The StarCraft Multi-Agent Challenge

Mikayel Samvelyan [*1] Tabish Rashid [*2] Christian Schroeder de Witt [2] Gregory Farquhar [2] Nantas Nardelli [2]
Tim G. J. Rudner [2] Chia-Man Hung [2] Philip H. S. Torr [2] Jakob Foerster [2] Shimon Whiteson [2]

Abstract

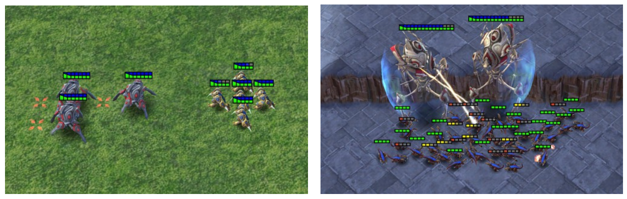
In the last few years, deep multi-agent reinforcement learning (RL) has become a highly active area of research. A particularly challenging class of problems in this area is partially observable, cooperative, multi-agent learning, in which teams of agents must learn to coordinate their behaviour while conditioning only on their private observations. This is an attractive research area since such problems are relevant to a large number of real-world systems and are also more amenable to evaluation than general-sum problems.

Standardised environments such as the ALE and MuJoCo have allowed single-agent RL to move beyond toy domains, such as grid worlds. However, there is no comparable benchmark for cooperative multi-agent RL. As a result, most papers in this field use one-off toy problems, making it difficult to measure real progress. In this paper, we propose the StarCraft Multi-Agent Challenge (SMAC) as a benchmark problem to fill this gap.[1] SMAC is based on the popular real-time strategy game StarCraft II and focuses on micromanagement challenges where each unit is controlled by an independent agent that must act based on local observations. We offer a diverse set of challenge maps and recommendations for best practices in benchmarking and evaluations. We also open-source a deep multi-agent RL learning framework including state-of-the-art algorithms.[2] We believe that SMAC can provide a standard benchmark environment for years to come.

Videos of our best agents for several SMAC scenarios are available at: https://youtu.be/VZ7zmQ_obZ0.

*Equal contribution [1]Russian-Armenian University, Armenia [2]University of Oxford, United Kingdom. Correspondence to: Mikayel Samvelyan <mikayel@samvelyan.com>, Tabish Rashid <tabish.rashid@cs.ox.ac.uk>.

[1]Code is available at https://github.com/oxwhirl/smac
[2]Code is available at https://github.com/oxwhirl/pymarl



(a) 3 Stalkers vs 5 Zealots (b) 2 Colossi vs 64 Zerglings

Figure 1. Decentralised micromanagement in StarCraft II. Each unit is an independent learning agent that needs to coordinate with its teammates to defeat the enemy units.

1. Introduction

Deep reinforcement learning (RL) promises a scalable approach to solving arbitrary sequential decision making problems, demanding only that a user must specify a reward function that expresses the desired behaviour. However, many real-world problems that might be tackled by RL are inherently multi-agent in nature. For example, the coordination of self-driving cars, autonomous drones, and other multi-robot systems is becoming increasingly critical. Network traffic routing, distributed sensing, energy distribution, and other logistical problems are also inherently multi-agent. As such, it is essential to develop multi-agent RL (MARL) solutions that can handle decentralisation constraints and deal with the exponentially growing joint action space of many agents.

Partially observable, cooperative, multi-agent learning problems are of particular interest. Cooperative problems avoid difficulties in evaluation inherent with general-sum games (e.g., which opponents are evaluated against). Cooperative problems also map well to a large class of critical problems where a single user that manages a distributed system can specify the overall goal, e.g., minimising traffic or other inefficiencies. Most real-world problems depend on inputs from noisy or limited sensors, so partial observability must also be dealt with effectively. This often includes limitations on communication that result in a need for decentralised execution of learned policies. However, there commonly is access to additional information during training, which may be carried out in controlled conditions or in simulation.

A growing number of recent works (Foerster et al., 2018a; Rashid et al., 2018; Sunehag et al., 2017; Lowe et al., 2017) have begun to address the problems in this space. However, there is a clear lack of standardised benchmarks for research and evaluation. Instead, researchers often propose one-off environments which can be overly simple or tuned to the proposed algorithms. In single-agent RL, standard environments such as the Arcade Learning Environment (Bellemare et al., 2013), or MuJoCo for continuous control (Plappert et al., 2018), have enabled great progress. In this paper, we aim to follow this successful model by offering challenging standard benchmarks for deep MARL, and to facilitate more rigorous experimental methodology across the field.

Some testbeds have emerged for other multi-agent regimes, such as Poker (Heinrich & Silver, 2016), Pong (Tampuu et al., 2015), Keepaway Soccer(Stone et al., 2005), or simple gridworld-like environments (Lowe et al., 2017; Leibo et al., 2017; Yang et al., 2018; Zheng et al., 2017). Nonetheless, we identify a clear gap in challenging and standardised testbeds for the important set of domains described above.

To fill this gap, we introduce the StarCraft Multi-Agent Challenge (SMAC). SMAC is built on the popular real-time strategy game StarCraft II[3] and makes use of the SC2LE environment (Vinyals et al., 2017). Instead of tackling the full game of StarCraft with centralised control, we focus on decentralised micromanagement challenges (Figure 1). In these challenges, each of our units is controlled by an independent, learning agent that has to act based only on local observations, while the opponent's units are controlled by the hand-coded built-in StarCraft II AI. We offer a diverse set of scenarios that challenge algorithms to handle high-dimensional inputs and partial observability, and to learn coordinated behaviour even when restricted to fully decentralised execution.

The full games of StarCraft: BroodWar and StarCraft II have already been used as RL environments, due to the many interesting challenges inherent to the games (Synnaeve et al., 2016; Vinyals et al., 2017). DeepMind's AlphaStar (DeepMind, 2019) has recently shown an impressive level of play on a StarCraft II matchup using a centralised controller. In contrast, SMAC is not intended as an environment to train agents for use in full StarCraft II gameplay. Instead, by introducing strict decentralisation and local partial observability, we use the StarCraft II game engine to build a new set of rich cooperative multi-agent problems that bring unique challenges, such as the nonstationarity of learning (Foerster et al., 2017), multi-agent credit assignment (Foerster et al., 2018a), and the difficulty of representing the value of joint actions (Rashid et al., 2018).

[3]StarCraft II is the sequel to the game StarCraft and its expansion set Brood War. StarCraft and StarCraft II are trademarks of Blizzard Entertainment^{TM}.

To further facilitate research in this field, we also open-source PyMARL, a learning framework that can serve as a starting point for other researchers and includes implementations of several key MARL algorithms. PyMARL is modular, extensible, built on PyTorch, and serves as a template for dealing with some of the unique challenges of deep MARL in practice. We include results on our full set of SMAC environments using QMIX (Rashid et al., 2018) and several baseline algorithms, and challenge the community to make progress on difficult environments in which good performance has remained out of reach so far. We also offer a set of guidelines for best practices in evaluations using our benchmark, including the reporting of standardised performance metrics, sample efficiency, and computational requirements.

We hope SMAC will serve as a valuable standard benchmark, enabling systematic and robust progress in deep MARL for years to come.

2. Related Work

Much work has gone into designing environments to test and develop MARL agents. However, not many of these focused on providing a qualitatively challenging environment that would provide together elements of strong partial observability, challenging dynamics, collaborative and adversarial settings, and high-dimensional observation spaces.

