Promoting Coordination through Policy Regularization in Multi-Agent Reinforcement Learning

Julien Roy*
Québec Institute for AI (Mila)
École Polytechnique de Montréal
julien-2.roy@polymtl.ca

Paul Barde*
Québec Institute for AI (Mila)
École Polytechnique de Montréal
paul.b.barde@gmail.com

Félix G. Harvey
Québec Institute for AI (Mila)
École Polytechnique de Montréal

Derek Nowrouzezahrai
Québec Institute for AI (Mila)
McGill University

Christopher Pal
Québec Institute for AI (Mila)
École Polytechnique de Montréal

Abstract

A central challenge in multi-agent reinforcement learning is the induction of coordination between agents of a team. In this work, we investigate how to promote inter-agent coordination and discuss two possible avenues based respectively on inter-agent modelling and common internal representation learning. We test each approach in five challenging continuous control tasks with sparse rewards and compare them against MADDPG, a state-of-the-art multi-agent reinforcement learning algorithm. To ensure a fair comparison, we rely on a thorough hyper-parameter selection and training methodology that allows a fixed hyper-parameter search budget for each algorithm. We consequently assess both the hyper-parameter sensitivity and asymptotic performance of each learning method.

1 Introduction

Multi-agent Reinforcement Learning (MARL) refers to the task of training an agent to maximize its expected return by interacting with an environment that contains other learning agents. It represents a challenging branch of reinforcement learning with interesting developments in recent years [7, 13, 3].

A popular framework for MARL is the use of a centralized training procedure and decentralized execution [13, 3, 6, 2, 16], typically done by training critics that approximate the value of the joint observations and actions, themselves used to train actors that are restricted to the observation-action pair of a single agent. Such critics, if exposed to coordinated joint actions leading to high returns, can steer the agents' policies toward these highly rewarding behaviors. However, these approaches depend on the agents luckily stumbling on coordinated joint actions in order to grasp their benefit. Thus, it might fail in scenarios where coordination is unlikely to occur by chance. We hypothesize that in such scenarios, coordination-promoting inductive biases on the policy search could discover coordinated behaviors more efficiently and supersede task related reward shaping and curriculum learning.

In this work, we assume two different priors to successful coordination and use these to regularize the learned policies. The first avenue (TeamMADDPG) supposes that agents must predict the behavior of its teammates in order to coordinate with them. The second (CoachMADDPG) assumes that coordinating agents share some common representation of the current situation. In the following sections we show how to derive practical regularization terms from these premises and meticulously evaluate them[1].

2 Background

In this work we consider the framework of Markov Games [12], a multi-agent extension of Markov Decision Processes (MDPs) (see Section 6.1 in Appendix) with N independant agents.

*Equal contribution
[1]The code for both the algorithms and environments will be made publicly available upon final publication of this work.

2.1 Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

MADDPG [13] is an adaptation of the Deep Deterministic Policy Gradient algorithm (DDPG) [11] to the multi-agent setting. It allows the training of cooperating and competing decentralized policies through the use of a centralized training procedure. In this framework, each agent i possesses its own deterministic policy μ^i for action selection and critic Q^i for state-action value estimation which are respectively parametrized by ϕ_i and θ_i. All parametric models are trained off-policy from previous transitions $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{o}_{t+1})$ uniformly sampled from a replay buffer $\mathcal{D}$. Note that $\mathbf{o}_t = [o_t^1, ..., o_t^N]$ is the joint observation vector and $\mathbf{a}_t = [a_t^1, ..., a_t^N]$ is the joint action vector, obtained by concatenating the individual observation vectors o_t^i and action vectors a_t^i of all N agents. Each centralized critic, is trained to estimate the expected return for a particular agent i using the Deep Q-Network (DQN) [14] update:

$$\nabla_{\phi_i}\mathcal{L}(\phi_i) = \mathbb{E}_{\mathbf{o}_t,\mathbf{a}_t,\mathbf{o}_{t+1},r_t^i\sim\mathcal{D}}\left[\nabla_{\phi_i}\frac{1}{2}\left(Q^i(\mathbf{o}_t,\mathbf{a}_t;\phi_i) - (r_t^i + \gamma Q^i_{\text{target}}(\mathbf{o}_{t+1},\mathbf{a}_{t+1}))\right)^2\Big|_{a^j_{t+1}=\mu^j(o^j_{t+1})}\right] \tag{1}$$

