SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning

Benjamin Ellis[1]* **Skander Moalla**[1] **Mikayel Samvelyan**[2]
Mingfei Sun[1] **Anuj Mahajan**[1] **Jakob N. Foerster**[1] **Shimon Whiteson**[1]
[1]University of Oxford [2]University College London

Abstract

The availability of challenging benchmarks has played a key role in the recent progress of machine learning. In cooperative multi-agent reinforcement learning, the StarCraft Multi-Agent Challenge (SMAC) has become a popular testbed for centralised training with decentralised execution. However, after years of sustained improvement on SMAC, algorithms now achieve near-perfect performance. In this work, we conduct new analysis demonstrating that SMAC is not sufficiently stochastic to require complex *closed-loop* policies. In particular, we show that an *open-loop* policy conditioned only on the timestep can achieve non-trivial win rates for many SMAC scenarios. To address this limitation, we introduce SMACv2, a new version of the benchmark where scenarios are procedurally generated and require agents to generalise to previously unseen settings (from the same distribution) during evaluation. We show that these changes ensure the benchmark requires the use of *closed-loop* policies. We evaluate state-of-the-art algorithms on SMACv2 and show that it presents significant challenges not present in the original benchmark. Our analysis illustrates that SMACv2 addresses the discovered deficiencies of SMAC and can help benchmark the next generation of MARL methods. Videos of training are available on our website.

1 Introduction

In many cases, control policies for cooperative multi-agent reinforcement learning (MARL) can be learned in a setting where the algorithm can access all agents' observations, e.g., in a simulator or laboratory, but must be deployed where such centralisation is not possible. Centralised training with decentralised execution (CTDE) [9, 24] is a paradigm for tackling such a setting and has been the focus of much recent research [31, 43, 11, 25, 26]. In CTDE, the learning algorithm can see all observations during training, as well as possibly the Markov state, but must learn policies that can be executed without such privileged information.

As in other areas of machine learning, benchmarks have played an important role in driving progress in CTDE. Examples of such benchmarks include the Hanabi Learning Environment [2], Google football [19], PettingZoo [45], and Multi-Agent Mujoco [30]. Perhaps the most popular is the StarCraft Multi-Agent Challenge (SMAC) [36], which focuses on decentralised micromanagement challenges in the game of StarCraft II. Rather than tackling the full game with centralised control [49], SMAC tasks a group of learning agents, each controlling a single army unit, to defeat the units of the enemy army controlled by the built-in heuristic AI. SMAC requires learning several complex joint

*Correspondence to benellis@robots.ox.ac.uk.

Preprint. Under review.

(a) Terran (b) Zerg (c) Protoss

Figure 1: Screenshots from SMACv2 showing agents battling the built-in AI.

action sequences such as focus fire,[2] and kiting enemy units,[3]. For these reasons, many prominent MARL papers [50, 25, 51, 32, 8, 54, 52] rely on SMAC to benchmark their performance.

However, there is significant evidence that SMAC is outliving its usefulness [54, 12, 10]. Recent work reports near-perfect win rates on most scenarios [54, 12] and strong performance without using the centralised state via independent learning [54, 7]. This suggests that, due to ceiling effects, SMAC may no longer reward further algorithmic improvements and that new benchmarks are needed.

In this paper, we present a new, more fundamental problem with SMAC and hence reason for new benchmarks. In particular, we show that an open-loop policy, which conditions only on the timestep and ignores all other observations, performs well on many SMAC scenarios. We also extend this analysis by examining the estimates of the joint Q-function learnt by QMIX. We demonstrate that, even with all features masked, regression to the joint Q-function is possible with less than 10% error in all but two scenarios from just timestep information.

Together these results suggest that SMAC is not stochastic enough to necessitate complex closed-loop control policies on many scenarios. Therefore, although SMAC scenarios may require complex action sequences such as focus fire, they do not require agents to apply this behaviour in small range of situations and hence agents can simply repeat a fixed sequence of actions that resemble it in a specific instance.

To address these shortcomings, we propose SMACv2, a new version of SMAC that uses procedural content generation (PCG) [35] to address SMAC's lack of stochasticity. In SMACv2, for each episode we randomly generate team compositions and agent start positions. Consequently, it is no longer sufficient for agents to repeat a fixed action sequence, but they must learn to cooperate in a diverse range of scenarios. Additionally, we update the sight and attack ranges to increase unit diversity. Finally, our implementation is extensible, allowing researchers to implement new distributions easily and hence further expand on the available SMACv2 scenarios. We repeat our analysis of stochasticity on SMACv2. The results indicate that the added stochasticity prevents an open-loop policy from learning successfully and requires conditioning on ally and enemy features.

Furthermore, we apply state-of-the-art algorithms to SMACv2 and show that they struggle with many scenarios, confirming that SMACv2 poses substantial new challenges. Finally, we perform an ablation on the new observation features to demonstrate how each contributes to the difficulty, highlighting the specific areas on which future MARL research should focus.

2 Related Work

The MARL community has made significant use of games for benchmarking cooperative MARL algorithms. The Hanabi Learning Environment [2] tasks agents with learning to cooperate in the card-game Hanabi. Here, each agent observes the cards of its teammates but not its own, which must be implicitly communicated via gameplay. Hanabi is partially observable and stochastic but only features teams of 2 to 5 players, which is fewer than all but the smallest SMACv2 scenarios. Kurach et al. [19] propose Google Football as a complex and stochastic benchmark, and also have a multi-agent setting. However, it assumes fully observable states, which simplifies coordination.

Peng et al. [30] propose a multi-agent adaptation of the MuJoCo environment featuring a complex continuous action space. Multi-Particle Environments (MPE) [24] feature simple communication-

[2] In focused fire, allied units jointly attack and kill enemy units one after another.

[3] Kiting draws enemy units to give chase while maintaining distance so that little damage is incurred.

oriented challenges where particle agents can move and interact with each other using continuous actions. In contrast to both multi-agent MuJoCo and MPE, SMACv2 has a discrete action space, but challenges agents to handle a wide range of scenarios using procedural content generation. A plethora of work in MARL uses grid-worlds [24, 22, 53, 3, 47, 34, 23] where agents move and perform a small number of discrete actions on a 2D grid, but these tasks have much lower dimensional observations than SMACv2. OpenSpiel [21] and PettingZoo [45] provide collections of cooperative, competitive, and mixed sum games, such as grid-worlds and board games. However, the cooperative testbeds in either of these suites feature only simple environments with deterministic dynamics or a small number of agents. Neural MMO [40] provides a massively multi-agent game environment with open-ended tasks. However, it focuses on emergent behaviour within a large population of agents, rather than the fine-grained coordination of fully cooperative agents. Furthermore, none of these environments combines partial observability, complex dynamics, and high-dimensional observation spaces, whilst also featuring more than a few agents that need to coordinate to solve a common goal.

StarCraft has been frequently used as a testbed for RL algorithms. Most work focuses on the full game whereby a centralised controller serves as a puppeteer issuing commands for the two elements of the game: *macromanagement*, i.e., the high-level strategies for resource management and economy, and *micromanagement*, i.e., the fine-grained control of army units. TorchCraft [44] and TorchCraftAI [1] provide interfaces for training agents on *StarCraft: BroodWar*. The StarCraft II Learning Environment (SC2LE) [48] provides a Python interface for communicating with the game of *StarCraft II* and has been used to train AlphaStar [49], a grandmaster-level but fully centralised agent that is able to beat professional human players. SMAC and SMACv2 are built on top of SC2LE and concentrate only on decentralised unit micromanagement for the CTDE setting.

One limitation of SMAC is the constant starting positions and types of units, allowing methods to memorise action sequences for solving individual scenarios (as we show in Section 4.1), while also lacking the ability to generalise to new settings at test time, which is crucial for real-world applications of MARL [27]. To address these issues, SMACv2 relies on procedural content generation [PCG; 35, 17] whereby infinite game levels are generated algorithmically and differ across episodes. PCG environments have recently gained popularity in single-agent domains [6, 5, 16, 20, 37] for improving generalisation in RL [18] and we believe the next generation of MARL benchmarks should follow suit. Iqbal et al. [14] and Mahajan et al. [27] consider updated versions of SMAC by randomising the number and types of the units, respectively, to assess the generalisation in MARL. However, these works do not include the random-start positions explored in SMACv2 or analyse the properties of SMAC to motivate these changes. They also do not change the agents' field-of-view and attack ranges or provide a convenient interface to generate new distributions over these features.

