K-Level Policy Gradients for Multi-Agent Reinforcement Learning

Aryaman Reddi[1,2,*], **Gabriele Tiboni**[5], **Jan Peters**[1,2,3,4], **Carlo D'Eramo**[1,2,5]
[1]Department of Computer Science, TU Darmstadt, Germany
[2]Hessian Center for Artificial Intelligence (Hessian.ai), Germany
[3]German Research Center for AI (DFKI), Systems AI for Robot Learning, Germany
[4]Center for Cognitive Science, TU Darmstadt, Germany
[5]Center for Artificial Intelligence and Data Science, University of Würzburg, Germany

Abstract

Actor-critic algorithms for deep multi-agent reinforcement learning (MARL) typically employ a policy update that responds to the current strategies of other agents. While being straightforward, this approach does not account for the updates of other agents at the same update step, resulting in miscoordination. In this paper, we introduce the K-Level Policy Gradient (KPG), a method that recursively updates each agent against the updated policies of other agents, speeding up the discovery of effective coordinated policies. We theoretically prove that KPG with finite iterates achieves monotonic convergence to a local Nash equilibrium under certain conditions. We provide principled implementations of KPG by applying it to the deep MARL algorithms MAPPO, MADDPG, and FACMAC. Empirically, we demonstrate superior performance over existing deep MARL algorithms in StarCraft II and multi-agent MuJoCo.

1 Introduction

Deep multi-agent reinforcement learning (MARL) research has made strides towards solving practical problems such as cooperative robotics [1], transportation management [2] and network traffic optimization [3]. While deep RL algorithms have garnered impressive results in complex single-agent control problems [4, 5], multi-agent systems present unique challenges. Roadblocks in MARL research include exploding joint state-action spaces and non-stationarity due to concurrent learning [6, 7].

Another challenge caused by simultaneous learning in MARL is a lack of mutual consistency, as each agent update does not account for the updates of other agents at the same update step, leading to miscoordination. To enhance the learning dynamics in MARL systems, we propose K-level Policy Gradients (KPG), a method which can be applied to any existing multi-agent policy gradient algorithm which can utilize a centralized policy gradient update. Most game-theoretic frameworks assume mutual consistency between agents; that is, each player's beliefs are consistent with what other players will actually do [8]. In practical scenarios, agents must infer the strategies of others using modeling or recurrent reasoning. The cognitive hierarchies framework formalizes *k-level thinking* as an iterative decision process whereby agents respond to the strategies they believe other agents will take, in accordance with other agent's updated strategies based on the same recursive reasoning: 'I think that you think that I think...' [9]. The prior distribution with which an agent acts is typically defined as recursion level k=0. MARL algorithms that update against the k=0 strategies of other agents can therefore be considered level k=1. KPG enables agents to update

*Correspondence to <aryaman.reddi@tu-darmstadt.de>

18th European Workshop on Reinforcement Learning (EWRL 2025).

their policies recursively against the updated policies of other agents, allowing them to utilize higher orders of k-level reasoning. Aided with an enhanced notion of mutual consistency, multi-agent systems are therefore able to converge faster on coordination problems. The intuition behind this algorithm is inspired by human behavior: research on theory of mind suggests that humans often model the decision-making processes of other humans in order to successfully collaborate on complex tasks [10, 11].

Research in both on-policy and off-policy MARL methods have produced impressive results from the extension of single-agent RL methods. In this work, we demonstrate the effectiveness of centralized learning with KPG across several metrics in on-policy and off-policy settings. We first conduct a theoretical study of the algorithm in the idealized case, followed by a theorem that demonstrates the convergence of KPG with finite iterates to a local Nash equilibrium under reasonable conditions. We accompany our theoretical analysis with an illustrative example in a simple toy problem. Finally, we present a comparison of KPG against existing deep MARL algorithms in three complex multi-agent coordination environments: the multi-agent StarCraft II environment in JaxMARL [12] (SMAX), the StarCraft II micromanagement benchmark in PyMARL [13] (SMAC), and multi-agent MuJoCo (MAMuJoCo) [14]. Note that SMAX and SMAC have different dynamics and should be considered separate baselines. In order to demonstrate the effectiveness of KPG in deep MARL, we practically implement it on top of the competitive MARL algorithms MAPPO [15], FACMAC [14], and MADDPG [16], which we denote **K-MAPPO**, **K-FACMAC**, and **K-MADDPG** respectively.

2 Related works

Policy gradient methods in multi agent reinforcement learning Policy gradient methods in cooperative MARL often fall within the CTDE regime [17] by conditioning on global state information and centralized actions during training. Algorithms such as MADDPG [16] utilize a multi-agent deterministic policy gradient using a centralized critic to exhibit more robust behavior than decentralized DDPG agents. Foerster et al. [18] similarly propose COMA, which uses a counterfactual baseline term to alleviate the noise in their centralized gradients, a common problem in cooperative MARL leading to difficulties in credit assignment. Concerned with credit assignment noise, Du et al. [19] implement a framework that learns a separate proxy reward in order to discriminatingly credit agents in multi-agent actor-critic methods. Certain actor-critic MARL algorithms such as FACMAC[14] borrow from the rich literature on multi-agent value decomposition methods [20–25] which attempt to improve credit assignment by decomposing global value estimates into factored ones.

Yu et al. [15] conduct a comprehensive study examining the performance of MAPPO, a CTDE variant of PPO which conditions its advantage estimation on global state information. While the policy gradient methods mentioned have displayed impressive results, they suffer from instability caused by a lack of mutual consistency, since they all update agents without taking the simultaneous updates of other agents into account.

LOLA [26] comes the closest to our approach methodologically, as it utilizes higher-order gradient terms to incorporate the future opponent policy into the learning update of each agent. This might be considered a k=2 level update. COLA [27] addresses the high variance and difficult computation of higher-level LOLA iterates by minimizing a differentiable consistency measure, requiring only second-order derivatives. POLA [28] builds on LOLA further, reinterpreting it as a proximal operator by penalizing divergence over policy behavior. This helps mitigate LOLA's sensitivity to parameterizations. While LOLA, COLA, and POLA are evaluated in the deep setting with two-player reciprocity-based games, their viability in more complex cooperative environments has not yet been sufficiently examined. We show in this work that the informed updates of KPG are more stable than POLA and can be successfully applied to achieve SOTA performance in complex environments. M-FOS [29] scales opponent shaping to general-sum games; while using a similar paradigm to LOLA, we do not benchmark against M-FOS as we consider its meta-game formulation out of the scope of this paper.

K-level reasoning in reinforcement learning K-level thinking has proven useful in several multi-agent opponent modeling scenarios. Notably, the cognitive hierarchies framework [9] has been combined extensively with deep RL techniques for the training of self-driving vehicles which must cooperate within a heterogeneous population of peer vehicles [30–36]. Cui et al. [37] propose a synchronous hierarchical k-level reasoning method known as SyKLRBR which obtains SOTA performance in multi-agent Hanabi [38].

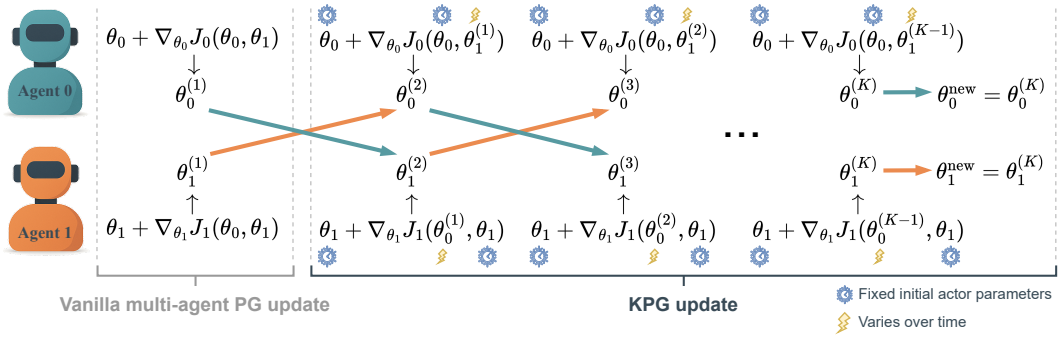


Figure 1: Schematic illustration of the K-level Policy Gradient (KPG) update for two agents.

Finally, Liu and Pavel [39] propose Level k gradient play, a recursive reasoning algorithm which stabilizes GAN training [40]. We extend their findings on the Semi-Proximal Point Method in this work to N-player, general sum games.

3 Preliminaries

We consider a multi-agent extension of a Markov decision process (MDP) [41] known as a Markov game [42], defined by a tuple $\mathcal{G} = \langle \mathcal{I}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, N, \iota \rangle$. $\mathcal{I} \equiv \{1, \ldots, N\}$ is the set of agents, $\mathcal{S}$ is the state space, $\mathcal{A} \equiv \times_{i \in \mathcal{I}} A_i$ is the joint action space of the agents. At each timestep, agent i samples an action $a_i \in A_i$ from policy $\pi_i(a_i|s)$ parameterized by $\boldsymbol{\theta}_i \in \mathbb{R}^{d_i}$. The joint action of all agents $\boldsymbol{a}$ from the joint policy $\boldsymbol{\pi}(.|s)$ determines the next state according to the joint state transition function $\mathcal{P}(s'|s, \boldsymbol{a})$. $\mathcal{R} \equiv \{R_1, \ldots, R_N\}$ are the set of agent reward functions. Each agent i has a learning rate η_i and receives rewards according to its reward function $R_i(s, \boldsymbol{a}, s')$. γ is the discount factor and ι is the initial state distribution. $\underset{max}{\lambda}(\boldsymbol{Z})$ and $\underset{min}{\lambda}(\boldsymbol{Z})$ refer to the maximum and minimum eigenvalues of a matrix $\boldsymbol{Z}$ respectively. Each agent aims to maximize its own discounted objective

$$J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}) = \underset{\tau \sim \iota, \pi_i, \boldsymbol{\pi}_{-i}, \mathcal{P}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t,i} \right], \tag{1}$$

where $\boldsymbol{\pi}_{-i}$ and $\boldsymbol{\theta}_{-i}$ refer to the other agent policies and parameters respectively, $r_{t,i}$ is the reward agent i receives at time t, and τ refers to the trajectories induced under the agent policies, ι, and $\mathcal{P}$.