Stone et al. (2005) presented Keepaway soccer, a domain built on the RoboCup soccer simulator (Kitano et al., 1997), a 2D simulation of a football environment with simplified physics, where the main task consists of keeping a ball within a pre-defined area where agents in teams can reach, steal, and pass the ball, providing a simplified setup for studying cooperative MARL. This domain was later extended to the Half Field Offense task, (Kalyanakrishnan et al., 2006; Hausknecht et al., 2016), which increases the difficulty of the problem by requiring the agents to not only keep the ball within bounds but also to score a goal. Neither task scales well in difficulty with the number of agents, as most agents need to do little coordination. There is also a lack of interesting environment dynamics beyond the simple 2D physics nor good reward signals, thus reducing the impact of the environment as a testbed.

Multiple gridworld-like environments have also been explored. Lowe et al. (2017) released a set of simple gridworld like environments for multi-agent RL alongside an implementation of MADDPG, featuring a mix of competitive and cooperative tasks focused on shared communication and low level continuous control. Leibo et al. (2017) show several mixed-cooperative Markov environment focused on testing social dilemmas, however, they did not release an implementation to further explore the tasks. Yang et al.

(2018); Zheng et al. (2017) present a framework for creating gridworlds focuses on many-agents tasks, where the number of agents ranges from the hundreds to the millions. This work, however, focuses on testing for emergent behaviour, since environment dynamics and control space need to remain relatively simple for the tasks to be tractable. Resnick et al. (2018) propose a multi-agent environment based on the game *Bomberman*, encompassing a series of cooperative and adversarial tasks meant to provide a more challenging set of tasks with a relatively straightforward 2D state observation and simple grid-world-like action spaces.

Learning to play StarCraft games also has been investigated in several communities: work ranging from evolutionary algorithms to tabular RL applied has shown that the game is an excellent testbed for both modelling and planning (Ontanón et al., 2013), however, most have focused on single-agent settings with multiple controllers, and classical algorithms. More recently progress has been made on developing frameworks that enable researchers working with deep neural network to test recent algorithms on these games; work on applying deep RL algorithms to single-agent and multi-agent versions of the micromanagement tasks has thus been steadily appearing (Usunier et al., 2016; Foerster et al., 2017; 2018a; Rashid et al., 2018; Nardelli et al., 2018; Hu et al., 2018; Shao et al., 2018; Foerster et al., 2018b) with the release of TorchCraft (Synnaeve et al., 2016) and SC2LE (Vinyals et al., 2017), interfaces to respectively *StarCraft: BroodWar* and *StarCraft II*. Our work presents the first standardised testbed for decentralised control in this space.

Other work focuses on playing the full game of StarCraft, including macromanagement and tactics (Pang et al., 2018; Sun et al., 2018; DeepMind, 2019). By introducing decentralisation and local observability, our agents are excessively restricted compared to normal full gameplay. SMAC is therefore not intended as an environment to train agents for use in full gameplay. Instead, we use the StarCraft II game engine to build rich and interesting multi-agent problems.

3. Multi-Agent Reinforcement Learning

In SMAC, we focus on tasks where a team of agents needs to work together to achieve a common goal. We briefly review the formalism of such *fully cooperative multi-agent tasks* as *Dec-POMDPs* but refer readers to Oliehoek & Amato (2016) for a more complete picture.

Dec-POMDPs

Formally, a Dec-POMDP G is given by a tuple $G = \langle S, U, P, r, Z, O, n, \gamma \rangle$, where $s \in S$ is the true state of the environment. At each time step, each agent $a \in A \equiv \{1, ..., n\}$ chooses an action $u^a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. This causes a transition of the environment according to the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$.

In contrast to partially-observable stochastic games, all agents in a Dec-POMDP share the same team reward function $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$, where $\gamma \in [0, 1)$ is the discount factor. Dec-POMDPs consider *partially observable* scenarios in which an observation function $O(s, a) : S \times A \to Z$ determines the observations $z^a \in Z$ that each agent draws individually at each time step. Each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a(u^a|\tau^a) : T \times U \to [0, 1]$. The joint policy π admits a joint *action-value function*: $Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}} [R_t | s_t, \mathbf{u}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the *discounted return*.

Centralised training with decentralised execution

If learning proceeds in simulation, as is done in StarCraft II, or in a laboratory, then it can usually be conducted in a centralised fashion. This gives rise to the paradigm of *centralised training with decentralised execution*, which has been well-studied in the planning community (Oliehoek et al., 2008; Kraemer & Banerjee, 2016). Although training is centralised, execution is decentralised, i.e., the learning algorithm has access to all local action-observation histories $\boldsymbol{\tau}$ and global state s, but each agent's learnt policy can condition only on its own action-observation history τ^a.

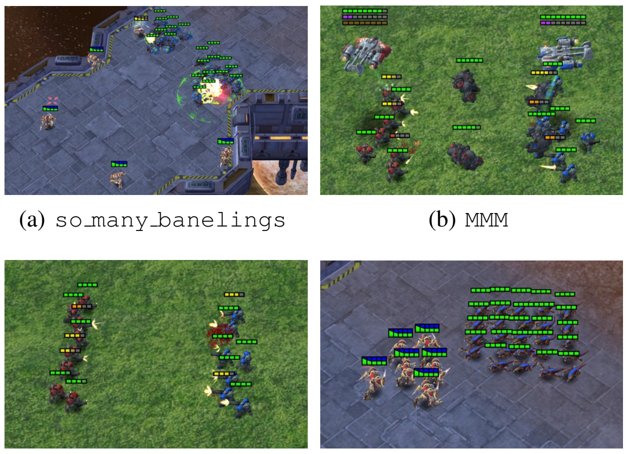
A number of recent state-of-the-art algorithms use extra state information available in centralised training to speed up the learning of decentralised policies. Among these, COMA (Foerster et al., 2018a) is an actor-critic algorithm with a special multi-agent critic baseline, and QMIX (Rashid et al., 2018) belongs to the Q-learning family.

4. SMAC

SMAC is based on the popular real-time strategy (RTS) game StarCraft II. In a regular full game of StarCraft II, one or more humans compete against each other or against a built-in game AI to gather resources, construct buildings, and build armies of units to defeat their opponents.

Akin to most RTSs, StarCraft has two main gameplay components: macromanagement and micromanagement. *Macromanagement* refers to high-level strategic considerations, such as economy and resource management. *Micromanagement* (micro), on the other hand, refers to fine-grained control of individual units.

StarCraft has been used as a research platform for AI, and more recently, RL. Typically, the game is framed as a competitive problem: an agent takes the role of a human player, making macromanagement decisions and performing micromanagement as a puppeteer that issues orders to individual



(a) `so_many_banelings` (b) `MMM`

(c) `8m` (d) `corridor`

Figure 2. Screenshots of four SMAC scenarios.

units from a centralised controller.

In order to build a rich multi-agent testbed, we instead focus solely on micromanagement. Micro is a vital aspect of StarCraft gameplay with a high skill ceiling, and is practiced in isolation by amateur and professional players. For SMAC, we leverage the natural multi-agent structure of micromanagement by proposing a modified version of the problem designed specifically for decentralised control. In particular, we require that each unit be controlled by an independent agent that conditions only on local observations restricted to a limited field of view centred on that unit. Groups of these agents must be trained to solve challenging combat scenarios, battling an opposing army under the centralised control of the game's built-in scripted AI.

Proper micro of units during battles will maximise the damage dealt to enemy units while minimising damage received, and requires a range of skills. For example, one important technique is *focus fire*, i.e., ordering units to jointly attack and kill enemy units one after another. When focusing fire, it is important to avoid *overkill*: inflicting more damage to units than is necessary to kill them.