3 Background

3.1 Dec-POMDPs

A partially observable, cooperative multi-agent reinforcement learning task can be described by a *decentralised partially observable Markov decision process (Dec-POMDP)* [29]. This is a tuple $(n, S, A, T, \mathbb{O}, O, R, \gamma)$ where n is the number of agents, S is the set of states, A is the set of individual actions, $T : S \times A^n \to \Delta(S)$ is the transition probability function, $\mathbb{O}$ is the set of joint observations, $O : S \times A^n \to \Delta(\mathbb{O})$ is the observation function, $R : S \times A^n \to \Delta(\mathbb{R})$ is the reward function, and γ is the discount factor. We use $\Delta(X)$ to denote the set of probability distributions over a set X. At each timestep t, each agent $i \in \{1, \dots, n\}$ chooses an action $a \in A$. The global state then transitions from state $s \in S$ to $s' \in S$ and yields a reward $r \in \mathbb{R}$ after the joint action $\mathbf{a}$, according to the distribution $T(s, \mathbf{a})$ with probability $\mathbb{P}(s'|s, \mathbf{a})$ and $R(s, \mathbf{a})$ with probability $\mathbb{P}(r'|s, \mathbf{a})$, respectively. Each agent then receives an observation according to the observation function O so that the joint observation $\mathbf{o} \in \mathbb{O}$ is sampled according to $O(s, \mathbf{a})$ with probability $\mathbb{P}(\mathbf{o}|s, \mathbf{a})$. This generates a trajectory $\tau_i \in (\mathbb{O} \times A^n)^*$. The goal is to learn n policies $\pi_i : (\mathbb{O}_i \times A)^* \to \Delta(A)$ that maximise the expected cumulative reward $\mathbb{E}[\sum_i \gamma^i r_{t+i}]$.

One of the key challenges in Dec-POMDPs is partial observability. However, this is only relevant when the environment is stochastic: stochasticity in the initial state and/or state transition makes it in general impossible to learn good policies for Dec-POMDPs by repeating a fixed sequence of actions without conditioning on the observation.

Instead, policies must generalise to different conditions. For example, a deterministic Dec-POMDP can be solved by repeating a fixed action sequence, and therefore only encountering a fixed sequence of states, whereas a stochastic Dec-POMDP could require policies that can perform good actions across the entire state-space. **Therefore, when assessing the ability of agents to generalise *in-distribution*, stochasticity is an important requirement**.

One way of measuring stochasticity is to measure how a policy conditions on its observation. A policy in a deterministic environment does not need to condition on its observation at all, but can repeat a fixed sequence of actions by conditioning only on the timestep. A policy which operates in a mostly deterministic environment may only need to attend to a few features for a small number of steps in an episode, and a policy in a complex environment may need to access most of its observation at most time steps. We use this concept in section 4.2 to evaluate SMAC's stochasticity more granularly.

3.2 StarCraft Multi-Agent Challenge (SMAC)

Rather than tackling the full game, the StarCraft Multi-Agent Challenge [36, SMAC] benchmark focuses on micromanagement challenges where each military unit is controlled by a single learning agent. Units have a limited field-of-view and no explicit communication mechanism when at test time. By featuring a set of challenging and diverse scenarios, SMAC has been used extensively by the MARL community for benchmarking algorithms. It consists of 14 micromanagement scenarios, which can be broadly divided into three different categories: *symmetric*, *asymmetric*, and *micro-trick*. The symmetric scenarios feature the same number of allied units as enemies. The asymmetric scenarios have one or more extra units for the enemy and are often more difficult than the symmetric scenarios. The micro-trick scenarios feature setups that require specific strategies to counter. For example, `3s_vs_5z` requires the three allied stalkers to kite[4]the five zealots, and `corridor` requires six zealots to block off a narrow corridor to defeat twenty-four zerglings without being swarmed from all sides. Two state-of-the-art algorithms on SMAC are MAPPO [54] and QMIX [31]. We provide background of these methods in Appendix B.

4 Limitations of SMAC

In this section, we analyse stochasticity in SMAC in two different ways. We show that SMAC is insufficiently stochastic to require complex closed-loop policies.

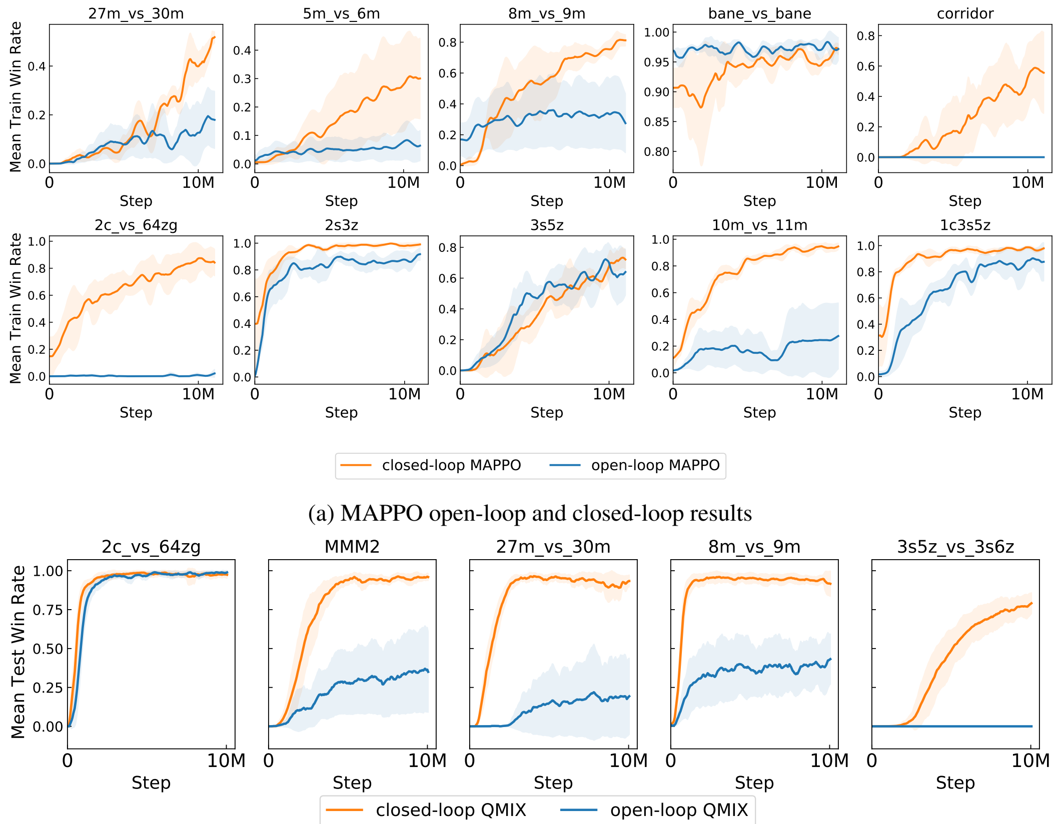
4.1 SMAC Stochasticity

We use the observation that if the initial state, observation and transition functions are all deterministic, there always exists an optimal deterministic policy that conditions solely on the timestep information to investigate stochasticity in SMAC. In particular, we use a range of SMAC scenarios to compare policies conditioned only on the timestep (which we refer to as *open-loop*) to policies conditioned on the full observation history (i.e., all the information typically available to each agent in SMAC), which we refer to as *closed-loop*. If stochasticity in the underlying state is a serious challenge when training policies on SMAC scenarios, then policies without access to environment observations should fail.

However, if the open-loop policies can succeed at the tasks, this demonstrates that SMAC is not representing the challenges of Dec-POMDPs well and so would benefit from added stochasticity. We use MAPPO and QMIX as our base algorithms and train open- and closed-loop versions of each. We train the *open-loop* policies on SMAC, but only allow the policies to observe the agent ID and timestep, whereas the closed-loop policies are given the usual SMAC observation as input with the timestep appended. The QMIX open-loop policy was trained only on the maps which MAPPO was not able to completely solve. For QMIX, the hyperparameters used are the same as in [12] and for MAPPO the hyperparameters are listed in Appendix C.

Figure 2 shows the mean win rates of both policies. Overall, the open-loop policies perform significantly worse than their closed-loop counterparts. However, this performance difference varies widely across scenarios and algorithms. For MAPPO, some scenarios, such as `bane_vs_bane`, `3s5z`, `1c3s5z` and `2s3z` achieve open-loop performance indistinguishable from that of closed-loop. In other scenarios, such as `8m_vs_9m` and `27m_vs_30m`, there are large performance differences,

[4]Making enemy units give chase while maintaining large distance so that little or no damage is incurred.



