The StarCraft Multi-Agent Challenges$^+$: Learning of Sub-tasks and Environmental Benefits without Precise Reward Functions

Anonymous Authors[1]

Abstract

In this paper, we propose a novel benchmark called SMAC$^+$, where agents are supposed to implicitly learn a way to complete sub-tasks or to use environmental benefits without precise reward functions. The StarCraft Multi-Agent Challenges (SMAC) recognized as a standard benchmark of Multi-Agent Reinforcement Learning (MARL) is mainly concerned with ensuring that all agents cooperatively eliminate approaching adversaries only through fine manipulation with obvious reward functions. SMAC$^+$, on the other hand, is interested in the ability of MARL algorithms to effectively discover sub-tasks. In the offensive scenarios, agents must learn to find opponents first and then eliminate them, and the defensive scenarios need agents to use topographic features such as placing behind structures to lower the possibility of being attacked by enemies. In these scenarios, MARL algorithms must learn indirectly how they accomplish sub-tasks without direct incentives. We investigate MARL algorithms under SMAC$^+$ and observe that recent approaches work well in similar settings to the previous challenges but misbehave in offensive scenarios, even when training time is significantly extended. We also discover that risk-based extra exploration approach has a positive effect on performance through the completion of sub-tasks.

1. Introduction

In the multi-agent reinforcement learning (MARL) studies, the StarCraft Multi-Agent Challenges (SMAC) is recognized as the standard benchmark simulator (Samvelyan et al., 2019). Tasks in the SMAC require micro-managed control of agents involving a variety of observation and action sets, as well as the consideration of a large number of agents. Specifically, the SMAC takes into account defensive scenarios in which all opponents with better strength naturally approach the trained agents. As seen in Table 1, some tough scenarios such as `2c_vs_64zg` and `corridor` need agents to indirectly learn environmental benefits or sub-tasks such as, exploiting different levels of terrains or avoiding enemies just rushing, without a specific reward for them. Recently, efficient exploration algorithms address these scenarios (Sun et al., 2021; Son et al., 2022), but these sub-tasks are relatively simplistic situations and not applicable in the real-world since they focus on in-game properties.

To address this issue, we propose a new class of the StarCraft Multi-Agent Challenges (SMAC$^+$) that accounts for realistic and sophisticated sub-tasks as well as environmental benefits agents must implicitly learn to accomplish. We present four offensive scenarios to encourage trained agents to initially find the adversaries while also considering topographical obstacles, and then rapidly defeat each adversarial troops. In addition, three defensive scenarios need agents to employ topographical features such as placing behind structures to lower the possibility of being attacked by enemies. In these situations, agents are still rewarded just for eliminating enemies, indicating that they indirectly learn how to do sub-tasks and use environmental benefits. This paper demonstrates that existing approaches work well in similar settings to the previous SMAC, but when environments need to perform complicated sub-tasks, most algorithms fail to learn adequately even when training time is significantly extended. To summarize, we make the following contributions:

- We propose a novel MARL benchmark; SMAC$^+$. The proposed benchmark presents four offensive scenarios for analyzing how agents learn implicitly to do sub-tasks when reward functions are designed only towards the final objective.
- SMAC$^+$ contains three defensive in which agents make use of environmental benefits in battlefields which reduces possibility of being damaged by opponents.
- We find that recent MARL algorithms shows marginal performance improvement in SMAC$^+$, but outcomes are inconsistent. We argue that the SMAC$^+$ is open-ended problem for efficient exploration toward MARL domains.

2. The StarCraft Multi-Agent Challenges$^+$

We propose a novel multi-agent environment referred to as StarCraft Multi-Agent Chllenges$^+$ (SMAC$^+$). We offer

Table 1: List of environmental benefits and sub-tasks for in SMAC and SMAC$^+$

	SMAC		SMAC$^+$	
	2c_vs_64zg	corridor	Defense(easy, hard, super-hard)	Offense(easy, hard, super-hard, ultra-hard)
Environmental Benefits	Different levels of terrains	-	Destroy obstacles-hiding enemies Place in less damage zones	Approach enemies strategically Discover a detour* Destroy movement impediments*
Sub-task	-	Avoiding rushing-enemies toward allies	-	Identify where-enemies locate

∗ : Ultra-hard scenario

more realistic and sophisticated environmental advantages such as destructible structures that can be used to conceal enemies and terrain features like hill. Furthermore, we develop a new MARL option; communication with among agents. This option supposes that the agents in close share partial information of observations to improve cooperation. We refer this option as to *relaxed centralized training and decentralized execution (CTDE)*. It does not violate decentralized execution because none of the agents utilize the absolute state information throughout the test time although a small portion of observation is shared. We believe this option more accurately depicts actual challenges as the communication technology is being rapidly developed. We describe the details about our environments in the rest of this section.

2.1. Terrain and Unit combination

We create a battlefield seen in Figure 1. The attacking troops are stationed on the lower side, while defending troops are placed on the hill. A topographic feature is the hill, which makes damage dealing stochastic. When an attacker stationed on the hill strikes opponents in the plain, the yellow line indicates deterministic damage dealing with a 100% probability. On the other hand, the orange line shows probabilistic damage with a 50% when an attacker positioned on the plain assaults an opponent placed on the hill. Hence, exploration of MARL algorithms must encourages agents to find these advantages during the training phase, as agents are not explicitly rewarded for using these features. Other features like trees and stones in battlefields block each unit's sight equally for both allies and opponents, allowing troops to avoid damage.

We devise a combination of units that can validate the cooperative decision-making of several agents. As seen in Table 2, we consider cooperation between armored troops and infantry troops. The marines and marauders can share their sights for armored troops to attack them. Second, the tank in SMAC$^+$ can transform from General Tank to Siege Tank in a single action and vice versa, so that the tank should be trained to utilize the best mode in response to the situation. As mentioned, SMAC$^+$ introduces a communication option allowing armored troops attack distant enemies by utilizing ally sights as seen Figure 1. The technical specifics of SMAC$^+$ communication and unit attributes are documented in Appendix C.

Table 2: List of units and their roles in SMAC$^+$

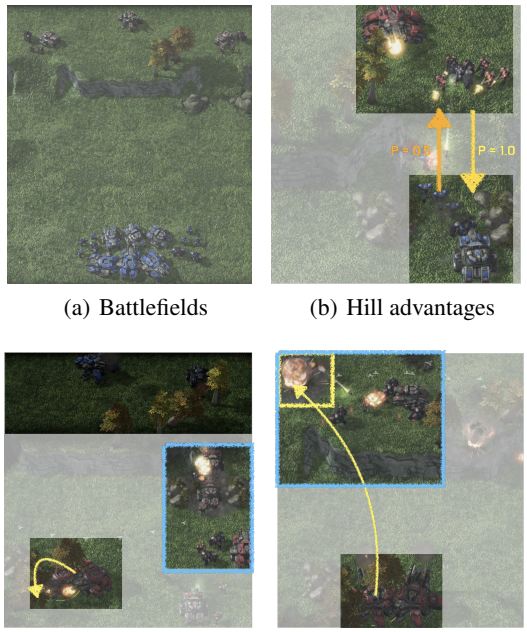
Unit	Roles
Marines	Intensive firepower Search for opponent positions
Marauders	Damage absorption from enemy Enemy's movement restriction
Tank	Strong firepower against individuals Limited range of fire
Siege Tank	Strong firepower against groups Limited range of sight Protection from close-range attack

2.2. Defensive Scenarios

The defensive scenarios emphasize the importance of agents defeating adversaries utilizing topographical advantage. We control the difficulties of the defensive settings as follows. We define different difficulties by making differences of the unit supply between allies and opponents, considering each unit's size, health, and firepower. Note that we do not make any difference between allies and enemies in the number of Tank units. Although we can easily increase the difficulty by adding more Tank units to the enemy, just one more Tank results in a huge imbalance because Tank units have overwhelming health points and higher firepower than other units. The second way is to change the approach direction of opponents. The opponents can attack forward to one or both sides. Offense from two sides is more difficult to win for defenders than from one side because agents must train on how to distribute ally units to each entry according to the direction of the adversary's approach. We categorize the problems into Easy, Hard, and Super-Hard as given in Table 3. For the detailed description of the defensive scenarios' sequential action selection of agents, see Appendix C.

2.3. Offensive Scenarios

Offensive scenarios provide learning of sub-tasks without direct incentives in MARL challenges. We suggest that agents should accomplish goals incrementally, such as eliminating adversaries after locating them. As detailed in Table 1, in SMAC$^+$, agents will not even have a chance to get a reward if they do not encounter adversaries. This is because



(a) Battlefields (b) Hill advantages

(c) Initial sights (d) Attack via shared sights

Figure 1: The characteristics of SMAC$^+$. The yellow line describes ally strike, while the orange line means enemy attack. The blue rectangle represents the shared sights between the agent and the alliance.

Table 3: Taxonomy of offensive and defensive scenarios. The complicated option in the offensive scenario means a case in that the possible route is extremely constrained. A spread formation of opponents may sustain more damage than a gathered one when agents initiate an offense.

	Defensive scenario		Offensive scenario	
	Supply difference	Opponents approach	Distance from opponents	Opponents formation
Easy	-2	One-sided	Near	Spread
Hard	-3	Two-sided	Near	Gathered
Super Hard	-6	Two-sided	Distant	Spread
Ultra Hard	-	-	Complicated	Spread

there are only three circumstances in which agents can get rewards: when agents defeat an adversary, kill an adversary, or inflict harm on an adversary. As a result, the combination of sub-tasks, environmental benefits and main challenges necessitates not only agent micro-management, but also a well-executed team strategy. We control the difficulties using formation and distance. As seen in Table 3, *Spread* means enemies placed on two entrances and *Gathered* is that enemies located at one entrance on the hill. The *Gathered* situations is more difficult since the winning strategy is limited to entering each entrance and attack adversaries simultaneously. Another method of altering difficulties is to change the distance from enemy units. There are *Near* and *Distant*. The *Complicated* scenario has the same distance as the *Distant* scenario, but includes extra hill topography. For a detailed description of offensive scenarios' sequential action selection of agents and some of their characteristics, see Appendix C.

3. Experiments

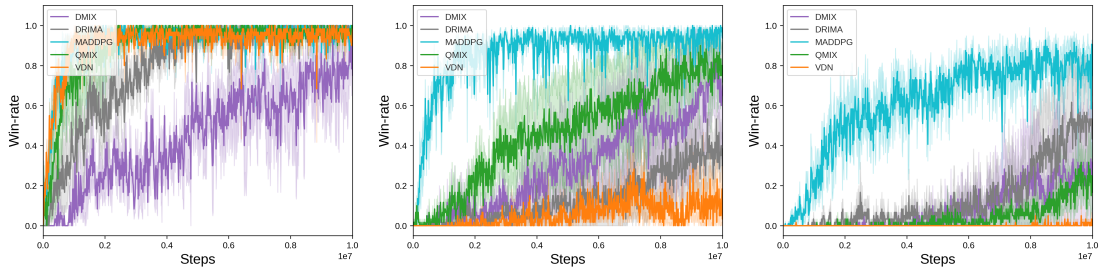
In this section, we present experimental results in SMAC+. We aim to demonstrate how the algorithms designed for micro-control perform in the proposed environment that requires completion of sub-tasks. We evaluated 11 MARL algorithms with CTDE paradigm and report results of 5 best baselines such as VDN (Sunehag et al., 2017), QMIX (Rashid et al., 2018), DMIX (Sun et al., 2021), DRIMA (Son et al., 2022) and MADDPG (Lowe et al., 2017). We ran each algorithm for a total of ten million timesteps with 3 different random seeds. During training, samples were obtained using parallel simulators (called parallel runner in SMAC) except MADDPG algorithm since it should update models at every episode hence use a single simulator (called episodic runner in SMAC). The trained model was test at every ten thousand timesteps during 20 episodes and the percentage of winning episodes is denoted as win-rate. More training details and hyperparameters are presented in Appendix C and the detailed description of all algorithms and the entire results can be found in Appendix D.