4 K-level policy gradients

In practice, $\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i})$ can be estimated in a variety of ways, such as sample-based estimates of the policy gradient [16, 43] or using the performance difference lemma [44]. The optimization performed by agent i is a response to the action distribution of the other agents before they have made an update. The update for the initial parameters $\boldsymbol{\theta}_i^{(0)}$ for agent i results in a k=1 policy denoted $\boldsymbol{\theta}_i^{(1)}$, where the superscript denotes the degree of k-level thinking (we drop the superscript for $\boldsymbol{\theta}_i^{(0)}$ from now on for brevity when possible). We can write the k=1 policy gradient update as:

$$\boldsymbol{\theta}_i^{(1)} \leftarrow \boldsymbol{\theta}_i + \eta_i \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}) \ , \forall i \in \mathcal{I}. \tag{2}$$

The generalized update in (2) is the standard one used in multi-agent policy gradient methods. The k=1 policy obtained in this way does not account for the fact that the other agent policies were also updated in the same step. This update step can be taken once again from the *initial* parameters of each agent while considering the k=1 policies of the other agents, allowing us to reach recursion level k=2:

$$\boldsymbol{\theta}_i^{(2)} \leftarrow \boldsymbol{\theta}_i + \eta_i \nabla_{\boldsymbol{\theta}_i}, J_i\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \ , \forall i \in \mathcal{I}, \tag{3}$$

This procedure can be applied recursively K times, with each agent responding to the other agents at the previous k-level. Notably, the final K-level policy of each agent is still only *one* gradient step away from the initial actor parameters, since each recursive update is applied to the same initial actor parameters $\boldsymbol{\theta}$. **The k-level updates are not moving further in parameter space, but rather finding a gradient direction that is more consistent with the updates of the other agents.** We show the full recursive reasoning procedure in Algorithm 1, with a schematic illustration in Figure 1 in the case of two agents.

Algorithm 1 K-Level Policy Gradients (KPG)

Input: Initial actor parameters $\boldsymbol{\theta}_i$ & actor learning rates $\eta_i, \forall i \in \mathcal{I}$; recursive reasoning steps K
for each update step **do**
  for $k = 1$ **to** K **do**
    $\boldsymbol{\theta}_i^{(k)} \leftarrow \boldsymbol{\theta}_i + \eta_i \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k-1)}),\ \forall i \in \mathcal{I}$
  end for
  return $\boldsymbol{\theta}_i^{(K)}, \forall i \in \mathcal{I}$
end for

4.1 Theoretical analysis

In this section, we present a theoretical study which shows that under certain conditions, KPG converges monotonically to a local Nash equilibrium, even with finite K iterates. Firstly, we demonstrate how an unbiased, infinite application of Algorithm 1 leads to perfect anticipation of other agents' future strategies.

Assumption 4.1. *The gradient $\nabla_{\boldsymbol{\theta_i}} J_i\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ is L_i-Lipschitz with respect to $\boldsymbol{\theta}_{-i}$, i.e.*

$$\|\nabla_{\boldsymbol{\theta_i}} J_i\left(\boldsymbol{\theta}_{i,}, \boldsymbol{\theta}_{-i,1}\right) - \nabla_{\boldsymbol{\theta_i}} J_i\left(\boldsymbol{\theta}_{i,}, \boldsymbol{\theta}_{-i,2}\right)\| \leq L_i \|\boldsymbol{\theta}_{-i,1} - \boldsymbol{\theta}_{-i,2}\|, \forall \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}, \tag{4}$$

where $\boldsymbol{\theta}_{-i,1}$ and $\boldsymbol{\theta}_{-i,2}$ are two arbitrary points in the joint parameter space of the other agents. We define the maximum objective function Lipschitzness $L := \max_i\{L_i\}$ and the maximum agent learning rate $\eta := \max_i\{\eta_i\}$. We define also the maximum objective function gradient across all agent parameters and objective functions $\nabla_{max} := \max_{i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}} \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i})\|$.

Theorem 4.2. *Suppose Assumption 4.1 holds. We define the combined agent parameter vector $\boldsymbol{\theta} := [\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n]^T \in \mathbb{R}^{\sum_i d_i}$. Let $\boldsymbol{\theta}^{(k)}$ be the combined agent parameter vector after k steps of KPG reasoning. For k-level KPG, consecutive update steps are bounded:*

$$\|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^{(k-1)}\| \leq \eta(\eta L)^{k-1} n(n-1)^{k-1} \nabla_{max}. \tag{5}$$

Assume the maximum learning rate η satisfies $\eta < \frac{1}{L(n-1)}$. Then, the sequence $\{\boldsymbol{\theta}^{(k)}\}_{k=0}^{\infty}$ is a convergent sequence. Since $\boldsymbol{\theta}$ exists in a complete subspace of $\mathbb{R}^{\sum_i di}$, the convergent sequence $\{\boldsymbol{\theta}^{(k)}\}_{k=0}^{\infty}$ is Cauchy, i.e.,

$$\exists C \in \mathbb{N} : \forall \epsilon > 0, (a > b > C \implies \|\boldsymbol{\theta}^{(a)} - \boldsymbol{\theta}^{(b)}\| < \epsilon). \tag{6}$$

Since every Cauchy sequence has a limit, we denote the limit of $\{\boldsymbol{\theta}^{(k)}\}_{k=0}^{\infty}$ as $\lim_{k\to\infty} \boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(\infty)}$.

According to Theorem 4.2, applying the update in Algorithm 1 with $k{=}\infty$ defines the following implicit algorithm:

$$\boldsymbol{\theta}_i^{(\infty)} \leftarrow \boldsymbol{\theta}_i + \eta_i \nabla_{\boldsymbol{\theta}_i} J_i\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(\infty)}\right), \forall i \in \mathcal{I}, \tag{7}$$

which we denote the Generalized Semi-Proximal Point Method (GSPPM). The implication of the GSPPM is that the update of each agent responds exactly to the updated strategies of the other agents, maintaining mutual consistency. Liu and Pavel [39] similarly establish the SPPM in 2-player minimax games, and we extend their findings to N-player general sum games in this work.

Following from Theorem 4.2, we show that the convergence of GSPPM iterates in a non-convex non-concave strategy space can be analyzed via the game Jacobian around a local stationary point:

Theorem 4.3. *Let $\boldsymbol{\theta}^*$ be a stationary point in an N-player general sum game. This stationary point is a local Nash equilibrium, i.e. a point at which no agent's objective function has a non-zero gradient under a unilateral change in policy. Let $\boldsymbol{\eta}$ be a block matrix of the agent learning rates η_i. Let the components of the Hessian of each objective function at $\boldsymbol{\theta}^*$ be denoted*

$$\begin{pmatrix} \nabla^2_{\boldsymbol{\theta}_i\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i^*\boldsymbol{\theta}_{-i}^*) & \nabla^2_{\boldsymbol{\theta}_i\boldsymbol{\theta}_{-i}} J_i(\boldsymbol{\theta}_i^*\boldsymbol{\theta}_{-i}^*) \\ \nabla^2_{\boldsymbol{\theta}_{-i}\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i^*\boldsymbol{\theta}_{-i}^*) & \nabla^2_{\boldsymbol{\theta}_{-i}\boldsymbol{\theta}_{-i}} J_i(\boldsymbol{\theta}_i^*\boldsymbol{\theta}_{-i}^*) \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_i & \boldsymbol{B}_i \\ \boldsymbol{B}_i^\top & \boldsymbol{C}_i \end{pmatrix}.$$

Furthermore, let $\boldsymbol{A}$ be the diagonal block matrix of all $\boldsymbol{A}_i$ matrices, and $\boldsymbol{B}$ be the diagonal block matrix of all $\boldsymbol{B}_i$ matrices:

$$\boldsymbol{A} = \mathrm{diag}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_n), \quad \boldsymbol{B} = \mathrm{diag}(\boldsymbol{B}_1, \ldots, \boldsymbol{B}_n)$$

Let $\boldsymbol{D}$ be a complement-selection matrix for each set of agent parameters $\boldsymbol{\theta}_i$ such that

$$\boldsymbol{D}\boldsymbol{\theta} = [\boldsymbol{\theta}_{-1}, ..., \boldsymbol{\theta}_{-n}]^\top$$

Suppose $\eta < \frac{1}{L(n-1)}$ such that the GSPPM iterates $\{\boldsymbol{\theta}_t^{(k)}\}_{k=0}^{\infty}$ form a Cauchy sequence. Then, there exists a neighborhood $\mathcal{U} \in \mathbb{R}^{\sum_i d_i}$ around $\boldsymbol{\theta}^*$ such that if GSPPM starts in $\mathcal{U}$, the iterates $\{\boldsymbol{\theta}_t^{(k)}\}_{k=0}^{\infty}$ satisfy:

$$\|\boldsymbol{\theta}^{(\infty)} - \boldsymbol{\theta}^*\| \leq \frac{\lambda_{max}(I + \boldsymbol{\eta}\boldsymbol{A})^2}{\lambda_{min}(I - \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|. \tag{8}$$

Moreover, for any $\boldsymbol{\eta}$ satisfying

$$\frac{\lambda_{max}(I + \boldsymbol{\eta}\boldsymbol{A})^2}{\lambda_{min}(I - \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2} < 1, \tag{9}$$

the iterates converge asymptotically to $\boldsymbol{\theta}^*$.

Hence, GSPPM iterates reach an ϵ-Nash equilibrium, i.e., an arbitrarily close ϵ-bound around the stationary point [45], under the conditions of Theorems 4.2 and 4.3.

