The StarCraft Multi-Agent Challenges$^+$: Learning of Multi-Stage Tasks and Environmental Factors without Precise Reward Functions

Mingyu Kim [* 1] **Jihwan Oh** [* 1] **Yongsik Lee** [1] **Joonkee Kim** [1] **Seonghwan Kim** [1] **Song Chong** [1] **Se-Young Yun** [1]

Abstract

In this paper, we propose a novel benchmark called the StarCraft Multi-Agent Challenges$^+$, where agents learn to perform multi-stage tasks and to use environmental factors without precise reward functions. The previous challenges (SMAC) recognized as a standard benchmark of Multi-Agent Reinforcement Learning are mainly concerned with ensuring that all agents cooperatively eliminate approaching adversaries only through fine manipulation with obvious reward functions. This challenge, on the other hand, is interested in the exploration capability of MARL algorithms to efficiently learn implicit multi-stage tasks and environmental factors as well as micro-control. This study covers both offensive and defensive scenarios. In the offensive scenarios, agents must learn to first find opponents and then eliminate them. The defensive scenarios require agents to use topographic features. For example, agents need to position themselves behind protective structures to make it harder for enemies to attack. 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. Additionally, we observe that an enhanced exploration approach has a positive effect on performance but is not able to completely solve all scenarios. This study proposes a new axis of future research.

1. Introduction

The StarCraft Multi-Agent Challenges (SMAC) is recognized as the standard benchmark simulator of Multi-Agent Reinforcement Learning (MARL) studies (Samvelyan et al., 2019). Tasks in the SMAC mainly require micro-managed

control of agents in defensive situations, where all opponents naturally approach the trained agents. This allows agents to obtain rewards directly. In these environments, the majority of algorithms concentrated on determining the relevance of each agent during training (Sunehag et al., 2017; Lowe et al., 2017; Rashid et al., 2018; Hu et al., 2021; Iqbal et al., 2021; Liu et al., 2021; Sun et al., 2021; Qiu et al., 2021). Some difficult scenarios, such as 2c_vs_64zg and corridor, on the other hand, require agents to indirectly learn environmental factors, such as exploiting different levels of terrains or discover multi-stage tasks like avoiding rushing enemies first and then eliminating individuals without a specific reward for them. Recently, efficient exploration approaches for MARL algorithms were reported to drastically increase performance in those tough scenarios (Sun et al., 2021; Son et al., 2022). However, those scenarios do not allow quantitative assessment of the algorithm's exploration capabilities, as they do not accurately reflect the difficulty of the task, which depends on the complexity of multi-stage tasks and the significance of environmental factors.

To address this issue, we propose a new class of the StarCraft Multi-Agent Challenges$^+$ (SMAC$^+$) that encompasses advanced and sophisticated multi-stage tasks, and involves environmental factors agents must learn to accomplish, as seen in Table 1. We present three defensive scenarios to encourage agents to employ topographical features such as positioning themselves behind structures to lower the probability of being attacked by enemies. In addition, offensive scenarios require agents to initially find the adversaries while also considering topographical obstacles and then rapidly defeating each of the adversarial troops. Like previous challenges, SMAC, in these situations, agents are still rewarded just for eliminating enemies, indicating that they indirectly learn how to do multi-stage tasks and use environmental factors. In the experiment results, we compare the performance of 11 MARL algorithms across all scenarios to establish a benchmark. We find that existing approaches perform well in similar settings to the previous challenge, but when environments need to complete sophisticated subtasks, most algorithms fail to learn adequately even when the training time is significantly extended. To summarize, we make the following contributions:

Table 1: List of environmental factors and multi-stage tasks for both SMAC and SMAC$^+$. Both SMAC and SMAC$^+$ employ the same final objective like eliminating all enemies.

	SMAC		SMAC$^+$	
	`2c_vs_64zg`	`corridor`	`Defense`	`Offense`
Environmental Factors	Different levels of the terrain	Limited sight range of enemies	Destroy obstacles hiding enemies Place in less damage zones	Approach enemies strategically Discover a detour* Destroy moving impediments*
Multi-Stage Tasks	-	Avoid enemies first, eliminate individually	-	Identify where enemies locate, then exterminate enemies

*: `Off_complicated` scenario

- We propose a novel MARL environment; SMAC$^+$. This intends to identify how agents explore to learn sequential completion of multi-stage tasks and environmental factors when reward functions are designed only towards the final objective.

- We present an extensive benchmark of MARL algorithms on SMAC$^+$. We find that recent MARL algorithms with enhanced exploration demonstrate stable performance in the proposed scenarios, but other baselines cannot be efficiently trained.

- We suggest the most challenging environments that demand simultaneously learning micro-control and multi-stage tasks. These scenarios are an open-ended problem for efficient exploration toward MARL domains because no algorithm attain satisfactory performance on the challenging scenario.

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

We propose a novel multi-agent environment referred to as StarCraft Multi-Agent Challenges$^+$ that features a quantitative evaluation of the exploration abilities of MARL algorithms. This challenge offers more advanced and sophisticated environmental factors such as destructible structures that can be used to conceal enemies and terrain features, such as a hill, that may be used to mitigate damages. Also, we newly introduce offensive scenarios that demand sequential completion of multi-stage tasks requiring finding adversaries initially and then eliminating them. Like in SMAC, both defensive and offensive scenarios in SMAC$^+$ employ the reward function proportional to the number of enemies removed. For unit combination, we select units to necessitate environmental factors and completion of multi-stage tasks cooperatively so that the trained agents can validate that they make effective use of these properties. In SMAC$^+$, agents must implicitly discover multi-stage tasks and factors by relying on their exploration strategy. Hence, it provides the evaluation of the exploration capability of MARL algorithms by comparing performance in similar but diverse scenarios. We describe the details of our environments in the rest of this section.

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

Unit	Role
Marine	Intensive firepower / Search for opponent positions
Marauder	Damage absorption from enemy / Enemy's movement restriction
Tank	Strong firepower against individuals / Limited range of fire
Siege Tank	Firepower against groups / Limited range of sight / Protection from close-range attack

2.1. General Description of SMAC$^+$: Terrain and Unit Combination

We design topographical features like trees, and stones, which block each unit's sight equally for both allies and opponents. Another topographic feature is the hill, which provides a stochastic environment for damage dealing, as seen in Figure 1c. 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. This is quite reasonable in the sense that someone in a topographically high position may easily hurt someone, but not vice versa. Hence, exploration of MARL algorithms must encourage agents to find these factors during the training phase, as agents are not explicitly rewarded for using these features. To determine the aforementioned factors, we devise a combination of units that can validate the cooperative decision-making of several agents. The role of each type of agent is presented in Table 2. Each unit in SMAC$^+$ has a unique set of attributes, such as shooting range, sight range, and firepower. Thus, it becomes important to determine how to combine and locate agents in battles considering various unit types. We explain more details in Appendix A.

2.2. Defensive Scenarios

In defensive scenarios, we place allies on the hill and adversaries on the plain. We emphasize the importance of agents defeating adversaries utilizing topographical factors. The defensive scenarios in SMAC$^+$ are almost identical to those in SMAC. However, our environment expands the exploration range of allies to scout for the direction of offense by allowing enemies to attack in several directions and adding

(a) General terrain   (b) Complex terrain   (c) Hill advantage

Figure 1: Summary of environmental factors in the SMAC$^+$. The defense troop is positioned on the hill in preparation for combat, while the offensive troop placed on the lower side goes forward to the defense troops. The yellow line describes an attack by troops on the hill, while the orange line indicates an attack from below the hill.

Table 3: Taxonomy of offensive and defensive scenarios. The term "Supply difference" refers to the gap in populations between opponents and allies in the StarCraft2. Enemies are scattered in all offensive scenarios such that they cannot fire fiercely, allowing agents to easily defeat them when they engage.

Defensive scenario			Offensive scenario	
Scenario	Supply difference	Opponents approach	Scenario	Distance from opponents
Def_infantry	-2	One-sided	Off_near	Near
Def_armored	-6	Two-sided	Off_distant	Distant
Def_outnumbered	-9	Two-sided	Off_complicated	Complicated

Table 4: The list of the most challenging offensive scenarios. New scenarios require to simultaneously learn micro-control and multi-stage tasks.

Scenario	Supply difference	Distance from opponents	Opponents formation
Off_hard	0	Complicated	Spread
Off_superhard	0	Complicated	Gather

topographical changes.

2.3. Offensive Scenarios

Offensive scenarios provide learning of multi-stage tasks without direct incentives in MARL challenges. We suggest that agents should accomplish goals incrementally, such as eliminating adversaries after locating them. To observe a clear multi-stage structure, we allocate thirteen supplies to the allies more than the enemies. Hence, as soon as enemies are located, the agents rapidly learn to destroy enemies. As detailed in Table 1, in SMAC$^+$, agents will not have a chance to get a reward if they do not encounter adversaries. This is because 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 main challenges necessitate not only micro-management, but also exploration to locate enemies. For instance, the

agents learn to separate the allied troops, locate the enemies, and effectively use armored troops like a long-ranged siege Tank. We measure the exploration strategy of effectively finding the enemy through this scenario. In this study, we examine the efficiency with which MARL algorithms explore to identify enemies by altering distance from them. In addition, to create more challenging scenarios, we show how enemy formation affects difficulty.

3. Experiments

In this section, we describe experimental results to answer three key questions: *1) Do the proposed scenarios provide a quantitative assessment of exploration capability by varying complexity of multi-stage tasks and the significance of environmental factors? 2) Do existing MARL algorithms efficiently utilize exploration to discover environmental factors and the completion of multi-stage tasks? 3) Can these algorithms reliably perform well on SMAC$^+$ despite repeated training?*

To demonstrate the need for assessment of exploration capabilities, we choose eleven algorithms of MARL algorithms classified into three categories; policy gradient algorithms, typical value-based algorithms, and distributional value-based algorithms. First, as an initial study of the MARL domain, policy gradient algorithms such as COMA (Foerster et al., 2018), MASAC (Haarnoja et al., 2018), MADDPG (Lowe et al., 2017) are considered. The typical value-based algorithm including IQL (Tan, 1993), VDN (Sunehag et al., 2017), QMIX (Rashid et al., 2018) and QTRAN (Son et al., 2019) are chosen as baselines. Last but not least, we choose DIQL, DMIX, DDN (Sun et al., 2021) and DRIMA (Son et al., 2022) as distributional value-based algorithms that recently reported high performance owing to the effective exploration of difficult scenarios in SMAC.