Each policy is updated to maximize the expected discounted return of the corresponding agent-i using the policy gradient:

$$\nabla_{\theta_i} J_{PG}(\theta_i) = \mathbb{E}_{\mathbf{o}_t,\mathbf{a}_t\sim\mathcal{D}}\left[\nabla_{\theta_i}\mu^i(o_t^i;\theta_i)\nabla_{a_t^i}\, Q^i(\mathbf{o}_t,\mathbf{a}_t)\big|_{a_t^i=\mu^i(o_t^i)}\right] \tag{2}$$

By guiding each agent's policy toward states that have been more positively evaluated when taking into account *all* agents' observation-action pairs, the centralized training of the value-function restores stationary in the multi-agent setting. In addition, this mechanism can allow to implicitly learn coordinated strategies that can then be deployed in a decentralized way. However, this procedure does not encourage the *discovery* of coordinated strategies since high-reward behaviors have to be randomly experienced through unguided exploration.

3 Related Work

Many works in multi-agent reinforcement learning consider explicit communication channels between the agents and distinguish between communicative actions (broadcasting a given message) and physical actions (moving in a given direction) [4, 15, 10]. Consequently, they often focus on the emergence of language, considering tasks where the agents must discover a common communication protocol in order to succeed. Deriving a successful communication protocol can already be seen as coordination in the communicative action space and can enable, to some extent, successful coordination in the physical action space [1]. However, explicit communication is not a necessary condition for coordination as agents can rely on physical communication [15, 5].

Approaches to shape RL agents' behaviours with respect to other agents have also been explored. Jacques et al. [8] use the mutual information between the agent's policy and a goal-independent policy to shape the agent's behaviour towards hiding or spelling out its current goal. However, this approach is only applicable for tasks with an explicit goal representation and is not specifically intended for coordination. Strouse et al. [17] use the direct causal effect between agent's actions as an intrinsic reward to encourage social empowerment, but rely on giving to some agents (the influencees) the action of other agents (the influencers) as input to their policy, which violates the decentralized execution framework.

4 Policy regularization

4.1 Team regularization

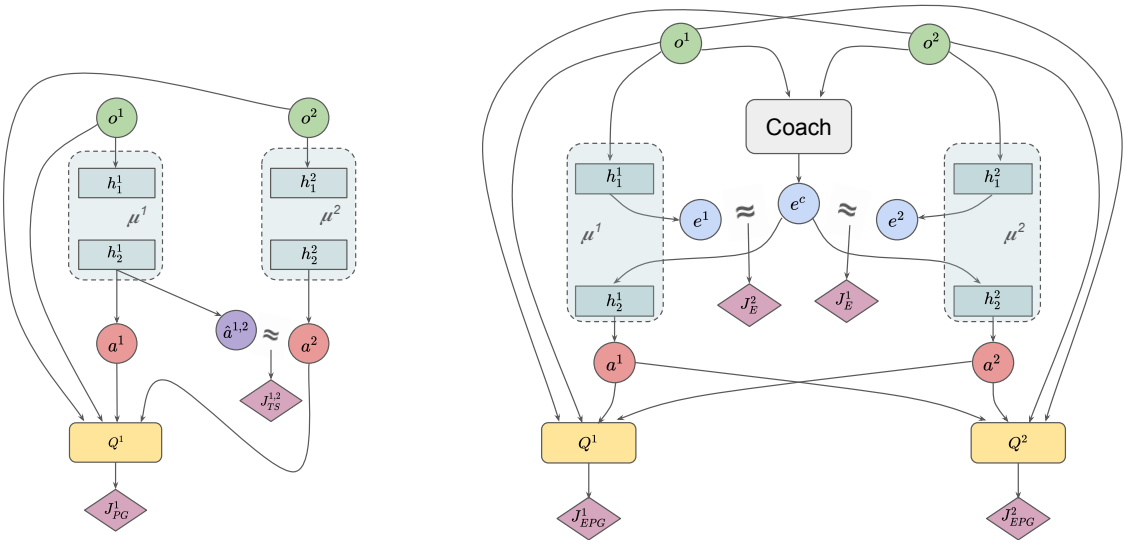
We hypothesize that coordination can be defined as the degree of predictability of one agent's behavior with respect to its teammate(s). In other words, in a coordinated team, there should be some predictable structure between the agent's actions. We cast this assumption into the decentralized framework by training agents to predict their teammates actions given only their own observation. We define this regularization term as the Mean Squared Error (MSE) between the predicted and real actions of the teammates, yielding a teammate-modelling secondary objective similar to the models of other agents used in [8]. Concurrently, the same objective is used to drive the teammates' behaviors closer to the prediction. We call this the Team-Spirit regularization term $J_{TS}^{i,j}$ between agents i and j:

$$J_{TS}^{i,j}(\theta_i,\theta_j) = \text{MSE}(\hat{a}_t^{i,j}, a_t^j) = \frac{1}{2}\sum_{k=1}^{|\mathcal{A}^j|}(\hat{a}_{t,k}^{i,j} - a_{t,k}^j)^2 \tag{3}$$

The total gradient for a given agent-i becomes:

$$\nabla_{\theta_i} J_{total}(\theta_i) = \nabla_{\theta_i} J_{PG}(\theta_i) + \lambda_1 \sum_j \nabla_{\theta_i} J_{TS}^{i,j}(\theta_i,\theta_j) + \lambda_2 \sum_j \nabla_{\theta_i} J_{TS}^{j,i}(\theta_j,\theta_i) \tag{4}$$

We call this modified algorithm TeamMADDPG. The diagram of Figure 1a summarizes these interactions.



(a) TeamMADDPG (b) CoachMADDPG

Figure 1: Illustrations of the proposed regularization schemes in a two-agents MARL problem. (a) Each agent's policy is equipped with additional heads that are trained to predict other agents' actions and every agent is regularized to produce actions that its teammates correctly predict. Note that this regularization is depicted for agent-1 only to avoid cluttering. (b) A central model called *Coach* takes all agents' observations as input and produces a single learned embedding. Agents are regularized to make their internal representation match the coach embedding and adequately behave with it.

4.2 Coach regularization

This regularization choice builds on the assumption that coordination can be more easily achieved if agents had a common description of the current situation. To promote this, we introduce the coach C, a new entity parametrized by ψ, that is only present at training time and has access to the observations of all the agents. The purpose of the coach is to produce an embedding, $C(\mathbf{o}_t) = e_t^c$ from the joint observations that can be used by each agent to succeed at the task at hand. In practice, the policy network μ^i of each agent i is divided into two stages. The first stage h_1^i takes as input the observation o_t^i and outputs a representation of the situation, e_t^i. The second stage h_2^i takes as input e_t^i and outputs the agent's action a_t^i. It follows that $\mu^i(o_t^i) = h_2^i(h_1^i(o_t^i)) = h_2^i(e_t^i)$. Then the coach is trained to output embeddings that yield high returns when passed to the agents and each agent is regularized so that (1) its hidden representation matches the coach's embedding and (2) it derives good actions from it. The gradient for the coach is given by:

$$\frac{1}{N}\sum_{i=1}^{N}\nabla_\psi J_{EPG}(\psi) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\mathbf{o}_t,\mathbf{a}_t\sim\mathcal{D}}\left[\nabla_\psi C(\mathbf{o}_t;\psi)\nabla_{e_t^c}h_2^i(e_t^c)\nabla_{a_t^i}Q^i(\mathbf{o}_t,\mathbf{a}_t)\Big|_{a_t^i=h_2^i(e_t^c),\, e_t^c=C(\mathbf{o}_t)}\right] \tag{5}$$