3.1. Defensive Scenarios

In defensive scenarios, the agents should learn utilizing topographical features as well as micro-controls. Figure 2 shows the average test win-rate of 5 best-performing algorithms in defensive scenarios. VDN failed to train except the easy scenario. A possible reason is that learning micromanagement of the agents was enough to overcome a small supply difference of the easy scenario, hence its assumption of linear value factorization worked. However, such assumption is too simple to address the additional sub-tasks of the harder scenarios. While QMIX, another value factorization algorithm, achieved the best result in defense hard scenario, DMIX, a variant of DFAC, and DRIMA performed better in defense Super-Hard case. It appears an additional consideration such as a distributional learning approach is required to accomplish the given sub-tasks. Note that MADDPG was trained with the episode runner and the model was updated 20 more times, hence the presented results are not directly comparable. Nonetheless, MADDPG solved all scenarios well. As MADDPG utilizes off-policy learning like other value-based algorithms, it could achieved the best performance due to its sample efficiency.

3.2. Offensive Scenarios

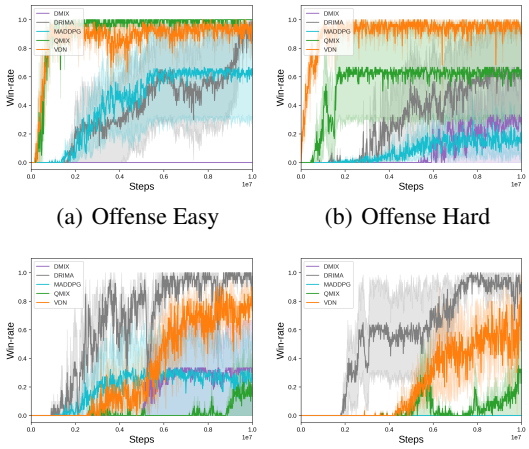
In offensive scenarios, an additional sub-task of finding enemies is also considered, hence an efficient exploration



(a) Defense Easy (b) Defense Hard (c) Defense Super-Hard

Figure 2: Baselines experiments on the defensive scenario. The experience buffers of all method except MADDPG are generated by running simulators in parallel.

strategy becomes more import. As shown in Figure 3, VDN and QMIX outperformed all the other algorithms in easy and hard scenarios. This is reasonable because the number of ally units larger than the number of enemies, it is relatively easy to learn the micro-management after finding opponents and such value function factorization learn micro-management methods well. However, in Super-Hard and Ultra-Hard scenarios, exploration becomes much more difficult as the distance between allies and enemies is complicated. In Super-Hard and Ultra-Hard cases, DRIMA outperformed VDN and QMIX, demonstrating the significance of the exploration strategy. We also found that unlike in defensive scenarios, MADDPG exhibited low performance even with the episodic runner. It appears policy-based algorithms suffers from poor exploration.



(a) Offense Easy (b) Offense Hard

(c) Offense Super-Hard (d) Offense Ultra-Hard

Figure 3: Baselines experiments on the offensive scenario. The experience buffers of all method except MADDPG are generated by running simulators in parallel.

We present comparison between well-learned agents and poorly-learned agents using sequence photographs in the offensive scenarios. As shown in Figure 4, the learned agents can move forward for approaching opponents. When it comes to DRIMA, the majority of agents go to opponents at the same time and begin fighting with them, however MADDPG does not push agents to move forward to enemies and few units go to hill, but are eliminated by enemies. This fact tells us that DRIMA that exploit risk-base additional exploration is effectively exploration for learning sub-task and boosts the performance.

(a) DRIMA

(b) MADDPG

Figure 4: Case study of Ultra-Hard scenario for DRIMA and MADDPG. We capture the scenes at the three seconds and nine seconds respectively for each algorithm.

4. Conclusion

We propose a novel benchmark called SMAC+, where agents are supposed to implicitly learn a way to complete sub-tasks or to use environmental benefits without precise reward functions. We anticipate that this work serves as a useful benchmark for researchers in MARL to address learning sub-tasks and environment advantages while concurrently to accomplish final objectives.

References

Albrecht, S. V. and Ramamoorthy, S. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. *arXiv preprint arXiv:1506.01170*, 2015.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pp. 1096–1105. PMLR, 2018a.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

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

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, 2019.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hausknecht, M. and Stone, P. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

Iqbal, S. and Sha, F. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Mordatch, I. and Abbeel, P. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Oliehoek, F. A., Spaan, M. T., and Vlassis, N. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

Papoudakis, G., Christianos, F., Schäfer, L., and Albrecht, S. V. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Peng, B., Rashid, T., Schroeder de Witt, C., Kamienny, P.-A., Torr, P., Böhmer, W., and Whiteson, S. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34, 2021.

Pu, Y., Wang, S., Yang, R., Yao, X., and Li, B. Decomposed soft actor-critic method for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2104.06655*, 2021.

Qiu, W., Wang, X., Yu, R., Wang, R., He, X., An, B., Obraztsova, S., and Rabinovich, Z. Rmix: Learning risk-sensitive policies forcooperative reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34, 2021.

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 *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Sohn, S., Oh, J., and Lee, H. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In *Advances in Neural Information Processing Systems*, 2018.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 5887–5896. PMLR, 2019.

Son, K., Kim, J., Yi, Y., and Shin, J. Disentangling sources of risk for distributional multi-agent reinforcement learning. 2022. URL https://openreview.net/forum?id=5qwA7LLbgP0.

Sun, W.-F., Lee, C.-K., and Lee, C.-Y. Dfac framework: Factorizing the value function via quantile mixture for multi-agent distributional q-learning. *arXiv preprint arXiv:2102.07936*, 2021.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Wolpert, D. H. and Tumer, K. Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pp. 355–369. World Scientific, 2002.

Xu, J., Zhong, F., and Wang, Y. Learning multi-agent coordination for enhancing target coverage in directional sensor networks. *Advances in Neural Information Processing Systems*, 33:10053–10064, 2020.

Yang, D., Zhao, L., Lin, Z., Qin, T., Bian, J., and Liu, T.-Y. Fully parameterized quantile function for distributional reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

A. Notation

Partially Observable MDP In Markov Decision Process (MDP), agents can observe all the environments like (Hausknecht & Stone, 2015) if we don't use screen flickering technique where environment is stochastic. But in real world, MDP is not achievable generally. Instead, Partially Observable MDP (POMDP) is observable in the real world rather than MDP. For example, human being can observe his around where he is located but cannot observe where he is not located. So, POMDP is an MDP that agents can observe things only in their sight and agents decide their actions based on their observations. This is called Decentralized-POMDP (Dec-POMDP). In MARL settings, we consider simulator's environment as a Dec-POMDP setting.

Formally, Dec-POMDP is given with a tuple G = $\langle S, U, P, r, Z, O, n, \gamma \rangle$. s $\in$ S is the true state that the environment provides. $a \in$ A $\equiv \{1, ..., n\}$ is an agent that chooses an action $u^a \in$ U which forms a joint action space $\mathbf{u} \in \mathbf{U}$ where n is the number of agents. P($s\prime$ | s, $\mathbf{u}$) : S $\times$ $\mathbf{U}$ $\times$ S $\rightarrow$ [0, 1] is a transition probability function. All agents in Dec-POMDP receive shared reward, so the reward function is $r(s, \mathbf{u}) : S \times \mathbf{U} \rightarrow \mathbb{R}$. Observation function is $O(s, a) : S \times A \rightarrow Z$ that determines agents' observation $z^a \in Z$. $\gamma \in [0, 1)$ is a discount factor for the reward.

B. Related works

Several reinforcement learning environments were designed to evaluate the ability to optimally execute sub-tasks to accomplish the desired goal. (Sohn et al., 2018) introduced **Playground** which is a 2-dimensional environment characterized by a randomly generated sub-task graph that represents dependencies among sub-tasks. The goal of the agent is to find the optimal sequence of sub-tasks (for example, picking up a particular object) to maximize the reward. **FrankaKitchen** environment contains several household items and the task is to learn manipulating a 9 DoF robot arm to reach a particular state configuration. (Gupta et al., 2019). For example, the desired state requires the light on and the microwave open and each sub-task gives the sparse-reward. In contrast, SMAC+ scenarios only give rewards for the final objective (defeating enemies) and not to sub-tasks, hence the agents should implicitly learn to do sub-tasks.

Centralized Training & Decentralized Execution (CTDE) is a widely adopted regime for MARL tasks (Oliehoek et al., 2008). The agents are trained with access to all global information. but act solely on local observations during execution. The CTDE research has been led by the development of environments. **Multi-agent Particle Environment** provides a suite of 2-dimensional cooperative and competitive tasks where an agent interacts with other agents and landmarks. (Mordatch & Abbeel, 2018) came up with the environment that allows for multi-agent's communication. Based on this framework, (Lowe et al., 2017; Iqbal & Sha, 2019) developed more general and challenging tasks. (Albrecht & Ramamoorthy, 2015) suggested **Level-Based Foraging** environment. Each agent is responsible for navigating the grid-world map and collecting items. This collection is successful only if the total level of all engaged agents levels is equal to or greater than the level of the item. (Xu et al., 2020) proposed a problem that requires sensor coordination in the two dimensional space in order to get ideal coverage while consuming low power, such as no missing targets or reducing redundant coverage. This environment regards action spaces as incrementally increasing or decreasing orientation of each sensor in 5 degrees. Recently, (Papoudakis et al., 2021) developed a **Multi-Robot Warehouse** environment in consideration of the sparse reward situation in MARL by rewarding agents for correctly completing a set of actions. Agents should identify and locate requested shelves to workstations on a grid-world warehouse. (Peng et al., 2021) developed **Multi-Agent MuJoCo**, allowing agents have cooperative and continuous robotic control. This environment assesses the ability of multiple agents inside a single robot to work together to complete a task.

Most of the aforementioned environments are composed of simple tasks on limited state and action spaces. To incorporate more challenging factors such as partial observability and high-dimensional spaces, (Samvelyan et al., 2019) offered **SMAC**, an environment based on the Real-Time Strategy game StarCraft II. SMAC is used to assess micro-control of each agent to cooperatively eliminate opponents under unfavorable situations. Specifically, all scenarios in the SMAC are categorized into three levels of difficulties, referred to Easy, Hard, Super-Hard and such difficulties are determined by the number of allies units and its combination. Along the SMAC environment, (Papoudakis et al., 2021) provided comprehensive benchmark results of 9 MARL algorithms total twenty-five tasks including 2-dimensional tasks and SMAC (Papoudakis et al., 2021). They illustrated that no algorithm outperforms all tasks; nevertheless policy gradient based MARL algorithms marginally have a high performance. However, since recent studies found that distributional value based algorithms consistently achieve high performance on the SMAC, this study does not reflect the current trend in research(Sun et al., 2021; Son et al., 2022).

Although its challenging characteristics, there is no topographical change in SMAC except one scenario and only simple

grounds are considered. Furthermore, as mentioned earlier, all scenarios considered only the cases against naturally approaching opponents. SMAC$^+$ introduces more complex scenarios where agents should learn to exploit environmental benefits and implicit sub-tasks. Furthermore, we take a step to evaluate the abilities of MARL algorithms such as exploration strategies in the proposed SMAC$^+$ environment.