Other common micromanagement techniques include: assembling units into formations based on their armour types, making enemy units give chase while maintaining enough distance so that little or no damage is incurred (*kiting*), coordinating the positioning of units to attack from different directions or taking advantage of the terrain to defeat the enemy. Figure 1a illustrates how three Stalker units kite five enemy Zealots units.

Learning these rich cooperative behaviours under partial observability is a challenging task, which can be used to evaluate the effectiveness of MARL algorithms.

Scenarios

SMAC consists of a set of StarCraft II micro scenarios which aim to evaluate how well independent agents are able to learn coordination to solve complex tasks. These scenarios are carefully designed to necessitate the learning of one or more micromanagement techniques to defeat the enemy. Each scenario is a confrontation between two armies of units. The initial position, number, and type of units in each army varies from scenario to scenario, as does the presence or absence of elevated or impassable terrain. Figures 1 and 2 include screenshots of several SMAC micro scenarios.

The first army is controlled by the learned allied agents. The second army consists of enemy units controlled by the built-in game AI, which uses carefully handcrafted non-learned heuristics. At the beginning of each episode, the game AI instructs its units to attack the allied agents using its scripted strategies. An episode ends when all units of either army have died or when a pre-specified time limit is reached (in which case the game is counted as a defeat for the allied agents). The goal for each scenario is to maximise the win rate of the learned policies, i.e., the expected ratio of games won to games played.

Perhaps the simplest scenarios are **symmetric** battle scenarios. The most straightforward of these scenarios are *homogeneous*, i.e., each army is composed of only a single unit type (e.g., Marines). A winning strategy in this setting would be to focus fire, ideally without overkill. *Heterogeneous* symmetric scenarios, in which there is more than a single unit type on each side (e.g., Stalkers and Zealots), are more difficult to solve. Such challenges are particularly interesting when some of the units are extremely effective against others (this is known as *countering*), for example, by dealing bonus damage to a particular armour type. In such a setting, allied agents must deduce this property of the game and design an intelligent strategy to protect teammates vulnerable to certain enemy attacks.

SMAC also includes more challenging scenarios, for example, in which the enemy army outnumbers the allied army by one or more units. In such **asymmetric** scenarios it is essential to consider the health of enemy units in order to effectively target the desired opponent.

Lastly, SMAC offers a set of interesting **micro-trick** challenges that require a higher-level of cooperation and a specific micromanagement trick to defeat the enemy. An example of such scenario is `so_many_baneling` (Figure 2a), where 7 allied Zealots face 32 enemy Baneling units. Banelings attack by running against a target and exploding when reaching them, causing damage to a certain area around the target. Hence, if a large number of Banelings attack a handful of Zealots located close to each other, the Zealots will be defeated instantly. The optimal strategy, therefore, is to cooperatively spread out around the map far from each other so that the Banelings' damage is distributed as thinly as possible. The `corridor` scenario (Figure 2d), in which 6 friendly Zealots face 24 enemy Zerglings, requires agents to make effective use of the terrain features. Specifically, agents should collectively wall off the choke point (the narrow region of the map) to block enemy attacks from different directions. Some of the micro-trick challenges are inspired by *StarCraft Master* challenge missions released by Blizzard (Blizzard Entertainment, 2012).

The complete list of challenges is presented in Table 1. The difficulty of the game AI is set to *very difficult* (7). Our experiments, however, suggest that this setting does significantly impact the unit micromanagement of the built-in heuristics.

State and Observations

At each timestep, agents receive local observations drawn within their field of view. This encompasses information about the map within a circular area around each unit and with a radius equal to the *sight range* (Figure 3). The sight range makes the environment partially observable from the standpoint of each agent. Agents can only observe other agents if they are both alive and located within the sight range. Hence, there is no way for agents to determine whether their teammates are far away or dead.

The feature vector observed by each agent contains the following attributes for both allied and enemy units within the sight range: `distance`, `relative x`, `relative y`, `health`, `shield`, and `unit_type`. Shields serve as an additional source of protection that needs to be removed before any damage can be done to the health of units. All Protos units have shields, which can regenerate if no new damage is dealt. In addition, agents have access to the last actions of allied units that are in the field of view. Lastly, agents can observe the terrain features surrounding them; particularly, the values of eight points at a fixed radius indicating height and walkability.

The global state, which is only available to agents during centralised training, contains information about all units on the map. Specifically, the state vector includes the coordinates of all agents relative to the centre of the map, together with unit features present in the observations. Additionally, the state stores the `energy` of Medivacs and `cooldown` of the rest of allied units, which represents the minimum delay between attacks. Finally, the last actions of all agents are attached to the central state.

All features, both in the state as well as in the observations of individual agents, are normalised by their maximum values. The sight range is set to 9 for all agents.

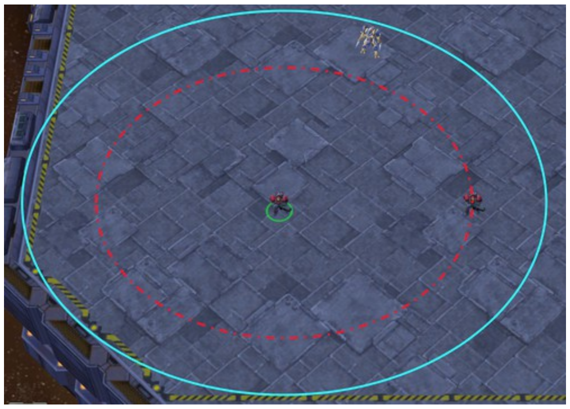


Figure 3. The cyan and red circles respectively border the sight and shooting range of the agent.

Action Space

The discrete set of actions which agents are allowed to take consists of `move[direction`[4]`]`, `attack[enemy_id]`, `stop` and `no-op`.[5] As healer units, Medivacs must use `heal[agent_id]` actions instead of `attack[enemy_id]`. The maximum number of actions an agent can take ranges between 7 and 70, depending on the scenario.

To ensure decentralisation of the task, agents are restricted to use the `attack[enemy_id]` action only towards enemies in their *shooting range* (Figure 3). This additionally constrains the ability of the units to use the built-in *attack-move* macro-actions on the enemies that are far away. We set the shooting range equal to 6 for all agents. Having a larger sight range than a shooting range forces agents to make use of the move commands before starting to fire.

Rewards

The overall goal is to have the highest win rate for each battle scenario. We provide a corresponding option for *sparse rewards*, which will cause the environment to return only a reward of +1 for winning and -1 for losing an episode. However, we also provide a default setting for a shaped reward signal calculated from the hit-point damage dealt and received by agents, some positive (negative) reward after having enemy (allied) units killed and/or a positive (negative) bonus for winning (losing) the battle. The exact values and scales of this shaped reward can be configured using a range of flags, but we strongly discourage disingenuous engineering of the reward function (e.g. tuning different reward functions for different scenarios).

[4]Four directions: north, south, east, or west.

[5]Dead agents can only take `no-op` action while live agents cannot.