(a) MAPPO open-loop and closed-loop results

(b) QMIX open-loop and closed-loop results

Figure 2: Plot of selected SMAC scenarios treated as an open-loop planning problem by limiting the observation to the current timestep and agent ID. Plots show the mean win rate and standard deviation across 3 training seeds for MAPPO and QMIX.

but the open-loop policy still achieves high win rates. There are also scenarios, such as `corridor` and `2c_vs_64zg`, where the MAPPO open-loop policy does not learn at all, whereas the closed-loop policies can learn successfully. QMIX also achieves good performance on `2c_vs_64zg`, `MMM2`, `27m_vs_30m` and `8m_vs_9m`, although for the latter maps the closed-loop policy strongly outperforms the open-loop QMIX. Altogether, there are only four SMAC maps where the open-loop approach cannot learn a good policy at all: `3s5z_vs_3s6z`, `corridor`, `6h_vs_8z` and `5m_vs_6m`. Overall, open-loop policies perform well on a range of scenarios in SMAC. The *easy* scenarios show no difference between the closed-loop and the open-loop policies, and some *hard* and *very hard* scenarios show either little difference or non-trivial win rates for the open-loop policy. This suggests that stochasticity is not a significant challenge for a wide range of scenarios in SMAC.

These results highlight an important deficiency in SMAC. Stochasticity is either not evaluated or not a significant part of the challenge for the vast majority of SMAC maps. Not testing stochasticity is a major flaw for a general MARL benchmark because without stochasticity the policy does not need to generalise, but instead only take optimal actions along a single trajectory. Given widespread use of SMAC as a benchmark to evaluate algorithms for Dec-POMDPs, this suggests that a new benchmark is required to holistically evaluate MARL algorithms.

4.2 SMAC Feature Inferrability & Relevance

In this section, we look at stochasticity from a different perspective. We mask (i.e. "zero-out") all features of the state and observations on which QMIX's joint Q-function conditions. If the Q-values can still be easily inferred, then SMAC does not require learning complex closed-loop policies as Dec-POMDPs in general do, but is solvable by ignoring key aspects of StarCraft II gameplay.

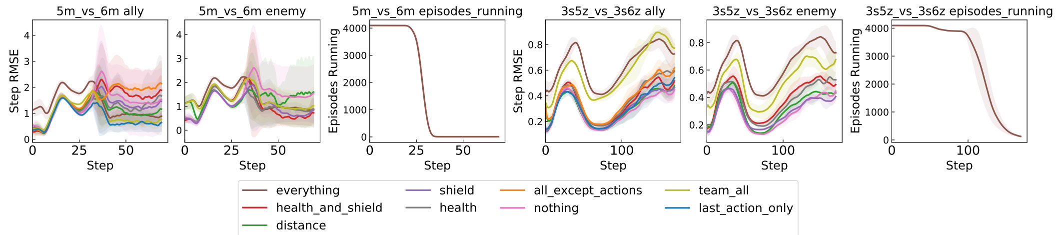


Figure 3: Comparison of the loss for different feature masks when regressing to a trained QMIX policy. *everything* masks the entire observation and *nothing* masks no attributes. *team_all* masks all attributes of a given team. *last_action_only* masks the last actions of all the allies in the state. The *all_except_actions* masks all the allied information *except* the last actions in the state. These latter two masks only apply to the allies. The mean is plotted for each mask and standard deviation across 3 seeds used as error bars.

We also extend this experiment to masking subsets of the observation and state features. By only masking a subset of the features, we measure the impact of different features on the learned policy, and therefore understand the information that is most important to the policy. The more observation features the Q-function conditions on, the more confident we can be that SMAC requires understanding decentralised agent micromanagement rather than learning action sequences because the observation features were chosen by humans to be relevant to the task.

To measure this, we first collect trajectories from a trained QMIX [31] policy. This uses the implementation and hyperparameters from Hu et al. [12], and achieves strong (near perfect) performance across all scenarios. Training curves for these policies are given in Appendix C.2. We then train another, identical QMIX network, which we call the *regression network* to regress to the estimated Q-values of the trained policy by training on the collected rollouts. This gives a baseline (i.e., minimum) error due to randomness in the training process.

To estimate feature importance, we redo this experiment, training a new regression network while masking all features from the observation and state history that make up its input. By masking sets of SMAC features like this, we measure the impact of a given set of features on this regression error and hence the Q-value. We group features that are related into masks, which can be found in a table in the appendix. If the regression error is high, the feature has a large impact on the estimate of the Q-values, meaning it must be both relevant and difficult to infer (since otherwise the regression network would learn to reconstruct the feature from others in the observation). We hypothesise that a lack of stochasticity in SMAC renders many features either irrelevant or inferrable. To investigate which features in SMAC are important, we compute results for several masks, including masking all features, health, and distance. Masks are applied to an entire team at a time, i.e., the given attribute is masked for all enemies or all allies. Additionally, attributes are always masked in both the state and the observation.

Figure 3, which plots this loss for each mask and scenario against steps in the RL episode, shows some interesting trends. First, the error of masking *everything* is low, reaching about 15% of the mean Q-value at peak, and 5-10% for most of the episode. Root-mean-squared error as a proportion of mean Q-value is given in the appendix. For all scenarios except `5m_vs_6m` this value is below 0.12. This is more evidence that stochasticity is not a concern when training policies on SMAC.

Secondly, there is redundancy between the ally actions and observations. `3s5z_vs_3s6z` shows this well. *last_action_only* is close to *nothing*, as is *all_except_actions*, but *team_all* has a high error. In other words, if the last action or the ally observations are obscured, then the Q-value is inferrable, but if both are hidden, then it is not. Thus there must be redundancy between these feature groups, which is an intuitive result – if an agent knows the features of another agent, it could infer its last action by modelling its policy. This redundancy is not necessarily a problem – part of learning a task is to understand and eliminate the use of redundant features. Finally, some features impact regression performance in some scenarios. Health information is an impactful feature for some harder Terran scenarios such as `27m_vs_30m`, `5m_vs_6m`, and `MMM2`. Additionally, enemy distance is also an impactful feature early on in the episode in `5m_vs_6m`. However, for most scenarios, most of the features individually do not impact regression performance.

A key insight is that even in these harder scenarios where an open-loop policy fails completely, it is still possible to regress accurately to the Q-value given no observations, which suggests the policies learned need not generalise well. This highlights the need for more stochastic environments in which the timestep is no longer such an important feature and policies must learn to condition on a larger portion of the observation space.

5 SMACv2

As shown in the previous section, SMAC has some significant drawbacks. To address these, we propose three major changes: random team compositions, random start positions, and increasing diversity among unit types by using the true unit attack and sight ranges. These changes increase stochasticity and so address the deficiency discovered in the previous section.

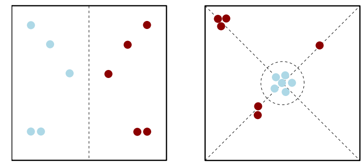


Figure 4: Examples of the two different types of start positions, *reflect* and *surround*. Allied units are shown in *blue* and enemy units in *dark red*.

Teams in SMACv2 are procedurally generated. Since StarCraft II does not allow units from different races to be on the same team, scenarios are divided depending on whether they feature Protoss, Terran or Zerg units. In scenarios with the same number of enemy and ally units, the unit types on each team are the same. Otherwise the teams are the same except for the extra enemy units, which are drawn identically to the other units. We use three unit types for each race, chosen to be the same unit types that were present in the original SMAC. We generate units in teams algorithmically, with each unit type having a fixed probability. The probabilities of generating each unit type are given in the appendix. These probabilities are the same at test time and train time.

Random start positions in SMACv2 come in two different flavours. In *reflect* scenarios, the allied units are spawned uniformly at random on one side of the scenario. The enemy positions are the allied positions reflected in the vertical midpoint of the scenario. This is similar to how units spawned in the original SMAC benchmark but without clustering them together. In *surround* scenarios, allied units are spawned at the centre and surrounded by enemies stationed along the four diagonals. Figure 4 illustrates both flavours of start position.

The final change is to the sight range and attack range of the units. In the original SMAC, all units types had a fixed sight and attack range, allowing short- and long-ranged units to choose to attack at the same distance. This did not affect the real attack range of units within the game – a unit ordered to attack an enemy while outside its striking range would approach the enemy before attacking. We change this to use the values from SC2, rather than the fixed values. However, we impose a minimum attack range of 2 because using the true attack ranges for melee units makes attacking too difficult.

Taken together, these changes mean that before an episode starts, agents no longer know what their unit type or initial position will be. Consequently, by diversifying the range of different scenarios the agents might encounter, they are required to understand the observation and state spaces more clearly, which should render learning a successful open-loop policy not possible.

We also define a convenient interface for defining distributions over team unit types and start positions. By only implementing a small `Distribution` class one can easily change how these are generated.