Theorem 4.4. *Suppose the conditions of Theorem 4.3 apply. Then, the distance of finite iterates $\{\boldsymbol{\theta}_t^{(k)}\}$ generated by KPG satisfy*

$$\|\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^*\|^2 \leq \left(\lambda_{max}(I + \boldsymbol{\eta}\boldsymbol{A})^2 + 2\lambda_{max}(I + \boldsymbol{\eta}\boldsymbol{A})\lambda_{max}(\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})\right)\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2 \tag{10}$$

$$+ 2\lambda_{max}(I + \boldsymbol{\eta}\boldsymbol{A})\lambda_{max}(\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})\left(\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|\nabla_{max}\right) \tag{11}$$

$$+ \lambda_{max}(\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2\|\boldsymbol{\theta}^{(k-1)} - \boldsymbol{\theta}^*\|^2. \tag{12}$$

Moreover, for any $\boldsymbol{\eta}$ satisfying

$$\lambda_{max}(\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2 < 1, \tag{13}$$

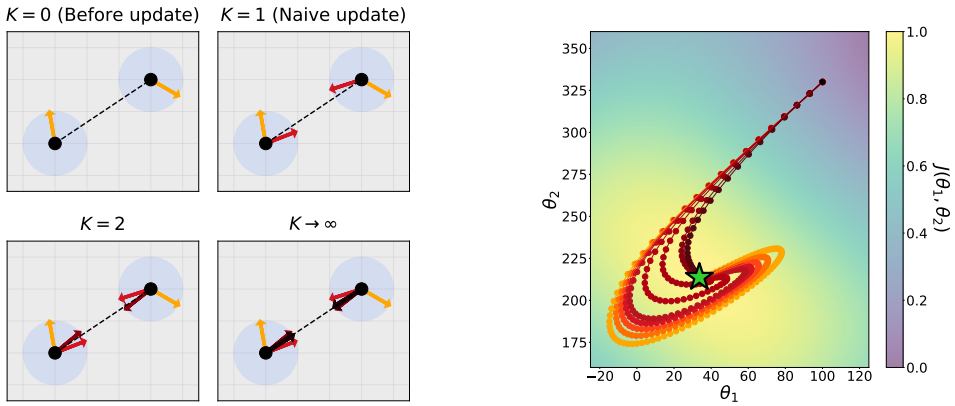
the KPG iterates converge asymptotically to $\boldsymbol{\theta}^*$.

Thus, KPG iterates with finite k reach an ϵ-Nash equilibrium under the conditions of Theorems 4.2, 4.3 and 4.4.

4.2 Illustrative example

We illustrate the effectiveness of KPG by studying a simple continuous cooperative game with two point agents taking continuous actions in a 2D space. The agents have one parameter each and produce a one-dimensional action (the angle of their next move).

The highest reward is achieved when the chosen direction of each agent points towards the *future* location of the other agent. Assuming the agents move kinematically and have no momentum, this game has a known optimal solution: moving towards each other in a straight line. Therefore, the



(a) Agents taking continuous actions with increasing k-levels (darkening arrows) converging on the optimal actions (dashed line) within a single update step.

(b) Gradient ascent to the optimal parameters (green star) with increasing k-levels (darker colors for higher k values).

Figure 2: An illustrative continuous cooperative game with two point agents using k-level policy gradient ascent.

two agents should cooperate to meet as quickly as possible (see Appendix B for details). However, the naive policy update for this game is for each agent to choose its next action to intercept with the *previous* action of the other agent. The top two figures of Figure 2a illustrate this problem: each agent's new action (red arrows) points towards the destination of the other agent under the other agent's old policy (yellow arrows). Hence, the naive update leads to a lack of mutual consistency since each agent does not consider the other agent's update.

Figure 2a bottom left shows the policy update after 2 levels of recursion: now the agents update to intercept the other agent *after* the other agent's naive policy update, resulting in better coordination. As the number of recursions increases, the policies converge on an ϵ-Nash bound of the optimal solution.

Figure 2b shows the progression of the agent parameters with gradient ascent and momentum; we use the true gradient and objective function $J(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$. The benefits of KPG recursion are evinced by the fact that increasing k-levels (darker points) exhibit monotonic convergence to the optimal parameters $\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*$ at each update step, as supported by Theorem 4.4. Figure 3 shows the distance from the Nash equilibrium for one update step with KPG near the stationary point in Figure 2b.

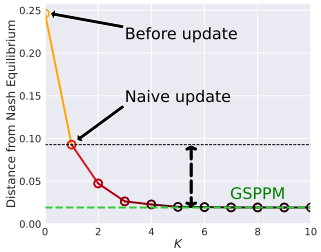


Figure 3: Monotonic convergence to GSPPM for one update step with KPG.

4.3 Algorithmic implementation of the k-level policy gradient

The KPG formulation is based on the recursive computation of the k-level policy gradient $\nabla_{\boldsymbol{\theta}_i} J(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)})$. However, unlike our illustrative example in Sec. 4.2, this quantity can generally only be estimated, e.g. through environment interactions. We here propose two implementations of KPG on top of existing algorithms that bridge the gap between its theoretical formulation and its application for deep RL problems. In particular, we show how the k-level policy gradient can be estimated using *only* the data collected by the initial joint policy $\boldsymbol{\pi}^{(0)}$ before the update, hence avoiding the need for additional environment interactions during the k-level reasoning process.

K-MAPPO (on-policy). We integrate KPG into the state-of-the-art algorithm MAPPO [15] by building on the surrogate objective provided by PPO [46]. MAPPO utilizes a surrogate loss with state-conditioned advantage estimation and an IS ratio, $r_{s,\boldsymbol{a}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}) = \frac{\pi_i(a_i|s)}{\pi_i^{(0)}(a_i|s)}$ (where π_i is the policy being updated), which does not consider the update of the other agents. We extend this ratio to

include the updated policies:

$$r_{s,\boldsymbol{a}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}) = \frac{\pi_i(a_i|s) \cdot \boldsymbol{\pi}_{-i}^{(k)}(\boldsymbol{a}_{-i}|s)}{\boldsymbol{\pi}^{(0)}(\boldsymbol{a}|s)} = \frac{\pi_i(a_i|s)}{\pi_i^{(0)}(a_i|s)} \cdot \frac{\boldsymbol{\pi}_{-i}^{(k)}(\boldsymbol{a}_{-i}|s)}{\boldsymbol{\pi}_{-i}^{(0)}(\boldsymbol{a}_{-i}|s)}. \tag{14}$$

In turn, this allows us to estimate the k-level policy gradient for each agent i as:

$$\begin{aligned} &\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}) \approx \nabla_{\boldsymbol{\theta}_i} \mathcal{L}_i^{\text{K-MAPPO}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}) = \\ &= \mathbb{E}_{(s,\boldsymbol{a})\sim\boldsymbol{\pi}^{(0)}} \left[\nabla_{\boldsymbol{\theta}_i} \min\left(r_{s,\boldsymbol{a}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}), \text{clip}(r_{s,\boldsymbol{a}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}), 1-\epsilon, 1+\epsilon) \right) \cdot A_i^{\boldsymbol{\pi}^{(0)}}(s, \boldsymbol{a}) \right], \end{aligned} \tag{15}$$

where $A_i^{\boldsymbol{\pi}^{(0)}}(s, \boldsymbol{a})$ is the advantage function of agent i under the 0-level policies. Intuitively, the k-level IS ratio in Eq. 14 can be seen as weighting the original surrogate loss; if the joint probability ratio of the other agent updates exceeds 1, it increases the magnitude of the gradient that would have been taken without considering their updates, and vice-versa. A comprehensive derivation of the K-MAPPO surrogate objective is available in the Appendix.

K-MADDPG & K-FACMAC (off-policy). In the case of off-policy approaches such as MADDPG [16], we can estimate the k-level policy gradient by leveraging an explicitly learned centralized Q-function. Analogously to how MADDPG relies on the gradient of the learned Q-function to execute a policy improvement step, we propose the following gradient estimate:

$$\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}) \approx \nabla_{\boldsymbol{\theta}_i} \mathcal{L}_i^{\text{K-MADDPG}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(k)}) = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D} \\ a_i, \boldsymbol{a}_{-i}\sim\pi_i, \boldsymbol{\pi}_{-i}^k}} \left[\nabla_{\boldsymbol{\theta}_i} Q_i^{\pi_i^{(0)}}(s, a_i, \boldsymbol{a}_{-i}) \right], \tag{16}$$

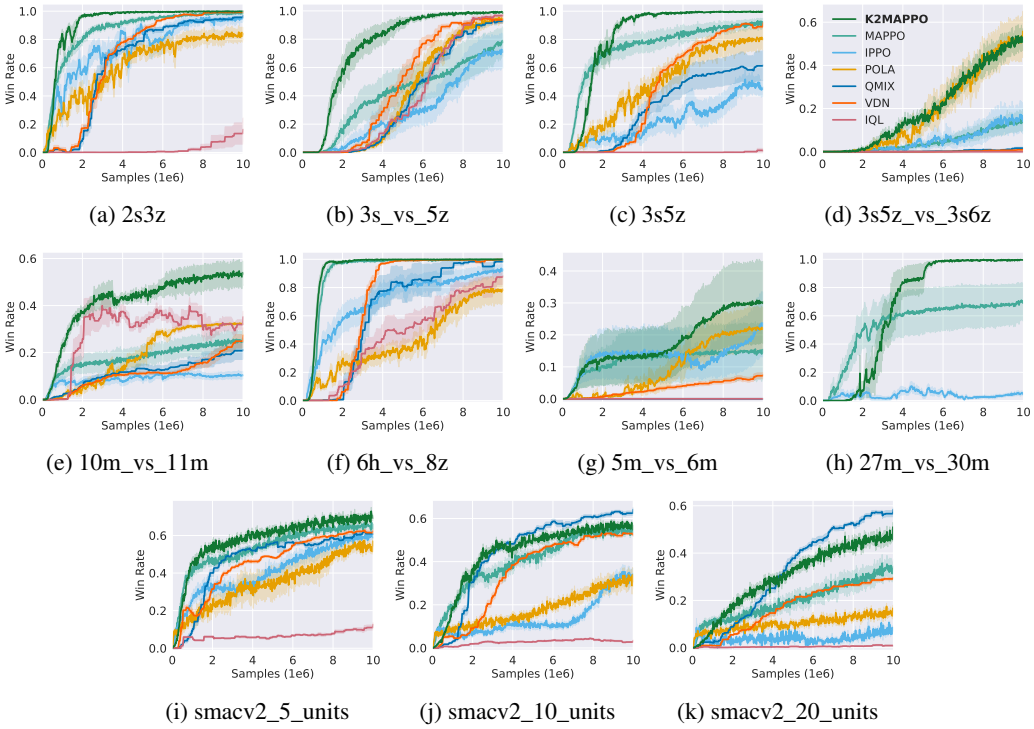
where $\mathcal{D}$ is a buffer of stored transitions and $Q_i^{\boldsymbol{\pi}^{(0)}}$ is agent i's action value function under the joint k=0 level policy. In other words, we adjust the policy improvement step at each k-level reasoning step by resampling the actions of the other agents from the newly updated policies $\boldsymbol{\pi}_{-i}^{(k)}$, whereas $\pi_i = \pi_i^{(0)}$ remains fixed. We also provide a theoretical motivation of this approximation in the Appendix. Note that the same principle can be applied to more sophisticated off-policy algorithms such as FACMAC.