In order to look into exploration capability, all baselines are trained in this experiment until the total number of cumulative episode steps respectively reaches five million steps for a sequential episodic setting. We respectively train each baseline three times and report average win-rates as the evaluation metric by conducting 32 test-runs in the episodic setting. Initially, we demonstrate the SMAC$^+$ benchmark utilizing sequential episodic buffers. We report experimental results by choosing a representative algorithm from each category because of the massive training time; MADDPG (Lowe et al., 2017), COMA (Foerster et al., 2018), QMIX (Rashid et al., 2018) and DRIMA (Son et al., 2022). We also conducted on sequential episodic buffer setting and note that the tendency of experimental results remains unchanged as shown in Figure 2, allowing us to analyze the exploration capabilities of MARL algorithms based on results of the parallel episodic buffer. More training information on sequential episodic buffer, details about algorithms and exper-

Table 5: Average win-rate (%) performance of QMIX, DRIMA, COMA and MADDPG. All methods used sequential episodic buffers. Note that MADDPG is only compatible with the sequential episodic buffer.

	Trial	Defensive scenarios			Offensive scenarios		
		infantry	armored	outnumbered	near	distant	complicated
COMA(Foerster et al., 2018)	1	75.0	0.0	0.0	0.0	0.0	0.0
	2	28.1	0.0	0.0	0.0	0.0	0.0
	3	21.9	0.0[1)]	0.0	0.0	0.0	0.0[2)]
QMIX(Rashid et al., 2018)	1	100	100	3.1	0.0	0.0	100
	2	93.8	0.0	0.0	100.0	100.0	87.5
	3	96.9	0.0	0.0	90.6	93.8	0.0[3)]
MADDPG(Lowe et al., 2017)	1	100	96.9	81.3	0.0	90.6	0.0
	2	100	84.4	81.3	75.0	0.0	75.0
	3	100	90.6	71.9	100.0	0.0	0.0
DRIMA(Son et al., 2022)	1	100	100	100	93.8	100[4)]	96.9
	2	100	96.9	96.9	93.8	100	100
	3	100	100	100	100	100	96.9
The total number of win-rate $\geq 80\%$		9	7	5	6	6	5

1) Takes total cumulative 3.29 million episode steps during training
2) Takes total cumulative 4.21 million episode steps during training
3) Takes total cumulative 4.59 million episode steps during training
4) Takes total cumulative 2.53 million episode steps during training
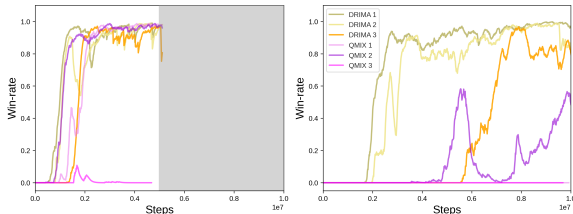
(a) Sequential episodic buffer   (b) Parallel episodic buffer

Figure 2: Average win-rates of QMIX and DRIMA according to the cumulative episodic steps during training. Two baselines are respectively trained three times. (a) learning curves of the sequential episodic setting for 5 million steps. (b) learning curves of parallel setting for 10 million steps. The horizontal axis means the total number of cumulative episode steps during the training, and the vertical axis is the win-rate.

Table 6: Average win-rate (%) performance on more challenging offensive scenarios. These scenarios require to address multi-stage tasks and micro-control at once.

	Trial	Episodic buffer		Parallel buffer	
		Off_hard	Off_superhard	Off_hard	Off_superhard
DRIMA(Son et al., 2022)	1	96.9	15.6	100	0.0
	2	93.8	3.1	80.0	0.0
	3	93.8	15.6	0.0	0.0

iment results are respectively documented at Appendix B, Appendix C, Appendix D and Appendix E.

3.1. Benchmark on Sequential Episodic Buffer

We first look into defensive scenario experiments on SMAC$^+$ to test whether MARL algorithms not only ad-

equately employ environmental factors but also learn micro-controls. As seen in Table 5, we find that supply difference and opponents' approach manipulate the difficulty, as the majority of baselines perform worse in response to those variants. In terms of algorithmic performance, we observe COMA and QMIX drastically degrade, but MADDPG gradually degrades. This fact reveals that MADDPG enables agents to effectively learn micro-control. However, among baselines, DRIMA achieves the highest score and retains performance even when the supply difference significantly increases. This is due to the fact that DRIMA efficiently explores micro-control but also environmental factors. This finding indicates that effective exploration uncovers intrinsic environmental factors. Regarding to offensive scenarios, we notice considerable performance differences of each baseline. Overall, even if an algorithm attains high scores at a trial, with exception of DRIMA, it is not guaranteed to train reliably in other trials. As mentioned, offensive scenarios do not require as much high micro-control as defensive scenarios, instead, it is important to locate enemies without direct incentives, such that when agents find enemies during training, the win-rate metric immediately goes to a high score. However, the finding enemies during training is decided by random actions drawn by ϵ-greedy or probabilistic policy, resulting in considerable variance in test outcome. In contrast, we see a perfect convergence of DRIMA in all offensive scenarios by employing its efficient exploration.

3.2. Evaluation on Challenging Scenarios

As previously stated, we identify that DRIMA reliably solves all offensive scenarios. To provide open-ended problems for the MARL domain, we suggest more challenging

scenarios as shown in Table 4. In these scenarios, the agents are required to simultaneously learn completion of multi-stage tasks and micro-control during training. DRIMA produces remarkable performance in past experiments, so that we provide the results acquired by DRIMA to verify that the proposed scenarios are sufficiently challenging. We use the same experimental condition as in the earlier experiments. As you can see Table 6, DRIMA still solves Off_hard, although its performance on Off_superhard is negligible. We argue that this scenario requires more sophisticated fine manipulation compared to other offensive scenarios. This is due to the fact that not only the strength of allies is identical to that of opponents, but also *Gathered* enables opponents to intensively strike allies at once. This indicates the necessity of more efficient exploration strategies for the completion of multi-stage tasks and micro-control.

4. Conclusion

We propose SMAC$^+$, a suite of environments to learn multi-stage tasks and environmental factors without specific rewards. We develop new MARL environments based on the previous challenge (Samvelyan et al., 2019). This point allows us to ensure that all baselines are completely compatible with our environments. Consequently, we evaluate a total eleven MARL algorithms on both defensive and offensive scenarios, and their experimental results show that an efficient exploration strategy is required to learn multi-stage tasks and environmental factors. We hope this work serves as a valuable benchmark to evaluate the exploration capabilities of MARL algorithms and give guidance for future research.

Acknowledgements

References

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.

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.

Hu, S., Zhu, F., Chang, X., and Liang, X. Updet: Universal multi-agent reinforcement learning via policy decoupling with transformers. *arXiv preprint arXiv:2101.08001*, 2021.

Iqbal, S., De Witt, C. A. S., Peng, B., Böhmer, W., Whiteson, S., and Sha, F. Randomized entity-wise factorization for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4596–4606. PMLR, 2021.

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.

Liu, B., Liu, Q., Stone, P., Garg, A., Zhu, Y., and Anandkumar, A. Coach-player multi-agent reinforcement learning for dynamic team composition. In *International Conference on Machine Learning*, pp. 6860–6870. PMLR, 2021.

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.

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.

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.

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.

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

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.

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.

Author Statement

The authors bear all responsibility in case of violation of rights, etc., and confirmation of the data license.

Accessibility

SMAC$^+$ environment is publicly available at the Github repository: `https://github.com/osilab-kaist/smac_plus` and we will maintain the released code for long-term accessibility.

License

The original SMAC environment and PyMARL code follow the MIT license and Apache 2.0 license respectively. The proposed SMAC$^+$ environment and the modified PyMARL code are also released under the MIT license and Apache 2.0 license each. The details of licenses can be found in our repository.

Reproducibility

The `README` file in the repository serves as a guide for installation and training. Several `yaml` files in `pymarl/src/config` directory contains the parameters we used for the paper. Further training details can be found in Appendix B.

A. Specification of StarCraft Multi-Agent Challenge$^+$

A.1. Environment

State and Observation features The features of original SMAC environment (Samvelyan et al., 2019) are described in from Table A.1 to Table A.2. ut means the number of different unit types in the given scenarios. Each agent has 6 moving actions and attacks 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 A.1: Global state information in SMAC

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

Table A.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}	The number of allies $\times(5 + ut)$
Enemy	{visibility, health, distance, relative x, relative y, unit type}	The number of enemies $\times(5 + ut)$
Total	{Move, Own, Ally, Enemy}	{4 + (1 + ut) + (The number of allies $\times(5 + ut)$) (The number of enemies $\times(5 + ut)$)}

The observation features in SMAC$^+$ are listed in Table A.3. The StarCraft2 in-game properties intrinsically provide 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, all agents can receive the last actions of all units within its sight range as a part of observations. These features are also provided in SMAC but not included in default setting. In addition, we incorporate the coordinate information perpendicular to the terrain surface and the

agent can have its own coordinate values. The unit type represents 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 A.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 within the agent's sight range. As the global state information, SMAC$^+$ provides two options. We might use same global 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 information in default.

Table A.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, unit type, last action }	The number of allies $\times$ $(14 + 7+$ The number of enemies and neutrals)
Enemy	{visibility, health, distance, relative x, relative y, relative z, unit type}	The number of enemies $\times$ 14
Neutral	{visibility, health, distance, relative x, relative y, relative z, unit type}	The number of neutrals $\times$ 14
Total	{Move, Own, Ally, Enemy, Neutral building }	$\{(4 + 8) + (4 + ut) +$ (The number of allies $\times$ $(14+7+$ The number of enemies and neutrals)) + $14 \cdot$ (The number of enemies + The number of neutrals)$\}$

Action space The total action set in SMAC$^+$ consists of basic action, number of enemies, and number of neutral buildings as shown in Table A.4. The basic action is almost identical to SMAC, but we additionally consider an units skill. The 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. Therefore, the action space of SMAC$^+$ is much larger than SMAC.