C. SMAC$^+$ details

C.1. Environment Features

State and Observation features The features of original SMAC environment (Samvelyan et al., 2019) are described in Table C.1 and Table C.2. ut means the number of different unit types in the given scenarios. Each agent has 6 basic actions and can attack a particular enemy, so the the dimension of action space is 6 plus the number of enemies. The global state information is provided during the centralized training phase while agents must utilize its own local observation during the decentralized execution phase.

Table C.1: Global state information in SMAC

Feature	Description	Dimension
Ally	health, cooldown, relative x, relative y unit type, last action	number of allies $\times$ $(4 + ut + 6+$number of enemies$)$
Enemy	health, relative x, relative y, unit type	number of enemies $\times(3 + ut)$

Table C.2: Observation information in SMAC

Feature	Description	Dimension
Move	basic movement	4
Own	health, unit type	$1 + ut$
Ally	visibility, health, distance, relative x, relative y, unit type	number of allies $\times(5 + ut)$
Enemy	visibility, health, distance, relative x, relative y, unit type	number of enemies $\times(5 + ut)$

The observation features in SMAC$^+$ are shown in Table C.3. There are additional information of terrain levels such as pathing grid and terrain height. We consider terrain features like hill and entrance, so that agents must distinguish opponents are located higher or not in these scenarios. Furthermore, the agent receive the last actions of all units in its sight range as part of observations. These features are also provided in SMAC but not included in default setting. In addition, we incorporated z-coordinate information and the agent receives its own coordinate values. The unit type indicates a total of eight entities in this scenario; Marine, Marauder, General Tank (which cannot switch to siege mode), Siege Tank in general mode, Siege Tank in siege mode (because siege tank may pick their mode) and three neutral building. The dimension of the last action is specifically described in Table C.4. Overall, an agent's observation space is composed of [movement feature, agent's own feature, enemies feature, allies feature, neutral buildings' feature] which are all visible when objects are in the agent's sight range. As the global state information, SMAC$^+$ provides two options. We may use same information of SMAC, or a concatenation of all agents' observations. We empirically found that a concatenation of all observations gave better performance across SMAC+ scenarios, hence we use concatenated observations as the global state in default.

Table C.3: Observation information in SMAC$^+$

Feature	Description	Dimension
Move	basic movement, pathing grid value	$4 + 8$
Own	health, x, y, z, unit type	$4 + ut$
Ally	visibility, health, distance, relative x relative y, relative z, unit type, last action	number of allies $\times(14 + 7+$ number of enemies and neutrals$)$
Enemy	visibility, health, distance relative x, relative y, relative z, unit type	number of enemies $\times$ 14
Neutral	visibility, health, distance relative x, relative y, relative z, unit type	number of neutrals $\times$ 14

Action space The total action set in SMAC$^+$ consists of basic action, number of enemies, and number of neutral buildings as shown in Table C.4. The basic action is basically same as SMAC, but we additionally consider an units skill. Unit skill is designed for a general tank to change to siege mode and vise versa. If others units choose unit skill action, it regards as stop action. We also add the number of neutral buildings into the action set because those can be removed by agents if needed. Therfore, the action space of SMAC$^+$ is much larger than SMAC.

Table C.4: Action space in SMAC$^+$

Basic action	Attack enemies	Attack neutral buildings
North, South, East, West, No-op, Stop, Skill	Enemy 1, $\cdots$ Enemy n	Builidng 1, $\cdots$, Building n

Unit features All units sight range and shooting range are different as shown Table Table C.5. The fire power has two types. Basically, every unit has its own fire power. Except marines, the remaining units give a special damage according to unit types. Tank has a machinery and heavy armor character, Marauder has a heavy armor character and Marine has nothing special. Then seeing the Table Table C.5, Marauder has a enhanced fire power on units who have machinery attributes. Similarly, Tank has a enhanced fire power on units who have heavy armor attributes. We reduced fire power of Siege Tank because of the property of long-ranged and splash damage for game balance. So according to the scenario and training step, usage of Siege Tank might be different.

Table C.5: Shooting, Sight range & Fire power in SMAC$^+$: Fire power that has two values means that left one is default fire power and right one is enhanced fire power according to the opponent's characteristic.

Range	Marine	Marauder	Tank	Siege Tank
Shooting	6	7	8	17
Sight	9	9	9	9
Fire power	6	10 / 30	15 / 25	5 / 10

Communication Feature We describe a critical feature of SMAC$^+$, communication in detail. The original SMAC environment uses a visibility matrix which is similar to adjacency matrix to indicate visible units for each agent. The dimension of the row is the number of the agents while the dimension of the column is the all the units in the environment including the agents. The entity of matrix (i, j) is 1 if unit j is in the sight range of the agent i and 0 otherwise. The agent's observation only receives the features of visible units.

In SMAC$^+$, to allow communication among the agents, we redesign the visibility matrix by setting communication range. If agent i and j is in the communication range of each other, then they share the visible units. For example, if the enemy k is in the sight range of the agent j but no in the agent i's, visibility matrix entity (i, k) is changed from 0 to 1. The agents use this modified visibility matrix to get observation features. The sight range is 9 for all agents, and the communication range is 16 for the Siege Tank and 12 for other units. Researchers can easily turn on or off the communication function by setting obs_communicate _info = True or False in environment setting file. The default value is True. Note that this setting does not violate CTDE assumption, as the agent receives additional Enemy feat, Ally feat, Neutral building feat of Table Table C.3 according to the updated visibility matrix, but the global state or the combination of the entire observations are not accessible. obs_broadcast _info is another communication function that has no limit on communication range. When you use these function, these two character should not be all True. Turning off all these function makes no communication among agents.

Figure C.1 shows the training curve of VDN algorithm in defensive easy scenario with and without the communication function. Even though there is no significant performance gap, the learned policies are quite different. When the agents are not allowed to communicate, as shown in Figure C.1(c) they show standing alongside at the entrance of the hill and directly engage the enemies. Even though the agents can benefit the stochastic damage dealing from feature of hill, not utilizing the feature makes it hard to win due to the supply difference between enemies and allies. On the other hand, in Figure C.1(b), the Marauder decoys the enemies while the Marines hide behind the neutral buildings which block the sight of the enemies and allies behind of the neutral buildings when they are positioned in a row with a neutral buildings. This implies that the agents learned to utilize the sight-blocking capabilities of the neutral building blocks with the communication function.

(a) Training Curve (b) No Communication (c) Inter-agent Communication

Figure C.1: Case Study on Communication among the agents

The possible reason is that the agents can behave apart from one another due to the observation sharing. It would be an interesting challenge to design an algorithm to learn such polices without communication.

Hyperparameters We describe about hyperparameters we used. Hyperparmeters are almost same with experiments conducted in the other papers for the fairness except training steps. Throughout the training, we anneal ϵ from 1.0 to 0.05 over 50000 training steps and fix the ϵ during the rest of the training time. We fix $\gamma = 0.99$. Replay buffer is capable for containing most recent 5000 episodes and we sample 32 size of batch from the replay buffer randomly. We update target network with current network every 200 time steps. We have 10050000 steps for each training. The details about hyperparameters are on Table C.6.

Table C.6: Hyperparameters of Network

Hyperparameter	Value	Description
Training step	10050000	how many steps we trained the model
Discount factor	0.99	how we estimate the future rewards
Learning rate	5×10^{-4}	learning rate by RMSProp optimizer
Target update period	200	update period of target network
Replay buffer size	5000	maximum container size of the past samples
Batch size	32	number of samples for each update
Batch size run	20	number of parallel simulator
ϵ	1.0 to 0.05	ϵ-greedy exploration over 50000 training steps
Number of sampling τ	32	number of quantile fraction samples in DFAC

C.2. Map Features

Defensive scenarios By the extended exploration problem, allies should scout nearby the hill where allies are located to find out the enemies' offense direction. And if trained well, the agents won't try to get out of the hill which make enemies' offense as a stochastic damage. Also allies remove the trees (neutral buildings) that block the agents' sight for making it easy to secure original sight range and find the enemies' offense direction which means eliminating the uncertainty which can be found in Figure C.2(a). After finding the enemies offense direction, allies need to be located at some proper location that provide reduce-able damage from enemies offense shown as Figure C.2(c) which shows that allies stand on the hill not allowing enemies to enter the hill. If agents are not trained well, they failed to find the enemies offense direction allowing occupation of the allies' respawn place to enemies.

Offensive scenarios In the offensive scenarios in SMAC$^+$, allies must find the place where the enemies are located. However, finding enemies far away from the allies is actually not an easy problem. Because the action space in SMAC$^+$ becomes much larger than the original SMAC (Samvelyan et al., 2019) environment caused by the neutral buildings. Therefore, it is difficult for agents to find enemies at the beginning of learning. To find the enemies, allies should endure the

(a) Removing buildings (b) Terrain advantage (c) Failed to scouting

Figure C.2: Sequential screenshots in Defensive Scenarios

damage from the enemies like shown as Figure C.3(a). When finding the enemies' location and deployment, allies attack the enemies with proper strategy if agents trained well as shown in Figure C.3(b) and Figure C.3(c). But if agents are not trained well in finding the enemies, agents get overfitted to the bad samples resulting the bad behavior as shown in Figure C.3(d). Offensive scenario is hard to solve for agents because of the sparse rewards problem and multi-goal objectives, we set the number of ally units quite larger than defensive scenarios' enemies' number for observing the training development. We can reduce the number of ally units for future work, if proper algorithms are developed.

(a) Finding enemies (b) Before communication (c) After communication (d) Failed to scouting

Figure C.3: Sequential screenshots in Offensive Scenarios

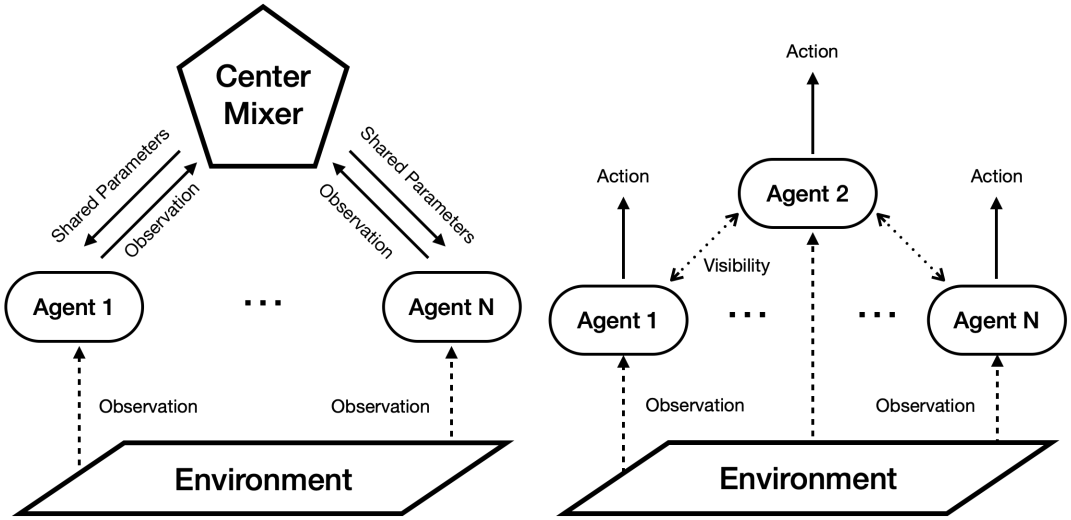
Table C.7: SMAC$^+$ Scenarios. Scenarios consist of 7 scenarios each 3 and 4 for defensive and offensive scenarios. All units are only from Terran race for the reflecting realistic part of the world. SG, Mar and M mean each Siege Tank, Marauder, Marine

Defense		
Scenario	**Ally Units**	**Enemy Units**
Easy	1 Mar & 4 M	1 Mar & 6 M
Hard	1 SG Tank & 1 Tank & 1 Mar & 5 M	2 Tank & 2 Mar & 9 M
Super Hard	1 SG Tank & 1 Tank & 1 Mar & 5 M	2 Tank & 3 Mar & 10 M
Offense		
Scenario	**Units Composition**	**Distance & formation**
Easy	3 SG Tank & 3 Tank & 3 Mar & 6 M	Near & Spread
Hard	(Allies)	Near & Gathered
Super Hard	**vs**	Distant & Spread
Ultra Hard	1 SG Tank & 2 Tank & 2 Mar & 4 M	Complicated & Spread

D. Algorithms

In this section, we describe how each algorithm works. We divided algorithms in detail taxonomies as Value-based, Policy-based, and Distribution-based in this section. Value-based and Distribution-based algorithms showed better performance so far than policy based algorithms. We think that this was induced by a sample efficiency and suitable action space for discrete action choice which is advantages of value-based algorithms. We will describe about Distribution-based algorithms which got popular from (Bellemare et al., 2017) that achieved state-of-the-art performance in Atari and Mujoco environments and also in SMAC's challenging hard scenario.