Then the first stage of the policy of each agent is regularized to match the coach's embedding with the following gradient:

$$\nabla_{\theta_i}J_E(\theta_i) = \mathbb{E}_{\mathbf{o}_t\sim\mathcal{D}}\left[\nabla_{\theta_i}\text{MSE}\left(h_1^i(o_t^i;\theta_i), e_t^c\right)\Big|_{e_t^c=C(\mathbf{o}_t)}\right] \tag{6}$$

Finally, the second stage of the policy of each agent is regularized to maximize the expected discounted return from the coach embedding:

$$\nabla_{\theta_i}J_{EPG}(\theta_i) = \mathbb{E}_{\mathbf{o}_t,\mathbf{a}_t\sim\mathcal{D}}\left[\nabla_{\theta_i}h_2^i(e_t^c;\theta_i)\nabla_{a_t^i}\,Q^i(\mathbf{o}_t,\mathbf{a}_t)\Big|_{a_t^i=h_2^i(e_t^c),\, e_t^c=C(\mathbf{o}_t)}\right] \tag{7}$$

The total update for a given agent-i becomes:

$$\nabla_{\theta_i}J_{total}(\theta_i) = \nabla_{\theta_i}J_{PG}(\theta_i) + \lambda_1\nabla_{\theta_i}J_E(\theta_i) + \lambda_2\nabla_{\theta_i}J_{EPG}(\theta_i) \tag{8}$$

where λ_1 and λ_2 are the regularization coefficients. We call this modified algorithm CoachMADDPG. The diagram of Figure 1b summarizes these interactions.

5 Results and Discussion

To test our methods, we designed five cooperative multi-agent tasks based on OpenAI multi-agent particle environment [15]. An agent receives as input its own global position and velocity as well as the relative position of other agents and

landmarks. All environments have continuous action and state spaces and only provide sparse rewards. A complete description of each environment along with illustrations are available in Section 6.2 in appendix. We compare our algorithms (TeamMADDPG and CoachMADDPG) against MADDPG, as well as its single-agent counterpart (DDPG) and another baseline called SharedMADDPG which shares the policy and value-function models across agents. We report in Figure 2 the results of our per-algorithm hyper-parameter tuning as described in Section 6.3.

Stability across hyper-parameter configurations is a recurring challenge in deep reinforcement learning. The average performance (across seeds) for each randomly sampled hyper-parameters configuration allows to evaluate the robustness of an algorithm with respect to its hyper-parameters. Figure 2 shows that while MADDPG, CoachMADDPG and TeamMADDPG are the best performing algorithms on the tuning environment (Spread, see Appendix Figure 4a), TeamMADDPG displays much more reliable performances with 11 configurations above an average performance of 60, compared to only 5 and 4 configurations respectively for MADDPG and CoachMADDPG, suggesting that our Team-Spirit regularization improves robustness across hyper-parameters. We hypothesize that this improved stability could be explained by either the policy-exploration bias that this second objective provides or the richer representations it induces.

Figure 2: Hyper-parameter tuning results for all algorithms. Each point represents performance at the end of training for one sampled hyper-parameters configuration, averaged over 3 trainings seeds.

We retrain all algorithms with their most successful configuration, this time on 10 different seeds and on all environments, and show the average learning curves in Figure 3. SharedDDPG performs significantly worse than the other algorithms on three environments (Spread, Chase and Bounce) and DDPG only solves Gather, the most simple one. While [13] suggest that DDPG bad performance is explained by the absence of centralized critics makes the environment appear stationary from any individual agent's perspective [13]. Trailing results with SharedMADDPG might be explained by the fact that sharing parameters across agents prevents specialization, a useful asset even in homogeneous cooperative tasks. Finally, while MADDPG and TeamMADDPG perform similarly well on all environments, CoachMADDPG is found to perform significantly better on two of them (Chase and Bounce), suggesting that regularizing toward a common internal representation between agents might foster efficient coordinated strategies. However, hyper-parameter tuning on each environment and for a greater number of samples would be necessary to validate these hypotheses.