Name	Ally Units	Enemy Units	Type
3m	3 Marines	3 Marines	homogeneous & symmetric
8m	8 Marines	8 Marines	homogeneous & symmetric
25m	25 Marines	25 Marines	homogeneous & symmetric
2s3z	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots	heterogeneous & symmetric
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots	heterogeneous & symmetric
MMM	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 2 Marauders & 7 Marines	heterogeneous & symmetric
5m_vs_6m	5 Marines	6 Marines	homogeneous & asymmetric
8m_vs_9m	8 Marines	9 Marines	homogeneous & asymmetric
10m_vs_11m	10 Marines	11 Marines	homogeneous & asymmetric
27m_vs_30m	27 Marines	30 Marines	homogeneous & asymmetric
3s5z_vs_3s6z	3 Stalkers & 5 Zealots	3 Stalkers & 6 Zealots	heterogeneous & asymmetric
MMM2	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 3 Marauders & 8 Marines	heterogeneous & asymmetric
2m_vs_1z	2 Marines	1 Zealot	micro-trick: alternating fire
2s_vs_1sc	2 Stalkers	1 Spine Crawler	micro-trick: alternating fire
3s_vs_3z	3 Stalkers	3 Zealots	micro-trick: kiting
3s_vs_4z	3 Stalkers	4 Zealots	micro-trick: kiting
3s_vs_5z	3 Stalkers	5 Zealots	micro-trick: kiting
6h_vs_8z	6 Hydralisks	8 Zealots	micro-trick: focus fire
bane_vs_bane	20 Zerglings & 4 Banelings	20 Zerglings & 4 Banelings	micro-trick: positioning
so_many_banelings	7 Zealots	32 Banelings	micro-trick: positioning
2c_vs_64zg	2 Colossi	64 Zerglings	micro-trick: positioning
corridor	6 Zealots	24 Zerglings	micro-trick: wall off

Table 1. SMAC scenarios

Environment Setting

SMAC makes use of the StarCraft II Learning Environment (*SC2LE*) (Vinyals et al., 2017) to communicate with the StarCraft II engine. SC2LE provides full control of the game by allowing to send commands and receive observations from the game. However, SMAC is conceptually different from the RL environment of SC2LE. The goal of SC2LE is to learn to play the full game of StarCraft II. This is a competitive task where a centralised RL agent receives RGB pixels as input and performs both macro and micro with the player-level control similar to human players. SMAC, on the other hand, represents a set of cooperative multi-agent micro challenges where each learning agent controls a single military unit.

SMAC uses the *raw API* of SC2LE. Raw API observations do not have any graphical component and include information about the units on the map such as health, location coordinates, etc. The raw API also allows sending action commands to individual units using their unit IDs. This setting differs from how humans play the actual game, but is convenient for designing decentralised multi-agent learning tasks.

Since our micro-scenarios are shorter than actual StarCraft II games, restarting the game after each episode presents a computational bottleneck. To overcome this issue, we make use of the API's debug commands. Specifically, when all units of either army have been killed, we kill all remaining units by sending a debug action. Having no units left launches a trigger programmed with the StarCraft II Editor that re-spawns all units in their original location with full health, thereby restarting the scenario quickly and efficiently.

Furthermore, to encourage agents to explore interesting micro-strategies themselves, we limit the influence of the StarCraft AI on our agents. In the game of StarCraft II, whenever an idle unit is under attack, it automatically starts a reply attack towards the attacking enemy units without being explicitly ordered. We disabled such automatic replies towards the enemy attacks or enemy units that are located closely by creating new units that are the exact copies of existing ones with two attributes modified: *Combat: Default Acquire Level* is set to *Passive* (default *Offensive*) and *Behaviour: Response* is set to *No Response* (default *Acquire*). These fields are only modified for allied units; enemy units are unchanged.

The sight and shooting range values might differ from the built-in *sight* or *range* attribute of some StarCraft II units. Our goal is not to master the original full StarCraft II game, but rather to benchmark MARL methods for decentralised control.

5. Evaluation Methodology

We propose the following methodology for evaluating MARL methods using SMAC.

Fixed Parameters

To ensure the fairness of the challenge and comparability of results, performance must be evaluated under standardised conditions. One must not undertake any changes to the environment used for evaluating the policies. This includes the observation and state spaces, action space, the game mechanics, and settings of the environment (e.g., frame-skipping rate). It is prohibited to modify the StarCraft II map files in any way or change the difficulty of the game AI. Episode limits of each scenario must remain unchanged.

SMAC also restricts the execution of the trained models to be decentralised, i.e., during testing each agent must base its policy solely on its own action-observation history and cannot use the global state or the observations of other agents.

Adjustable Parameters

Although restricted to a number of fixed configurations, some parameters of the environment can be adjusted. For instance, it is allowed to add or remove some features from observations or state using corresponding flags (depending on the scenario, some of the features might be irrelevant, such as features concerning terrain information). Reward functions can also be adjusted using the corresponding environment flags.

Moreover, it is acceptable to train the decentralised policies in centralised fashion. Specifically, agents can exchange individual observations during training and use the global state.

Evaluation Metrics

Our main evaluation metric is the mean win percentage of evaluation episodes as a function of environment steps observed, over the course of training. Such progress can be estimated by periodically running a fixed number of evaluation episodes (i.e. with any exploratory behaviours disabled). Each experiment should be repeated using a sufficiently large number of independent training runs (e.g., more than 10) and the resulting plot should include the median performance as well as the 25-75% percentiles. We recommend using the median instead of the mean in order to avoid the affect of any outliers. The number of simultaneous runs, as well as environment steps used in training, must be reported. Figure 4 contains an example for such graphs. It can also be informative to report this and additional metrics at the end of the training period, summarised in a single table as in Table 2.

It can prove helpful to other researchers to include the computational resources used, and the wall clock time for running each experiment. SMAC provides functionality for saving StarCraft II replays, which can be viewed using a freely available client. The resulting videos can be used to comment on interesting behaviours observed.

6. PyMARL

To make it easier to develop algorithms for SMAC, we are also open-sourcing our software engineering framework *PyMARL*. PyMARL has been designed explicitly with deep MARL in mind and allows for out-of-the-box experimentation and development.

PyMARL's codebase is organized in a modular fashion in order to enable the rapid development of new algorithms, as well as providing implementations of current deep MARL algorithms to benchmark against. It is built on top of PyTorch to facilitate the fast execution and training of deep neural networks, and take advantage of the rich ecosystem built around it. PyMARL's modularity means it's easy to extend, and components can be readily isolated for testing purposes.

Since the implementation and development of deep MARL algorithms comes with a number of additional challenges beyond those posed by single-agent deep RL, it is crucial to have simple and understandable code. In order to improve the readability of code and simplify the handling of data between components, PyMARL encapsulates all data stored in the buffer within an easy to use data structure. This encapsulation provides a cleaner interface for the necessary handling of data in deep MARL algorithms, whilst not obstructing the manipulation of the underlying PyTorch Tensors. In addition, PyMARL aims to maximise the batching of data when performing inference or learning so as to provide significant speed-ups over more naive implementations.

PyMARL features implementations of the following algorithms: QMIX (Rashid et al., 2018) and COMA (Foerster et al., 2018a) as state-of-the-art methods, and VDN (Sunehag et al., 2017) and IQL (Tan, 1993) as baselines.

7. Results

In this section we present results for scenarios included as part of SMAC. The purpose of these results is to demonstrate the performance of the current state-of-the-art methods in our chosen MARL paradigm.

We present comprehensive results for QMIX. In our experiments we found it to perform better than IQL and COMA, but we include some further results for IQL and COMA to give a rough idea for how independent learning and on-policy actor-critic methods perform. Better results can potentially be achieved by adjusting hyperparameters or network architectures per scenario. Furthermore, the number of episode rollouts we perform in between training steps for

QMIX can be decreased to achieve better sample efficiency at the expense of wall clock time.

The evaluation procedure is similar to the one in (Rashid et al., 2018). The training is paused after every 20000 timesteps during which 24 test episodes are run with agents performing action selection greedily in a decentralised fashion. The percentage of episodes where the agents defeat all enemy units within the permitted time limit is referred to as the *test win rate*.