Centralized Training & Decentralized Execution In early stage of MARL, decentralized training & decentralized execution (DTDE) is the main framework of training model like training single agent RL model. But it was very hard to train model with the DTDE framework in multi-agent setting, because of the more enhanced randomness by other agents' action selection, Partially Observable MDP setting (POMDP), and insufficient information depending only on one's observation during training the model's parameters. So, now centralized training & decentralized execution (CTDE) (Oliehoek et al., 2008) framework is highly employed in Multi-Agent Reinforcement Learning (MARL) domain because of the advantage of informative training data like global state or gathering all of the observation of the agents. Most of the recent MARL algorithms have developed on the basis of CTDE learning framework which has access on all information at training step but has access on individual information only at execution step as shown in Figure D.4. Gathering all agents' information at the center enables learning in POMDP and complex setting. We describe the development of the algorithms based on CTDE from VDN(Sunehag et al., 2017) to DFAC(Sun et al., 2021) which uses distributional RL algorithm.



(a) Centralized Training (b) Decentralized Execution

Figure D.4: CTDE Framework

D.1. Value-based

Value-based algorithms were developed for environments that require discrete action space. So, SMAC environment had started with value-based algorithms naturally and showed better performance than policy-based algorithms. Value-based algorithms use n-step (1-step in general) temporal difference error (TD-error) for updating critic parameters. Basically these algorithms follow below equation as a basis of loss function:

$$\delta_{td-error} = (R + \gamma * \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a))^2 \tag{D.1}$$

where s and a represent state and action. Prime $'$ means next step and θ, θ^- means parameters of behavior and target network each. In general, state and action are denoted as s, a, but from now we will use the notation of h and u for state and action which are also utilized as a method of notation state and action in RL domains. Before entering to the value-based algorithms, we have to know about the condition which is needed for stable learning and avoiding lazy agents that is called IGM(Individual Global Max) condition which means individual maxiaml Q-values consist of total (or global) maximal Q-values. It can be represented as follows:

$$\arg\max_u Q_{total}(\mathbf{h}, \mathbf{u}) = \begin{pmatrix} \arg\max_{\mathrm{u}_1} Q_1(\mathrm{h}_1, \mathrm{u}_1) \\ \vdots \\ \arg\max_{\mathrm{u}_\mathrm{N}} Q_N(\mathrm{h}_\mathrm{N}, \mathrm{u}_\mathrm{N}) \end{pmatrix} \tag{D.2}$$

where h and u represents history of observation and action each. For CTDE framework, value-based algorithms are consist of utility function and mixer. Utility function receive individual observation every time step and outputs action which is

comprised of DRQN(Hausknecht & Stone, 2015) for POMDP setting that take history of observation. And mixer takes all agents' individual Q-values and outputs joint Q-values for calculating temporal difference error for centralized training. For decentralized execution, mixer is not used but only utility function is used. Most of the utility function in various algorithms is almost same, but architecture of mixer is highly related to the training model's architecture.

IQL Value-based algorithms for MARL start from decentralized training & decentralized execution (DTDE) setting which means that train agents independently. Independent Q-Learning (IQL) trains agents and execute action in a decentralized manner as the same way of training single agent not caring about whether there are other agents near itself or not. IQL has only utility function for training and execution and no mixer for centralized training. So generally, agents do not share parameters contrary to the current CTDE based algorithm that share parameters among all agents. For updating IQL model's parameters, temporal difference (TD) error is calculated individually and in this point IQL violates Markov Decision Process (MDP) assumption which is needed for converging Q learning. This is because environment appears to be non-stationary for each agent for other agents' actions. So, IQL shows not good performance in SMAC(Samvelyan et al., 2019) and SMAC+ scenarios but sometimes show good performance in specific scenarios like *2s vs 1sc* and *bane vs bane* that obstinately don't demand cooperative strategy between agents.

VDN Value Decomposition Network(Sunehag et al., 2017) (VDN) is a simple methods for cooperative MARL algorithm that adapt CTDE training framework first time to overcome IQL's learning shortcoming that only use individual observation for training the model. VDN just sum up the individual action-state value (which is called additivity), and use it as a joint Q-value which is utilized for calculating joint TD-error that is needed for centralized training. So the mixer in VDN is summation function of individual Q-values. The joint Q-value can be acquired as follows which satisfy the IGM condition by additivity:

$$Q_{joint}\left(\mathbf{h}, \mathbf{u}\right) = \sum_{i=1}^{N} Q_i\left(h_i, u_i\right) \tag{D.3}$$

where Q and N means Q-value and number of agents. In this algorithm all agents share parameters and the performance and the performance start to increase substantially compared to IQL algorithm.

QMIX VDN(Sunehag et al., 2017) has limitation on expressing complexity of the centralized joint Q-value that can ignore the additional global state or gathered observational information by just summing up the all individual Q-values. QMIX (Rashid et al., 2018) develop the architecture of VDN which utilizes additivity for making Q_{joint} satisfying IGM condition. QMIX adapt mixing network as a mixer for weighted linear summation of individual Q-values rather than simple linear summation like VDN where the parameters for mixing network are made by hypernetwork. And the IGM condition is satisfied with the monotonocity that make the parameters of mixing network positive values and enforce the joint-Q-value as monotonic in the per agent Q-values, which can be represented as:

$$\frac{\partial Q_{joint}}{\partial Q_i} \geq 0, \forall i \in \mathbb{N} \tag{D.4}$$

which enable tractable optimization of the joint Q-value in off-policy learning. To make joint Q-value using mixing network, algorithm utilizes value factorization like VDN, meanwhile it uses a mixing network to compute the total value function. In the Equation D.5, ψ means a linear combination function by a mixing network which can be denoted as follows:.

$$Q_{total}\left(\mathbf{h}, \mathbf{u}\right) = \psi\big(Q_1\left(\mathrm{h}_1, \mathrm{u}_1\right), \ldots, Q_n\left(\mathrm{h}_N, \mathrm{u}_N\right)\big) \tag{D.5}$$

By the mixing network and monotonicity, QMIX outperformed other algorithms like IQL, VDN, COMA etc. QMIX algorithms has become the most popular algorithms in MARL and still lots of variants of QMIX have continued to emerge.

QTRAN Former VDN and QMIX algorithms can handle the MARL issue as a way of being trapped in a constrained structures which are called additivitiy and monotonicity. QTRAN tried to relax the constraints on architecture's structure for abundant expressiveness by transforming the Q_{joint} into an easily factorizable value. Mitigating the constraints, QTRAN proved guarantee for general factorization better than VDN and QMIX. The relaxed condition can be represented as:

$$\sum_{i=1}^{N} Q_i\left(\mathrm{h}_i, \mathrm{u}_i\right) - Q_{joint}\left(\mathbf{h}, \mathbf{u}\right) + V_{joint}(\mathbf{h}) = \begin{cases} 0 & \mathbf{u} = \bar{\mathbf{u}} \\ \geq 0 & \mathbf{u} \neq \bar{\mathbf{u}} \end{cases} \tag{D.6}$$

where,

$$V_{joint}(\mathbf{h}) = \max_{\mathbf{u}} Q_{joint}(\mathbf{h}, \mathbf{u}) - \sum_{i=1}^{N} Q_i(\mathrm{h}_i, \bar{\mathrm{u}}_i) \tag{D.7}$$

where $\bar{u}$ means optimal action and $V(\mathbf{h_{joint}})$ is a discrepancy that occurs from the Partially Observable MDP (POMDP) environment which can corrects the discrepancy between centralized Q_{joint} and the sum of individual Q which is $\sum_{i=1}^{N} Q_i(h_i, u_i)$. Despite of the expressive power that relaxed constraints on architecture's structure, QTRAN shows poor performance in many complex scenarios.

D.2. Distribution-based

These days policy-based distributional RL algorithms are emerging in MARL domain but by the the fact that SMAC demands discrete control optimization, usually distributional RL (DRL) for SMAC is based on the value-based RL algorithm so far. DRL algorithm has been developing substantially with great attention since (Bellemare et al., 2017), (Bellemare et al., 2017), (Dabney et al., 2018b) suggested new concept of value-based RL which outputs a distribution of return per action. (Bellemare et al., 2017) fixed possible return value and made model to predict probability of returns with projected KL-divergence loss function. (Dabney et al., 2018b) approximate quantile regression as a output of return distribution per action and fixed probability of each returns with $\frac{1}{N}$ where N means number of returns per action and made model to predict value of returns. And (Dabney et al., 2018b) utilize Wasserstein metric as a loss function measuring distance of TD-error between $Q(s_t, a_t)$ and $R(s_{t+1}, a_{t+1}) + \gamma * max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$. (Dabney et al., 2018a) samples quantile fractions uniformly in $[0, 1]$ which were fixed in (Dabney et al., 2018b). And also model outputs value of returns corresponds to quantile fractions embeded with cosine function. (Yang et al., 2019) approximates both of probability and value of returns contrary to former algorithms that only approximate probability or value of returns. So, DRL algorithms develop following the sequence of C51(Bellemare et al., 2017)$\longrightarrow$QR-DQN(Dabney et al., 2018b)$\longrightarrow$IQN(Dabney et al., 2018a)$\longrightarrow$FQF(Yang et al., 2019).

DFAC DFAC(Sun et al., 2021) is a first algorithm that combines distributional RL and multi-agent RL which is based on the IQN(Dabney et al., 2018a) algorithm. Especially, IQN was used for distributional output sampling quantile fractions from $\mathrm{U}[0, 1]$ and approximating return values with quantile regression. By mean-shape decomposition, authors integrated distributional perspective in the multi-agent setting not violating the IGM condition which can be written as:

$$\arg\max_{\mathbf{u}} \mathbb{E}[Z_{joint}(\mathbf{h}, \mathbf{u})] = \begin{pmatrix} \arg\max_{\mathrm{u}_1} \mathbb{E}[Z_1(\mathrm{h}_1, \mathrm{u}_1)] \\ \vdots \\ \arg\max_{\mathrm{u}_\mathrm{N}} \mathbb{E}[Z_N(\mathrm{h}_\mathrm{N}, \mathrm{u}_\mathrm{N})] \end{pmatrix} \tag{D.8}$$

that can be satisfied by the following DFAC Theorem (mean-shape decomposition):

$$\begin{aligned} Z_{joint}(\mathbf{h}, \mathbf{u}) &= \mathbb{E}[Z_{joint}(\mathbf{h}, \mathbf{u})] + Z_{joint}(\mathbf{h}, \mathbf{u}) - \mathbb{E}[Z_{joint}(\mathbf{h}, \mathbf{u})] \\ &= Z_{mean}(\mathbf{h}, \mathbf{u}) + Z_{shape}(\mathbf{h}, \mathbf{u}) \\ &= \psi(Q_1(\mathrm{h}_1, \mathrm{u}_1), \ldots, Q_N(\mathrm{h}_N, \mathrm{u}_N)) \\ &+ \Phi(Z_1(\mathrm{h}_1, \mathrm{u}_1), \ldots, Z_N(\mathrm{h}_N, \mathrm{u}_N)) \end{aligned} \tag{D.9}$$

which is proved to satisfy IGM condition. DFAC(Sun et al., 2021) shows outperforming performance than any other algorithms especially in *Hard* scenarios. Also this algorithm can be adapted to IQL, VDN(Sunehag et al., 2017), QMIX(Rashid et al., 2018). So the DFAC algorithm's variants are named as DIQL, DDN, DMIX that were used as our baselines.