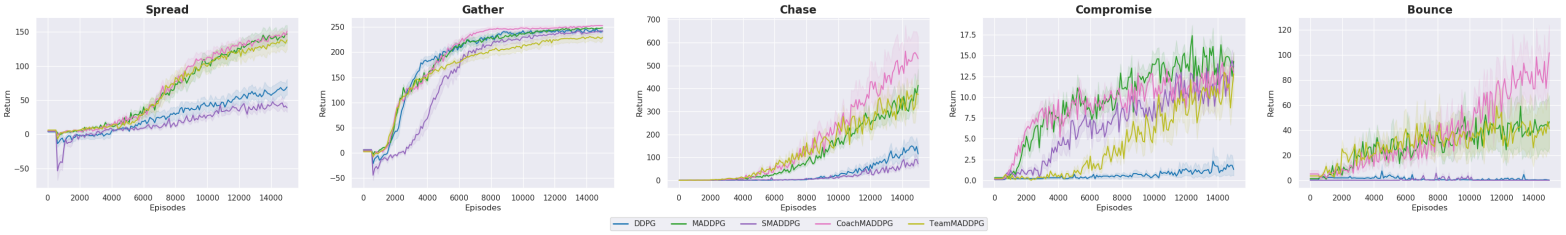


Figure 3: Average learning curves for all algorithms on all five environments. Solid lines are the mean across 10 training seeds of the evaluation performance on 10 fixed episodes. The envelopes are the standard error across the 10 training seeds.

6 Conclusion

In this work we introduced two policy regularization methods to promote multi-agent coordination. The preliminary results are encouraging: acting from a common embedding seems to increase the sample efficiency on two coordination tasks where agents must jointly tune their behavior in order to succeed. On the other hand being able to predict teammates' intentions appears to increase training robustness across hyper-parameter configurations. However, a thorough investigation is due before drawing clear conclusions on the benefits of these methods.

Acknowledgments

We wish to thank Olivier Delalleau for providing insightful comments on previous versions of these methods as well as *Fonds de Recherche Nature et Technologies* (FRQNT), Ubisoft Montréal and Mitacs for providing part of the funding for this work.

References

[1] Sanjeevan Ahilan and Peter Dayan. "Feudal multi-agent hierarchies for cooperative reinforcement learning". In: *arXiv preprint arXiv:1901.08492* (2019).

[2] Jakob N Foerster et al. "Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1811.01458* (2018).

[3] Jakob N Foerster et al. "Counterfactual multi-agent policy gradients". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[4] Jakob Foerster et al. "Learning to communicate with deep multi-agent reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2137–2145.

[5] Jayesh K. Gupta, Maxim Egorov, and Mykel J. Kochenderfer. "Cooperative Multi-agent Control Using Deep Reinforcement Learning". In: *AAMAS Workshops*. 2017.

[6] Shariq Iqbal and Fei Sha. "Actor-Attention-Critic for Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1810.02912* (2018).

[7] Max Jaderberg et al. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning". In: *arXiv preprint arXiv:1807.01281* (2018).

[8] Natasha Jaques et al. "Intrinsic Social Motivation via Causal Influence in Multi-Agent RL". In: *arXiv preprint arXiv:1810.08647* (2018).

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. "Multi-agent cooperation and the emergence of (natural) language". In: *arXiv preprint arXiv:1612.07182* (2016).

[11] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[12] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[13] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6379–6390.

[14] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[15] Igor Mordatch and Pieter Abbeel. "Emergence of grounded compositional language in multi-agent populations". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[16] Tabish Rashid et al. "QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning". In: *arXiv preprint arXiv:1803.11485* (2018).