Akin to the original version of SMAC [36], we split the scenarios into *symmetric* and *asymmetric* groups. In the symmetric scenarios, allies and enemies have the same number and type of units. In the asymmetric scenarios, the enemies have some extra units chosen from the same distribution as the allies. There are 5 unit, 10 unit and 20 unit symmetric scenarios, and 10 vs 11 and 20 vs 23 unit aymmetric scenarios for each of the three races.

6 SMACv2 Experiments

In this section, we present results from a number of experiments on SMACv2. We first train state-of-the-art SMAC baselines on SMACv2 to assess its difficulty. We then ablate the environment features to assess how each contributes to the difficulty and highlight specific areas on which future research should focus. Currently, both value-based and policy optimisation approaches achieve strong performance on all SMAC scenarios [54, 12]. As baselines we use QMIX [31], a strong value-based method and MAPPO [54], a strong policy optimisation method.

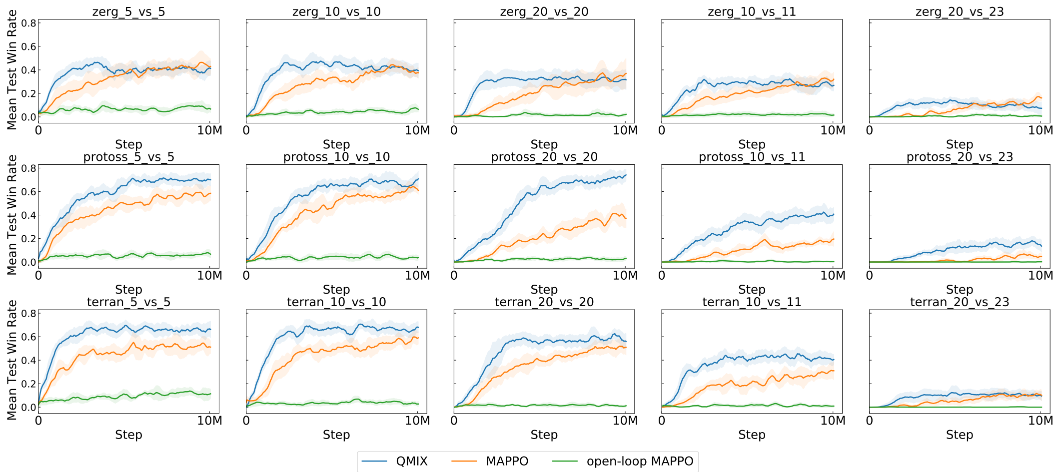


Figure 5: Comparison of the mean test win rate of QMIX, MAPPO and an open-loop policy on SMACv2. Plots show the mean and standard deviation across 3 seeds.

6.1 SMACv2 Baseline Comparisons

First we compare the performance of MAPPO, QMIX, and an open-loop policy, based on MAPPO, on SMACv2. We run each algorithm for 10M training steps and 3 environment seeds. For MAPPO, we use the implementation from Sun et al. [41] and for QMIX the one from Hu et al. [12]. The results of the runs of the best hyperparameters are shown in Figure 5.

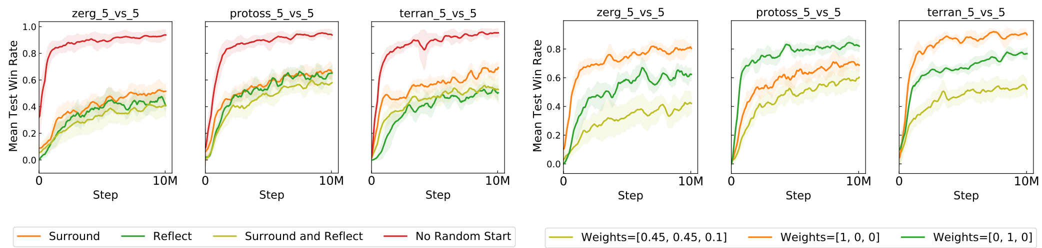
These results shown in Figure 5 reveal a few trends. First, QMIX generally performs better than MAPPO across most scenarios. QMIX strongly outperforms MAPPO in two Protoss scenarios (`protoss_20_vs_20` and `protoss_10_vs_11`), but otherwise final performance is similar, although QMIX is more sample efficient. However, QMIX is memory-intensive because of its large replay buffer and so requires more compute to run than MAPPO. Additionally, MAPPO appears to still be increasing its performance towards the end of training in several scenarios, such as the `10_vs_11` and `20_vs_20` maps. It is possible that MAPPO could attain better performance than QMIX if a larger compute budget were used.

Roughly even win rates are attained on the symmetric maps. By contrast, on the asymmetric maps, win rates are very low, particularly on the `20_vs_23` scenarios. Additionally, there is no real change in difficulty as the number of agents scales. However, the maps do not seem uniformly difficult, with all algorithms struggling with the Zerg scenarios more than other maps. Reassuringly the open-loop method cannot learn a good policy on any maps – even those where QMIX and MAPPO both achieve high win rates. This is strong evidence that SMAC's lack of stochasticity has been addressed.

6.2 Analysis of Changes in SMACv2

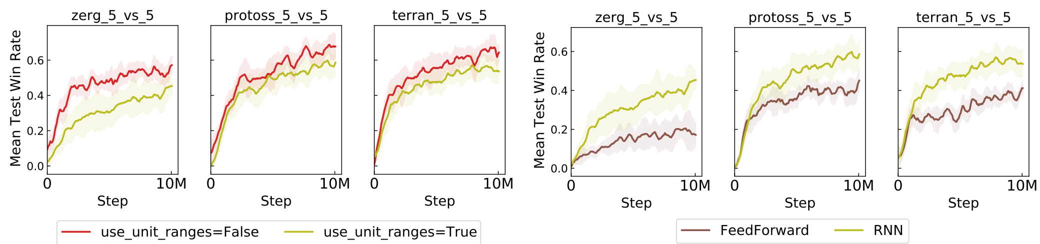
To investigate the impact of the changes introduced in SMACv2, we perform three ablation studies. We focus only on the 5 unit scenarios, and only train MAPPO on the ablations because it is significantly faster to train than QMIX. We use the same hyperparameters and setup as in Section 6.1. We ablate different team compositions by using a single unit type for each of the non-special units in each race. We investigate random start positions by only using the *surround* or *reflect* scenarios and by removing the random start positions entirely. We also ablate the effect of using the true unit ranges. Additionally, we investigate partial observability by comparing feed-forward and recurrent networks.

The results of these experiments are shown in Figure 6. The position ablations show that stochastic starting positions and hence stochasticity itself contributes greatly to the challenge of SMACv2. Without random start positions, MAPPO achieves win rates of close to 100%. Both *surround* and *reflect* scenarios have a similar win rate, suggesting similar difficulty. There does not appear to be additional difficulty from combining the two. This is logical as the scenarios can be disambiguated using start positions at the beginning of the episode.



(a) Position ablations

(b) Unit type ablations



(c) Field-of-view ablations

(d) Feed-forward vs RNN ablation

Figure 6: Ablation of SMACv2 unit type, field-of-view and starting position features. Figure 6d shows the performance of a feed-forward policy compared to an RNN baseline. The unit weights are relative to a fixed order of units. For Zerg this is zergling, hydralisk, baneling, for Terran it is marine, marauder, medivac and for Protoss it is stalkers, zealots and colossi. Plots show mean and standard deviation across three seeds.

Our experiments of unit type ablations, shown in Figure 6b, indicate that unit variety significantly impacts the difficulty of the tasks. All three races show large differences between the easiest unit distribution and the baseline, suggesting that a diverse range of units contributes to SMACv2's difficulty. In both the Zerg and Protoss scenarios, the melee-only scenarios are easier than the ablation with ranged units. From observing episodes, the enemy AI tends to aggressively pursue allied units, which allows the allies to lure it out of position. This is easier to exploit with melee units than ranged ones. Terran scenarios have no melee units, but the marine scenarios are slightly easier. Overall these results show that generalising to different unit types is a significant part of the challenge in SMACv2.

Our field-of-view ablations, shown in Figure 6c, compare the fixed unit sight and attack ranges to the SMACv2 versions. There is an increase in difficulty from varying the unit ranges, but this effect is small. This suggests that the difficulty of SMACv2 is better explained by the diversity of start positions and unit types than the change to the true sight and attack ranges.

Finally, to evaluate the effect that the changes had on the partial observability of SMACv2, we compare a feedforward network with the performance of the RNN baseline, shown in Figure 6d. There is a significant decrease in performance from using the feedforward network. However, the feedforward network can still learn reasonable policies. This suggests that although being able to resolve partial observability is useful in SMACv2, the primary difficulty consists in generalising to a range of micro-management scenarios. The zerg scenario has a larger relative performance difference. Partial observability may be more of a problem here because the splash damage done to allies by banelings creates an incentive for allies to spread out more.

