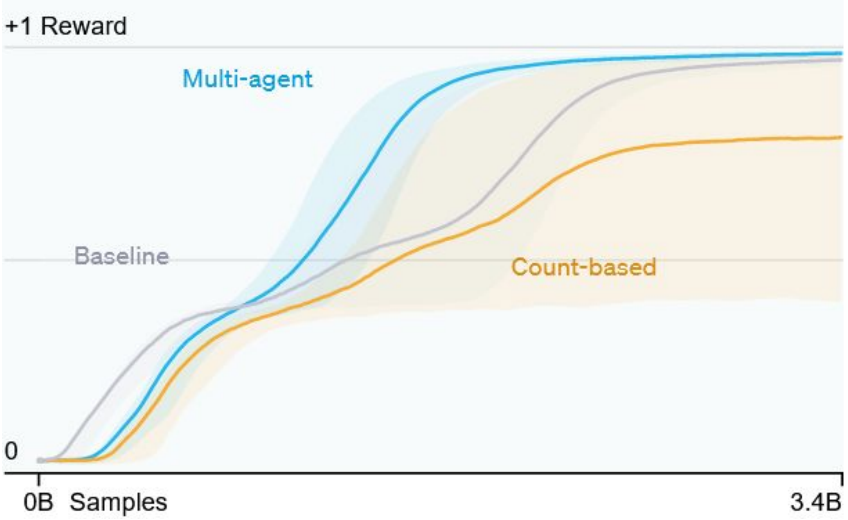




Emergent behaviours - Manipulation

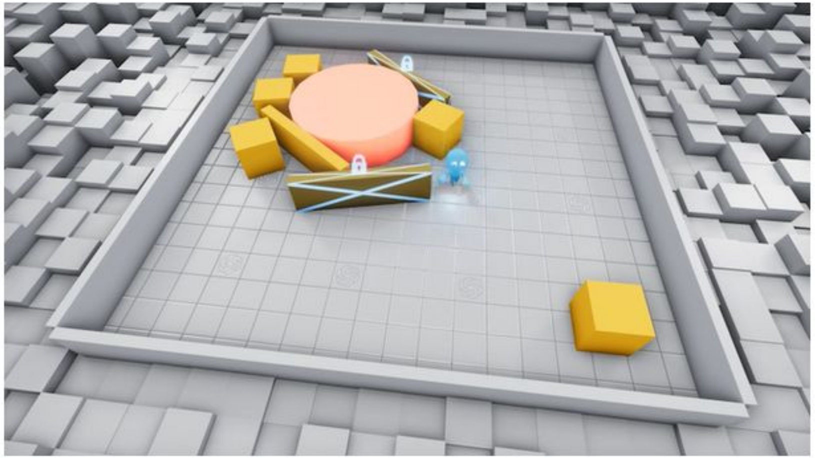
Blueprint construction The agent must move boxes to the target locations.





Emergent behaviours - Manipulation

Shelter construction The agent must construct a shelter around the cylinder.



Emergent Behaviours - Challenges

Open-endedness: how to design environments that don't admit "dominating" strategies?

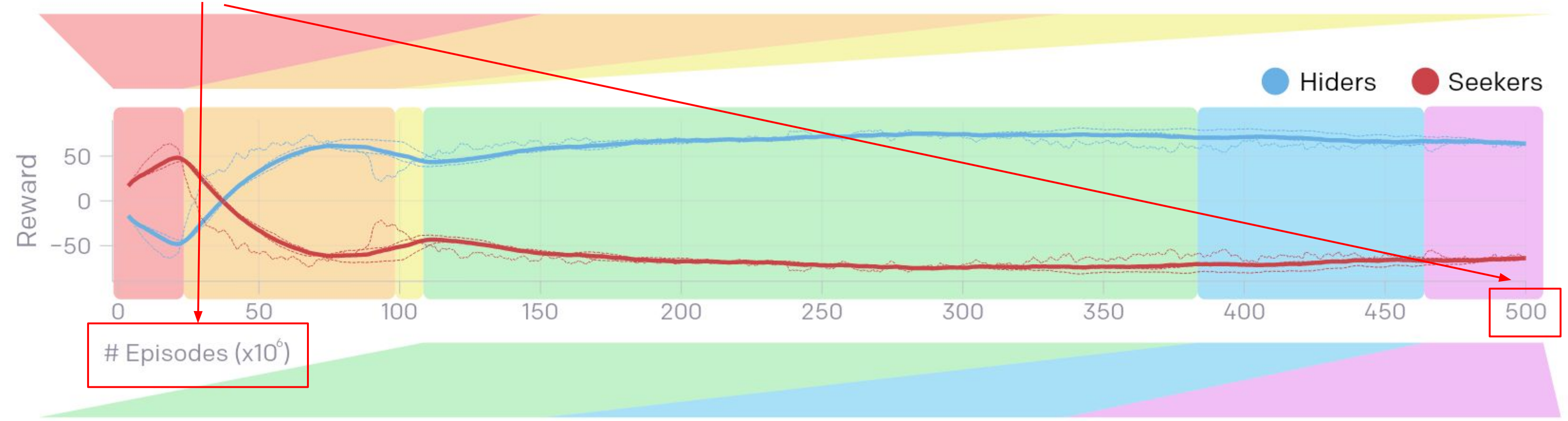
- Example: in hide & seek, when using a **deterministic** environment **only 4 stages** of emergence

Evaluation:

- how to choose test skills?
- is transfer learning performance the best metric?

Emergent Behaviours - Challenges

Sample efficiency: look at the training time!



500 Million episodes x 200 steps per episode = 10^11 steps!