DRIMA DFAC only considers a simple risk source but DRIMA(Son et al., 2022) considers separating risk sources into agent-wise risk and environment-wise risk which makes another hyperparameter contrary to DFAC algorithm that has a hyperparameter in risk setting, agent-wise risk. Environment-wise risk can be interpreted as transition stochasticity and agent-wise risk can be seen as the randomness induced by the other agents' action which can't be modeled by environment MDP (Markov Decision Process). In distributional multi-agent reinforcement learning algorithms((Sun et al., 2021), (Qiu et al., 2021)), models take risk level as an input to the agent utility function that output a distribution of return per an action which can be considered as a randomness by agents. But in DRIMA, agent receives agent-wise risk w_{agt} and in the process of making joint distribution of returns, joint action-value network takes w_{env} as input where agent utility function and joint action-value network composes of hierarchical architecture that resembles with QTRAN(Son et al., 2019) structure.

Network architecture of DRIMA consists of agent-wise utility function, true action-value network, and transformed action-value network. Agent-wise utility function is structured by DRQN(Hausknecht & Stone, 2015) taking w_{agt} as a input. True action-value network approximates true distribution of returns with additional representation power that receives environment-wise risk w_{env}, state s and utility functions' outputs $\mathbb{Z}_i$. Transformed action-value network is similar to the QMIX's(Rashid et al., 2018) mixing network as follows:

$$Z_{trans}\left(s, \tau, \mathbf{u}, \mathbf{w_{agt}}\right) = f_{mix}\left(z_1\left(\tau_1, u_1, w_{agt}\right), \ldots, z_N\left(\tau_N, u_N, w_{agt}\right); \theta_{mix}(s, w_{agt})\right) \tag{D.10}$$

where not considering environment-wise risk and $\theta_{mix}(s, w_{agt})$ consists of a non-negative values obtained from hypernetwork like QMIX. DRIMA outperforms other algorithms especially in offense scenarios and in our experiments, we follow the default risk setting, agent-wise risk to be seeking and environment-wise risk to be averse.

D.3. Policy-based

Policy-based algorithm directly optimize the parameters θ to approximate the optimal policy π which is especially specialized in continuous action control setting like robot system and autonomous vehicle. The simplest way of training policy-based model is independently taking policy gradient per agent. As expected, this works poorly more than IQL algorithm. Therefore, centralized training and decentralized execution is main framework for training models in policy-based algorithms. Policy based algorithms maximizes objective function that take expectation on initial state value as follows which is usually noted as J(θ):

$$J(\theta) = \mathbb{E}_{\pi_\theta}[V_0] \tag{D.11}$$

where the gradient of objective is represented as $\nabla_\theta J(\theta) = \mathbb{E}_{s,a}[\nabla_\theta \log \pi_\theta(a|s) G(s)]$ which update actor's parameter realized in algorithm REINFORCE(Sutton et al., 1999). In actor-critic algorithm, Return $G(s)$ is substituted by Q-value $Q_\pi(s, a)$ represented as same as Equation D.1 or advantage function $A_\pi(s, a)$ which demands another parameterized model to approximate critic. In multi-agent setting, we derive gradient of objective as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s_i, a_i}[\nabla_{\theta_i} \log \pi_i(a_i|s_i) Q_i^\pi(\mathbf{s}, a_1, \ldots, a_N)] \tag{D.12}$$

where Q_i^π is a centralized individual critic that takes as inputs other agents' actions and additional information like global state or gathering all observations of the agents'.

COMA COMA (Foerster et al., 2018) is policy-based method that uses actor-critic algorithm that takes contribution to solving credit assignment problem by using concept of Difference rewards(Wolpert & Tumer, 2002). In COMA (Foerster et al., 2018), a new form of advantage function was proposed to solve the 'credit assignment' problem in a multi-agent environment. Original advantage function is calculated through differences between state-value functions and action-state value function, but in COMA (Foerster et al., 2018), all other agents' actions are fixed and use the average value of the action-state value function for a particular agent's action as follows:

$$A^a\left(h, u\right) = Q\left(h, u\right) - \sum_{u'^a} \pi^a\left(u'^a | h^a\right) Q\left(h, \left(\mathbf{u}^{-a}, u'^a\right)\right) \tag{D.13}$$

which estimate credit $r\left(s, (\mathbf{u}^{-a}, c^a)\right)$ where c means default action of agents that is hard to estimate. This is why COMA uses Equation D.13 as a estimation. The average of the action-state value function for each agent's action becomes the reference value for that agent's action. It serves to determine how good an agent's action is compared to the average of action-state value.

MASAC MASAC(Pu et al., 2021) is soft actor critic (SAC)(Haarnoja et al., 2018) algorithm for multi-agent system. SAC is a actor-critic algorithm that maximizes exploration in action space utilizing entropy of distribution of action probability based on the maximum entropy RL theory. Like the original SAC for single agent, MASAC substitute individual Q-value with Q_{total} and calculate TD-error and objective gradient using same value network with QMIX(Rashid et al., 2018). So, the TD-error is calculated as follows:

$$\delta_{td-error} = \left(R + \gamma \min_{u'} Q_{\theta^-}^{target}(h', u') - Q_\theta^{total}(h, u)\right)^2 \tag{D.14}$$

which update the critic parameters. And for optimal policy, derived from soft policy iteration, objective is as follows:

$$J(\theta) = \mathbb{E}_D[\alpha \log \pi(u|h) - Q^{tot}_{\phi'}(h, u)] \tag{D.15}$$

where α controls exploration and exploitation trade-off that if α is near to 1, it means exploration more, on the contrary α is near to 0, it means exploitation more.

MADDPG MADDPG(Lowe et al., 2017) has main contribution of reducing uncertainty by taking input as actions by other agents and learns centralized individual critic and decentralized actor for policy that enable both of cooperative and competitive strategy by splitting centralized critic in individual manner. MADDPG(Lowe et al., 2017) has emerged for continuous actions like MASAC(Haarnoja et al., 2018) algorithm contrary to COMA(Foerster et al., 2018) algorithm. MADDPG(Lowe et al., 2017) is based on the DDPG(Lillicrap et al., 2015) algorithm that select deterministic action which is different from general policy-based algorithm that sample actions based on the action's distribution. The MADDPG(Lowe et al., 2017)'s gradient of objective function can be written as a new version of Equation D.12 adapting deterministic policy $a = \mu(s)$ as follows:

$$\nabla_{\mu_i} J(\mu_i) = \mathbb{E}_{h,u\sim D}[\nabla_{\theta_i}\mu_i(u_i|h_i)\nabla_{u_i} Q^{\mu}_i(\mathbf{h}, u_1, \ldots, u_N)|_{u_i=\mu_i(u_i)}] \tag{D.16}$$

where replay buffer D contains tuples of (h, u, r, h') that also utilized for critic update like Equation D.1 as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{u,h,r,h'}[(Q^{\mu}_i(\mathbf{h}, u_1, \ldots, u_N) - y)^2] \tag{D.17}$$

where $y = r_i + \gamma Q^{\mu'}_i(\mathbf{h'}, u'_1, \ldots, u'_N)|_{u'_j=\mu'_j(h_j)}$. Updating actor parameters only needs for state and action information from replay buffer not next state and next action. From this point, MADDPG takes off-policy algorithm that enable to have replay buffer which makes sample efficient algorithms like value-based. This is why MADDPG showed good performance than general actor-critic algorithm that it has similar algorithm architecture with value-based RL. And because of the nature of the MADDPG algorithm, actor and critic should be updated at every step, making that distributed training (known as a *parallel runner*) in SMAC(Samvelyan et al., 2019)) unable. In SMAC(Samvelyan et al., 2019), we can select distributed training with multiple simulator or basic training with single simulator. In this paper, we use only non-distributed setting for MADDPG as shown in Figure E.10.

E. Hyperparameter Optimization in Algorithms

We experiment all of the algorithms in Appendix D on every scenario and report the win-rate performance following the 4 taxonomy (Value-based, Distribution-based, Policy-based, Well-performing). In addition, we tried to find optimal hyperparameter setting in exploration perspective about sweeping out ϵ-greedy steps and adjusting risk-sensitiveness. Because of the architecture's constraint of MADDPG that is not compatible with training in parallel method, we present the update interval difference between episodic training and parallel training as shown in Table E.8 for reference. We mainly conduct experiment in parallel setting for the time limitation.

Table E.8: Comparing model update interval between episodic and parallel training when training 10 million steps.

	MODEL UPDATE INTERVAL		NUMBER OF MODEL UPDATE	
	TARGET NETWORK	BEHAVIOR NETWORK	TARGET NETWORK	BEHAVIOR
EPISODE	200 EPISODES	1 EPISODE	420	80000
PARALLEL	200 EPISODES	20 EPISODES	420	4240

E.1. Baseline results in SMAC$^+$

E.1.1. PARALLEL TRAINING

We show experiment results in parallel training setting in all scenarios with all algorithms as shown in Figure E.5, Figure E.7, Figure E.6. At the last of this section, we present the best top 2 algorithms from each taxonomy in a Figure E.8.

E.1.2. EPISODIC TRAINING

We show experiment results in episodic training setting in all scenarios with several algorithms, QMIX, DRIMA, COMA, and MADDPG as shown in Figure E.9 and Figure E.10 for 5 million steps.

E.2. Optimization for Exploration

We conducted several kinds of balancing exploration-exploitation trade-off in value-based and distribution based algorithm with ϵ-greedy decaying and taking various risk-level each to find optimal hyperparameters for algorithms which does not solve offensive scenarios.