Overall, the results in Figures 6a and 6b show that current MARL methods struggle with the increased stochasticity due to map randomisation. To solve this problem, agents must be able to use their observations to make inferences about relevant state or joint observation features in the Dec-POMDP in more general settings. Current inference capabilities could be improved by research on more effective sequence models, such as specialised transformers [46, 15, 4]. The work in this area either applies transformer models to traditional RL methods [14], or focuses on paradigms similar to upside-down RL [38, 4]. While the latter work shows promising generalisation [33], it is mostly in the offline setting. Hu et al. [13] demonstrate a promising sequence model that can be applied to any MARL algorithm, but only demonstrate their work on marine units, and rely on the Dec-POMDP

Table 1: Summary values for *everything* and *nothing* masks for SMACv2 5 unit scenarios and some SMAC maps. ϵ_{rmse} = root mean squared error, $\bar{Q}$ = mean Q-value, and ϵ_{abs} = mean absolute error.

Map	Mask	$\bar{Q}$	ϵ_{rmse}	$\epsilon_{\text{rmse}}/\bar{Q}$	ϵ_{abs}
`terran_5_vs_5`	everything	8.2 ± 0.3	2.0 ± 0.1	0.243 ± 0.009	1.6 ± 0.1
	nothing	8.2 ± 0.4	0.78 ± 0.03	0.0945 ± 0.0007	0.56 ± 0.03
`protoss_5_vs_5`	everything	8.4 ± 0.2	1.50 ± 0.05	0.179 ± 0.006	1.18 ± 0.04
	nothing	8.4 ± 0.3	0.50 ± 0.03	0.060 ± 0.006	0.37 ± 0.03
`zerg_5_vs_5`	everything	7.2 ± 0.4	2.41 ± 0.05	0.33 ± 0.02	1.86 ± 0.04
	nothing	7.3 ± 0.4	1.14 ± 0.02	0.157 ± 0.007	0.81 ± 0.02
`3s5z_vs_3s6z`	everything	6.97 ± 0.03	0.61 ± 0.01	0.087 ± 0.001	0.418 ± 0.006
	nothing	6.96 ± 0.04	0.286 ± 0.008	0.041 ± 0.001	0.161 ± 0.007
`5m_vs_6m`	everything	8.1 ± 0.9	1.71 ± 0.08	0.21 ± 0.02	1.24 ± 0.09
	nothing	8.1 ± 1.00	1.11 ± 0.05	0.14 ± 0.02	0.71 ± 0.04

having a certain structure. Expanding such work to handle more diverse scenarios is an interesting avenue of future work.

Additionally, although we have addressed a lack of stochasticity in SMAC, we believe that SMAC is also insufficiently partially observable to require communication via actions, which is an important component of the challenge of Dec-POMDPs. For this to be necessary, an agent's ally must have access to a feature that the agent does not know, but needs to know to decide its action. Adding a more explicit requirement of communication via actions to SMAC is an interesting option for future work – presently agents learn to group up, whereupon the partial observability is resolved.

6.3 SMACv2 Feature Inferrability & Relevance

In this section we repeat the experiments from Section 4.2 on the 5 unit SMACv2 scenarios. We use three previously trained QMIX policies to generate 3 different datasets for training, and compute error bars in the regression across these three policies. Table 1 summarises the results. The full results are given in Figure 9 in the Appendix.

When comparing results between SMAC and SMACv2, we focus on the difference between the *everything* (all features not visible) and *nothing* (no features hidden) masks. This is to account for some scenarios having much higher errors in the *nothing* mask than others. The SMACv2 scenarios have higher errors than any of the SMAC scenarios when subtracting $\frac{\epsilon_{\text{rmse}}}{\bar{Q}}$ for the *everything* and *nothing* masks. Additionally, the difference between the root-mean-squared error between the *everything* and *nothing* masks is higher for the SMACv2 maps. These results suggest that SMACv2 is significantly more stochastic than SMAC. Second, examining stepwise results in the appendix shows that SMACv2 maps have similar patterns of feature importance to SMAC maps. Enemy health is the most important individual feature and is important throughout an episode for all scenarios tested. Importantly, both the ally and enemy all masks reach significant thresholds of the average Q-value, suggesting that it is important in SMACv2 for methods to attend to environment features.

7 Conclusion

Using a set of novel experimental evaluations, we showed that the popular SMAC benchmark for cooperative MARL suffers from a lack of stochasticity. We then proposed SMACv2 and repeated these evaluations, demonstrating significant alleviation of this problem. We also took a number of baselines that achieve strong performance on SMAC and used them to demonstrate that current methods struggle to achieve high win rates on the new scenarios in SMACv2. We then performed ablations and discovered that the stochasticity of the unit types and random start positions combine to explain the tasks' difficulty. We hope that SMACv2 helps further MARL research as a significantly challenging domain capturing practical challenges.

References

[1] Torchcraftai: A bot platform for machine learning research on starcraft: Brood war. `https://github.com/TorchCraft/TorchCraftAI`, 2018. GitHub repository.

[2] 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.

[3] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f5b1b89d98b7286673128a5fb112cb9a-Paper.pdf`.

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[5] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[6] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2048–2056, 2020.

[7] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[8] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alch'e Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/07a9d3fed4c5ea6b17e80258dee231fa-Paper.pdf`.

[9] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf`.

[10] Rihab Gorsane, Omayma Mahjoub, Ruan de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *arXiv preprint arXiv:2209.10485*, 2022.

[11] Tarun Gupta, Anuj Mahajan, Bei Peng, Wendelin Böhmer, and Shimon Whiteson. Uneven: Universal value exploration for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 3930–3941. PMLR, 2021.

[12] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and SW Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*, 2021.

[13] Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. Updet: Universal multi-agent rl via policy decoupling with transformers. In *International Conference on Learning Representations*, 2020.

[14] Shariq Iqbal, Christian A Schroeder De Witt, Bei Peng, Wendelin Boehmer, Shimon Whiteson, and Fei Sha. Randomized entity-wise factorization for multi-agent reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4596–4606. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/iqbal21a.html`.

[15] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34, 2021.

[16] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning. In *IJCAI*, 2019.

[17] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Procedural level generation improves generality of deep reinforcement learning. *CoRR*, abs/1806.10729, 2018.

[18] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning, 2022.

[19] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4501–4510, 2020.

[20] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[21] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019. URL `http://arxiv.org/abs/1908.09453`.

[22] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas, 2017. URL `https://arxiv.org/abs/1702.03037`.

[23] Joel Z. Leibo, Edgar Dué nez Guzmán, Alexander Sasha Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charles Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. PMLR, 2021.

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017.

[25] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in Neural Information Processing Systems*, 32, 2019.

[26] Anuj Mahajan, Mikayel Samvelyan, Lei Mao, Viktor Makoviychuk, Animesh Garg, Jean Kossaifi, Shimon Whiteson, Yuke Zhu, and Animashree Anandkumar. Tesseract: Tensorised actors for multi-agent reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7301–7312. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/mahajan21a.html`.

[27] Anuj Mahajan, Mikayel Samvelyan, Tarun Gupta, Benjamin Ellis, Mingfei Sun, Tim Rocktäschel, and Shimon Whiteson. Generalization in cooperative multi-agent systems. *arXiv preprint arXiv:2202.00104*, 2022.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[29] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276.

[30] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. FACMAC: Factored multi-agent centralised policy gradients. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=WxH774N0mEu.

[31] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[32] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10199–10210. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/73a427badebe0e32caa2e1fc7530b7f3-Paper.pdf.

[33] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[34] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground, 2018.

[35] Sebastian Risi and Julian Togelius. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence*, 2, 08 2020. doi: 10.1038/s42256-020-0208-z.

[36] 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. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

[37] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[38] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[39] 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.

[40] Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents, 2019.

[41] Mingfei Sun, Sam Devlin, Katja Hofmann, and Shimon Whiteson. Monotonic improvement guarantees under non-stationarity for decentralized ppo. *arXiv preprint arXiv:2202.00082*, 2022.

[42] Mingfei Sun, Vitaly Kurin, Guoqing Liu, Sam Devlin, Tao Qin, Katja Hofmann, and Shimon Whiteson. You may not need ratio clipping in ppo. *arXiv preprint arXiv:2202.00079*, 2022.

[43] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

[44] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games, 2016. URL https://arxiv.org/abs/1611.00625.

[45] Justin K Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[47] Eugene Vinitsky, Natasha Jaques, Joel Leibo, Antonio Castenada, and Edward Hughes. An open source implementation of sequential social dilemma games. `https://github.com/eugenevinitsky/sequential_social_dilemma_games/issues/182`, 2019. GitHub repository.

[48] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017. URL `https://arxiv.org/abs/1708.04782`.