The architectures and training details are presented in the Appendix A.

Discussion

Table 2 presents the test win rates across all SMAC scenarios using QMIX, COMA, and IQL methods. To provide a better insight into the performances, it includes the Median, Mean, Min, and Max win rates of evaluation episodes after 10 million environment timesteps. Based on these results, we broadly categorise the scenarios into three categories for our discussion below: *Easy*, *Hard*, and *Super Hard*.

The *Easy* scenarios are fully solved (above 95% median win rate) by our QMIX setup within the timeframe given, whereas we make progress on but do not solve the *Hard* scenarios. The *Super Hard* scenarios are perhaps the most interesting since our QMIX agent makes little to no progress on them.

In the remainder of this section, we present the results of running our chosen agents across the scenarios, and offer some discussion of the learnt policies we have observed. Finally, we investigate the necessity of using an RNN in our agent networks to tackle partial observability. The learning curves for all 22 SMAC scenarios are presented in the Appendix B.

Easy scenarios

The symmetric scenarios generally provide an easier challenge to learning agents. The enemy utilises a fixed strategy and possesses no advantage in unit composition or number compared to the allied team, which allows the team of learning agents to develop a strategy to win in almost every encounter.

The `3m` scenario provides the easiest challenge of all the scenarios, and is fully solved by both IQL and QMIX. It is an ideal scenario to debug algorithm implementations and investigate hyperparameter configurations on, serving a similar role to Pong in the ALE. The `8m` scenario (Figure 2c) provides a larger challenge, owing mainly to the increased number of agents that must learn to coordinate when focussing fire. Our QMIX agents were able to learn to first position into a line, in order to ensure that they can all begin attacking at the same time and to focus fire without overkill. They also learned to run back from the attack line upon absorbing severe hit-point damage. COMA and IQL agents also learned to win the `8m` scenario most of the time, although the cooperation between agents is less coordinated compared with QMIX.

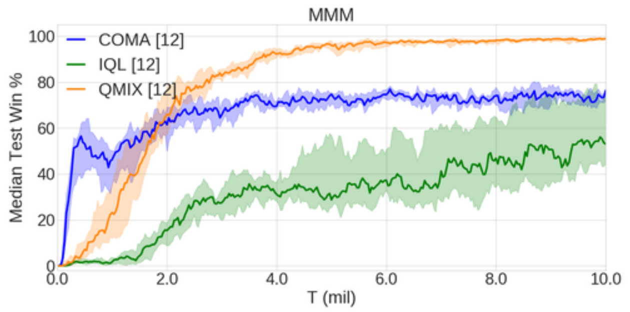


Figure 4. Median win rate of QMIX, COMA, and IQL on `MMM` scenario. 25%-75% percentile is shaded.

The `MMM` and `2s3z` scenarios prove harder, but are still solvable by QMIX within the training period. Notably, we begin to see a large disparity between QMIX and both IQL and COMA on these scenarios. Figure 4 shows the performance of the 3 methods across the training. Our QMIX agents are able to learn the strategy of focus fire and protecting the Medivac from damage on the `MMM` scenario, whereas IQL and COMA do not. On the `2s3z` scenario QMIX successfully learned to intercept the enemy Zealots with allied Zealots in order to protect the allied Stalkers from severe damage which requires deliberate movement.

On the `2m_vs_1z` scenario we can see that both IQL and QMIX are able to fully solve it, which requires the 2 Marines to alternately fire on the enemy Zealot in order to draw its attention away from the other allied Marine. COMA, on the other hand does not learn to draw the enemy Zealot away from the Marines, and thus takes heavy damage and loses the scenario in the vast majority of cases. `2m_vs_1z`, which is a similar scenario where 2 allied Stalkers face a Spine Crawler, is also solved by QMIX and IQL. Here, the agents learned to make the enemy retarget once any of the Stalkers absorbs severe hit-point damage in order not to get killed.

The `so_many_banelings` scenario requires the agents to sacrifice themselves to the enemy Banelings. The optimal strategy is to spread out in order to both minimise the number of allied units which are damaged and maximise the number of enemy Banelings which are hit within the damage radius. The QMIX agents learn this winning strategy, whereas the COMA and IQL agents do not spread out enough and take unnecessary damage leading to an unstable and mediocre win rate.

Banelings also play a key role in the `bane_vs_bane` sce-

	QMIX				COMA				IQL			
Scenario	Median	Mean	Min	Max	Median	Mean	Min	Max	Median	Mean	Min	Max
3m	100	99	95	100	91	92	83	100	100	97	87	100
8m	100	96	62	100	95	94	83	100	91	90	79	100
25m	72	70	0	100	10	11	0	29	16	18	0	45
2s3z	100	97	91	100	66	64	25	87	39	43	16	66
3s5z	16	25	0	79	0	0	0	4	0	3	0	25
MMM	100	97	79	100	79	80	62	91	56	56	33	79
5m_vs_6m	58	55	25	91	0	0	0	0	20	21	4	45
8m_vs_9m	81	76	33	95	0	1	0	8	4	10	0	58
10m_vs_11m	83	75	37	91	0	3	0	16	37	33	0	79
27m_vs_30m	4	7	0	33	0	0	0	0	0	0	0	0
3s5z_vs_3s6z	0	0	0	0	0	0	0	0	0	0	0	0
MMM2	0	12	0	87	0	0	0	0	0	0	0	4
2m_vs_1z	100	99	95	100	0	25	0	100	100	99	95	100
2s_vs_1sc	100	97	83	100	89	76	0	100	100	97	91	100
3s_vs_3z	100	99	95	100	0	7	0	95	91	90	75	100
3s_vs_4z	95	95	87	100	0	0	0	0	6	14	0	58
3s_vs_5z	31	38	8	70	0	0	0	0	0	0	0	4
6h_vs_8z	0	1	0	4	0	0	0	0	0	0	0	0
bane_vs_bane	100	98	95	100	45	46	8	87	97	97	87	100
so_many_banelings	100	97	83	100	79	64	4	100	50	54	0	87
2c_vs_64zg	62	58	29	75	0	1	0	20	56	56	8	91
corridor	0	8	0	75	0	0	0	0	0	1	0	8

Table 2. Win rates of QMIX, COMA, and IQL methods on all SMAC scenarios across 12 independent runs.

nario. Here, QMIX and IQL agents were able to learn a winning strategy in which the allied Banelings attack the enemy first to deal severe damage, only after which Zerglings start to attack. COMA agents were not able to coordinate well in order to master such a winning strategy.

An optimal strategy for the 3s_vs_3z, 3s_vs_4z, and 3s_vs_5z scenarios is to kite the enemy Zealots. On the 3s_vs_3z scenario only QMIX is able to learn such a strategy. A similar strategy is learnt by QMIX for the 3s_vs_4z scenario, but only the best runs are able to solve it within the training period indicating that the learning speed of our algorithm is the main bottleneck here.

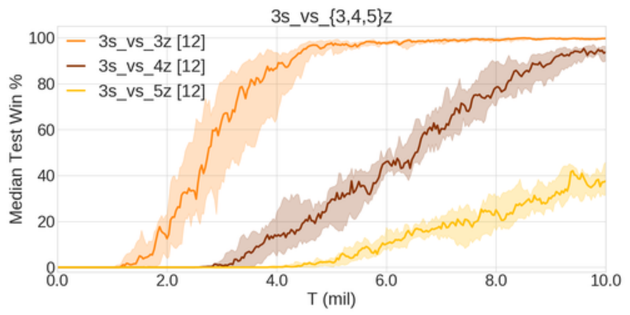


Figure 5. Median win rate of QMIX on 3s_vs_3z, 3s_vs_4z and 3s_vs_5z scenarios. 25%-75% percentile is shaded.