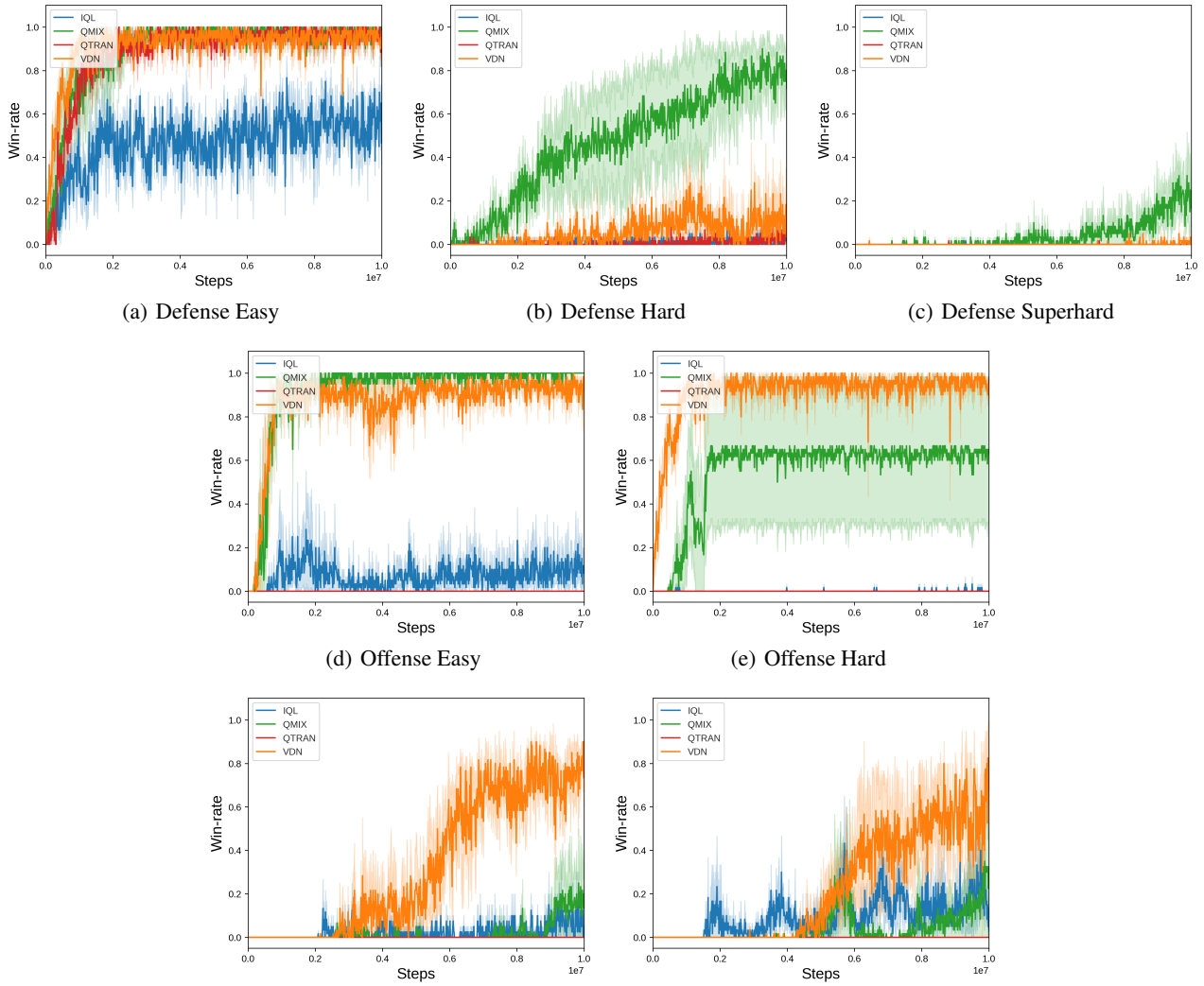
E.2.1. VALUE-BASED ALGORITHMS

We try to search optimal hyperparmater for exploration with ϵ-greedy in space $\{$10k, 50k, 100k, 500k, 5000k$\}$ for QMIX especially in hard defensive and offensive scenarios which requires countless exploration. In addition to the linear decaying method, we try exponential and piece-wise decaying method for 50k steps. We find that there's no specific tendency on adjusting ϵ-greedy, exponential, and piece-wise decaying steps for exploration in exploration-demanding scenarios. We conclude that additional factor for exploration in offensive scenario is needed for value-based algorithms to win the scenarios. Results are as shown as Figure E.11.

E.2.2. RISK-BASED ALGORITHMS

Quantile function Distributioanl deep reinforcement learning (DRL) got popular from (Bellemare et al., 2017) which suggested C51 algorithm. DRL makes output as a distribution of return given state and action. In DRL, they utilize Wasserstein metric for computing TD-error between distributional bellman updated distribution and current distribution of return. It is different from algorithms but most of them use quantile function to approximate distribution of return to compute the Wasserstein distance between current distribution and target distribution (which is updated by distributional bellman update). So we can imagine that we have a model that approximate a quantile function that domain is [0, 1] and range is $(-\infty, +\infty)$ which means return variable. Output of a model could approximate an inverse of cumulative density function as a quantile function.

Risk-sensitive criteria We can notice that using distribution of returns can lead to risk-sensitive reinforcement learning. Actually, we don't know what shape of function our trained model approximate. Then we can just make behavior criteria



(a) Defense Easy (b) Defense Hard (c) Defense Superhard

(d) Offense Easy (e) Offense Hard

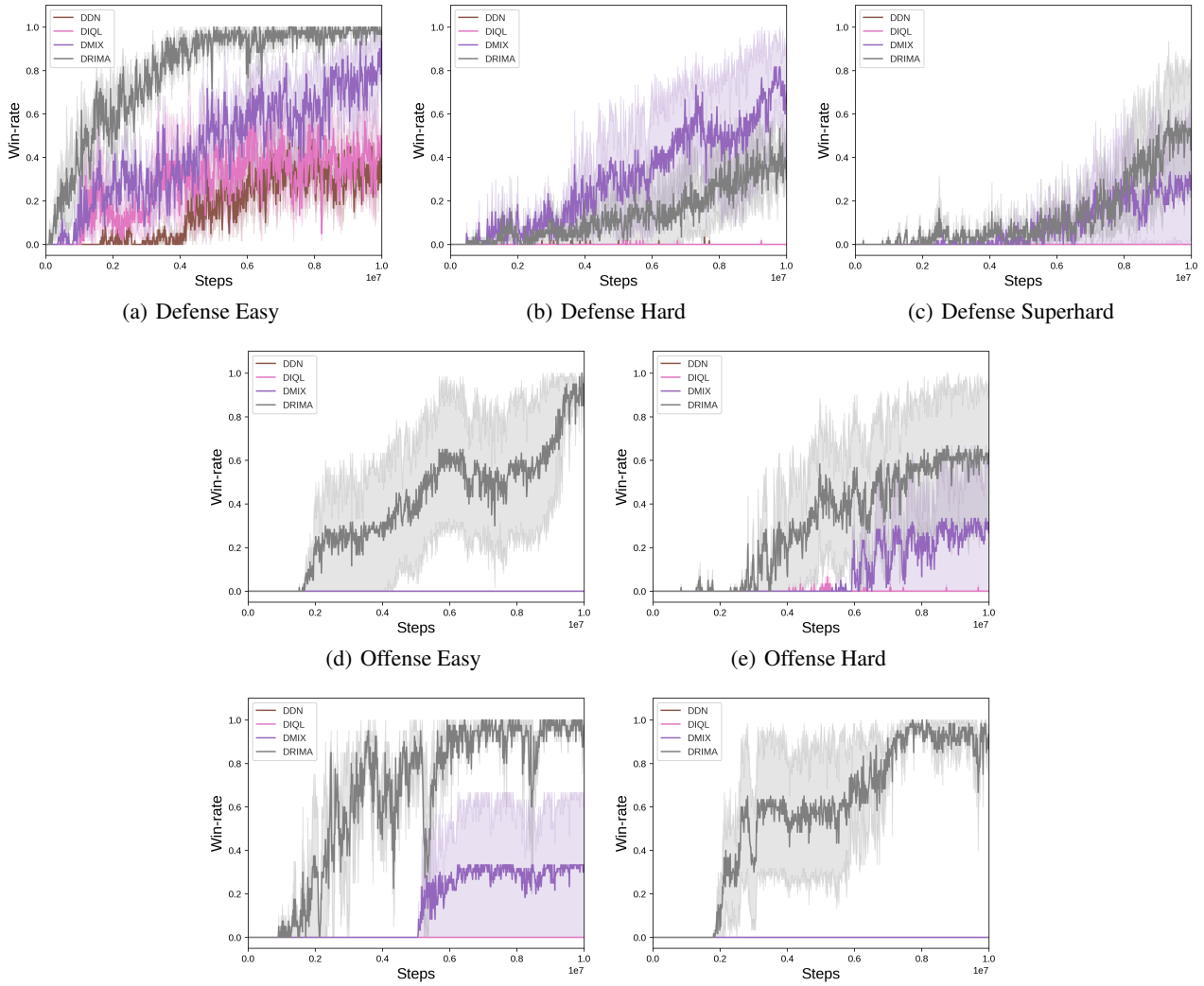
(f) Offense Superhard (g) Offense Ultrahard

Figure E.5: Value-based baselines experiments on every scenario with parallel training.

with a sampling method. Sampling τ in low interval like [0, 0.25] make the behavior criteria as a risk-averse by force no matter how the approximated quantile function looks like. Sampling τ in high interval like [0.75, 1] make criteria risk-seeking contrary to the sampling τ in low interval. Also we can think about risk-sensitive behavior with variance which is called uncertainty. Lower variance of an action's return distribution means lower uncertainty that means risk-averse action and higher variance of an action's return distribution means higher uncertainty which means risk-seeking action because it is very hard to expect what return we will get. With these properties of distributional RL, we can train a model with a risk-sensitive behavior criteria.

E.3. Risk-sensitive criteria experiments

Experiments with various criteia We conducted risk-sensitive criteria experiments in our setting, and we adapted DFAC variants for distributional RL named DDN and DMIX. In DRIMA algorithm which is also risk-based distributional RL, we just follow the default setting of the hyperparameter not conducted with further experiments. We divide the sampling quantile fractions into 4 portion ([0, 0.25], [0.25, 0.5], [0.5, 0.75], [0.75, 1.0]) from uniform distribution each $\tau \sim U([\cdot, \cdot])$. We name each portion as risk-averse, risk-neutral-averse, risk-neutral-seeking and risk-seeking sampling. The results are shown in Figure E.12. Looking the results, we notice that risk-neutral-averse criteria with DMIX work best in most of the



(a) Defense Easy (b) Defense Hard (c) Defense Superhard

(d) Offense Easy (e) Offense Hard

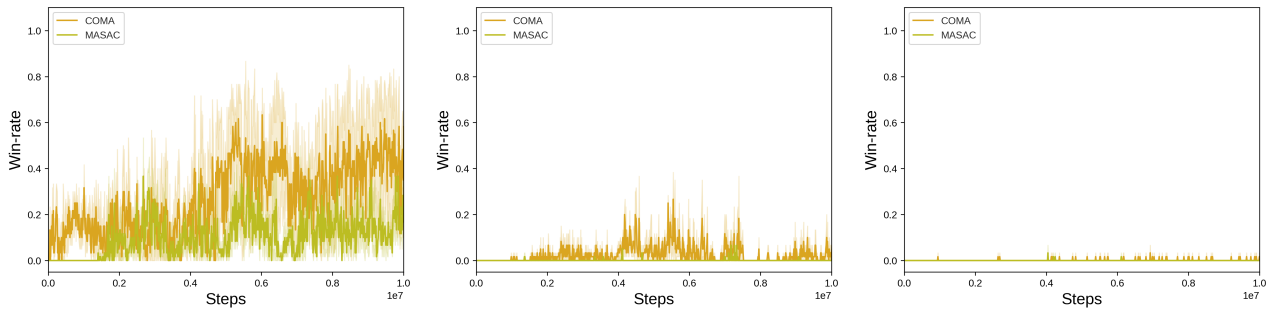
(f) Offense Superhard (g) Offense Ultrahard

Figure E.6: Distribution-based baselines experiments on every scenario with parallel training.

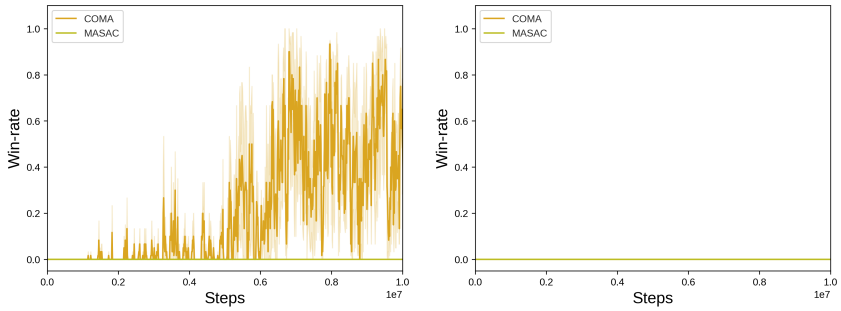
setting except some scenarios that no training has developed. Also, we can find that risk-seeking criteria in DMIX and DDN never worked in this setting. With these experiments, we knew that distributional RL is very fragile to the sampling τ or risk-sensitive criteria so as to make the needs for research about the risk-sensitive behavior in distributional reinforcement learning which is very critical in performance.