Table A.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 A.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. When seeing the Table Table A.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 A.5: Shooting, Sight range & Fire power in SMAC$^+$. Fire powers are categorized into two types. The left one is default fire power and the 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 among agents In this paper, we study a new type option of MARL referred as communicating with all agents. It means all agents share partial information of observations for improving cooperation. For example, all units can access opponent's position when one of allies observe even though opponents are located outside of the unit's sight range. This assumption still assures a decentralized evaluation because none of agents utilize absolute state information and only share a small fraction of observations throughout the test time. We argue that this assumption more accurately reflects the actual challenge because real-world communication technology is being rapidly developed. Specifically, the communication option allows agents to share a portion of their observations when they are within certain agents' sight range. When you see Figure A.1a, due to limited field of view, armored troops that are capable of long-distance assaults, such as tank and siege-armed tank, cannot engage the adversaries. However, the communication option enables armored troops attack distant enemies by utilizing ally sights as seen Figure A.1b. These attributes help each agent to move in a more cooperative manner.
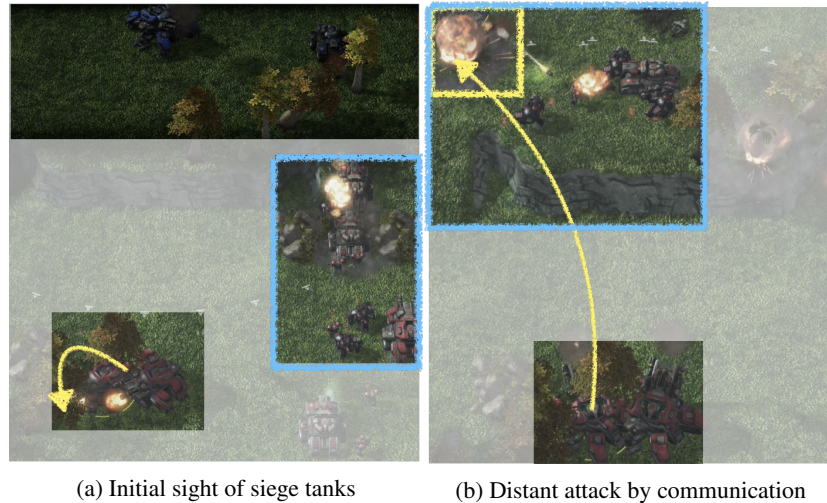

(a) Initial sight of siege tanks   (b) Distant attack by communication

Figure A.1: The explanation of environmental factors in the SMAC$^+$. The yellow line describes range attack by armored troops. The blue rectangle represents the alliance sights between the agent and the alliance.

We technically describe a communication option. 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. To allow communication among the agents in SMAC$^+$, we re-design 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 not 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 the 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 A.3 according to the updated visibility matrix, but the global state or the combination of the entire observations are not accessible during a test phase. obs_broadcast _info is another communication function that has no limit on communication range. Hence, when we set either one option to True, the communication option is active, but turning off all these functions makes no communication among agents.

Figure A.2a shows the training curve of VDN algorithm in the def_infantry with and without the communication option. 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 A.2b 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. In contrast, in Figure A.2c, the Marauder distracts the enemies while the Marines hide behind the neutral buildings which block the sight of the enemies. This implies that the agents learned to utilize the sight-blocking capabilities of the neutral building blocks with the communication option. The possible reason is that the agents can behave apart from one another due to the observation sharing. It would be an

(a) Training curve

(b) No communication

(c) Inter-agent communication

Figure A.2: Case study of communication option. We present the results of `def_infantry` by VDN and its associated test trial.

interesting challenge to design an algorithm to learn such polices without communication.

A.2. The Detail Information of Scenarios

Defensive scenarios By the extended exploration problem, allies should scout nearby the hill where allies are located to find out the enemies' offense direction as shown in Figure A.3b. If trained well, the agents won't try to get out of the hill which make enemies' offense as a stochastic damage. Plus, 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 A.3a. 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 A.3c 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.

(a) Removing buildings

(b) Scouting

(c) Terrain advantage

(d) Failed to scouting

Figure A.3: Sequential screenshots in Defensive Scenarios

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 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 damage from the enemies like shown as Figure A.4a. When finding the location and deployment of enemies, allies begin to attack the enemies with proper strategy if agents trained well as shown in Figure A.4b and Figure A.4c. 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 A.4d. 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 A.4: Sequential screenshots in Offensive Scenarios

Table A.6: SMAC$^+$ defensive 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
Infantry	1 Mar & 4 M	1 Mar & 6 M
Armored	1 SG Tank, 1 Tank, 1 Mar & 5 M	2 Tank, 2 Mar & 9 M
Outnumbered	1 SG Tank, 1 Tank, 1 Mar & 5 M	2 Tank, 3 Mar & 10 M

B. Training Details

Training Protocol We evaluate the following eleven MARL algorithms using the CTDE paradigm. For the value-based category, widely-adopted baselines employing value function factorization (IQL, VDN (Sunehag et al., 2017), QMIX (Rashid et al., 2018), QTRAN (Son et al., 2019)) and more recent state-of-the-art algorithms (DIQL, DMIX, DDN (Sun et al., 2021), DRIMA (Son et al., 2022)) are selected. COMA (Foerster et al., 2018), MASAC (Haarnoja et al., 2018), MADDPG (Lowe et al., 2017) are chosen for the policy-based category by general usage in MARL domain. Additionally, we report the four most effective baselines based on the results of parallel training in Table D.13. For the fair comparison with the MADDPG algorithm (Lowe et al., 2017) which is only compatible to the episodic setting, we respectively select the best performing algorithms in value-based, distribution-based and policy-based algorithms such as QMIX (Rashid et al.,

Table A.7: SMAC$^+$ offensive scenarios.

Offense			
Scenario	**Ally Units**	**Enemy Units**	**Distance & formation**
Near Distant Complicated	{3 SG Tank, 3 Tank, 3 Mar & 4 M}	{1 SG Tank, 2 Tank, 2 Mar & 4 M}	Near & Spread Distant & Spread Complicated & Spread
Hard Superhard	{1 SG Tank, 2 Tank, 2 Mar & 4 M}	{1 SG Tank, 2 Tank, 2 Mar & 4 M}	Complicated & Spread Complicated & Gathered

2018), DRIMA (Son et al., 2022) and COMA (Foerster et al., 2018) and train those algorithms using the episodic buffer setting. We run each algorithm for a total of ten million timesteps with 3 different random seeds for parallel training using 20 runners and 5 million timestep for episodic training. The trained model is tested at every ten thousand timesteps during 32 episodes for episodic training and 20 episodes for parallel training. As a evaluation metric, the percentage of winning episodes referred as to win-rate is employed. In these experiments, we set all rewards to positive values.

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 B.8.

Table B.8: 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

Computational Cost We use a machine containing 512GB memory, 4 GPUs(GeForce RTX 3080) with 10240MB memory each and AMD EPYC 7543 Processor with 32 cores. On the basis of an offensive scenario, SMAC$^+$ requires about 70-80GB of main memory and 3000-7000 MB of GPU capacity per scenario with parallel 20 simulators settings, and we are able to obtain results within 12-24 hours. Meanwhile, when we train for total 5 million cumulative episode steps via the sequential episodic buffer, it takes at least 24 hours and grows up to 48 hours. The detailed information of computation cost is listed in Table B.9.

Table B.9: Approximate training hours of SMAC$^+$. Results are evaluated with sequential episodic buffers during 5 million training timesteps and parallel episodic buffers during 10 million training timesteps.

	Defensive scenarios			Offensive scenarios				
	infantry	armored	outnumbered	near	distant	complicated	hard	superhard
Episode	23	35	36	36	38	46	44	47
Parallel	5	8	8	13	13	16	9	9

C. 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.

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.

Notation 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 $\mathbf{u} \in U$ where n is the number of agents. $P(s\prime \mid s, \mathbf{u}) : S \times U \times S \to [0, 1]$ is a transition probability function. All agents in Dec-POMDP receive shared reward, so the reward function is $r(s, \mathbf{u}) : S \times U \to \mathbb{R}$. Observation function is $O(s, a) : S \times A \to Z$ that determines agents' observation $z^a \in Z$. $\gamma \in [0, 1)$ is a discount factor for the reward.

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 C.5. 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.
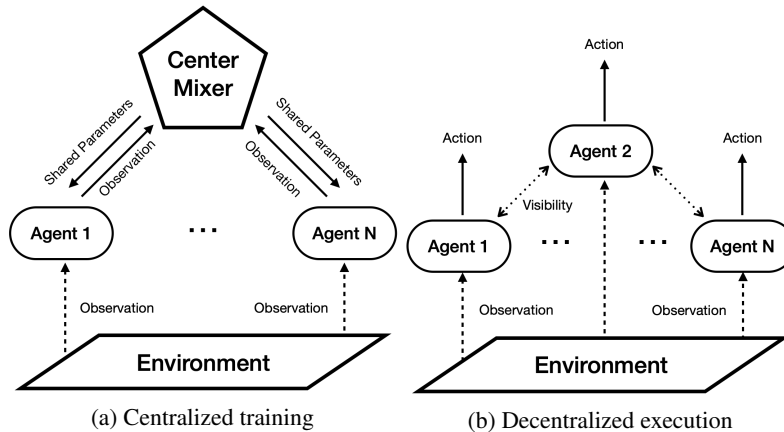

(a) Centralized training (b) Decentralized execution

Figure C.5: CTDE Framework

C.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:

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

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:

$$\arg\max_u Q_{total}(\mathbf{h}, \mathbf{u}) = \begin{pmatrix} \arg\max_{u_1} Q_1(h_1, u_1) \\ \vdots \\ \arg\max_{u_N} Q_N(h_N, u_N) \end{pmatrix} \tag{C.2}$$

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:

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

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:

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

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 C.5, ψ means a linear combination function by a mixing network which can be denoted as follows:.

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

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:

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

where,

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

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.

C.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(Dabney et al., 2018a) was used for distributional output sampling quantile fractions from $U[0, 1]$ 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:

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

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

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

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:

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

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.

C.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):

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

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 C.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:

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

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:

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

which estimate credit $r(s, (\mathbf{u}^{-a}, c^a))$ where c means default action of agents that is hard to estimate. This is why COMA uses Equation C.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:

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

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

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

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(Pu et al., 2021) 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 C.12 adapting deterministic policy $a = \mu(s)$ as follows:

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

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

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

where $y = r_i + \gamma Q_i^{\mu'}(\mathbf{h'}, u_1', \ldots, u_N')|_{u_j' = \mu_j'(h_j)}$. 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.

D. Completed Experimental Results of All Possible Cases

We present a part of the experimental results in the manuscript. Here, we show experiment results in all possible cases and report the win-rate performances. As mentioned, due to the constraint on MADDPG that is not compatible with training in the parallel method, we respectively show the experimental results based on both sequential and parallel episodic buffers. Prior to the explanation of all results, we note the update interval difference between the sequential episodic buffer and the parallel episodic buffer as shown in Table D.10.