[49] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z.

[50] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2020.

[51] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Roma: Multi-agent reinforcement learning with emergent roles. *arXiv preprint arXiv:2003.08039*, 2020.

[52] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. In *International Conference on Learning Representations*, 2020.

[53] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5571–5580. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/yang18d.html`.

[54] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

A Links

- SMACv2: https://github.com/oxwhirl/smacv2. MIT License.
- MAPPO implementation: https://github.com/benellis3/mappo.
- QMIX implementation: https://github.com/benellis3/pymarl2. Apache-2 License.

B Further Background

B.1 StarCraft

StarCraft II is a real-time strategy game featuring 3 different races, Protoss, Terran and Zerg, with different properties and associated strategies. The objective is to build an army powerful enough to destroy the enemy's base. When battling two armies, players must ensure army units are acting optimally. This is called *micromanagement*. An important micromanagement strategy is *focus firing*, which is ordering all allied units to jointly target the enemies one by one to ensure that damage taken is minimised.

Another important strategy is *kiting*, where units flee from the enemy and then pick them off one by one as they chase.

B.2 QMIX

QMIX can be thought of as an extension of DQN[28] to the Dec-POMDP setting. The joint optimal action is found by forcing the joint Q to adhere to the individual global max (IGM) principle[39], which states that the joint action can be found by maximising individual agents' Q_i functions:

$$\arg\max_{\boldsymbol{a}} Q(s, \boldsymbol{\tau}, \boldsymbol{a}) = \begin{cases} \arg\max_a Q_1(\tau_1, a_1) \\ \arg\max_a Q_2(\tau_2, a_2) \\ \cdots \\ \arg\max_a Q_n(\tau_n, a_n) \end{cases}$$

This central Q is trained to regress to a target $r + \gamma\hat{Q}(s, \boldsymbol{\tau}, \boldsymbol{a})$ where $\hat{Q}$ is a target network that is updated slowly. The central Q estimate is computed by a mixing network, whose weights are conditioned on the state, which takes as input the utility function Q_i of the agents. The weights of the mixing network are restricted to be positive, which enforces the IGM principle[39] by ensuring the central Q is monotonic in each Q_i.

B.3 Independent and Multi-agent PPO

Proximal Policy Optimisation (PPO) is a method initially developed for single-agent reinforcement learning which aims to address performance collapse in policy gradient methods. It does this by heuristically bounding the ratio of action probabilities between the old and new policies. To this end, it optimises the below objective function.

$$\mathbb{E}_{s \sim d^\pi, a \sim \pi}\left[\min\left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)} A^\pi(s,a), \text{clip}\left(\frac{\tilde{\pi}(a|s)}{\pi(a|s)}, 1-\epsilon, 1+\epsilon\right) A^\pi(s,a)\right)\right]$$

where $\text{clip}(t, a, b)$ is a function that outputs a if $t < a$, b if $t > b$ and t otherwise.

Extending this to the multi-agent setting can easily done in two ways. The first is to use independent learning, where each agent treats the others as part of the environment and learns a critic using its observation and action history. The second is to use a centralised critic conditioned on the central state. This is called multi-agent PPO. Both independent PPO (IPPO) and multi-agent PPO (MAPPO) have demonstrated strong performance on SMAC[54, 7]. Note that we do not apply the observation and state changes suggested by Yu et al. [54] anywhere in this paper. This is because these changes were not implemented as a wrapper on top of SMAC, but instead by modifying SMAC directly, leading to the environment code becoming unmanageably complicated.

C Experimental Details

In this section we describe extra details of the experiments run as part of the paper.

C.1 Stochasticity

This section describes details of the experiments run in Section 4.1 in the main paper. Both the closed-loop and open-loop algorithms were based on the MAPPO implementation from Sun et al.

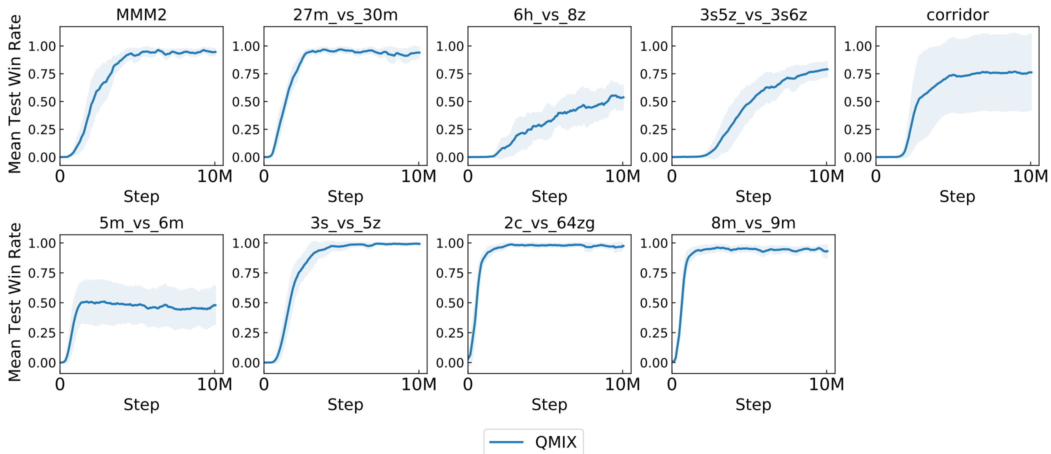


Figure 7: QMIX training results on SMAC

[41]. The code used for these experiments can be found here. Both the open-loop and closed-loop algorithms use the same neural network architecture as in [41]. Both were also provided with access to the available actions mask of the environment and conditioned the critic on the state. We used the hyperparameters from Sun et al. [41], with the exception of the actor's learning rate, which we set to 0.0005 and decayed linearly throughout training. This is because learning rate decay has been found to be important for bounding PPO's policy updates [42]. Full hyperparameters can be found in Table 6.

C.2 SMAC Feature Inferrability & Relevance

This section describes details of the experiments run in Section 4.2 and Section 6.3 in the main paper. The code used for these experiments can be found here.

For each scenario, we had 3 QMIX policies trained using the implementation and hyperparameters from Hu et al. [12], each with a different network and SMAC initialization. Training results for the policies for SMAC are shown in Figure 7. Each policy constitutes a seed and is used to collect a dataset of episodes for the feature-relevance experiment on the scenario. We call these policies *expert policies*. QMIX hyperparameters used are given in Table 5. A dataset consists of two folds: 8192 episodes used for training and 4096 episodes used for evaluation.

For a given mask, the experiment then consists of training a new QMIX network, termed *regression network*, whose input is trajectories with observations and states masked according to the mask, and whose task is to predict the Q-values output by the expert policy on the unmasked trajectories.

The *nothing* mask does not apply any effect on the observations and states in a trajectory. Otherwise, a mask, say 'health (ally)', masks the health feature of every ally in the observation of each agent (except its own health) and masks the health feature of all the units controlled by QMIX in the state. Table 2 shows the feature sets zeroed out for different masks.

The regression network has the same architecture as the expert network. We use the mean squared error (MSE) as a loss function and optimise the network via Adam with a batch size of 512 episodes and learning rate equal to 0.005. (The other Adam hyper-parameters are the default PyTorch values.) Training is performed with early stopping according to the validation fold with an evaluation every 5 epochs and a patience of 10 tries (i.e. training is stopped when the MSE on the validation fold does not decrease in 50 consecutive epochs and performance at the best epoch is retained). Models for all masks and scenarios hit early stopping and trained for about 200 epochs on average, except a few ones which trained to a limit of 500 epochs.

All hyper-parameters were tuned on a few different scenarios from both SMAC and SMACv2 to minimise validation MSE using datasets and expert policies not used for the reported results. The sizes of the training and validation folds are respectively 1.6 times and 0.8 times the size of the QMIX

Table 2: Masked features for each setting of the feature quality experiment. ✓means a feature is masked and ✗means a feature is not masked.