5 Experimental results

In this section, we demonstrate the effectiveness of KPG in deep MARL when applied to actor-critic algorithms. In the SMAX environment, we benchmark $K2$-MAPPO (MAPPO with k=2 levels of recursion) against several actor-critic and value-based algorithms. Due to the highly parallelized nature of SMAX [12], MAPPO performs well in terms of performance and wall-clock time, making it a good candidate to demonstrate the benefits of recursive reasoning. In the SMAC and MAMuJoCo environments, we demonstrate the effectiveness of $K2$-FACMAC, as FACMAC achieves SOTA performance among off-policy multi-agent methods. In our experiments, we find that k=2 levels of policy recursion achieves most of the attainable performance benefits, with higher levels of recursion yielding small further improvements.

In addition to MAPPO and FACMAC, we benchmark against QMIX/COMIX due to their prevalence in MARL research and competitive performance, and POLA due to its use of higher-order opponent updates. We implement Outer-POLA with generalized advantage estimation as seen in Zhao et al. [28]. In SMAX, we also benchmark against VDN due to its relevance as a centralized value-based method and IPPO/IQL as examples of decentralized training. In SMAC and MAMuJoCo, we additionally benchmark against MADDPG with a centralized and monolithic critic. Note that our results may appear slightly different to previous works with SMAC/SMAX [23, 22] since previous works choose to report median instead of mean win rates, which reduces the impact of failing seeds, especially on certain difficult maps like Corridor.

K2-MAPPO outperforms related baselines in SMAX Figure 4 compares the test win rate of $K2$-MAPPO against related baselines on 11 SMAX maps (8 SMACv1-based and 3 SMACv2-based). Notably, $K2$-MAPPO performs equal or better than other baselines on 9 out of 11 maps, and is the only algorithm to solve 3s5z and 27m_vs_30m with 100% accuracy. $K2$-MAPPO also maintains a consistent advantage over MAPPO, demonstrating the viability of recursive reasoning. The effects

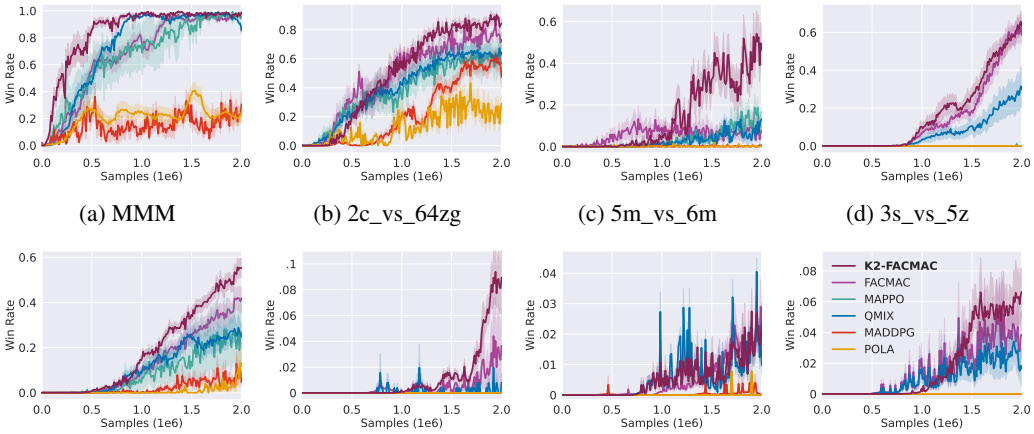


(a) 2s3z (b) 3s_vs_5z (c) 3s5z (d) 3s5z_vs_3s6z

(e) 10m_vs_11m (f) 6h_vs_8z (g) 5m_vs_6m (h) 27m_vs_30m

(i) smacv2_5_units (j) smacv2_10_units (k) smacv2_20_units

Figure 4: Mean win rate and standard error of $K2$-MAPPO and baselines on SMAX maps across 10 seeds. Note that 27m_vs_30m is very large and could only be benchmarked with PPO-based methods due to computational constraints.



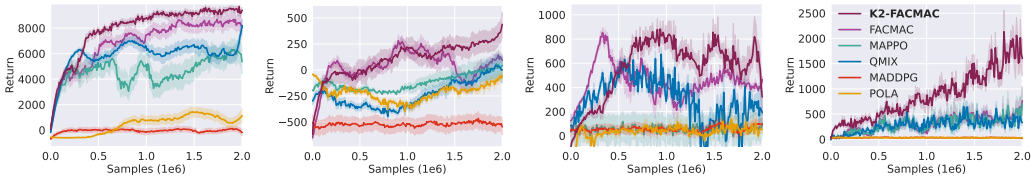
(a) MMM (b) 2c_vs_64zg (c) 5m_vs_6m (d) 3s_vs_5z

(e) MMM2 (f) Corridor (g) 6h_vs_8z (h) 3s5z_vs_3s6z

Figure 5: Mean win rate and standard error of $K2$-FACMAC and baselines on SMAC maps across 10 seeds.

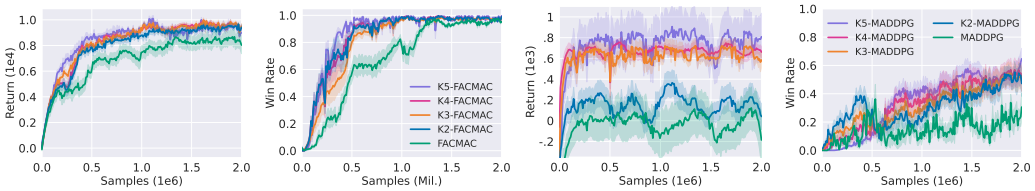
of recursive reasoning are seen in higher overall performance as well as faster convergence, as $K2$-MAPPO often reaches its maximum performance sooner than other methods. Despite utilizing opponent-shaping, POLA failed to match the performance of $K2$-MAPPO in every map except 3s5z_vs_3s6z. It is likely that the typical formulation of POLA, which performs well on 2-player reciprocity-based games, does not generalize well to more complex environments with many agents.

K2-FACMAC outperforms related baselines in SMAC and MAMuJoCo Figure 5 compares the win rate of $K2$-FACMAC and related baselines across 8 SMAC maps (Figure 5), with a focus on



(a) HalfCheetah-2x3 (b) HalfCheetah-1p1 (c) Ant-2x4 (d) Walker-2x3

Figure 6: Mean performance and standard error of $K2$-FACMAC and baselines on four Multi-Agent MuJoCo environments across 10 seeds.



(a) K-FACMAC (HalfCheetah-2x3) (b) K-FACMAC (MMM) (c) K-MADDPG (HalfCheetah-2x3) (d) K-MADDPG (MMM)

Figure 7: Ablations for higher K-level FACMAC and MADDPG on MAMuJoCo (HalfCheetah-2x3) and SMAC (MMM).

Hard and Super Hard maps. $K2$-FACMAC achieves higher or equal final success rates compared to baselines in every scenario. Using recursive reasoning, also achieves its maximum win rate sooner in maps which show equal final win rates (MMM, 3s_vs_5z, 6h_vs_8z). $K2$-FACMAC particularly stands out in Corridor (Figure 5f) and 3s5z_vs_3s6z (Figure 5h), two notoriously difficult SMAC maps in which $K2$-FACMAC is the only algorithm to surpass a 5% success rate. Note that MAPPO performs much worse on SMAC maps than SMAX maps due to SMAC being CPU-based and less parallelizable, resulting in far less sample availability (2e6 for each map rather than 1e7 in SMAX).

Figure 6 compares the performance of $K2$-FACMAC against the baselines on four selected MAMuJoCo environments. In all four environments, $K2$-FACMAC achieves superior performance at the end of training and tends to reach its peak performance earlier. In Ant 2x4 (Figure 6c), learning a solution is difficult due to the asymmetric positioning of the agents which control opposing diagonal halves of a 4-legged Ant agent. When observing the final learned policies, only $K2$-FACMAC, FACMAC, and COMIX demonstrate a positive improvement over the initial policy. $K2$-FACMAC and FACMAC both achieve the same level of maximum performance during training, with $K2$-FACMAC maintaining an advantage despite the unstable learning dynamics. $K2$-FACMAC also achieves the *only* policy in Walker 2x3 (Figure 6d) which adequately solves the task, obtaining SOTA performance on this very difficult locomotion benchmark.

k=2 level KPG attains most of the available benefits Figure 7 compares the test performance of higher-level KPG applied to FACMAC and MADDPG upto k=5 in HalfCheetah-2x3 (MAMuJoCo) and MMM (SMAC). Note the performance increases over nominal MADDPG and FACMAC when KPG is applied, supporting the claim that our approach can reliably improve any centralized policy gradient algorithm. While the wall clock time increases fairly linearly as the number of policy recursions increases (Table 2 in Appendix F), the additional benefits of increasing K saturate quickly after $k = 2$. Despite this, higher k-levels do exhibit monotonic performance increases and remain stable. Interestingly, Figure 7c demonstrates a situation where higher-level recursions above $k = 2$ are necessary for the algorithm to find a better policy mode.

6 Conclusion and discussion

We present K-Level Policy Gradients, an approach which harnesses the game theoretical paradigm of k-level thinking to improve the convergence of multi-agent policy gradient algorithms. We show that KPG in the limit (the Generalized Semi-Proximal Point Method) converges to a fixed

point for N-player, general sum games. Furthermore, we show that our algorithm reaches an ϵ-Nash equilibrium with a finite sequence of iterates under certain conditions. Our empirical results demonstrate KPG's ability to outperform existing MARL algorithms in the deep setting when applied to MAPPO, FACMAC, and MADDPG, enabling a further stepping stone to real-world applications of MARL. Future work in this area will consist of an analysis of KPG in the competitive setting and the applications of k-level thinking to other MARL approaches such as message passing.

Limitations. The biggest practical limitation of KPG is its computational expense, since the number of backpropagation steps at each update scales linearly with k. In the future, we hope to reduce the computational expense of KPG using policy estimation methods such as opponent-policy modeling [47] or parameter-based value estimation [48].

Acknowledgments

This work was funded by the German Federal Ministry of Education and Research (BMBF) (Project: 01IS22078). This work was also funded by Hessian.ai through the project 'The Third Wave of Artificial Intelligence – 3AI' by the Ministry for Science and Arts of the state of Hessen.