Hard scenarios

Despite having the same number of Marines on both side, the 25m scenario is much harder compared with 3m and 8m. Defeating the enemy here requires a higher-level of coordination when focussing fire, which is difficult when the number of independent agents is large. None of the methods were able to learn a consistently winning strategy.

Unlike the 3s_vs_3z and 3s_vs_4z scenarios, the QMIX agents are unable to solve the 3s_vs_5z scenario within the training period. A look at the learned policies shows better kiting in some situations compared to 3s_vs_4z, but overall worse performance. We hypothesise that the longer episodes make it more difficult to train, since temporal credit assignment is harder. This further highlights the need for faster training. Figure 5 shows the performance of the QMIX agents across the 3 scenarios.

Compared to its easier counterpart, 3s5z is a much harder scenario. The learnt strategies for the best QMIX agents are very different compared to the best strategies learnt for 2s3z. In this scenario the QMIX Zealots do not purposefully intercept the enemy Zealots, and thus the allied Stalkers die very quickly leading to a guaranteed loss. We have observed that a longer training period does indeed lead to much better performance on this scenario. We hypothesise that a better exploration scheme would help in this regard, since the agents would be exposed to better strategies more often, as opposed to requiring many training iterations and some luck to discover a better positioning for their units.

The asymmetric scenarios `5m_vs_6m`, `8m_vs_9m`, and `10m_vs_11m` offer a substantial challenge. The allied agents must learn to focus fire without overkill and to correctly position themselves with considerable precision. The QMIX agents make good progress, but are unable to learn to both focus fire appropriately and position themselves correctly before encountering the enemy. This is particularly prevalent on the `10m_vs_11m` scenario, where a number of the allied agents are still positioning themselves when the enemy is encountered. The lack of a performance hit despite this is due to the easier nature of the `10m_11m` scenario compared to `5m_vs_6m` owing to the relative army sizes.

The difficulty of the `2c_vs_64zg` scenario lies in the number of enemy units, which makes the action space of the agents much larger than in other scenarios. The allied Colossi need to make use of the cliff to buy time and kill as many Zerglings as possible before can they reach them from all sides. Our QMIX agents learned to make use of this terrain feature but were unable to fully master this scenario.

Super Hard scenarios

Of the 22 scenarios featured in initial release of SMAC, our QMIX agent makes very little progress on 5 of them: `27m_vs_30m`, `3s5z_vs_3s6z`, `MMM2`, `6h_vs_8z` and `corridor`.

`corridor` presents a task in which active state space exploration is required. An optimal strategy for this scenario requires the allied Zealots to move to a choke point on the map, so as to not be surrounded by the enemy army. None of our tested methods utilise any form of directed state space exploration, which leads to them all remaining in the local optimum of merely damaging the enemy units for reward, instead of first moving to the choke point and then attacking.

The hardest of all Marine scenarios is `27m_vs_30m` on which all three methods make little to no progress. Only the QMIX agents were able to display a non-zero win rate on this challenge. The tested methods were also unsuccessful in solving the `6h_vs_8z` scenario, which requires a fine-grained focus-fire by the allied Hydralisks.

`MMM2` and `3s5z_vs_3s6z` are the harder asymmetric counterparts to `MMM` and `3s5z` respectively. Since the allied army is outnumbered by the enemy army, winning the battle requires significantly more coordination and strategy than in the symmetric case. These scenarios are examples of open-problems and will require algorithmic innovation. We plan to release further scenarios in this category but will also open the challenge for community contributions of new scenarios, which can be created using the StarCraft II map editor.

Scenario	RNN Agents	FF Agents
`2s_vs_1sc`	100	10
`3s_vs_3z`	100	85
`so_many_banelings`	100	100

Table 3. Median win rate of QMIX with RNN vs QMIX with feed-forward agents across 12 seeds.

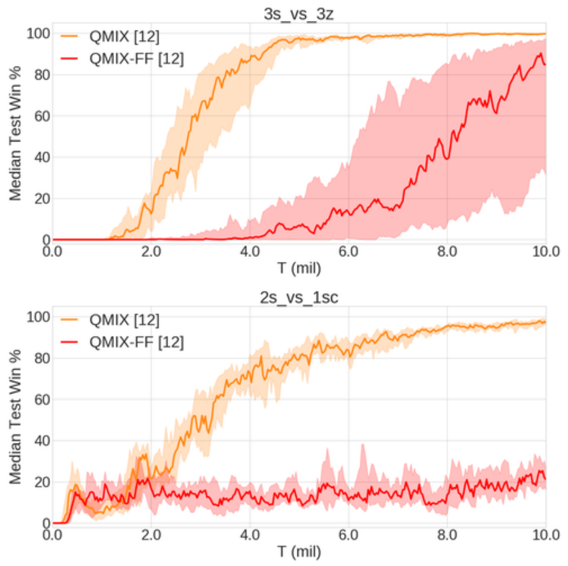


Figure 6. Median win rate of QMIX with RNN vs QMIX with feed-forward agents. 25%-75% percentile is shaded.

Feed-forward Agent Network

Table 3 shows the final median win rate for the QMIX agents with RNN and feed-forward agent networks. Although the final performance is comparable for 3 of the scenarios, the speed of learning can be quite different as shown in Figure 6. On the `2m_vs_1z` scenario, only the agents with RNNs are able to learn a good strategy.

8. Conclusion and Future Work

This paper presented SMAC as a set of benchmark problems for cooperative MARL. Based on the real-time strategy game StarCraft II, SMAC focuses on decentralised micromanagement tasks and features 22 diverse combat scenarios which challenge MARL methods to handle partial observability and high-dimensional inputs. We offer recommendations for reporting evaluations using standardised performance metrics and provide a thorough report and discussion for several state-of-the-art MARL algorithms, such as QMIX and COMA. Additionally, we are open-sourcing PyMARL - our framework for design and analysis of deep MARL algorithms.

In the near future, we aim to extend SMAC with new challenging scenarios that feature a more diverse set of units and require a higher level of coordination amongst agents. Particularly, we plan to make use of the rich skill set of StarCraft II units, and host scenarios that require the agents to utilise the features of the terrain. With harder multi-agent coordination problems, we aim to explore the gaps in existing MARL approaches and motivate further research in this domain, particularly in areas such as multi-agent exploration and coordination.

We look forward to accepting contributions from the community to SMAC and hope that it will become a standard benchmark for measuring progress in cooperative MARL.

Acknowledgements

The authors would like to thank Davit Hayrapetyan for his helpful suggestions about the StarCraft II scenarios. We also thank Phil Bates and Jon Russell (Oracle Corporation) for guidance deploying and executing SMAC on Oracles public Cloud Infrastructure (https://www.oracle.com/uk/cloud/infrastructure.html) and the SC2LE teams at DeepMind and Blizzard for their work on the interface.

The work is supported by the European Union's Horizon 2020 research and innovation programme (grant agreement number 637713), the National Institutes of Health (grant agreement number R01GM114311) and EPSRC/MURI grant EP/N019474/1. It is also supported by the an EPSRC grant (EP/M508111/1, EP/N509711/1), the Oxford-Google DeepMind Graduate Scholarship, Microsoft and Free the Drones (FreeD) project under the Innovation Fund Denmark. The experiments were made possible by generous cloud credit grants from Oracles Cloud Innovation Accelerator and NVIDIA.