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

| | Model update interval | | The number of model update | |
	Target network	Behavior network	Target network	Behavior network
Episode	200 episodes	1 episode	420	80000
Parallel	200 episodes	20 episodes	420	4240

We pick numerous benchmark algorithms based on the selection of traditional value-based and policy-based algorithms as well as contemporary algorithms exhibiting state-of-the-art performance in the MARL domain. Consequently, we chose IQN, VDN, QMIX, and QTRAN for value-based algorithms, COMA, MASAC, and MADDPG for policy-based algorithms, and DMIX, DDN, DIQL, and DRIMA for more modern algorithms. In this part, we discuss the performance of each algorithm trained on the parallel episodic buffer and sequential episodic buffer. As demonstrated in Table D.12, and Figure D.6 - D.16, for 5 million training time steps, we provide experiment results in an sequential episodic buffer for most of the SMAC$^+$ scenarios using the all algorithms. In episodic buffer settings, the performance in defense scenarios tends to decline as the supply differential increases, while the performance in offensive scenarios is marginally getting worse due to the complexity of the path from the starting place of allied troops to enemy units. Whereas many of the scenarios are quite solved by the algorithms, all the algorithms struggle to solve offense hard and superhard scenarios. Only QMIX and DRIMA can solve the offense hard scenario, as shown in Figure D.7g and Figure D.9g whereas none of the algorithms can solve offense superhard scenario. We report only defensive scenarios for some algorithms for the time limitation.

As seen in Table D.13, and Figure D.17 - D.26, for 10 million steps, we present the experimental outcomes of a parallel training setup with 20 parallel runners for each scenario and algorithms except MADDPG for not compatible with parallel episodic buffer setting. This setting contains 20 times fewer updates than episodic training, which may result in a performance decrease. With a parallel buffer, the overall tendency of performance is the same as that of episodic buffer setting, which shows that the performance in defense scenarios tends to decline as the supply differential increases. But in offensive scenarios, the pattern that the performance decreases as the complexity of the scenario increases is strongly shown than in sequential episodic settings. But likewise sequential episodic setting, none of the algorithms can solve the offense hard and superhard scenarios, which are also very challenging in a parallel setting. The only algorithm which solved the offense hard scenario are DRIMA, as shown in Figure D.26g, which uses risk-based exploration.

D.1. Benchmark on Parallel Episodic Buffer

While the majority of studies have reported performance using the sequential episodic buffer, as we mentioned above, the parallel episodic buffer would be more practical due to reduced training time. For this reason, we provide a comprehensive benchmark of MARL algorithms in this setting. Since the previous challenge (Samvelyan et al., 2019) already provided technical implementation of the parallel setup, we merely determine the parallelism level. With consideration of computation resources, we intend to run twenty simulators simultaneously to gather episodes. In these experiments, we evaluate 10 MARL algorithms on SMAC$^+$. Overall, we observe that the tendency of experimental results is identical to those of the results from the episodic buffer as seen in Table D.13. In addition, this result supports that both defensive and offensive scenarios can also be adjusted by the complexity of multi-tasks and environmental factors. Instead, as stated, due to the reduced update frequency, we observe that the performance of all baselines is marginally decreased and learning instability is increased. For example, in relatively simple scenarios like `off_near` and `off_distant`, typical value-based algorithms such as VDN and QMIX outperform other algorithms, whereas as complexity increases, DRIMA with an enhanced exploration capability by risk-based information attains the highest scores only in `off_complicated` and `def_outnumbered`. When we look closely at distributional value-based algorithms, we find that enhanced exploration by distributional value functions positively affects the performance on SMAC$^+$, meanwhile the DMIX shows unstable outcomes despite that it occasionally gains the best scores in complex scenarios. Contrary, we see that DRIMA captures robust scores compared to

Table D.11: Average win-rate (%) performance of QMIX, DRIMA, COMA and MADDPG. All methods used sequential episodic buffers. Note that MADDPG is only compatible with the sequenced experience buffer.

	Trial	Defensive scenarios			Offensive scenarios				
		infantry	armored	outnumbered	near	distant	complicated	hard	superhard
COMA(Foerster et al., 2018)	1	75.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	2	28.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	3	21.9	0.0[1]	0.0	0.0	0.0	0.0[2]	0.0	0.0
QMIX(Rashid et al., 2018)	1	100	100	3.1	0.0	0.0	100	96.9	0.0
	2	93.8	0.0	0.0	100.0	100.0	87.5	0.0	0.0
	3	96.9	0.0	0.0	90.6	93.8	0.0[3]	96.9	0.0
MADDPG(Lowe et al., 2017)	1	100	96.9	81.3	0.0	90.6	0.0	0.0	0.0
	2	100	84.4	81.3	75.0	0.0	75.0	0.0	0.0
	3	100	90.6	71.9	100.0	0.0	0.0	0.0	0.0
DRIMA(Son et al., 2022)	1	100	100	100	93.8	100[4]	96.9	96.9	15.6
	2	100	96.9	96.9	93.8	100	100	93.8	3.1
	3	100	100	100	100	100	96.9	93.8	15.6
The total number of win-rate $\geq 80\%$		9	7	5	6	6	5	5	0

1) Takes total cumulative 3.29 million episode steps during training
2) Takes total cumulative 4.21 million episode steps during training
3) Takes total cumulative 4.59 million episode steps during training
4) Takes total cumulative 2.53 million episode steps during training

Table D.12: Average win-rate (%) performance of additional seven algorithms with sequential episodic buffers in defensive scenarios.

	Trial	Defensive scenarios		
		infantry	armored	outnumbered
MASAC(Pu et al., 2021)	1	50.0	0.0	0.0
	2	37.5	0.0	0.0
	3	0.0	0.0	0.0
IQL(Tan, 1993)	1	96.9	9.4	0.0
	2	93.8	90.6	0.0
	3	84.4	6.3	0.0
VDN(Sunehag et al., 2017)	1	96.9	84.4	15.6
	2	93.8	96.9	9.4
	3	100	100	37.5
QTRAN(Son et al., 2019)	1	100	25.0	81.3
	2	100	96.9	65.6
	3	100	93.8	93.8
DDN(Sun et al., 2021)	1	81.3	96.9	0.0
	2	96.9	71.9	68.8
	3	90.6	71.9	0.0
DIQL(Sun et al., 2021)	1	93.8	68.8	78.1
	2	93.8	25.0	0.0
	3	93.8	53.1	0.0
DMIX(Sun et al., 2021)	1	100	81.3	0.0
	2	100	93.8	0.0
	3	96.9	46.9	53.1
The total number of win-rate $\geq 80\%$		18	9	2

other distributional algorithms.

In another aspect, we find the moment at which win-rate begins to rise is intriguing. As you can see Figure 2, the learning curves of QMIX obviously depicts distinct tendency according to the buffer setting, on the other hand, the learning curves of

Table D.13: Average win-rate (%) performance of 10 algorithms in both defensive and offensive scenarios using parallel episodic buffers with 20 parallel simulators.

Category	Algorithm	Trial	Defensive scenarios			Offensive scenarios				
			infantry	armored	outnumbered	near	distant	complicated	hard	superhard
Policy gradient	MASAC(Pu et al., 2021)	1	40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		2	25.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	COMA(Foerster et al., 2018)	1	85.0	0.0	0.0	20.0	0.0	0.0	0.0	0.0
		2	50.0	0.0	0.0	80.0	0.0	0.0	0.0	0.0
		3	5.0	0.0	0.0	20.0	10.0	0.0	0.0	0.0
Value based	IQL(Tan, 1993)	1	20.0	5.0	0.0	0.0	25.0	10.0	0.0	0.0
		2	40.0	0.0	0.0	5.0	0.0	35.0	0.0	0.0
		3	45.0	0.0	0.0	10.0	0.0[1]	40.0	0.0	0.0
	QTRAN(Son et al., 2019)	1	100	5.0	0.0	0.0	0.0	0.0	0.0	0.0
		2	80.0	25.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	100	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	QMIX(Rashid et al., 2018)	1	100.0	75.0	30.0	95.0	20.0	0.0	0.0	0.0
		2	95.0	100	0.0	95.0[2]	0.0	25.0	0.0	0.0
		3	95.0	75.0	65.0	0.0	0.0	0.0	0.0	0.0
	VDN(Sunehag et al., 2017)	1	100	0.0	0.0	100	90.0	85.0	15.0	0.0
		2	95.0	5.0	20.0	90.0	70.0	55.0	50.0	0.0
		3	95.0	5.0	0.0	85.0	85.0	70.0[3]	10.0	0.0
Distributional value based	DDN(Sun et al., 2021)	1	20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		2	30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	10.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DIQL(Sun et al., 2021)	1	70.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		2	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		3	45.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DMIX(Sun et al., 2021)	1	95.0	100	0.0	0.0	100	0.0	0.0	0.0
		2	90.0	55.0	5.0	0.0	0.0	0.0	0.0	0.0
		3	85.0	90.0	90.0	0.0	0.0	0.0[4]	0.0	0.0
	DRIMA(Son et al., 2022)	1	100	70.0	0.0	95.0	95.0	85.0	100	0.0
		2	100	60.0	70.0	90.0	100	100	80.0	0.0
		3	95.0	50.0	80.0	95.0	90.0	100	0.0	0.0
The total number of win-rate $\geq 80\%$			16	3	2	9	6	4	2	0

1) Takes total cumulative 2.41 million episode steps during training
2) Takes total cumulative 7.85 million episode steps during training
3) Takes total cumulative 8.15 million episode steps during training
4) Takes total cumulative 8.10 million episode steps during training