Calculations for this research were conducted on the Lichtenberg high-performance cluster of TU Darmstadt, and the Intelligent Autonomous Systems (IAS) cluster at TU Darmstadt. The authors also gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) under the NHR project b187cb. NHR funding is provided by federal and Bavarian state authorities. NHR@FAU hardware is partially funded by the German Research Foundation (DFG) -440719683.

References

[1] Zool Hilmi Ismail, Nohaidda Sariff, and E Gorrostieta Hurtado. A survey and analysis of cooperative multi-agent robot systems: challenges and directions. *Applications of Mobile Robots*, 5:8–14, 2018.

[2] Ammar Haydari and Yasin Yılmaz. Deep reinforcement learning for intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(1):11–32, 2020.

[3] Yue Pi, Wang Zhang, Yong Zhang, Hairong Huang, Baoquan Rao, Yulong Ding, and Shuanghua Yang. Applications of multi-agent deep reinforcement learning communication in network management: A survey. *arXiv preprint arXiv:2407.17030*, 2024.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[5] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review of Control, Robotics, and Autonomous Systems*, 8, 2024.

[6] Hui Li, X. Liao, and L. Carin. Multi-task reinforcement learning in partially observable stochastic environments. *J. Mach. Learn. Res.*, 10:1131–1186, 2009. doi: 10.5555/1577069.1577109.

[7] W. Barfuss and R. Mann. Modeling the effects of environmental and perceptual uncertainty using deterministic reinforcement learning dynamics with partial observability. *Physical review. E*, 105 3-1:034409, 2021. doi: 10.1103/PhysRevE.105.034409.

[8] D Robertson. General theory of employment, interest and money. *QJ Econ*, 51:791–795, 1936.

[9] Colin F Camerer, Teck-Hua Ho, and Juin-Kuan Chong. A cognitive hierarchy model of games. *The Quarterly Journal of Economics*, 119(3):861–898, 2004.

[10] Nikolos Gurney, Stacy Marsella, Volkan Ustun, and David V Pynadath. Operationalizing theories of theory of mind: a survey. In *AAAI Fall Symposium*, pages 3–20. Springer, 2021.

[11] Sara M Schaafsma, Donald W Pfaff, Robert P Spunt, and Ralph Adolphs. Deconstructing and reconstructing theory of mind. *Trends in cognitive sciences*, 19(2):65–72, 2015.

[12] Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

[13] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[14] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

[15] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

[16] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[17] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[18] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[19] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[20] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[21] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

[22] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020.

[23] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

[24] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020.

[25] Ling Pan, Tabish Rashid, Bei Peng, Longbo Huang, and Shimon Whiteson. Softmax with regularization: Better value estimation in multi-agent reinforcement learning. *arXiv preprint arXiv:2103.11883*, 2021.

[26] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[27] Timon Willi, Alistair Hp Letcher, Johannes Treutlein, and Jakob Foerster. Cola: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, pages 23804–23831. PMLR, 2022.

[28] Stephen Zhao, Chris Lu, Roger B Grosse, and Jakob Foerster. Proximal learning with opponent-learning awareness. *Advances in Neural Information Processing Systems*, 35:26324–26336, 2022.

[29] Akbir Khan, Timon Willi, Newton Kwan, Andrea Tacchetti, Chris Lu, Edward Grefenstette, Tim Rocktäschel, and Jakob Foerster. Scaling opponent shaping to high dimensional games. *arXiv preprint arXiv:2312.12568*, 2023.

[30] Nan Li, Dave Oyler, Mengxuan Zhang, Yildiray Yildiz, Anouck Girard, and Ilya Kolmanovsky. Hierarchical reasoning game theory based approach for evaluation and testing of autonomous vehicle control systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 727–733. IEEE, 2016.

[31] Mario Garzón and Anne Spalanzani. Game theoretic decision making for autonomous vehicles' merge manoeuvre in high traffic scenarios. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3448–3453. IEEE, 2019.

[32] Maxime Bouton, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Mykel J Kochenderfer. Reinforcement learning with iterative reasoning for merging in dense traffic. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2020.

[33] Berat Mert Albaba and Yildiray Yildiz. Driver modeling through deep reinforcement learning and behavioral game theory. *IEEE Transactions on Control Systems Technology*, 30(2):885–892, 2021.

[34] Xinpeng Wang, Songan Zhang, and Huei Peng. Comprehensive safety evaluation of highly automated vehicles at the roundabout scenario. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):20873–20888, 2022.

[35] Shahab Karimi, Arash Karimi, and Ardalan Vahidi. Level-k reasoning, deep reinforcement learning, and monte carlo decision process for fast and safe automated lane change and speed management. *IEEE Transactions on Intelligent Vehicles*, 8(6):3556–3571, 2023.

[36] Siyu Dai, Sangjae Bae, and David Isele. Game theoretic decision making by actively learning human intentions applied on autonomous driving. *arXiv preprint arXiv:2301.09178*, 2023.

[37] Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. K-level reasoning for zero-shot coordination in hanabi. *Advances in Neural Information Processing Systems*, 34:8215–8228, 2021.

[38] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

[39] Zichu Liu and Lacra Pavel. Recursive reasoning in minimax games: A level k gradient play method. *Advances in Neural Information Processing Systems*, 35:16903–16917, 2022.

[40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[41] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

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

[43] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.04402*, 2021.

[44] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the nineteenth international conference on machine learning*, pages 267–274, 2002.

[45] Drew Fudenberg and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[47] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *International conference on machine learning*, pages 1804–1813. PMLR, 2016.

[48] Francesco Faccio, Louis Kirsch, and Jürgen Schmidhuber. Parameter-based value functions. *arXiv preprint arXiv:2006.09226*, 2020.

[49] Yulai Zhao, Zhuoran Yang, Zhaoran Wang, and Jason D Lee. Local optimization achieves global optimality in multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 42200–42226. PMLR, 2023.

[50] Yueheng Li, Guangming Xie, and Zongqing Lu. Difference advantage estimation for multi-agent policy gradients. In *International Conference on Machine Learning*, pages 13066–13085. PMLR, 2022.

[51] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[52] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134:19–67, 2005.

A Proofs

A.1 Proof of Theorem 4.2

Recall Assumption 4.1 and Algorithm 1. We first analyze the pattern in successive updates of $\boldsymbol{\theta}$ as k increases.

Consider level $K = 1$ KPG:

$$\boldsymbol{\theta}_{t,i}^{k=1} = \boldsymbol{\theta}_{t,i} + \eta_i \nabla_{\boldsymbol{\theta}_i} J(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}) \quad \forall i \in \mathcal{I}. \tag{17}$$

The jump between $\boldsymbol{\theta}_t^{k=1}$ and $\boldsymbol{\theta}_t$ is

$$\|\boldsymbol{\theta}_{t,i}^{k=1} - \boldsymbol{\theta}_{t,i}\| = \eta_i \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i})\| \quad \forall i \in \mathcal{I}. \tag{18}$$

Thus, for all agents

$$\begin{aligned}
\|\boldsymbol{\theta}_t^{k=1} - \boldsymbol{\theta}_t\| &\leq \sum_{i=1}^{n} \|\boldsymbol{\theta}_{t,i}^{k=1} - \boldsymbol{\theta}_{t,i}\| = \sum_{i=1}^{n} \eta_i \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i})\| \\
&\leq \eta \sum_{i=1}^{n} \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i})\| \\
&\leq \eta n \max_i \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i})\| \\
&\leq \eta n \nabla_{\max}.
\end{aligned} \tag{19}$$

Now consider level $K = 2$ KPG:

$$\boldsymbol{\theta}_{t,i}^{k=2} = \boldsymbol{\theta}_{t,i} + \eta_i \nabla_{\boldsymbol{\theta}_i} J(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}^{k=1}) \quad \forall i \in \mathcal{I}. \tag{20}$$

The jump between $\boldsymbol{\theta}_t^{k=2}$ and $\boldsymbol{\theta}_t^{k=1}$ is

$$\begin{aligned}
\|\boldsymbol{\theta}_{t,i}^{k=2} - \boldsymbol{\theta}_{t,i}^{k=1}\| &= \|\boldsymbol{\theta}_{\boldsymbol{t,i}} + \eta_i \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}^{k=1}) - \boldsymbol{\theta}_{\boldsymbol{t,i}} - \eta_i \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i})\| \\
&= \eta_i \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}^{k=1}) - \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i})\| \\
&\leq \eta_i L_i \|\boldsymbol{\theta}_{t,-i}^{k=1} - \boldsymbol{\theta}_{t,-i}\| \quad \forall i \in \mathcal{I}.
\end{aligned} \tag{21}$$

Thus, for all agents

$$\begin{aligned}
\|\boldsymbol{\theta}_t^{k=2} - \boldsymbol{\theta}_t^{k=1}\| &\leq \sum_{i=1}^{n} \|\boldsymbol{\theta}_{t,i}^{k=2} - \boldsymbol{\theta}_{t,i}^{k=1}\| \\
&\leq \sum_{i=1}^{n} \eta_i L_i \|\boldsymbol{\theta}_{t,-i}^{k=1} - \boldsymbol{\theta}_{t,-i}\| \\
&\leq \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \|\boldsymbol{\theta}_{t,j}^{k=1} - \boldsymbol{\theta}_{t,j}\| \\
&= \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \eta_j \|\nabla_{\boldsymbol{\theta}_j} J_j(\boldsymbol{\theta}_{t,j}, \boldsymbol{\theta}_{t,-j})\| \\
&\leq \sum_{i=1}^{n} \eta_i L_i (n-1) \eta \nabla_{max} \\
&\leq \eta^2 L n (n-1) \nabla_{max}.
\end{aligned} \tag{22}$$

Now consider level $K = 3$ KPG:

$$\boldsymbol{\theta}_{t,i}^{k=3} = \boldsymbol{\theta}_{t,i} + \eta_i \nabla_{\boldsymbol{\theta}_i} J(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}^{k=2}) \quad \forall i \in \mathcal{I}. \tag{23}$$