F. Reward Engineering

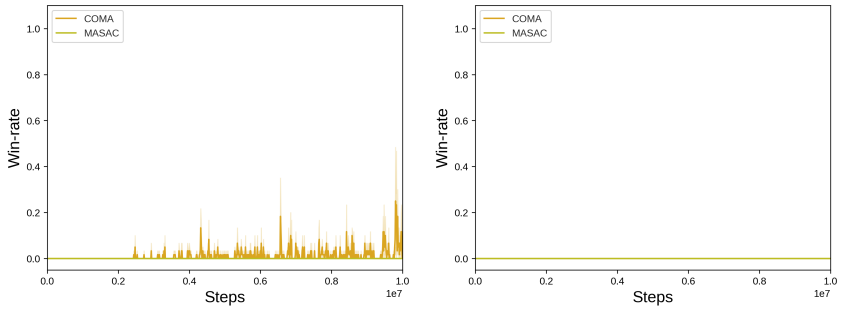
Sub-task oriented reward In offense scenarios, there is a clear sub-task that is finding enemies to fight. Since if agents didn't find any opponents, they can't get rewards from them. In offense hard scenario, QMIX does not show proper improvement in the winning rate with learning. It is because of the failure in searching. I which described in Figure 1(a), In Figure F.13(b), Although they go near enemies, they cannot climb the uphill. We want to solve this problem with simple reward engineering while expecting to give not only gain from fighting against opponents but also gain from distance from them. The alternative reward at time step t before normalize is Equation F.18. We normalize this reward with the maximum



(a) Defense Easy (b) Defense Hard (c) Defense Superhard



(d) Offense Easy (e) Offense Hard



(f) Offense Superhard (g) Offense Ultrahard

Figure E.7: Policy-based baselines experiments on every scenario with parallel training.

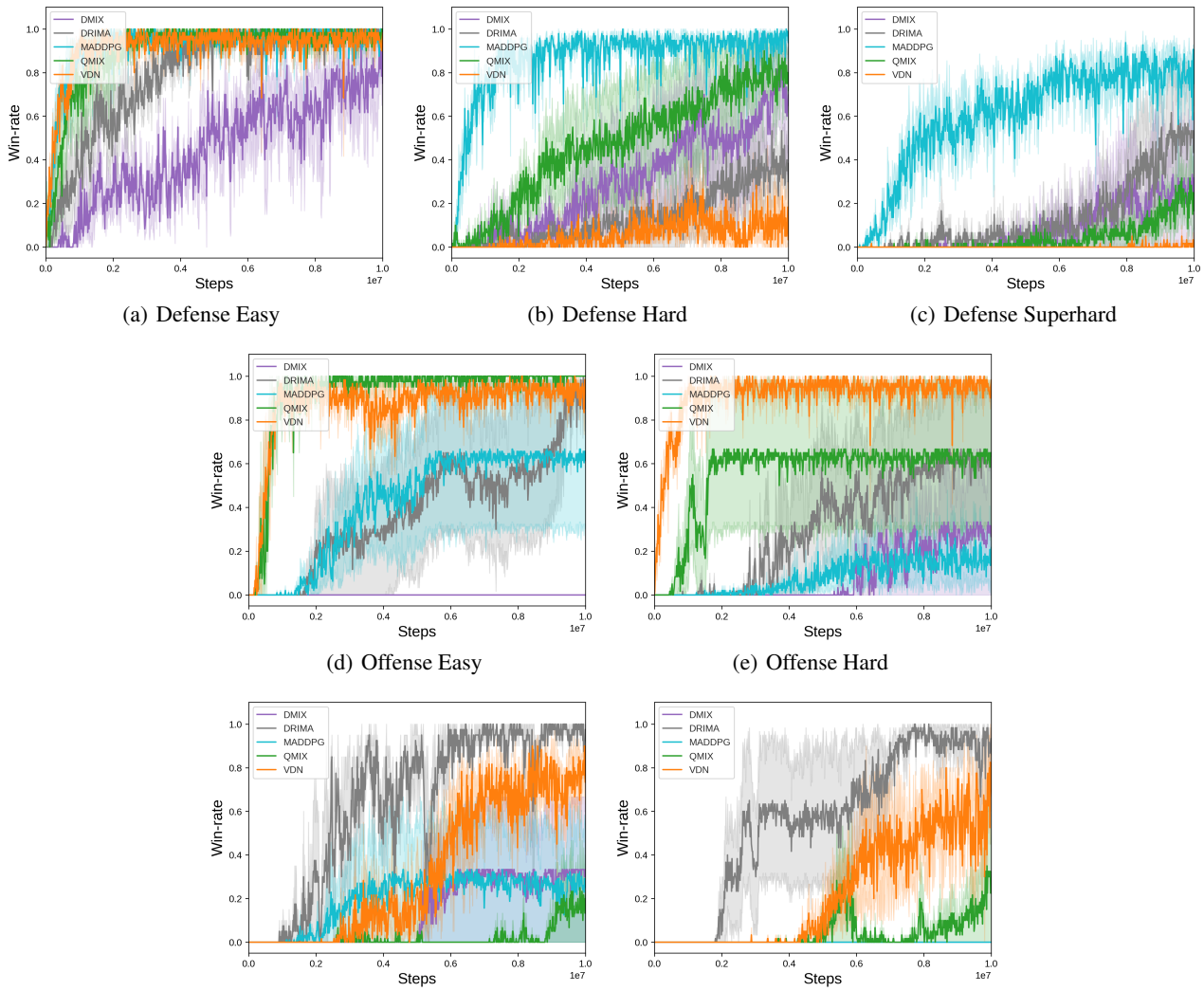
alternative reward of the map.

$$R^t_{alt} = (\sum_a \sum_e d(a^t_{pos}, e^t_{pos}) - \sum_a \sum_e d(a^{t-1}_{pos}, e^{t-1}_{pos})) \tag{F.18}$$

where each a and e is agents and enemies, d is euclidian distance and pos means position. We test the alternative reward function in two ways. Using the alternative function until certain time steps like a step function, mixing it with basic reward function until the end, 2:8.

Result Figure F.13(a) shows drastic improved results. After applying alternative reward function for sub-task, QMIX agents directly head for the location where enemies at, as shown in Figure F.13 (c). This change makes QMIX appears to have completely solved the problem of sub-task. Even though agents don't fight a lot to approach enemies, they naturally fight with enemies to get closer to other enemies.

However, in offense ultra hard scenario, the simple distance reward function doesn't work as in Figure F.14. Although Figure F.14(a) show QMIX could learn with basic reward function in ultra hard scenario. Figure F.14(b) illustrate finding enemies with reward engineering is failed. It doesn't show any improvement, even it has worse performance.



(a) Defense Easy (b) Defense Hard (c) Defense Superhard

(d) Offense Easy (e) Offense Hard

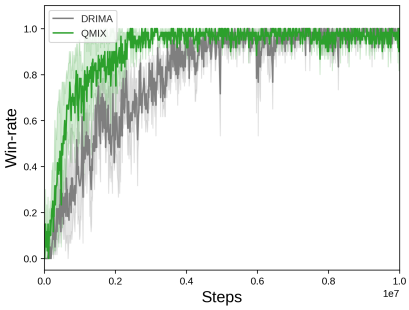
(f) Offense Superhard (g) Offense Ultrahard

Figure E.8: Well-performing baselines experiments on every scenario with parallel training.

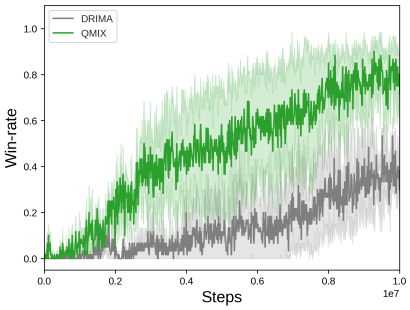
G. Case Study

In this section, we compare the learned policies with case studies of the results in 3.

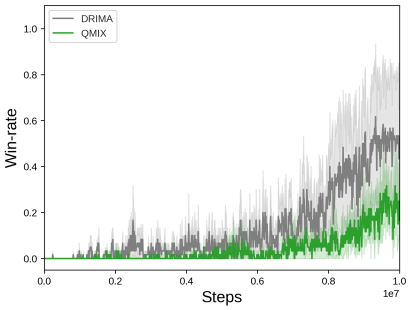
Defensive Scenario In the `super hard` in defense scenarios, enemies approach along with both left and right sides. Figure G.15 shows how algorithms are learned to utilize environmental benefits. In case of MADDPG that records the best performance in this scenario, move two tanks to the left side and the remaining agents to the right side. As in Figure G.15(a) and (b), this fully exploits environmental benefits by placing units on the hill at the left side and units behind tree at the right side. On the other hand, DRIMA decides to move one tank to the left side and all other agents to the right side. The reason the win-rate of DRIMA is less than MADDPG is that beating all enemies on the left side by only one tank is not easy. When you see Figure G.15(c) and (d), we identify that QMIX makes it worse. It move all agents forward the left side and then agents defeat some enemies and do not try to fight them in the other side.



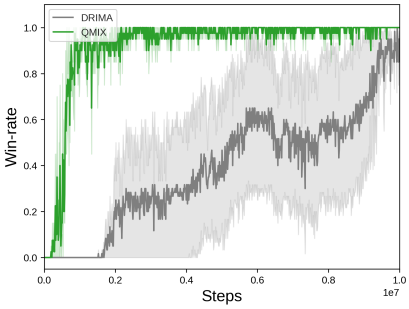
(a) Defense Easy



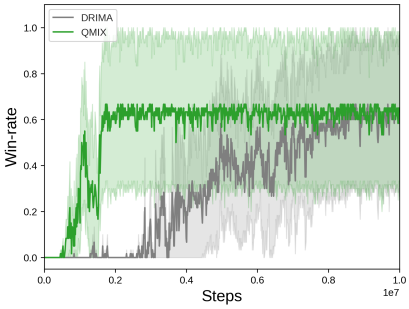
(b) Defense Hard



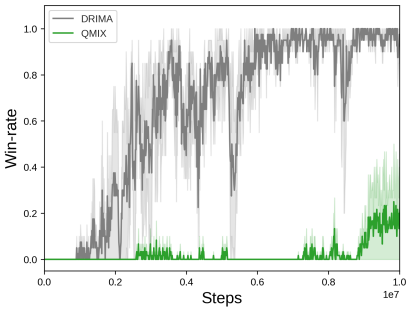
(c) Defense Superhard



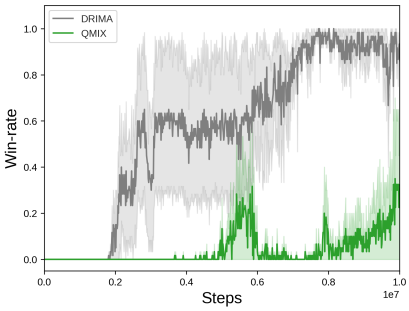
(d) Offense Easy



(e) Offense Hard

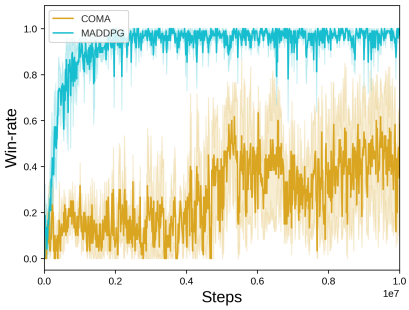


(f) Offense Superhard

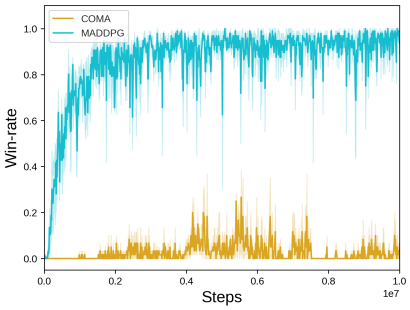


(g) Offense Ultrahard

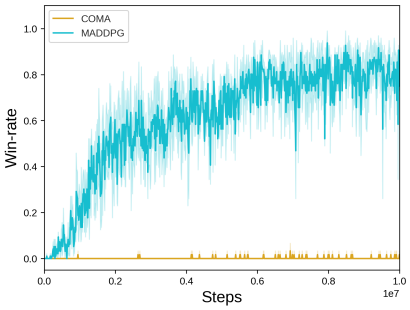
Figure E.9: Experiments results of value-based baseline algorithms on 7 scenarios with episodic training.