Mask	Ally						Enemy				
	Health	Shield	x	y	Distance	Actions	Health	Shield	x	y	Distance
everything	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
nothing	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
health (ally)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
shield (ally)	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
distance (ally)	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
health and shield (ally)	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
actions only	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
all except actions	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
ally all	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
health (enemy)	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
shield (enemy)	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
distance (enemy)	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
enemy all	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓

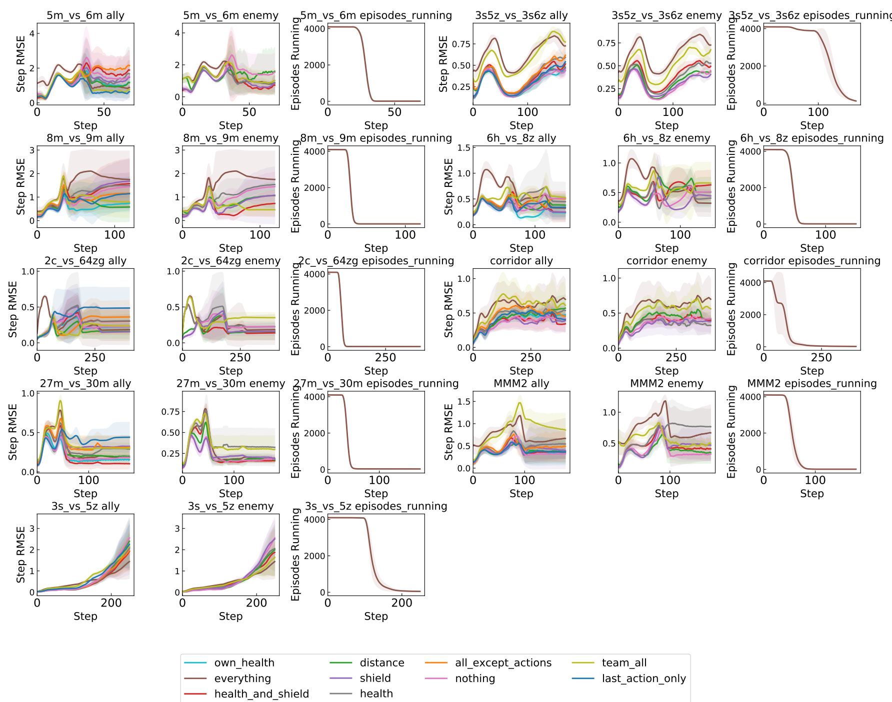


Figure 8: Results of the feature quality experiments for all tested SMAC scenarios not in the main paper.

replay buffer (5000 episodes) and have been chosen to be large enough to minimize validation MSE while keeping experiments practical.

Figure 8 shows the full results of the feature quality experiments for SMAC and Figure 9 shows the full results for SMACv2. Table 4 shows summary results for the *everything* and *nothing* masks for SMACv2 and Table 3 shows the same data for SMAC.

Experiments for this section were conducted on 80-core CPU machines with NVIDIA GeForce RTX 2080 Ti or Tesla V100-SXM2-16GB GPUs. A single (scenario, seed, mask) combination took around 1 hour to train. In total, the experiments in this section took about 1500 GPU hours.

Table 3: Mean Q values in the different feature quality experiments on SMAC scenarios

Map	Mask	$\bar{Q}$	ϵ_{rmse}	$\frac{\epsilon_{\text{rmse}}}{Q}$	ϵ_{abs}
`5m_vs_6m`	everything	8.1419 ± 0.9	1.7084 ± 0.08	0.2116 ± 0.02	1.2395 ± 0.09
	nothing	8.1084 ± 1.00	1.1121 ± 0.05	0.1399 ± 0.02	0.7077 ± 0.04
`3s5z_vs_3s6z`	everything	6.9666 ± 0.03	0.6092 ± 0.01	0.0874 ± 0.001	0.4183 ± 0.006
	nothing	6.9608 ± 0.04	0.2861 ± 0.008	0.0411 ± 0.001	0.1611 ± 0.007
`8m_vs_9m`	everything	12.2900 ± 0.2	0.7489 ± 0.07	0.0610 ± 0.006	0.4626 ± 0.06
	nothing	12.3225 ± 0.2	0.4381 ± 0.07	0.0356 ± 0.006	0.2445 ± 0.04
`6h_vs_8z`	everything	7.4810 ± 0.8	0.8674 ± 0.09	0.1182 ± 0.02	0.6493 ± 0.09
	nothing	7.4961 ± 0.8	0.3847 ± 0.03	0.0522 ± 0.009	0.2654 ± 0.03
`2c_vs_64zg`	everything	7.7224 ± 0.08	0.5568 ± 0.02	0.0721 ± 0.002	0.4229 ± 0.01
	nothing	7.7195 ± 0.09	0.1375 ± 0.03	0.0178 ± 0.004	0.0729 ± 0.02
`corridor`	everything	$7.0455 \pm 1e+00$	0.5491 ± 0.08	0.0783 ± 0.002	0.3888 ± 0.05
	nothing	$7.0641 \pm 1e+00$	0.2541 ± 0.02	0.0365 ± 0.003	0.1636 ± 0.006
`27m_vs_30m`	everything	9.6275 ± 0.4	0.4906 ± 0.04	0.0512 ± 0.006	0.3317 ± 0.03
	nothing	9.6409 ± 0.4	0.3398 ± 0.05	0.0355 ± 0.006	0.2088 ± 0.04
`MMM2`	everything	9.3504 ± 0.4	0.7080 ± 0.1	0.0763 ± 0.02	0.4704 ± 0.09
	nothing	9.3665 ± 0.4	0.3499 ± 0.05	0.0377 ± 0.007	0.2014 ± 0.04
`3s_vs_5z`	everything	8.0589 ± 0.2	0.2975 ± 0.04	0.0371 ± 0.007	0.1989 ± 0.03
	nothing	8.0562 ± 0.2	0.2006 ± 0.02	0.0250 ± 0.003	0.0783 ± 0.01

Table 4: SMACv2 feature quality experiment results

Map	Mask	$\bar{Q}$	ϵ_{rmse}	$\frac{\epsilon_{\text{rmse}}}{Q}$	ϵ_{abs}	$\frac{\epsilon_{\text{rmse}}^{\text{everything}} - \epsilon_{\text{rmse}}^{\text{nothing}}}{Q}$
`5_gen_terran`	everything	8.2011 ± 0.3	1.9918 ± 0.1	0.2427 ± 0.009	1.5676 ± 0.1	0.1482 ± 0.008
	nothing	8.2023 ± 0.4	0.7753 ± 0.03	0.0945 ± 0.0007	0.5616 ± 0.03	
`5_gen_protoss`	everything	8.3641 ± 0.2	1.4973 ± 0.05	0.1791 ± 0.006	1.1779 ± 0.04	0.1189 ± 0.004
	nothing	8.4059 ± 0.3	0.5021 ± 0.03	0.0599 ± 0.006	0.3652 ± 0.03	
`5_gen_zerg`	everything	7.2294 ± 0.4	2.4087 ± 0.05	0.3339 ± 0.02	1.8556 ± 0.04	0.1762 ± 0.009
	nothing	7.2756 ± 0.4	1.1383 ± 0.02	0.1569 ± 0.007	0.8117 ± 0.02	

C.3 SMACv2 Runs

Here we describe the hyperparameter and training procedure used in section 6.1. As in previous experiments, we used the implementation by Hu et al. [12] for QMIX and by Sun et al. [41] for MAPPO. These implementations have both achieved very strong results on SMAC. We therefore tuned the hyperparameters by taking deviations from these for a few key parameters. For MAPPO, we tuned the actor's learning rate and the clipping range. Additionally, we added linear learning rate decay to the MAPPO implementation because this has been shown to be important to bound the policy update[42]. For QMIX, we tuned the ϵ-annealing time and λ in the eligibility trace $Q(\lambda)$. All other hyperparameters, including neural network architecture, were unchanged from the implementations mentioned previously.

The MAPPO code for these experiments can be found here, and the QMIX code here. The QMIX code is distributed under the Apache license, and the MAPPO code under the MIT license. The only differences in these branches and the implementations used for previous experiments on SMAC are important to changing the environment from SMAC to SMACv2.

The open-loop policy is identical to MAPPO, except the policy receives as input only the timestamp and agent ID, as in Section 4.1.