The jump between $\boldsymbol{\theta}_t^{k=3}$ and $\boldsymbol{\theta}_t^{k=2}$ is

$$\begin{aligned} \|\boldsymbol{\theta}_{t,i}^{k=3} - \boldsymbol{\theta}_{t,i}^{k=2}\| &= \eta_i \|\nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}^{k=2}) - \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}, \boldsymbol{\theta}_{t,-i}^{k=1})\| \\ &\leq \eta_i L_i \|\boldsymbol{\theta}_{t,-i}^{k=2} - \boldsymbol{\theta}_{t,-i}^{k=1}\| \quad \forall i \in \mathcal{I}. \end{aligned} \tag{24}$$

Thus, for all agents

$$\begin{aligned} \|\boldsymbol{\theta}_t^{k=3} - \boldsymbol{\theta}_t^{k=2}\| &\leq \sum_{i=1}^{n} \|\boldsymbol{\theta}_{t,i}^{k=3} - \boldsymbol{\theta}_{t,i}^{k=2}\| \\ &\leq \sum_{i=1}^{n} \eta_i L_i \|\boldsymbol{\theta}_{t,i}^{k=2} - \boldsymbol{\theta}_{t,i}^{k=1}\| \\ &\leq \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \|\boldsymbol{\theta}_{t,j}^{k=2} - \boldsymbol{\theta}_{t,j}^{k=1}\| \\ &\leq \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \eta_j L_j \|\boldsymbol{\theta}_{t,-j}^{k=1} - \boldsymbol{\theta}_{t,-j}\| \\ &\leq \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \eta_j L_j \sum_{l \neq j} \|\boldsymbol{\theta}_{t,m}^{k=1} - \boldsymbol{\theta}_{t,m}\| \\ &= \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \eta_j L_j \sum_{l \neq j} \eta_m \|\nabla_{\boldsymbol{\theta}_m} J_m(\boldsymbol{\theta}_{t,m}, \boldsymbol{\theta}_{t,-m})\| \\ &= \sum_{i=1}^{n} \eta_i L_i \sum_{j \neq i} \eta_j L_j \sum_{l \neq j} \eta_m \|\nabla_{\boldsymbol{\theta}_m} J_m(\boldsymbol{\theta}_{t,m}, \boldsymbol{\theta}_{t,-m})\| \\ &\leq \eta^3 L^2 n(n-1)^2 \nabla_{max}. \end{aligned} \tag{25}$$

We see by induction that any consecutive states during the $K - level$ procedure are bounded by

$$\|\boldsymbol{\theta}_t^k - \boldsymbol{\theta}_t^{k-1}\| \leq \eta(\eta L)^{k-1} n(n-1)^{k-1} \nabla_{max}. \tag{26}$$

Let $\eta < \frac{1}{L(n-1)}$ such that the difference between between two K-steps is a contraction. Consider the difference $\|\boldsymbol{\theta}_t^a - \boldsymbol{\theta}_t^b\|$, where $a > b > 0$:

$$
\begin{aligned}
\|\boldsymbol{\theta}_t^{k=a} - \boldsymbol{\theta}_t^{k=b}\| &= \| \sum_{j=b+1}^{a} \left(\boldsymbol{\theta}_t^{k=j} - \boldsymbol{\theta}_t^{k=j-1}\right) \| \\
&\leq \sum_{j=b+1}^{a} \|\boldsymbol{\theta}_t^{k=j} - \boldsymbol{\theta}_t^{k=j-1}\| \\
&\leq \sum_{j=b+1}^{a} \eta(\eta L)^{j-1} n(n-1)^{j-1} \nabla_{max} \\
&\leq n\nabla_{max} \left(\sum_{j=b+1}^{a} (n-1)^{j-1} L^{j-1} \right) \eta \left(\sum_{j=b+1}^{a} \eta^{j-1} \right) \\
&\leq n\nabla_{max} \left(\sum_{j=b+1}^{a} (n-1)^{j-1} L^{j-1} \right) \eta \left(\sum_{j=b+1}^{a} \eta^{b-1} \right) \approx \mathcal{O}(\eta^b).
\end{aligned} \tag{27}
$$

Thus, for any $\epsilon > 0$, we can solve for b such that $\eta(\eta L)^{k-1} n(n-1)^{k-1} \nabla_{max} < \epsilon$, or

$$
\exists N \in \mathbb{N} : \forall \epsilon > 0, (a > b > N \implies \|\boldsymbol{\theta}^a - \boldsymbol{\theta}^b\| < \epsilon). \tag{28}
$$

Hence $\{\boldsymbol{\theta}_t^k\}_{k=0}^{\infty}$ is a Cauchy sequence. Since $\boldsymbol{\theta}_t$ lies in a complete subspace of $\mathbb{R}^{\sum_i d_i}$, the Cauchy sequence has a limit: $\lim_{k\to\infty} \boldsymbol{\theta}_t^k = \boldsymbol{\theta}_t^{\infty}$. □

A.2 Proof of Theorem 4.3

Let us define $\hat{\boldsymbol{\theta}}_{t,i} = \boldsymbol{\theta}_{t,i} - \boldsymbol{\theta}_i^*$ and $\hat{\boldsymbol{\theta}}_t = [\hat{\boldsymbol{\theta}}_{t,1}, ...\hat{\boldsymbol{\theta}}_{t,n}]^T$ for all $i \in \mathcal{I}$. It follows by linearizing the system about the stationary point $\boldsymbol{\theta}^*$,

$$
\begin{aligned}
\hat{\boldsymbol{\theta}}_{t+1,i} &= \boldsymbol{\theta}_{t,i} + \eta_i \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i \boldsymbol{\theta}_{-i}) - \boldsymbol{\theta}^* \\
&\approx \left(I + \eta_i \nabla^2_{\boldsymbol{\theta}_i,\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_i^*, \boldsymbol{\theta}_{-i}^*), \eta_i \nabla^2_{\boldsymbol{\theta}_i,\boldsymbol{\theta}_{-i}} J_i(\boldsymbol{\theta}_i^*, \boldsymbol{\theta}_{-i}^*)\right) \begin{pmatrix} \hat{\boldsymbol{\theta}}_{t,i} \\ \hat{\boldsymbol{\theta}}_{t+1,-i} \end{pmatrix} \quad \textit{First order Taylor expansion} \\
&= \hat{\boldsymbol{\theta}}_{t,i} + \eta_i \boldsymbol{A}_i \hat{\boldsymbol{\theta}}_{t,i} + \eta_i \boldsymbol{B}_i \hat{\boldsymbol{\theta}}_{t+1,-i} \\
\therefore \hat{\boldsymbol{\theta}}_{t+1} &= \hat{\boldsymbol{\theta}}_t + \boldsymbol{\eta}\boldsymbol{A}\hat{\boldsymbol{\theta}}_t + \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D}\hat{\boldsymbol{\theta}}_{t+1}.
\end{aligned} \tag{29}
$$

By analyzing the distance r_t of the GSPPM iterates from the stationary point,

$$
\begin{aligned}
r_{t+1}^2 &= \|\hat{\boldsymbol{\theta}}_{t+1}\|^2 \\
&= \hat{\boldsymbol{\theta}}_t^T (I + \boldsymbol{\eta}\boldsymbol{A})^T (I - \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^{-T} (I - \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^{-1} (I + \boldsymbol{\eta}\boldsymbol{A}) \hat{\boldsymbol{\theta}}_t \\
&\leq \frac{\lambda_{max}(I + \boldsymbol{\eta}\boldsymbol{A})^2}{\lambda_{min}(I - \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2} r_t^2.
\end{aligned} \tag{30}
$$

Thus, for any $\{\eta_i\}$ satisfying $\frac{\lambda_{max}(I+\boldsymbol{\eta}\boldsymbol{A})^2}{\lambda_{min}(I-\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2} < 1$, GSPPM iterates converge asymptotically to the local Nash equilibrium. □

A.3 Proof of Theorem 4.4

Let us define $\hat{\boldsymbol{\theta}}_{t,i}^k = \boldsymbol{\theta}_{t,i}^k - \boldsymbol{\theta}_i^*$ and $\hat{\boldsymbol{\theta}}_t^k = [\hat{\boldsymbol{\theta}}_{t,1}^k, ...\hat{\boldsymbol{\theta}}_{t,n}^k]^T$ for all $i \in \mathcal{I}$. It follows by linearizing the system about the stationary point $\boldsymbol{\theta}^*$,

$$
\begin{aligned}
\hat{\boldsymbol{\theta}}^k_{t+1,i} &= \boldsymbol{\theta}_{t,i} + \eta_i \nabla_{\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}_{t,i}\boldsymbol{\theta}^{k-1}_{t,-i}) - \boldsymbol{\theta}^* \\
&\approx \left(I + \eta_i \nabla^2_{\boldsymbol{\theta}_i,\boldsymbol{\theta}_i} J_i(\boldsymbol{\theta}^*_i, \boldsymbol{\theta}^*_{-i}), \eta_i \nabla^2_{\boldsymbol{\theta}_i,\boldsymbol{\theta}_{-i}} J_i(\boldsymbol{\theta}^*_i, \boldsymbol{\theta}^*_{-i})\right) \begin{pmatrix} \hat{\boldsymbol{\theta}}_{t,i} \\ \hat{\boldsymbol{\theta}}^{k-1}_{t,-i} \end{pmatrix} \quad \textit{First order Taylor expansion} \\
&= \hat{\boldsymbol{\theta}}_{t,i} + \eta_i \boldsymbol{A}_i \hat{\boldsymbol{\theta}}_{t,i} + \eta_i \boldsymbol{B}_i \hat{\boldsymbol{\theta}}^{k-1}_{t,-i} \\
\therefore \hat{\boldsymbol{\theta}}^k_t &= \hat{\boldsymbol{\theta}}_t + \boldsymbol{\eta}\boldsymbol{A}\hat{\boldsymbol{\theta}}_t + \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D}\hat{\boldsymbol{\theta}}^{k-1}_t.
\end{aligned} \tag{31}
$$

By analyzing the distance r^k_t of the KPG iterates from the stationary point,

$$
\begin{aligned}
(r^k_t)^2 &= \|\hat{\boldsymbol{\theta}}^k_t\|^2 \\
&= \hat{\boldsymbol{\theta}}^T_t (I + \boldsymbol{\eta}\boldsymbol{A})^T (I + \boldsymbol{\eta}\boldsymbol{A}) \hat{\boldsymbol{\theta}}_t + \hat{\boldsymbol{\theta}}^T_t (I + \boldsymbol{\eta}\boldsymbol{A})^T \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D}\hat{\boldsymbol{\theta}}^{k-1}_t \\
&+ (\hat{\boldsymbol{\theta}}^{k-1}_t)^T \boldsymbol{D}^T \boldsymbol{B}^T \boldsymbol{\eta}^T (I + \boldsymbol{\eta}\boldsymbol{A}) \hat{\boldsymbol{\theta}}_t + (\hat{\boldsymbol{\theta}}^{k-1}_t)^T \boldsymbol{D}^T \boldsymbol{B}^T \boldsymbol{\eta}^T \boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D}\hat{\boldsymbol{\theta}}^{k-1}_t \\
&\leq \left(\underset{max}{\lambda} (I + \boldsymbol{\eta}\boldsymbol{A})^2 + 2 \underset{max}{\lambda} (I + \boldsymbol{\eta}\boldsymbol{A}) \underset{max}{\lambda} (\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D}) \right) (r^0_t)^2 + \\
&2 \underset{max}{\lambda} (I + \boldsymbol{\eta}\boldsymbol{A}) \underset{max}{\lambda} (\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D}) \left(r^0_t \nabla_{max} \right) + \underset{max}{\lambda} (\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2 (r^{k-1}_t)^2.
\end{aligned} \tag{32}
$$

defining the bound of the finite-k iterates to the stationary point $\hat{\boldsymbol{\theta}}^*$.