[17] Daniel Strouse et al. "Learning to share and hide intentions using information regularization". In: *Advances in Neural Information Processing Systems*. 2018, pp. 10270–10281.

Appendix

6.1 Markov Games

A Markov Game involves N independent agents and is defined by the tuple $\langle \mathcal{S}, \mathcal{T}, \mathcal{P}, \{\mathcal{O}^i, \mathcal{A}^i, \mathcal{R}^i\}_{i=1}^N \rangle$. With $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{P}$ being respectively the set of all possible states, the transition function and the initial state distribution. While these are global properties of the environment, $\mathcal{O}^i$, $\mathcal{A}^i$ and $\mathcal{R}^i$ are individually defined for each agent i. They are respectively the observation functions, the sets of all possible actions and the reward functions. At each time-step t, the global state of the environment is given by $s_t \in \mathcal{S}$ and every agent's individual action vector is denoted by $a_t^i \in \mathcal{A}^i$. To select their action, each agent i has only access to its own observation vector o_t^i which is extracted by its observation function $\mathcal{O}^i$ from the global state s_t. The initial global state s_1 is sampled from the initial state distribution $\mathcal{P} : \mathcal{S} \mapsto [0, 1]$ and the next states of the environment s_{t+1} are sampled from the probability distribution over the possible next states $P(\mathcal{S})$ given by the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto P(\mathcal{S})$. Finally, at each time-step each agent receives an individual scalar reward r_t^i from its reward function $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathbb{R}$. Agents aim at maximizing their expected discounted return $\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t^i\right]$ over the time horizon T, where $\gamma \in [0, 1]$ is a discount factor.

6.2 Training environments

All of our environments are based on a modified version of the OpenAI multi-agent particle environments [15] and require a certain degree of team-coordination to maximize the return. Figure 4 presents visualizations of all five environments.

Spread (Figure 4a): In this environment, there are 3 agents (small orange circles) and 3 landmarks (bigger gray circles). At every timestep, agents receive a team-reward $r_t = n$ where n is the number of landmarks occupied by at least one agent. To maximize their return, agents must therefore split themselves on all landmarks.

Compromise (Figure 4b): In this environment, two agents (small blue and pink circles) are linked together with a spring that pulls them toward each-other when stretched above its relaxation length. Each agent is rewarded by $r_t = 1$ for reaching the landmark (bigger circles) of its corresponding color. When a landmark is reached, it is automatically relocated randomly elsewhere in the environment. Importantly, the two agents receive as input the relative position of the other agent as well as the relative position of its own landmark, but not the position of the other agent's landmark. To maximize their return, agents must learn to reach their own landmark, while sometimes compromising and follow the other agent to avoid both of them pulling in opposite directions and consequently remain immobile.

Chase (Figure 4c): In this environment, two predators (red circles) are chasing one prey (green circle). The prey moves w.r.t a hardcoded policy consisting of repulsion forces from the walls and predators. At every timestep, the learnable agents (predators) receive a team-reward of $r_t = n$ where n is the number of predator touching the prey. The prey has a greater top speed and acceleration than the predators. To maximize their return, they must coordinate to squeeze the prey into a corner or a wall in order to catch it.

Bounce (Figure 4d): In this environment, two agents (small purple circles) are linked together with a spring that pulls them toward each-other when stretched above its relaxation length. At mid-point during the episode, a ball (smaller red circle) falls from the top of the environment. Agents must position correctly to use the spring linking them to as a trampoline and make the ball bounce towards the target (bigger beige circle). They receive a team-reward of $r_t = 0.1$ if the ball reflects towards the side walls, $r_t = 0.2$ if the ball reflects towards the top of the environment, and $r_t = 1$ if the ball reflects towards the target.

Gather (Figure 4e): In this environment, there are 5 agents (small blue circles) and 1 landmark (bigger gray circle). At every timestep, agents receive a team-reward of $r_t = 1$ for all agents in the landmark, and $r_t = -2$ for all pair of agents colliding with one another. To maximize their return, agents must therefore gather as quickly together in the landmark, while avoiding collisions.