We tuned hyperparameters by running each set of hyperparameters on all scenarios and then choosing the best results. The grids of hyperparameters for QMIX and MAPPO can be found in Table 8 and

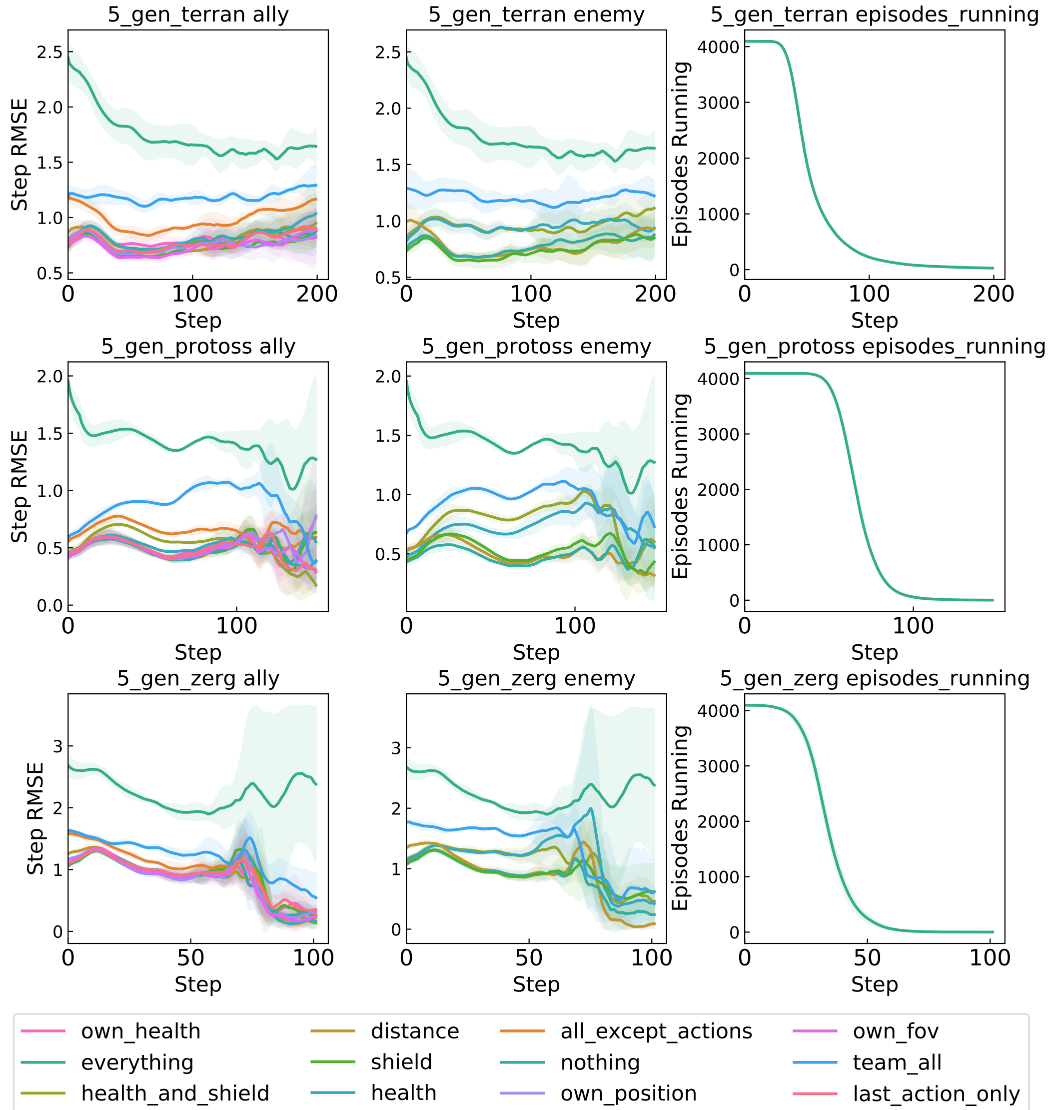


Figure 9: Feature Quality Experiments for the 5 and 10 unit SMACv2 scenarios.

9. All algorithms were trained for 10M environment steps with evaluations of 20 episodes every 2k steps. The hyperparameters used for QMIX are given in Table 5, and for MAPPO in Table 6.

For the ablations in Section 6.2, we used the same hyperparameters as in Section 6.1. The code for the ablations can be found here. This is distributed under an MIT license. The changes on this branch enable easy running of the ablations and add associated environment configurations for this purpose.

D SMACv2 Additional Details

In this section we describe the generation of teams in SMACv2. For each race, 3 different types of units are used. Each race has a special unit that should not be generated too often. For Protoss this is the colossus. This is a very powerful unit. If a team has many colossi, the battle will devolve into a war about who can use their colossi most effectively. Terran has the medivac unit. This is a healing unit that cannot attack the enemy and so is only spawned sparingly. Zerg has the baneling unit. This is a suicidal unit which deals area-of-effect damage to enemy units by rolling into them and exploding. In scenarios with many banelings, the agents learn to spread out and hide in the corners with the hope that the enemy banelings explode and the allies win by default. All of these special

Table 5: QMIX hyperparameters used for experiments. Parameters with (SMAC) or (SMACv2) after them denote that parameter setting was only used for SMAC or SMACv2 experiments respectively. These are the values in the corresponding configuration file in PyMarl[36, 12]. Mac is the code responsible for marshalling inputs to the neural networks, learner is the code used for learning and runner determines whether experience is collected in serial or parallel.

Parameter Name	Value
Action Selector	epsilon greedy
ϵ Start	1.0
ϵ Finish	0.05
ϵ Anneal Time	100000
Runner	parallel
Batch Size Run	4
Buffer Size	5000
Batch Size	128
Optimizer	Adam
$t_{\max}$	10050000
Target Update Interval	200
Mac	n_mac
Agent	n_rnn
Agent Output Type	q
Learner	nq_learner
Mixer	qmix
Mixing Embed Dimension	32
Hypernet Embed Dimension	64
Learning Rate	0.001
λ (SMAC)	0.6
λ (SMACv2)	0.4

units are spawned with a probability of 10%. The other units used spawn with a probability of 45%. This is summarised in Table 10.

There are two changes to the observation space from SMAC. First, each agent observes their own field-of-view direction. Secondly, each agent observes their own position in the map as x- and y-coordinates. This is normalised by dividing by the map width and height respectively. The only change to the state from SMAC was to add the field-of-view direction of each agent to the state.

Additionally, we made one small change to the reward function in SMACv2. This was to fix a bug where the enemies healing can give allied units reward. There are more details available about this problem in the associated Github issue. SMACv2 also has an identical API to the original SMAC, allowing for very simple transition between the two frameworks.

The code for SMACv2 can be found in the Github repo, where there is a README detailing how to run the benchmark with random agents.

E Limitations

The main limitation of SMACv2 is that it is confined to scenarios within the game of StarCraft II. This is an environment which, while complex, cannot represent the dynamics of all multi-agent tasks. Evaluation of MARL algorithms therefore should not be limited to one benchmark, but should target variety with a range of tasks.

Whilst the scenarios in SMACv2 involve battles between two armies of units, only one side can be controlled by RL agents. It is technically possible to control two armies using two StarCraft II clients that communicate via LAN, which would allow to train two groups of decentralised agents against each other, e.g., via self-play. We leave the implementation of this functionality as future work.

Table 6: MAPPO hyperparameters used for the experiments on SMAC and SMACv2. Parameters with (SMAC) or (SMACv2) after them denote that parameter setting was only used for SMAC or SMACv2 experiments respectively.

Hyperparameter	Value
Action Selector	multinomial
Mask Before Softmax	True
Runner	parallel
Buffer Size	8
Batch Size Run	8
Batch Size	8
Target Update Interval	200
Learning Rate Actor (SMAC)	0.001
Learning Rate Actor (SMACv2)	0.0005
Learning Rate Critic	0.001
τ	0.995
λ	0.99
Agent Output Type	pi_logits
Learner	trust_region_learner
Critic	centralV_rnn_critic
Detach Every	20
Replace Every	None
Mini Epochs Actor	10
Mini Epochs Critic	10
Entropy Loss Coeff	0.0
Advantage Calc Method	GAE
Bootstrap Timeouts	False
Surrogate Clipped	True
Clip Range	0.1
Is Advantage Normalized	True
Observation Normalized	True
Use Popart	False
Normalize Value	False
Obs Agent Id	True
Obs Last Action	False

Table 7: Hyperparameters used for DDN in the experiments.

Hyperparameter	Value
Action Selector	epsilon_greedy
Epsilon Start	1.0
Epsilon Finish	0.05
Epsilon Anneal Time	50000
Runner	episode
Buffer Size	5000
Target Update Interval	200
Agent Output Type	q
Learner	iqn_learner
Double Q	False
Mixer	ddn
Name	ddn
Agent	iqn_rnn
Optimizer	Adam
Quantile Embed Dim	64
N Quantiles	1
N Target Quantiles	1
N Approx Quantiles	32

Table 8: QMIX hyperparameter tuning grid

Hyperparameter	Values Tried
ϵ annealing time	$[100 \times 10^3, 500 \times 10^3]$
λ	[0.6, 0.8, 0.4]

Table 9: MAPPO hyperparameter tuning grid

Hyperparameter	Value
Actor Learning Rate	$[0.0007, 0.0004, 0.0001]$
Clip Range	$[0.15, 0.05, 0.1]$

Table 10: Unit types used per-race in the SMACv2 scenarios.

Race	Unit Types	Probability of Generation
Terran	Marine	0.45
	Marauder	0.45
	Medivac	0.1
Zerg	Zergling	0.45
	Baneling	0.1
	Hydralisk	0.45
Protoss	Stalker	0.45
	Zealot	0.45
	Colossus	0.1