Hence, for any $\{\eta_i\}$ satisfying $\underset{max}{\lambda} (\boldsymbol{\eta}\boldsymbol{B}\boldsymbol{D})^2 < 1$, KPG iterates converge asymptotically to the local Nash Equilibrium. □

B Details of the illustrative example

We report the full details of the toy problem introduced in Sec. 4.2, which we here refer to as the *Meet-up* problem.

Environment properties. The problem is designed as a simple 2-player continuous cooperative game in a 2D space. The state of the game $s = (s_1, s_2) \in \mathbb{R}^4$ encodes the location of the two players, with $s_i \in \mathbb{R}^2$. For the sake of simplicity, agents can only move by a fixed distance step of 1 around their current position, towards a chosen direction. The initial state of the two agents is deterministic and fixed to $\iota = (\iota_1 = (0, 0), \iota_2 = (3, 2))$. We assume undiscounted returns ($\gamma = 1$) and terminate an episode when the agents effectively meet each other as a result of their actions.

Policy parameterization. Although one-dimensional continuous actions are trivially tractable for one-step games, sequential decision making problems demand finding policies that respond optimally for any possible configuration s of the game. Here, we reduce the complexity of the problem by conveniently parameterizing each agent as single-parameter policies. In particular, we define an agent action as a 1-DoF unit vector $a_i \in \mathbb{R}^2$, and parameterize the deterministic policy of agent i with $\theta_i \in \mathbb{R}$, as

$$
\pi_i(s) = \begin{cases} (\cos\theta_i, \sin\theta_i)^\top & \text{if, } s = \iota \\ \pi^*_i(s) & \text{if, } s \neq \iota \end{cases} \tag{33}
$$

where, $\pi^*_i(s) = \frac{s_{-i} - s_i}{\|s_{-i} - s_i\|}$ is the optimal policy that goes straight towards the other agent. In other words, we assume that both agents will act optimally after taking the first action, and we only parametrize the agents decisions at the starting state. This design choice allows to easily study the joint policy space directly, as well as computing the closed-form solution of the return $J(\cdot)$ (see below).

Solving the Meet-up problem with KPG. We design the reward function of the Meet-up problem to reward each agent for getting closer to the other agent after the effect of both actions. We achieve this by computing the cosine similarity between the agent's action a_i and the actual direction that would have led closest to the other agent:

$$R_i(s, a, s') = a_i \cdot \pi_i^*(s_i, s'_{-i}) - 1 = a_i \cdot \frac{s'_{-i} - s_i}{\|s'_{-i} - s_i\|} - 1. \tag{34}$$

Here, we denote the joint action as $a = (a_1, a_2)$, and the next state as $s' = (s_1 + a_1, s_2 + a_2)$. Note that a -1 offset is added so that both the reward signal and the return $J_i(\theta_i, \theta_{-i})$ of each agent is always ≤ 0. In turn, this makes the computation of the optimal value function $V^*(s)$ of this game trivial: the strategy of moving towards each other in a straight line leads to returns of 0 from any state s; since this is the maximum return, this joint policy must also be optimal, and $V^*(s) = 0 \; \forall s$ is the unique optimal value function. We now derive the analytical form of $J_i(\theta_i, \theta_{-i})$, as needed by KPG to compute the recursive gradient updates. Given that both agents are assumed to act optimally in any state besides the starting state, we can conveniently write the return as

$$\begin{aligned} J_i(\theta_i, \theta_{-i}) &= R_i(\iota, a, s') + V_i(s') \\ &= R_i(\iota, a, s') + V^*(s') = \\ &= R_i(\iota, a, s') \end{aligned} \tag{35}$$

where s' is the resulting state after the players' first actions $a_1 = (\cos\theta_1, \sin\theta_1)$ and $a_2 = (\cos\theta_2, \sin\theta_2)$. Following this, we may therefore compute the gradient of the return for any pair of agent policies θ_1, θ_2 in closed form:

$$\begin{aligned} \nabla_{\theta_i} J_i(\theta_i, \theta_{-i}) &= \nabla_{\theta_i} R_i(\iota, a, s') \\ &= \nabla_{\theta_i} \Big(\; a_i \cdot \pi_i^*(\iota_i, s'_{-i}) \; \Big) \\ &= \nabla_{\theta_i} \Big(a_i \Big) \cdot \pi_i^*(\iota_i, s'_{-i}) \\ &= \nabla_{\theta_i} \begin{pmatrix} \cos\theta_i \\ \sin\theta_i \end{pmatrix} \cdot \pi_i^*(\iota_i, s'_{-i}) \\ &= \begin{pmatrix} -\sin\theta_i \\ \cos\theta_i \end{pmatrix} \cdot \pi_i^*(\iota_i, s'_{-i}) \end{aligned} \tag{36}$$

In conclusion, Eq. 36 allows us to compute the recursive K-level reasoning steps with the true analytical gradient of the return, as shown in Algorithm 1 of the main manuscript.

C Practical Implementation of the K-Level Policy Gradient

The performance difference lemma (PDL) [44] **can be extended to the k-level multi-agent setting** [49, 50].

Lemma C.1. *It can be shown that for agent i,*

$$J_i(\pi_i, \boldsymbol{\pi}_{-i}^{(0)}) - J_i(\pi_i^{(0)}, \boldsymbol{\pi}_{-i}^{(0)}) = \frac{1}{1-\gamma} \mathbb{E}_{(s,\boldsymbol{a}) \sim d^{\pi_i, \boldsymbol{\pi}_{-i}^{(0)}}, \pi_i, \boldsymbol{\pi}_{-i}^{(0)}} \left[A_i^{\boldsymbol{\pi}^{(0)}}(s, \boldsymbol{a}) \right] \tag{37}$$

$$= \frac{1}{1-\gamma} \mathbb{E}_{(s,\boldsymbol{a}) \sim d^{\pi_i, \boldsymbol{\pi}_{-i}^{(0)}}, \pi_i^{(0)}, \boldsymbol{\pi}_{-i}^{(0)}} \left[\frac{\pi_i}{\pi_i^{(0)}} A^{\boldsymbol{\pi}^{(0)}}(s, \boldsymbol{a}) \right], \tag{38}$$

where π_i is the policy of agent i, $\boldsymbol{\pi}_{-i}^{(0)}$ is the 0-level joint policy of the other agents, $J_i(.)$ is agent i's objective function, $\boldsymbol{a}$ is the joint action, $A_i^{(0)}(s, \boldsymbol{a})$ is agent i's advantage function, and $d^{(.)}$ is the discounted state distribution. This is the version of the PDL used by MAPPO; since it does not consider the update of the other agents, they are treated like part of the environment.

Now consider the PDL when updating against k-level agents:

$$J_i(\pi_i, \boldsymbol{\pi}_{-i}^{(k)}) - J_i(\pi_i^{(0)}, \boldsymbol{\pi}_{-i}^{(0)}) = \frac{1}{1-\gamma}\mathbb{E}_{(s,\boldsymbol{a})\sim d^{\pi_i,\boldsymbol{\pi}_{-i}^{(k)}},\pi_i,\boldsymbol{\pi}_{-i}^{(k)}}\left[A_i^{\boldsymbol{\pi}^{(0)}}(s,\boldsymbol{a})\right] \tag{39}$$

$$= \frac{1}{1-\gamma}\mathbb{E}_{(s,\boldsymbol{a})\sim d^{\pi_i,\boldsymbol{\pi}_{-i}^{(0)}},\pi_i^{(0)},\boldsymbol{\pi}_{-i}^{(0)}}\left[\frac{\pi_i}{\pi_i^{(0)}}\frac{\boldsymbol{\pi}_{-i}^{(k)}}{\boldsymbol{\pi}_{-i}^{(0)}}A^{\boldsymbol{\pi}^{(0)}}(s,\boldsymbol{a})\right], \tag{40}$$

which includes a correction ratio accounting for the change in action distribution of the k-level agents. Note that both cases feature a mismatch in the discounted state distribution d of the PDL and the sample distribution ($d^{\pi_i,\boldsymbol{\pi}_{-i}}$); this mismatch is ignored in practice.

The deterministic policy gradient can be estimated for k-level policies. Centralized deterministic policy gradient algorithms utilize an action-value function trained using off-policy transitions. In our implementation of K-MADDPG and K-FACMAC, we use an off-policy critic $Q_i^{\boldsymbol{\pi}^{(0)}}(s,.,.)$, which estimates the action-value function for agent i under the $k=0$ policies of all the agents. Since the critic is trained using off-policy data, it is reasonably well-estimated for higher k-level updates which are obtained by estimating $Q_i^{\boldsymbol{\pi}^{(0)}}(s, a_i, \boldsymbol{a}_{-i}^{(k)})$ for each agent i, since the joint k-level actions $\boldsymbol{a}_{-i}^{(k)}$ are only obtained by policies one gradient step away from $\boldsymbol{\pi}_{-i}^{(0)}$. This is similar to Peng et al. [14], where the centralized critic is trained using off-policy actions and used to estimate the action-value function for the online actions of all agents to prevent relative overgeneralization.