DRIMA are comparable in both settings if convergence occurs. Therefore, we claim that learning multi-stage tasks and environmental factors without direct incentives are influenced by the exploration capability of MARL algorithms.
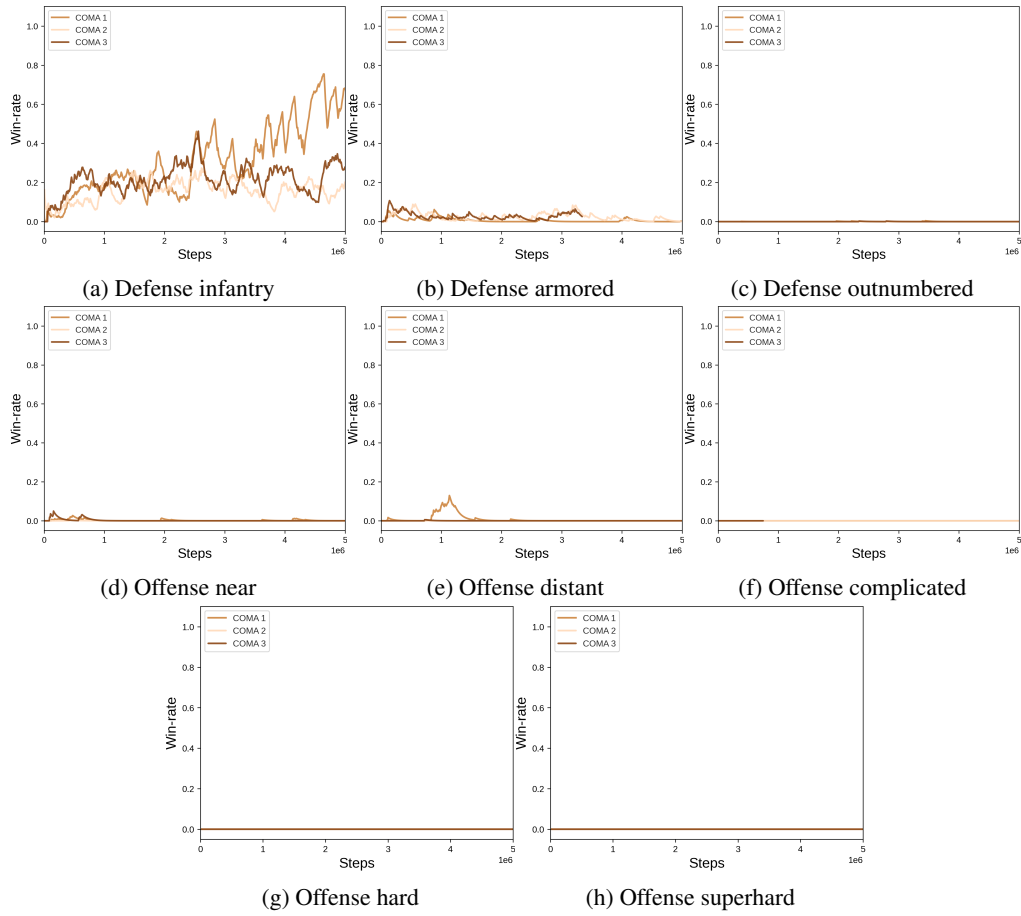
(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard

Figure D.6: COMA trained on the sequential episodic buffer

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

(d) Offense near   (e) Offense distant   (f) Offense complicated

(g) Offense hard   (h) Offense superhard
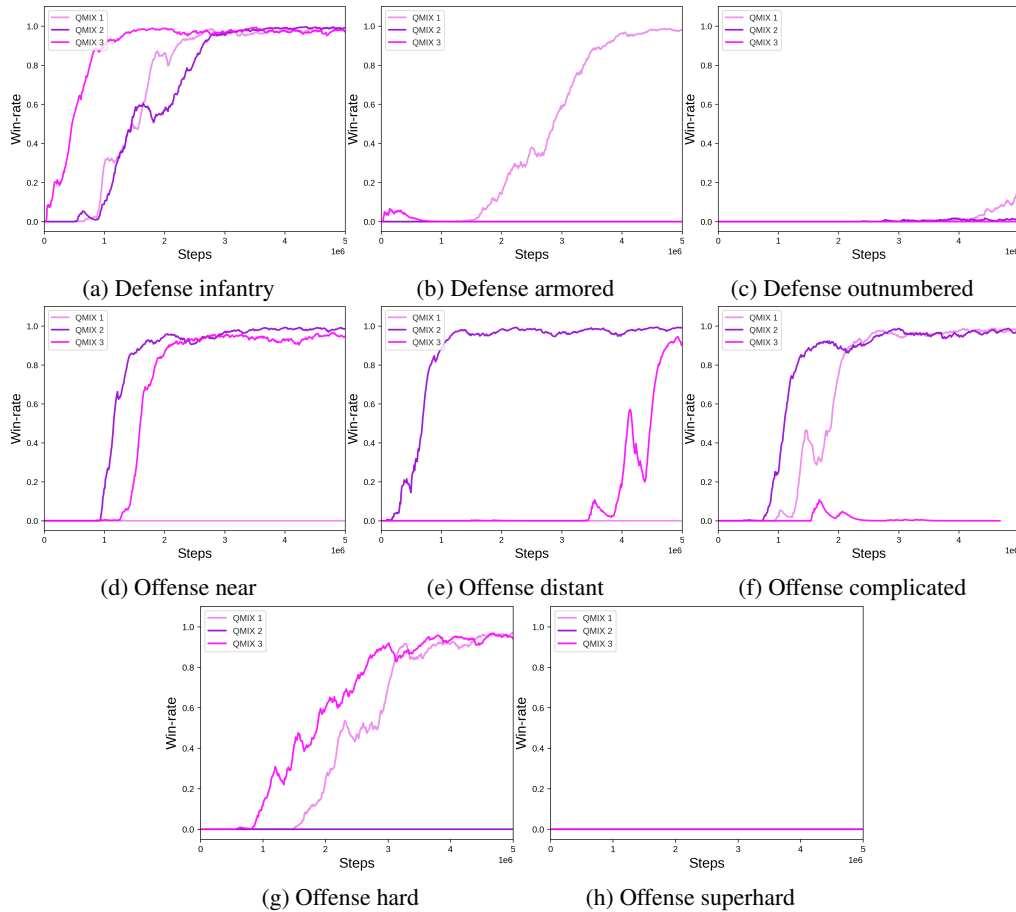
Figure D.7: QMIX trained on the sequential episodic buffer

E. Ablation Study

We additionally investigate the reason why some algorithms perform worse in offensive scenarios than the defensive scenarios. We point out that the exploration is a main issue to increase performance in offensive scenarios. For this reason, we try to find the optimal hyper-parameter setting with respect to exploration perspective by sweeping out ϵ-greedy steps and adjusting risk-sensitiveness.

E.1. Optimization for Exploration

To determine the optimal hyperparameters for algorithms that do not solve scenarios requiring exploration, we conducted multiple types of exploration-exploitation trade-off balancing in value-based, distribution-based, and policy-based algorithms with ϵ-greedy decaying, various risk levels, and entropy term control. We choose QMIX, DFAC (DMIX, DDN), and MASAC for each category that can be trained in parallel.

E.1.1. VALUE-BASED ALGORITHMS

We explore for the optimal hyperparamater for exploration with ϵ-greedy in space {10k, 50k, 100k, 500k, 5000k} for QMIX, particularly in difficult defensive and offensive scenarios that demand extensive explorations. In addition to the linear decaying approach, we utilize the exponential and piece-wise decaying methods for a total of 50k steps. We discover that there is no discernible trend in adjusting ϵ-greedy, exponential, and piece-wise decaying steps for exploration in scenarios requiring exploration. We find that more exploratory factors are required for value-based algorithms to win offensive scenarios. The results are depicted in Figure Figure E.27.

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

(d) Offense near   (e) Offense distant   (f) Offense complicated
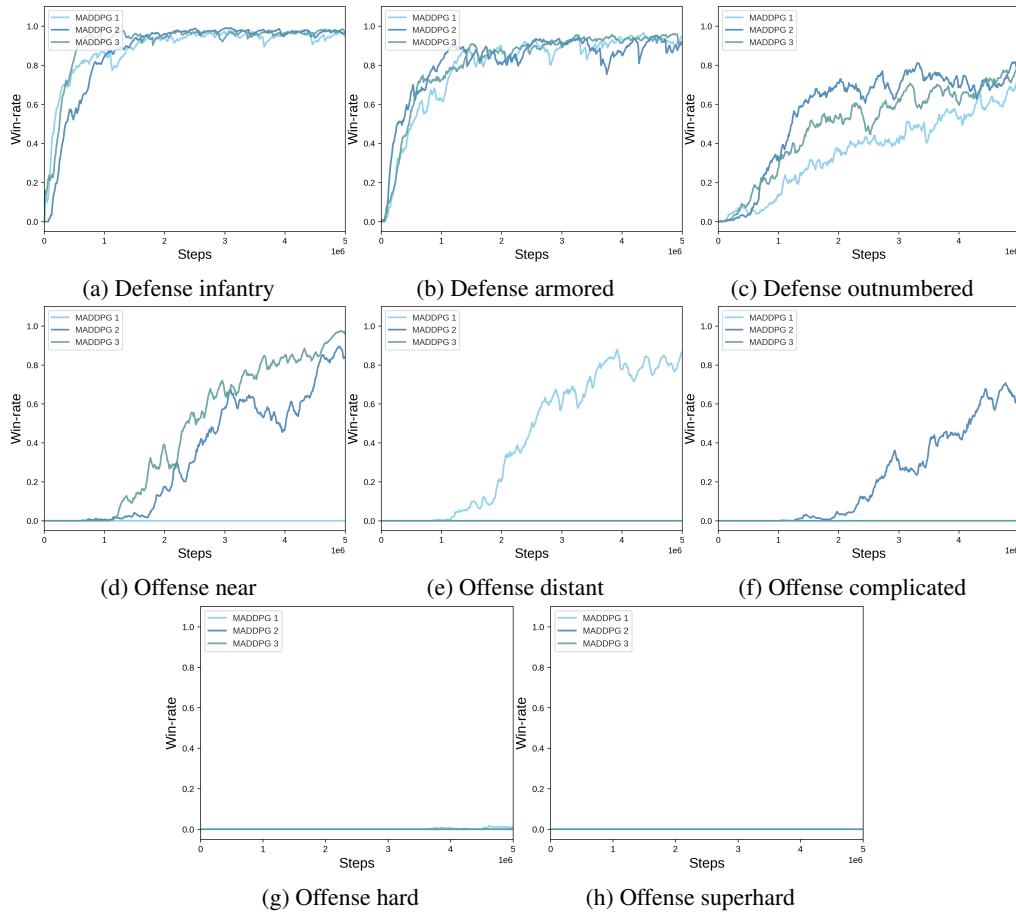
(g) Offense hard   (h) Offense superhard

Figure D.8: MADDPG trained on the sequential episodic buffer

E.1.2. DISTRIBUTION-BASED ALGORITHMS

In this section, we discuss the main concept of Distribution-based algorithms, followed by the outcomes of controlling risk-sensitivity.