6.3 Hyper-parameter tuning and training details

To offer a fair comparison of all methods, we tune the hyper-parameters individually for all algorithms on *Spread*, one of the most challenging task, and reuse their best configuration to train them on all other tasks. Random searches are performed to carry out these tuning procedures. We train the model over 50 sets of randomly sampled hyper-parameter values and over 3 different seeds for each of these sets. The best set is then used to train this method on all tasks over 10 seeds each. The tuned hyper-parameters are the learning rates of the actor and critic, the gradient-clipping threshold and the initial noise scale for all algorithms. Additionnally, TeamMADDPG and CoachMADDPG also search over the λ_1 and λ_2 regularization coefficients and finally CoachMADDPG additionnally tunes the learning rate for the coach model.

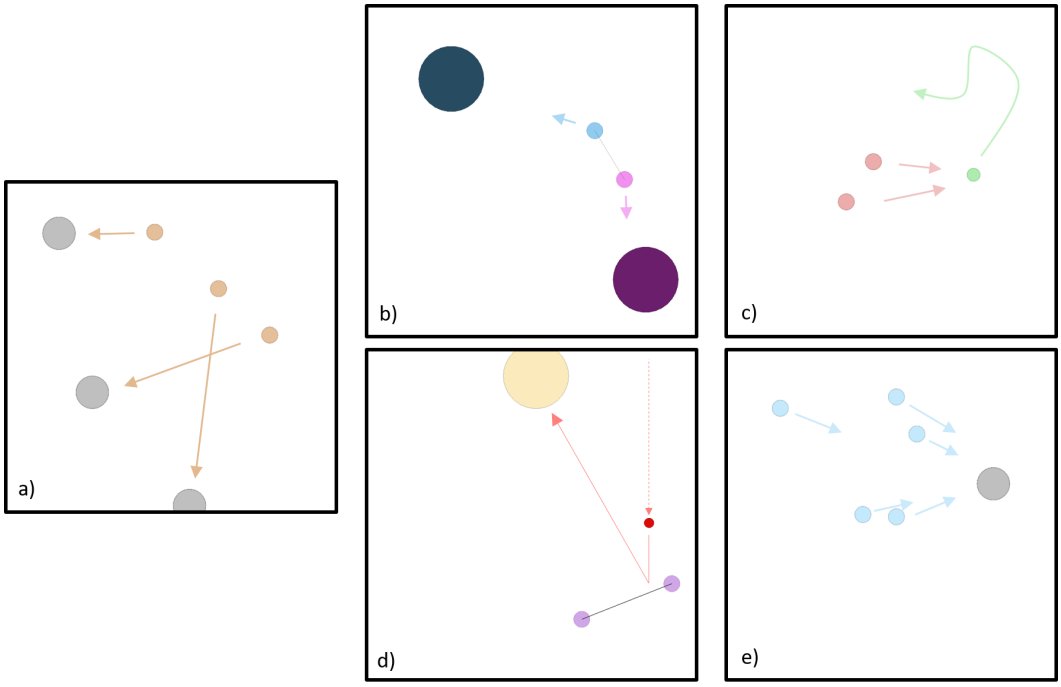


Figure 4: Multi-agent environments used in this work.

In all of our experiments, we use Adam optimizer [9] to perform the parameter update. All models (actors and critics) are parametrized with MLPs containing two hidden layers of 128 units each. We use the Rectified Linear Unit (ReLU) as activation function and layer normalization to stabilize the learning. We use a buffer-size of 10^6 entries and a batch-size of 1024. We collect 100 transitions by interacting with the environment for each learning update. For all tasks, we train the agents for 15,000 episodes of 100 steps. The scale of the exploration noise is kept constant for the first 7,500 episodes and then decreases linearly until the end of the 15,000 training episodes.