D Environment Details

All SMAX maps were configured according to the default settings seen in Rutherford et al. [12]. All MAMuJoCo environments and agents are configured according to the default configurations used in Peng et al. [14] where they were introduced. Each agent observes the positions of its own body parts, receives a common team reward that depends on the task, and controls only its joints. The exact configurations and rewards can be seen at `https://robotics.farama.org/envs/MaMuJoCo/`. All experiments using SMAC mostly used the default team configurations, rewards, and observations as the SMAC benchmark [13]. The state space was modified slightly by including the last actions of each agent (using the inbuilt feature in the StarCraft II environment) as this was found to stabilize learning for all algorithms.

E Experimental Details

E.1 SMAX

Each baseline is run with the settings seen in Rutherford et al. [12]. Each baseline uses $1e7$ total training steps and is trained against the 'HeuristicEnemySMAX' AI, updated every 128 steps, and uses a γ of 0.99. **Off-policy algorithms** (QMIX, VDN, IQL) are trained with 16 parallel environments with a buffer size of 5000 and a batch size of 32. Each uses Adam optimizers with a learning rate of $5e-5$, and performs ϵ-greedy exploration during training time with and ϵ that decays from 1 to 0.05 over the first 10% of total steps (learning is also paused until the ϵ decay is concluded). Neural networks use a hidden size of 512 and relu activations, hard target updates every 10 updates, 8 update epochs, and a reward scale of 10 (the reward scale of the original SMAC environments). The maximum gradient norm is constrained to be 10. In QMIX, the mixer embedding dimension is 64, the mixer hypernet hidden dimension is 256, and the initial scale of the kernel weights of the mixer weights is set to 0.001. Baselines are evaluated every 5% of total steps for 128 steps across 512 environments. **On-policy algorithms** (K-MAPPO, MAPPO, IPPO, POLA) are trained with 64 parallel environments. Each uses Adam optimizers with a learning rate of $4e-3$ which is annealed to 0 over the entire course of training. Neural networks use a hidden size of 128 and relu activations, 2 minibatch updates, and 2 update epochs. The maximum gradient norm is constrained to be 0.5. The value of λ for the GAE is set to 0.95, the value of ϵ for surrogate clipping is 0.2, the value loss coefficient is 0.5, and no entropy bonus is provided.

E.2 Multi-Agent MuJoCo and SMAC

For each algorithm, we evaluate the performance in the following manner: we pause training after 10,000 steps and run a fixed number of independent test episodes (10 for MAMuJoCo and 32 for SMAC). During these test episodes, each agent acts greedily in a decentralized fashion (DDPG agents don't use action noise, DQN agents don't select random actions, etc.). The mean performance of the agents is reported in MAMuJoCo (the performance for each agent is identical since they share a common objective) and the mean success rate is reported for SMAC. Note that we chose to report the mean success rate rather than the median, as certain SMAC maps (especially Super Hard maps) commonly result in very success rates. Using the mean success rate better reflects the difficulty of these maps, as the median success rate can sometimes skew results to look more positive than they should. We set $\gamma = 0.99$ for all experiments.

Since our results are primarily obtained by applying KPG to FACMAC, we mostly kept the algorithmic implementation standards used in FACMAC for reproducibility. We use parameter sharing for all actor and critic networks to speed up learning. All actor, critic, mixer, and Q-networks have target networks.

In torch environments, we find that RMSProp provides the most stable updates, since the momentum effects of Adam appear to mitigate the benefits of KPG by carrying over a running average of gradients from previous timesteps (which does not help when KPG's benefit is a better gradient for the *current* timestep.) As a practical detail, note also that at the end of each non-final intermediate gradient step taken by KPG, the statistics of the optimizer must be reset to what they were at the beginning of that timestep, as the carryover effects of any optimizer from the previous timestep must apply to each K-level gradient independently.

To clarify our implementation of K-FACMAC, we provide Algorithm 2 below, which provides further details on K-level action sampling, intermediate fitting using optimizers, and resets of the actors and optimizers after each K-level step *except* the final one.

MAMuJoCo The architecture of all deep Q-networks is an MLP with 2 hidden layers with 400 and 300 units respectively. In all actor-critic methods, the architecture of the shared actor and critic networks is an MLP with 2 hidden layers with 400 and 300 hidden units respectively. All hidden layers for all networks use ReLU activations. All critic networks provide raw outputs while actor networks have a tanh activation at the output. Actor networks and DQNs receive the local observations of that agent as an input, appended with a one-hot vector due to the parameter sharing. All centralized critics and mixing networks are conditioned on the global state provided by the environment.

Each episode has a maximum length of 1000 steps. The total training time for each algorithm is set to 2 million steps. To improve initial exploration, each agent takes 10,000 random steps at the beginning of each run. During training we apply uncorrelated, mean-zero noise with a standard deviation of 0.1 to further encourage exploration. Each agent has a replay buffer with a maximum size of 1 million and trains with a batch size of 100 after every new sample. Target networks are updated using Polyak averaging with $\tau = 0.001$. All neural networks are trained using the Adam optimizer [51] (except for KPG which uses RMSProp) with a learning rate of 0.001.

SMAC The architecture of all shared deep Q-networks is a DRQN with a recurrent layer comprised of a GRU with a 64-dimensional hidden state, with fully connected layers on either side. In all actor-critic methods, the architecture of all shared actors is a recurrent MLP comprised of a GRU with a 64-dimensional hidden state, with fully connected layers on either side. We train the GRU networks on batches of 32 fully unrolled episodes (with 0-padding to account for temporal mismatch between episodes). The architecture of all shared critic networks is an MLP with 2 hidden layers with 64 units. All networks use ReLU activations for the hidden layers. All actor critic methods select discrete actions using the Gumbel-Softmax estimator [52] in order to turn continuous softmaxed logits into discrete one-hot actions while retaining the ability to backpropagate through the network. Actor networks and DRQNs receive the local observations of that agent as an input, appended with the last action taken by the agent, as well as a one-hot vector due to the parameter sharing. All agents use ϵ-greedy action selection and we anneal ϵ from 0.5 to 0.05 over 50k training steps. The replay buffer contains the most recent 5000 episodes. All target networks are updated hard every 200 training steps. All networks are trained using Adam (except for KPG which uses RMSProp) with a learning rate of 0.0025 for the actor network and 0.0005 for the critic network (except for QMIX which uses the learning rates specified in Samvelyan et al. [13] as they have already been tuned for SMAC).

Algorithm 2 K-FACMAC

Input: Initial actor parameters $\boldsymbol{\theta}_i$; Initial critic parameters $\boldsymbol{\phi}_i$; Initial mixer parameters $\boldsymbol{\psi}$; Actor targets $\overline{\boldsymbol{\theta}}_{\boldsymbol{i}}$; Critic targets $\overline{\boldsymbol{\phi}}_{\boldsymbol{i}}$; Mixer target $\overline{\boldsymbol{\psi}}$; Actor optimizers Ω_i; Critic and mixer update function $\mathcal{F}()$; recursive reasoning steps k'; $\forall i \in \mathcal{I}$

for each update step **do**
  Update critics and mixer using $\mathcal{F}(\{\boldsymbol{\phi}_{t,i}\}, \boldsymbol{\psi_t}, \{\overline{\boldsymbol{\theta}}_{\boldsymbol{t,i}}\})$ {typical FACMAC critic and mixer update}
  Save initial actor parameters $\boldsymbol{\theta}_{reset,i} \leftarrow \boldsymbol{\theta}_{t,i}$ $\forall i \in \mathcal{I}$
  Save initial actor optimizer states $\Omega_{reset,i} \leftarrow \Omega_{t,i}$ $\forall i \in \mathcal{I}$
  Sample 0-level actions $a_i^{k=0} \sim \boldsymbol{\theta}_{t,i}$, $\forall i \in \mathcal{I}$
  for $j = 1$ **to** k' **do**
    Sample 0-level actions $a_i \sim \boldsymbol{\theta}_{t,i}$, $\forall i \in \mathcal{I}$
    Update actor parameters $\boldsymbol{\theta}_{t,i}^{k=j} \leftarrow \Omega_{t,i}(a_{t,i}, a_{-i}^{k=j-1}, \boldsymbol{\theta}_{t,i})$, $\forall i \in \mathcal{I}$ {this implicitly updates the optimizer states}
    if $j \neq k'$ **then**
      Sample j-level actions $a_i^{k=j} \sim \boldsymbol{\theta}_{t,i}^{k=j}$ $\forall i \in \mathcal{I}$
      Reset actor parameters $\boldsymbol{\theta}_{t,i} \leftarrow \boldsymbol{\theta}_{reset,i}$, $\forall i \in \mathcal{I}$
      Reset actor optimizers $\Omega_{t,i} \leftarrow \Omega_{reset,i}$, $\forall i \in \mathcal{I}$
    end if
  end for
  Update targets $\overline{\boldsymbol{\theta}}_{\boldsymbol{i}}, \overline{\boldsymbol{\phi}}_{\boldsymbol{i}}, \overline{\boldsymbol{\psi}}$, $\forall i \in \mathcal{I}$
  $\boldsymbol{\theta}_{t+1,i} \leftarrow \boldsymbol{\theta}_{t,i}^{k=k'}$, $\forall i \in \mathcal{I}$
end for

Table 1: Final mean performance and success rate improvement w.r.t. FACMAC for $K2$-FACMAC and baselines across MAMuJoCo and SMAC maps.

Algorithm	MAMuJoCo	SMAC
FACMAC	$+0.0\%$	$+0.0\%$
COMIX/QMIX	-53%	-26%
MADDPG	-238%	-85%
POLA	-106%	-88%
K2-FACMAC	$\mathbf{+114\%}$	$\mathbf{+98\%}$

F Additional Results

K2-FACMAC doubles the performance of FACMAC on average. Table 1 shows the average performance and success rate changes for each algorithm relative to FACMAC. Note that the extreme jump for $K2$-FACMAC over FACMAC is skewed higher in these experiments due to the number of difficult environments which FACMAC never solved reasonably, such as Walker2d, 5m_vs_6m, Corridor, and 6h_vs_8z.

Table 2: Mean wall clock increases to run K-FACMAC and K-MADDPG relative to nominal FACMAC and MADDPG, respectively.

K-Level	FACMAC	MADDPG
2	$+28\%$	$+27\%$
3	$+55\%$	$+52\%$
4	$+84\%$	$+84\%$
5	$+106\%$	$+99\%$