Quantile function Distributioanl deep reinforcement learning gained popularity when (Bellemare et al., 2017) proposed the C51 method, which generates output as a distribution of return given state and action. In this domain, algorithms compute TD-error between distributional bellman's updated distribution and the current distribution of return using the Wasserstein metric. It differs among methods, but the majority of them estimate the return distribution in order to compute the Wasserstein distance between the present distribution and the desired distribution (which is updated by distributional bellman update). Consequently, we may pretend that we have a model that approximates a quantile function with domain $[0, 1]$ and return value range $(-\inf, +\inf)$. A model's output may approximate the inverse of the cumulative density function.

Risk-sensitive criteria We can observe that utilizing return distributions can result in risk-sensitive reinforcement learning. When a model approximates the quantile function, sampling τ uniformly from $\mathcal{U}[0, 0.25]$ yields relatively low output values. This is known as risk-averse behavior since we anticipate low rewards. If, on the other hand, we randomly pick τ from $\mathcal{U}[0.75, 1]$, we will obtain relatively high values, which may be viewed as an optimistic behavior. This is referred to as risk-seeking behavior. If we randomly choose τ from $\mathcal{U}[0, 1]$, we get a risk-neutral distribution that is not skewed toward risk-averse or risk-seeking criteria. Also, we may consider risk-sensitive behavior from the perspective of variance, also known as uncertainty. Lower variance of an action's return distribution indicates lower uncertainty, which indicates risk-averse action, whereas higher variance of an action's return distribution indicates greater uncertainty, which indicates risk-seeking action, because it is extremely difficult to predict what return we will receive. With these distributional RL
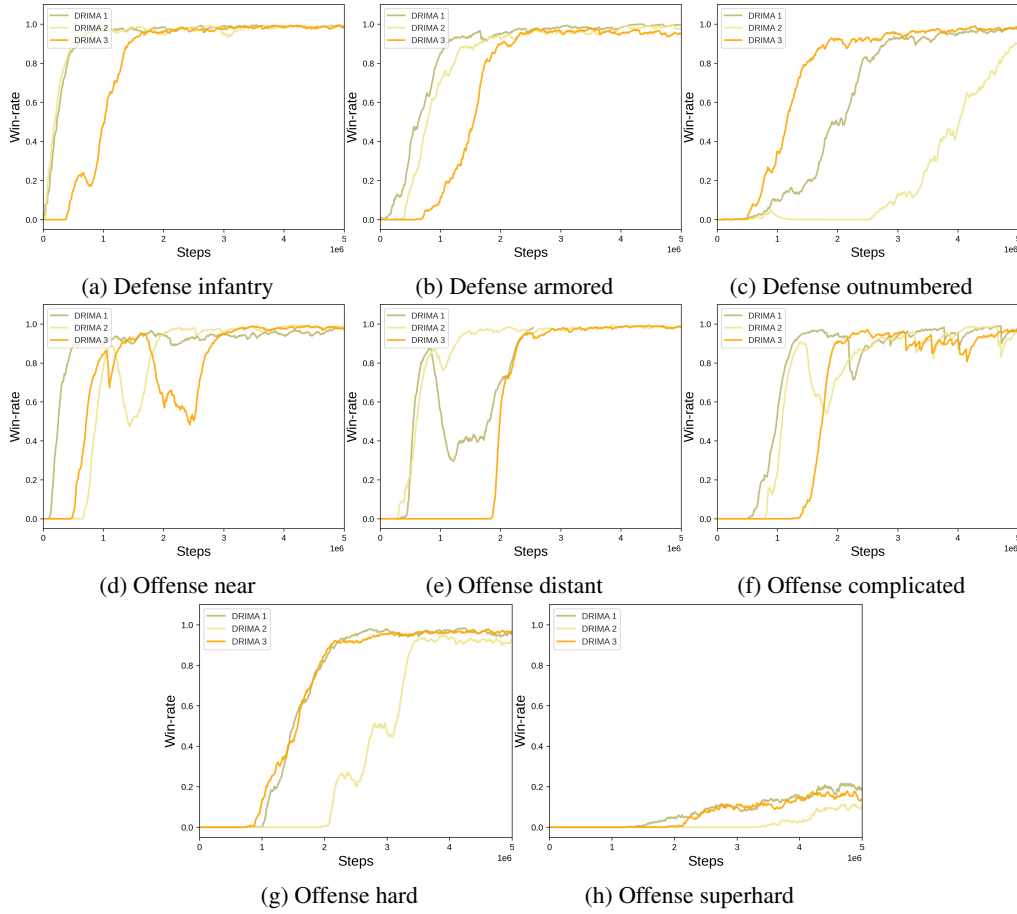
(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

(d) Offense near   (e) Offense distant   (f) Offense complicated

(g) Offense hard   (h) Offense superhard

Figure D.9: DRIMA trained on the sequential episodic buffer

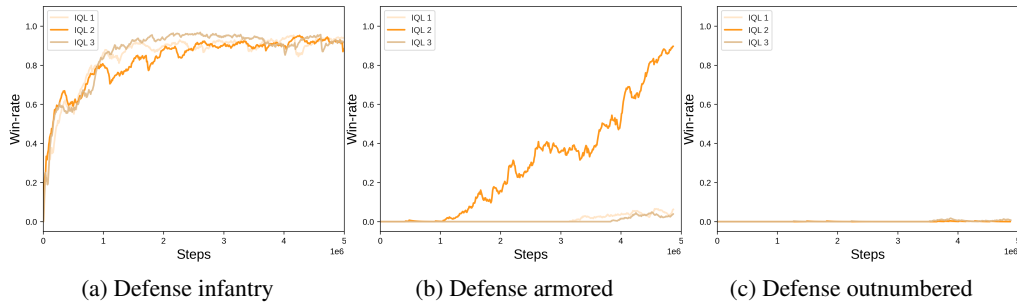(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

Figure D.10: IQL trained on the sequential episodic buffer

characteristics, we may train a model with risk-sensitive behavior criteria.

Risk-sensitive experiments 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 $\tau \sim \mathcal{U}([\cdot, \cdot])$. We name each portion as risk-averse, risk-neutral-averse, risk-neutral-seeking and risk-seeking sampling. The results are shown in Figure E.29. Looking the results, we notice that risk-neutral-averse criteria with DMIX work best in most of the 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

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered
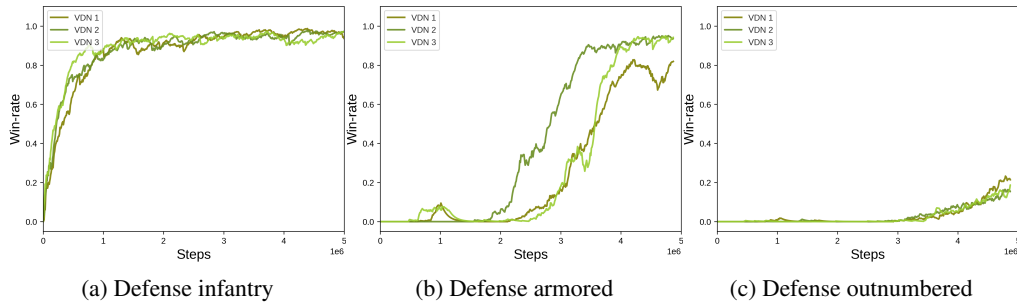
Figure D.11: VDN trained on the sequential episodic buffer

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered
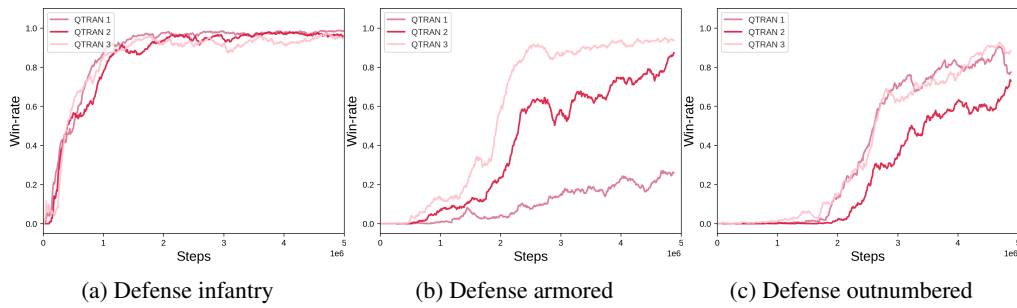
Figure D.12: QTRAN trained on the sequential episodic buffer

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered
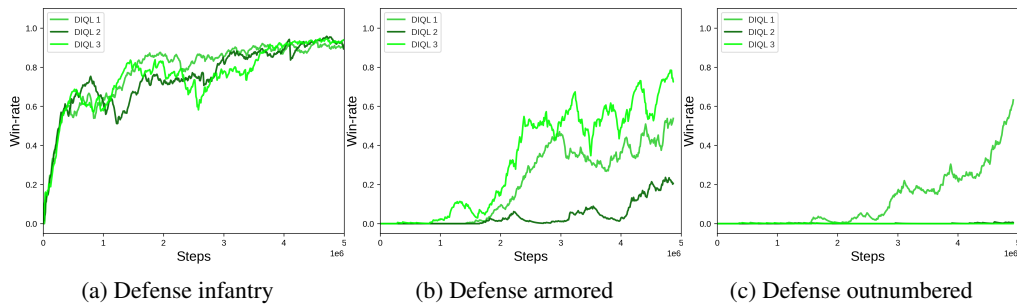
Figure D.13: DIQL trained on the sequential episodic buffer

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered
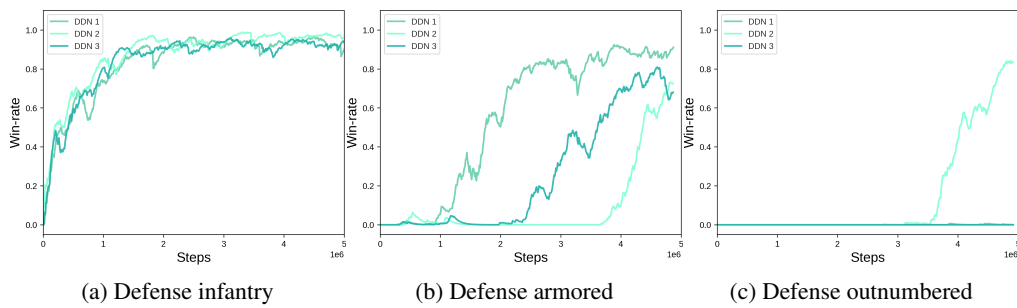
Figure D.14: DDN trained on the sequential episodic buffer

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.