References

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

DeepMind. Alphastar: Mastering the real-time strategy game starcraft ii, 2019. URL https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., and Whiteson, S. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1146–1155, 2017.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018a.

Foerster, J. N., de Witt, C. A. S., Farquhar, G., Torr, P. H., Boehmer, W., and Whiteson, S. Multi-agent common knowledge reinforcement learning. *arXiv preprint arXiv:1810.11702*, 2018b.

Hausknecht, M. and Stone, P. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.

Hausknecht, M., Mupparaju, P., Subramanian, S., Kalyanakrishnan, S., and Stone, P. Half field offense: an environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, 2016.

Heinrich, J. and Silver, D. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016. URL http://arxiv.org/abs/1603.01121.

Hu, Y., Li, J., Li, X., Pan, G., and Xu, M. Knowledge-guided agent-tactic-aware learning for starcraft micromanagement. In *IJCAI*, pp. 1471–1477, 2018.

Kalyanakrishnan, S., Liu, Y., and Stone, P. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In *Robot Soccer World Cup*, pp. 72–85. Springer, 2006.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pp. 340–347. ACM, 1997.

Kraemer, L. and Banerjee, B. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. February 2017. URL http://arxiv.org/abs/1702.03037. arXiv: 1702.03037.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. June 2017. URL http://arxiv.org/abs/1706.02275. arXiv: 1706.02275.

Nardelli, N., Synnaeve, G., Lin, Z., Kohli, P., Torr, P. H., and Usunier, N. Value propagation networks. *arXiv preprint arXiv:1805.11199*, 2018.

Oliehoek, F. A. and Amato, C. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent

Systems. Springer, 2016. URL http://www.springer.com/us/book/9783319289274.

Oliehoek, F. A., Spaan, M. T. J., and Nikos Vlassis. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *JAIR*, 32:289–353, 2008.

Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

Pang, Z.-J., Liu, R.-Z., Meng, Z.-Y., Zhang, Y., Yu, Y., and Lu, T. On reinforcement learning for full-length game of starcraft. *arXiv preprint arXiv:1809.09095*, 2018.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4295–4304, 2018.

Resnick, C., Eldridge, W., Ha, D., Britz, D., Foerster, J., Togelius, J., Cho, K., and Bruna, J. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018.

Shao, K., Zhu, Y., and Zhao, D. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

Stone, P., Kuhlmann, G., Taylor, M. E., and Liu, Y. Keepaway soccer: From machine learning testbed to benchmark. In *Robot Soccer World Cup*, pp. 93–105. Springer, 2005.

Sun, P., Sun, X., Han, L., Xiong, J., Wang, Q., Li, B., Zheng, Y., Liu, J., Liu, Y., Liu, H., et al. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *arXiv preprint arXiv:1809.07193*, 2018.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv:1706.05296 [cs]*, June 2017. URL http://arxiv.org/abs/1706.05296. arXiv: 1706.05296.

Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., and Usunier, N. TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. *arXiv preprint arXiv:1611.00625*, 2016.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent Cooperation and Competition with Deep Reinforcement Learning. November 2015. URL http://arxiv.org/abs/1511.08779. arXiv: 1511.08779.

Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Blizzard Entertainment. New blizzard custom game: Starcraft master. http://us.battle.net/sc2/en/blog/4544189/new-blizzard-custom-game-starcraft-master-3-1-2012, 2012. Accessed: 2018-11-16.

Usunier, N., Synnaeve, G., Lin, Z., and Chintala, S. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Kttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. StarCraft II: A New Challenge for Reinforcement Learning. *arXiv:1708.04782 [cs]*, August 2017. URL http://arxiv.org/abs/1708.04782. arXiv: 1708.04782.

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

Zheng, L., Yang, J., Cai, H., Zhang, W., Wang, J., and Yu, Y. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *arXiv preprint arXiv:1712.00600*, 2017.

A. Experimental Setup

A.1. Architecture and Training

The architecture of all agent networks is a DRQN (Hausknecht & Stone, 2015) with a fully connected layer followed by a recurrent layer comprised of a GRU with a 64-dimensional hidden state, and then a fully-connected layer with $|U|$ outputs.

For IQL and QMIX, exploration is performed during training using independent ϵ-greedy action selection, where each agent performs ϵ-greedy action selection over its own outputs. We anneal ϵ from 1.0 to 0.05 over 20k timesteps, and then keep it constant. We keep a replay buffer of the 5000 most recent episodes and sample a batch of size 32 entire episodes for training. The target networks are updated after every 200 episodes. We do a single training step after every 8 episodes. The architecture of the mixing network is the same as in (Rashid et al., 2018).

The architecture of the COMA critic is a feedforward fully-connected neural network with the first 2 layers having 128 units, followed by a final layer of $|U|$ units. We set $\lambda = 0.8$. We utilise the same ϵ-floor scheme as in (Foerster et al., 2018a) for the agents' policies, linearlly annealing ϵ from 0.5 to 0.01 over 100k timesteps.

For all experiments we set $\gamma = 0.99$, use RMSprop with a learning rate of 5×10^{-4} and $\alpha = 0.99$ without weight decay or momentum and share parameters across agent networks. We run each method for 10 million environment steps, which takes roughly 18 hours in our PyMARL framework on a NVIDIA GTX 1080Ti GPU.

A.2. Reward and Observation

All experiments use the default shaped rewards throughout all scenarios. At each timestep, agents receive positive rewards, equal to the hit-point damage dealt, and bonuses of 10 and 200 points for killing each enemy unit and winning the scenario, respectively. The rewards are scaled so that the maximum cumulative reward achievable in each scenario is around 20.

The agent observations used in the experiments include all features from Section 4, except for the he last actions of the allied units (within the sight range), terrain height and walkability.

B. Full Results

The median test win rates during the training of QMIX, COMA, and IQL methods on easy, hard and super hard SMAC scenarios are presented in Tables 7, 8, and 9, respectively. All experiments feature 12 independent runs.