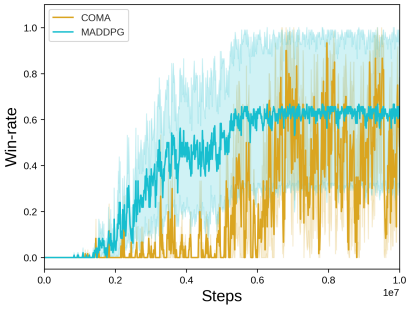
(a) Defense Easy



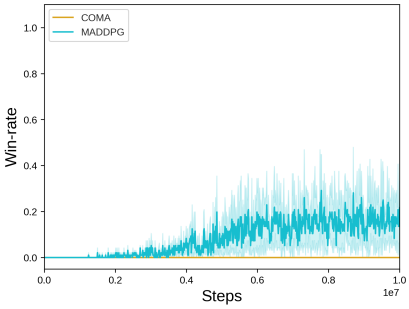
(b) Defense Hard



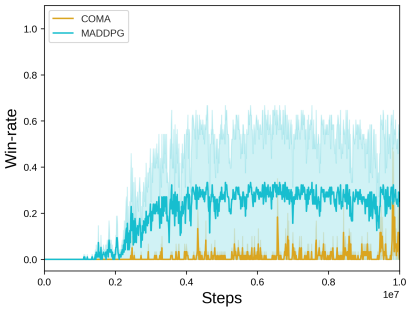
(c) Defense Superhard



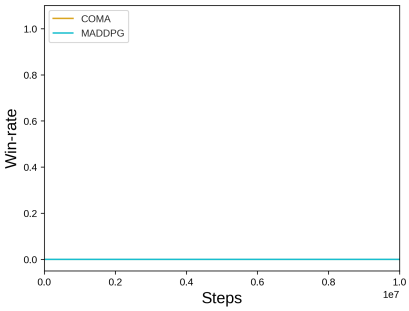
(d) Offense Easy



(e) Offense Hard

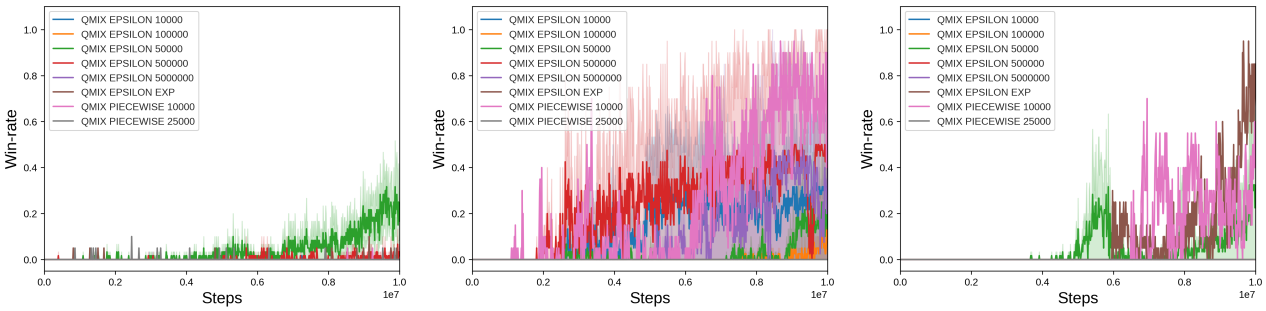


(f) Offense Superhard



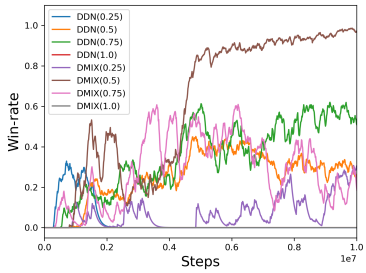
(g) Offense Ultrahard

Figure E.10: Experiments results of policy-based baseline algorithms on 7 scenarios with episodic training.

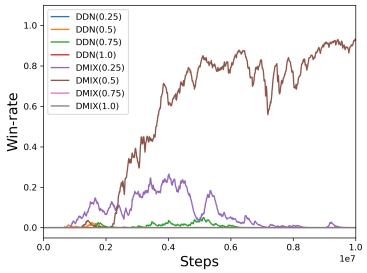


(a) Defense Superhard (b) Offense Superhard (c) Offense Ultrahard

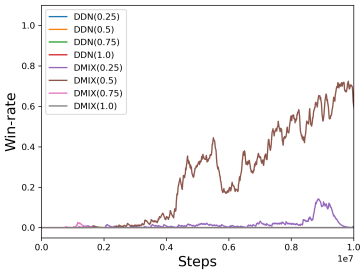
Figure E.11: Sweeping out ϵ-decaying steps with QMIX algorithm. The numbers next to the EPSILON letter means the decaying steps. In piece-wise setting, decaying line consist of combination of two linear line and set ϵ decaying point to 0.1. Arriving that point, again decay epsilon to 0.5. The number '10000 to 50000' in label means how to use piece-wise decaying strategy That is, for example, '10000 to 50000' means decay epsilon from 1.0 to 0.1 for 10000 steps and then decay from 0.1 to 0.05 for 40000 steps



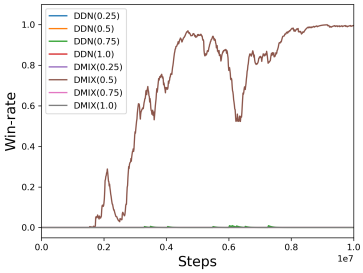
(a) Defense Easy



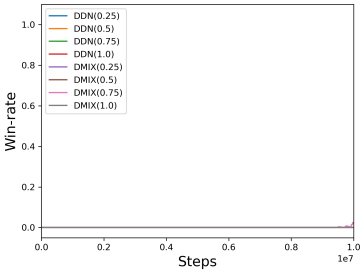
(b) Defense Hard



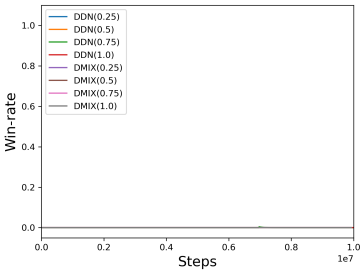
(c) Defense Superhard



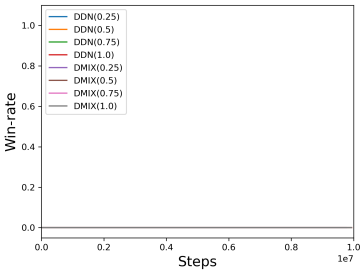
(d) Offense Easy



(e) Offense Hard

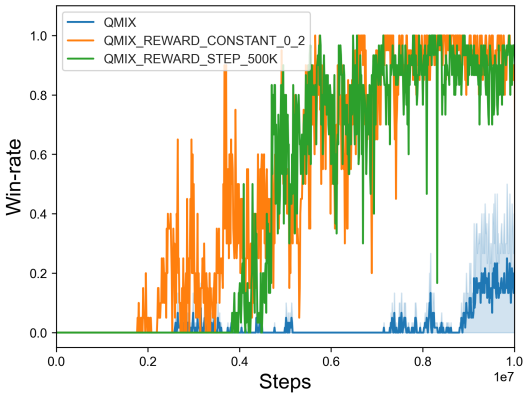


(f) Offense Superhard

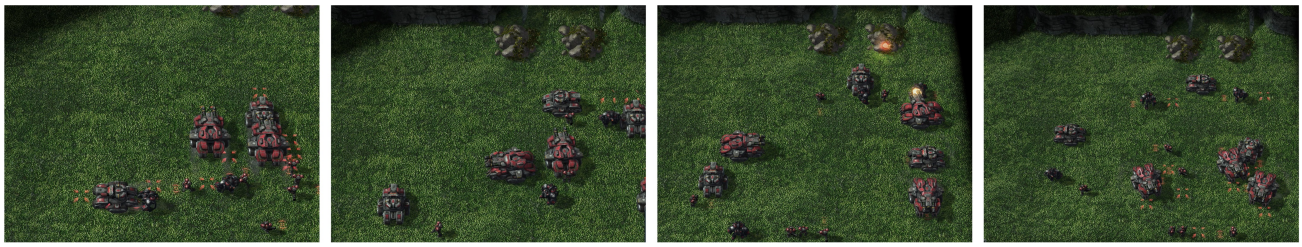


(g) Offense Ultrahard

Figure E.12: Risk-sensitive experiments with DFAC algorithm according to the sampling methods. DDN(0.25) and DDN(0.5) means sampling $\tau \sim U([0, 0.25])$, $U([0.25, 0.5])$ each using DDN algorithm.



(a) Comparing the alternative reward and basic reward



(b) QMIX with the basic reward at training step 10M

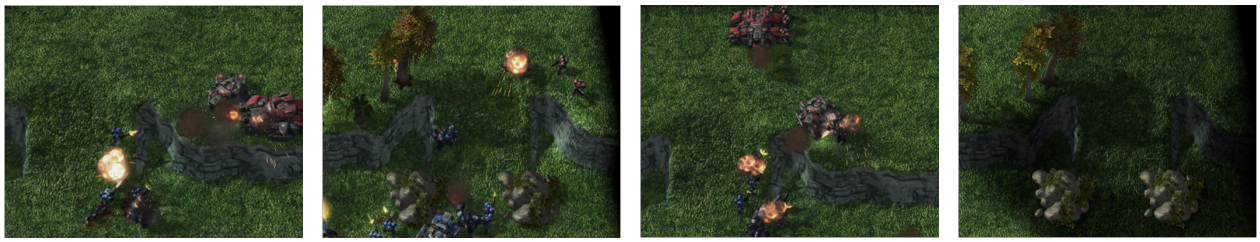
(c) QMIX with the alternative reward at training step 500k

Figure F.13: (a) Performance comparison with basic and alternative reward function. (b),(c) Time ordered scene in replays in offense superhard. The sequences are captured at the same time. (c) is trained with the alternative reward function until 300k timestep.

(a) QMIX with the basic reward at training step 10M

(b) QMIX with the alternative reward at training step 800k

Figure F.14: Time ordered scene in replays in offense ultrahard. The sequences are captured at the same time. (b) is trained with the alternative reward function until 800k timestep.



(a) Left-side battle by MADDPG (b) Right-side battle by MADDPG (c) Left-side battle by QMIX (d) Right-side battle by QMIX

Figure G.15: Defense super hard scenario replays with MADDPG, DRIMA, QMIX.