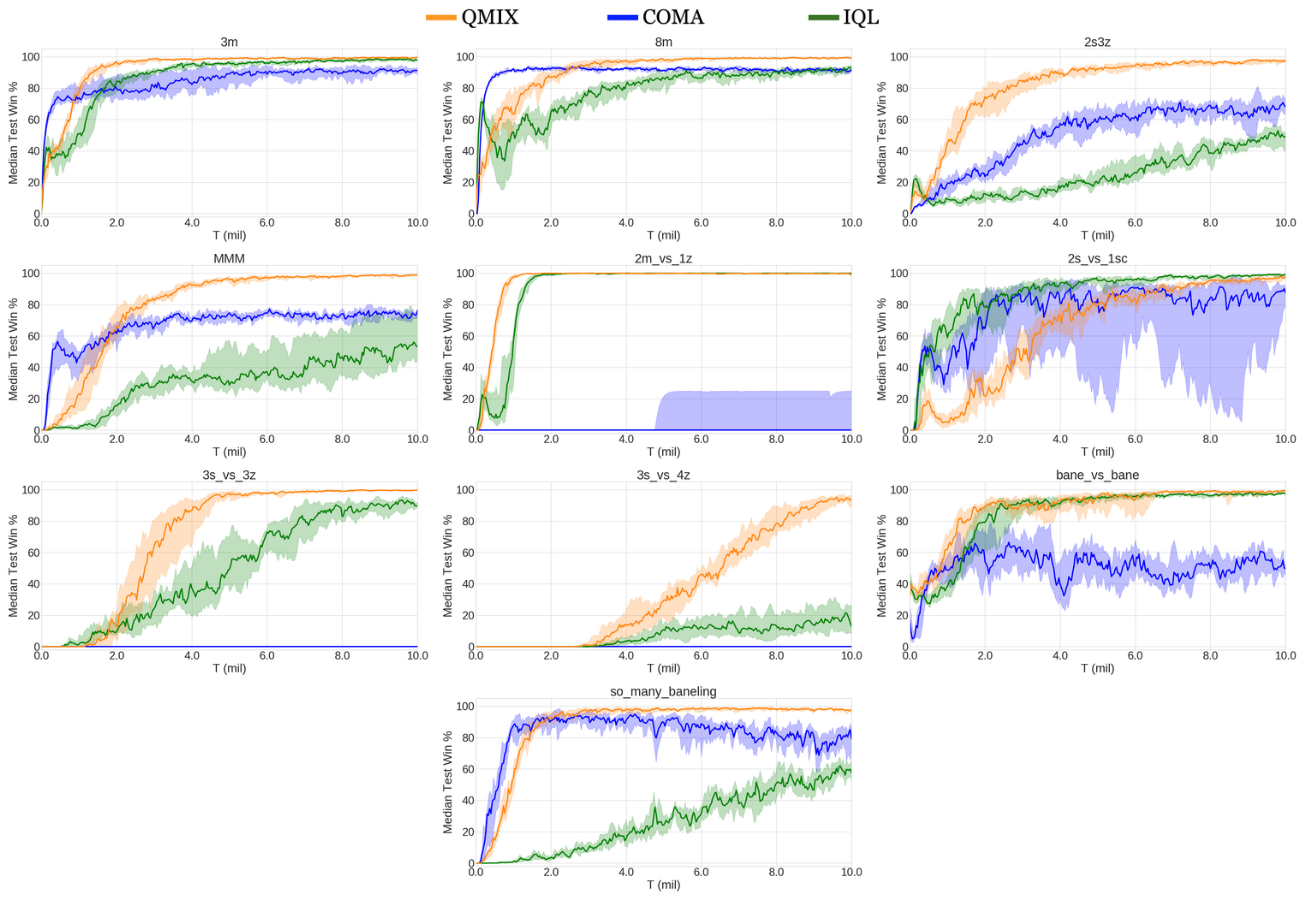


Figure 7. Median win rates for QMIX, COMA, and IQL on easy SMAC scenarios. 25%-75% percentile is shaded.

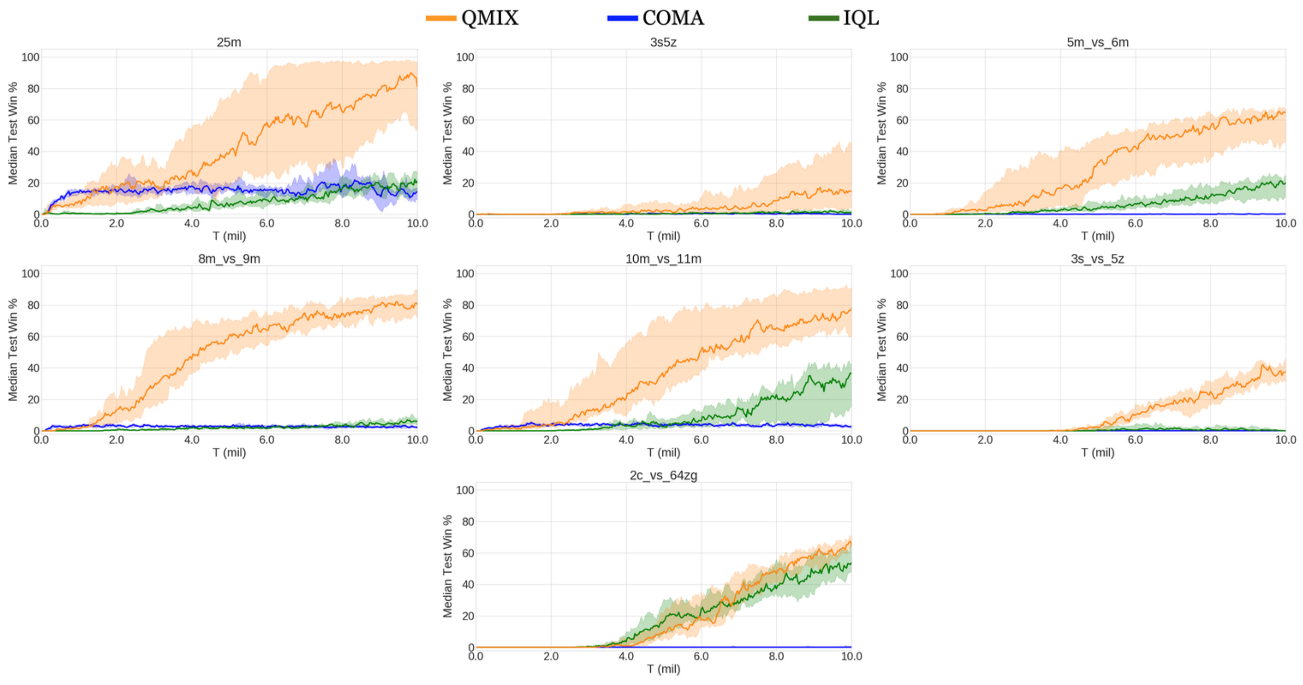


Figure 8. Median win rates for QMIX, COMA, and IQL on hard SMAC scenarios. 25%-75% percentile is shaded.

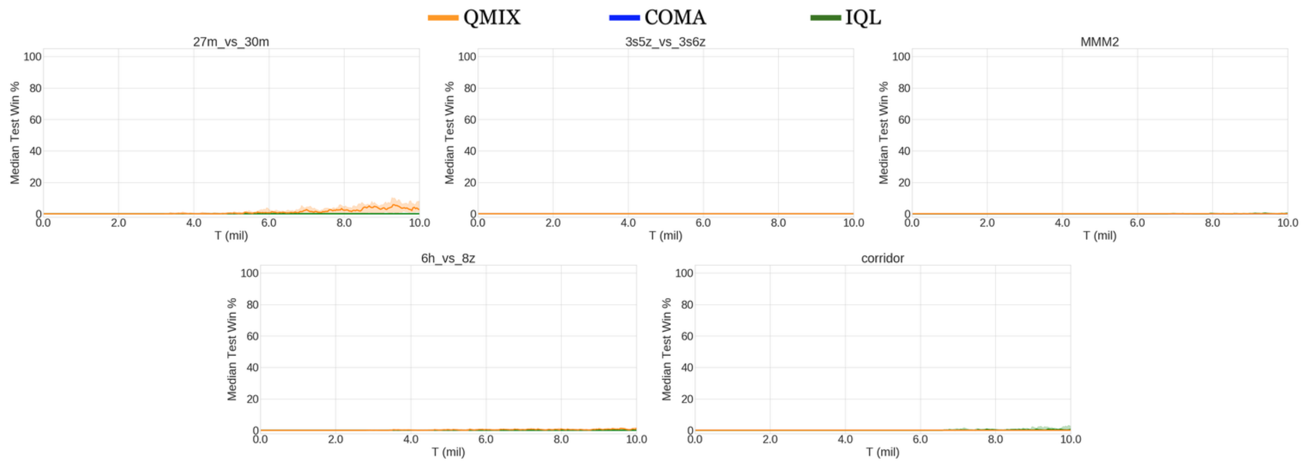


Figure 9. Median win rates for QMIX, COMA, and IQL on super hard SMAC scenarios. 25%-75% percentile is shaded.