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered
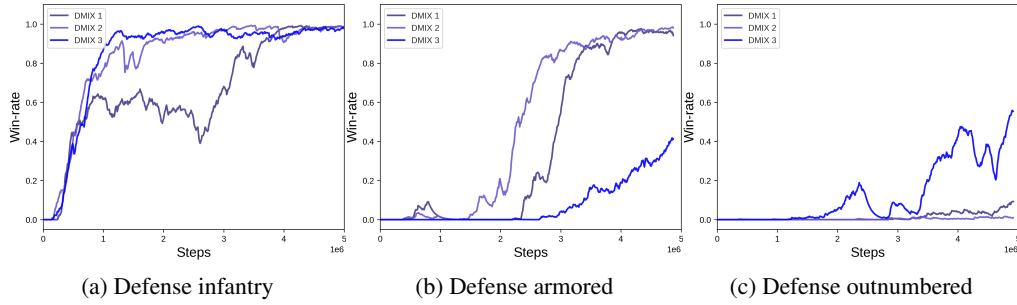
Figure D.15: DMIX trained on the sequential episodic buffer

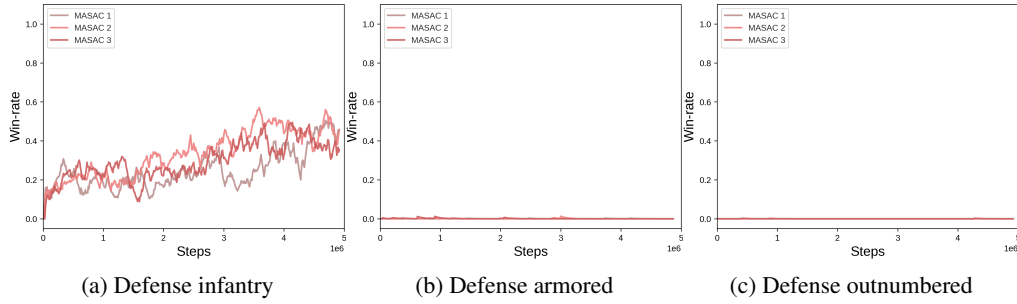(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

Figure D.16: MASAC trained on the sequential episodic buffer

E.1.3. POLICY-BASED ALGORITHMS

We choose MASAC as a representative model among policy-based algorithms since SAC was designed to improve exploration strategy by adding an entropy term to the loss function with coefficient α, as described in Equation C.15. The default value for α is 0.01, which favors exploration over exploitation. We set α space to the values $\{0.01, 0.1, 0.5, 0.9\}$. Results indicate that increasing the entropy term for exploration in our scenario has little effect, and algorithms may require additional parameters for exploration. The outcome is in Figure E.28

E.2. Reward Engineering

We evaluate the extent to the reward function engineering can solve the offense scenarios. The alternative reward function is designed to reward as agents get closer to the enemies. Since the offensive scenarios require agents to find the enemies before defeating them, agents with the explicit reward for finding the enemies to agents early in training can solve tasks sequentially. In detail, agents are rewarded by how much they get close to the enemies with respect to euclidean distance in the map until the 100k training time step. After that, they get the basic reward that is used in SMAC, SMAC$^+$. The equation of the alternative reward function is as follows

$$R_{alt}(h, u) = \Delta \frac{1}{|e|} \sum_a \sum_e distance(pos_a - pos_e) \tag{E.18}$$

where pos is 2D cartesian coordinates, a and e denote agents and enemies respectively.

We test QMIX in `Off_distant` with parallel episodic buffer and identical hyper-parameter setting. As in Figure E.30a, the results show that reward engineering makes significant improvements both in final win-rate and convergence speed compared to QMIX with a basic reward function. Figure E.30b illustrate agents find the enemies properly in the early stage of training. Even though the reward function is changed drastically at 100k time step, the agents utilize the information learned through the alternative reward function until the end of training Figure E.30c.

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

(d) Offense near   (e) Offense distant   (f) Offense complicated
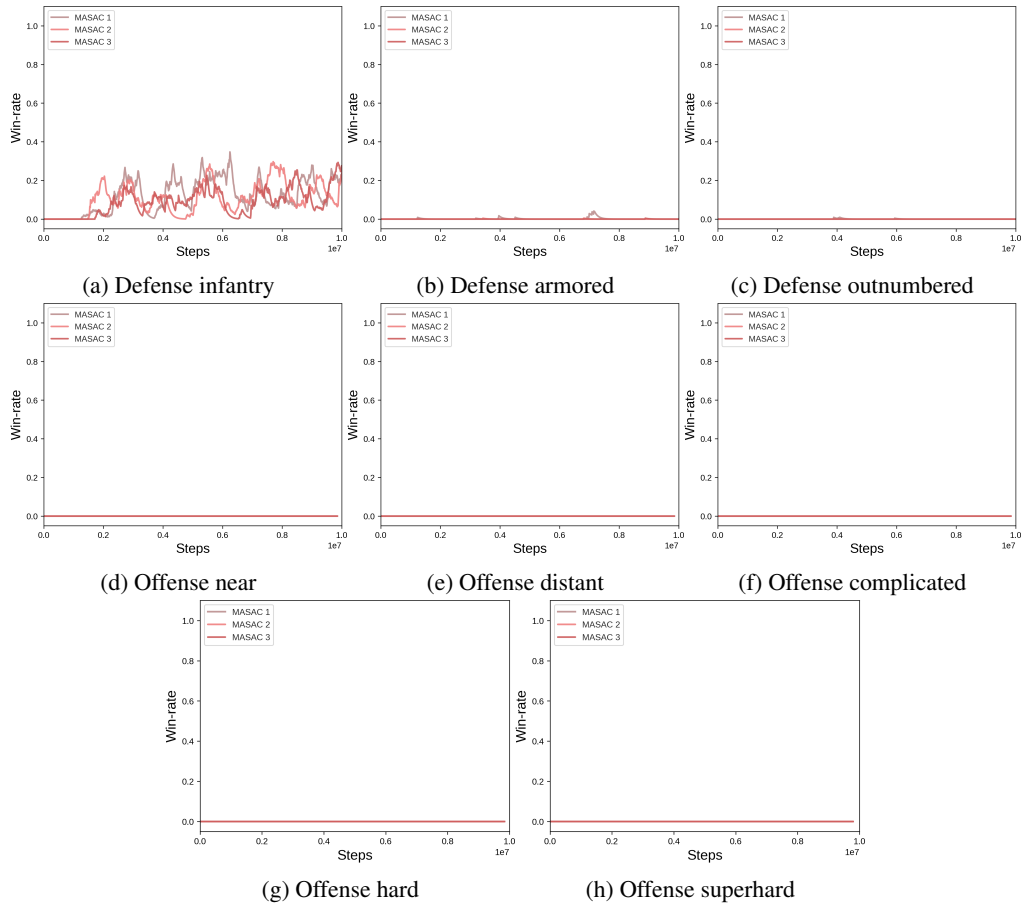
(g) Offense hard   (h) Offense superhard

Figure D.17: MASAC trained on the parallel episodic buffer

E.3. Heat-map Analysis of Offensive Scenarios

We argue that heat-maps of agent movement in offensive scenarios prove efficient exploration of MARL algorithms in **??**. In this subsection, we show all heat-maps generated by four algorithms like COMA, QMIX, MADDPG, and DRIMA trained on the sequential episode buffer. As previously indicated, as the number of training steps increases, all agents (lower side) must move forward to the enemies positioned on the hill (upper side). As shown in Figure E.31, we identify that DRIMA can make all agents go for the enemies in all scenarios meanwhile, the other algorithms do not succeed in making agents move forward in some scenarios. When compared of Table D.12 and Figure E.31, we can see that the win-rate performance at the test phase is highly correlated to agents going for enemies without direct incentives. Therefore, we claim that the exploration capability of MARL algorithms enables agents to train to discover enemies even though an algorithm at early stage does not move agents forward to enemies.

Additionally, we observe that these heat-maps might be used to determine the complexity of offensive scenarios. In the `Off_near`, `Off_near`, two algorithms, at least, seems to find enemies, however all algorithms except DRIMA fail to find enemies in the `Off_complicated`, `Off_hard` and `Off_superhard`. When you see the results of QMIX in the `Off_complicated` and `Off_hard`, the agents appear to be approaching enemies, but they are not capable of precisely locating enemies. In these situations, where the enemies are scattered two-sided in front of each entrance of the hill, QMIX removes just one-sided opponents and does not find the remaining regions. On the other hand, DRIMA perfectly complete to locate enemies scattered and eliminate them in this situation. Especially in the `Off_superhard`, the other algorithms do not completely train, but DRIMA makes agents move forward to the opponents and begin to beat them. Nonetheless, as seen in Table D.12, DRIMA attains about 10% win-rate performance on `Off_superhard`, leading us to conclude that a longer training horizon is necessary in order to simultaneously learn fine-manipulation and enemy detection.
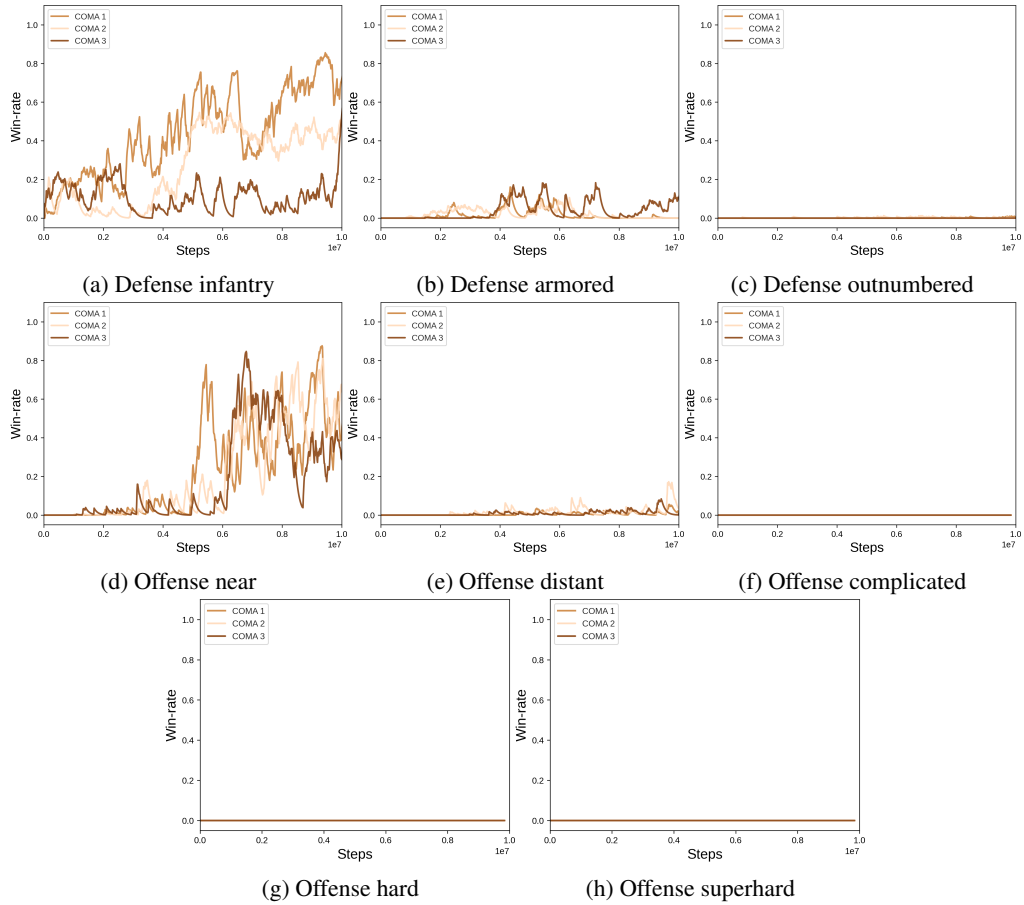
(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard

Figure D.18: COMA trained on the parallel episodic buffer

(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard
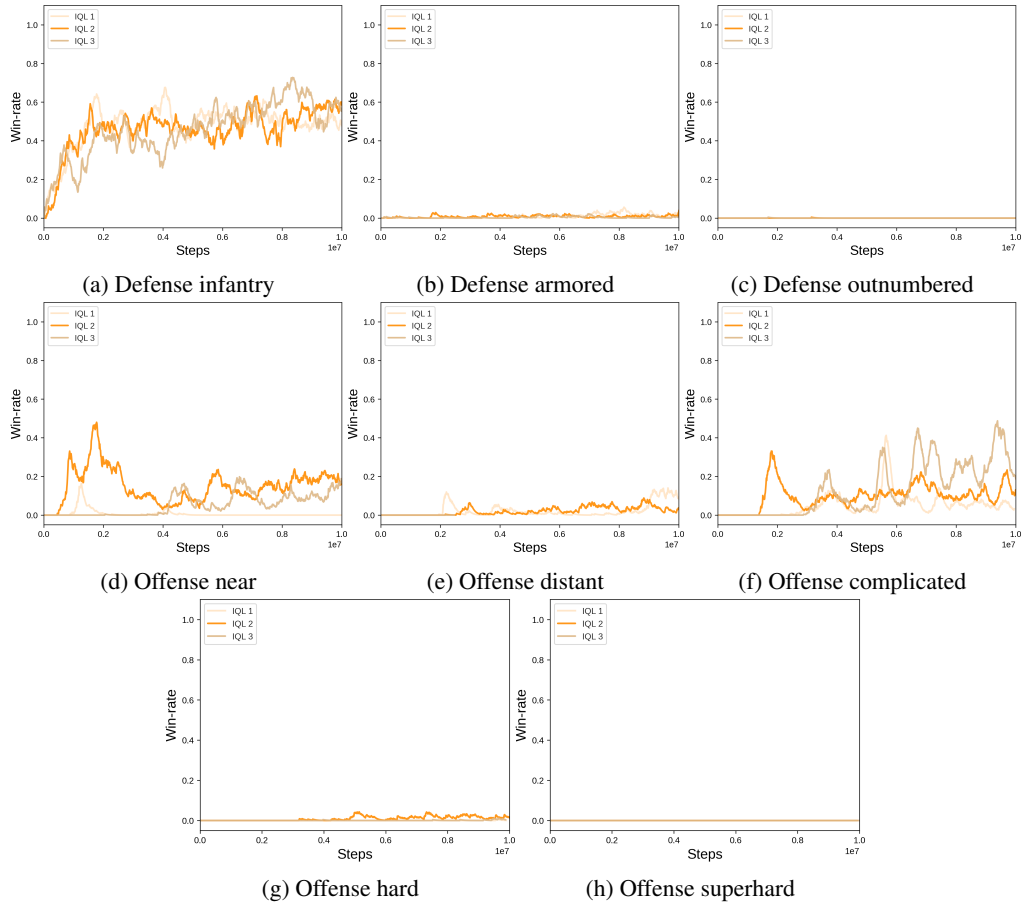
Figure D.19: IQL trained on the parallel episodic buffer

(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard
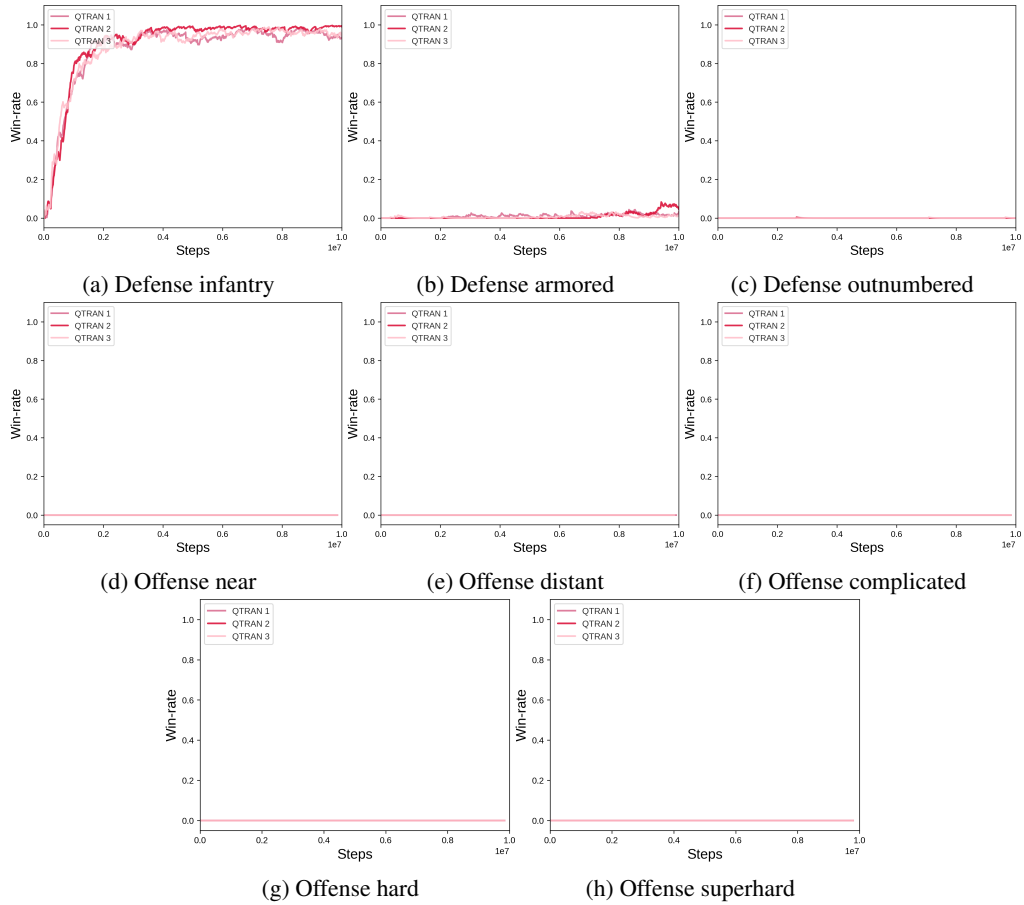
Figure D.20: QTRAN trained on the parallel episodic buffer

(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard
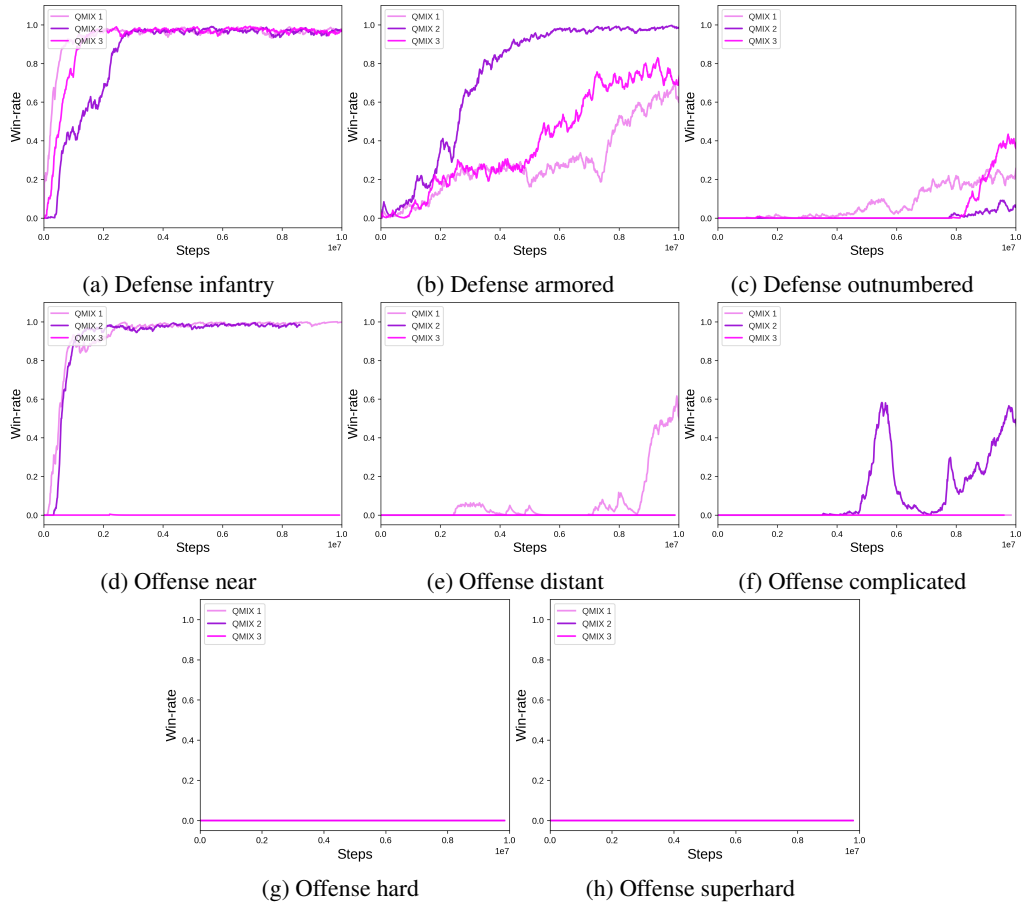
Figure D.21: QMIX trained on the parallel episodic buffer

(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard
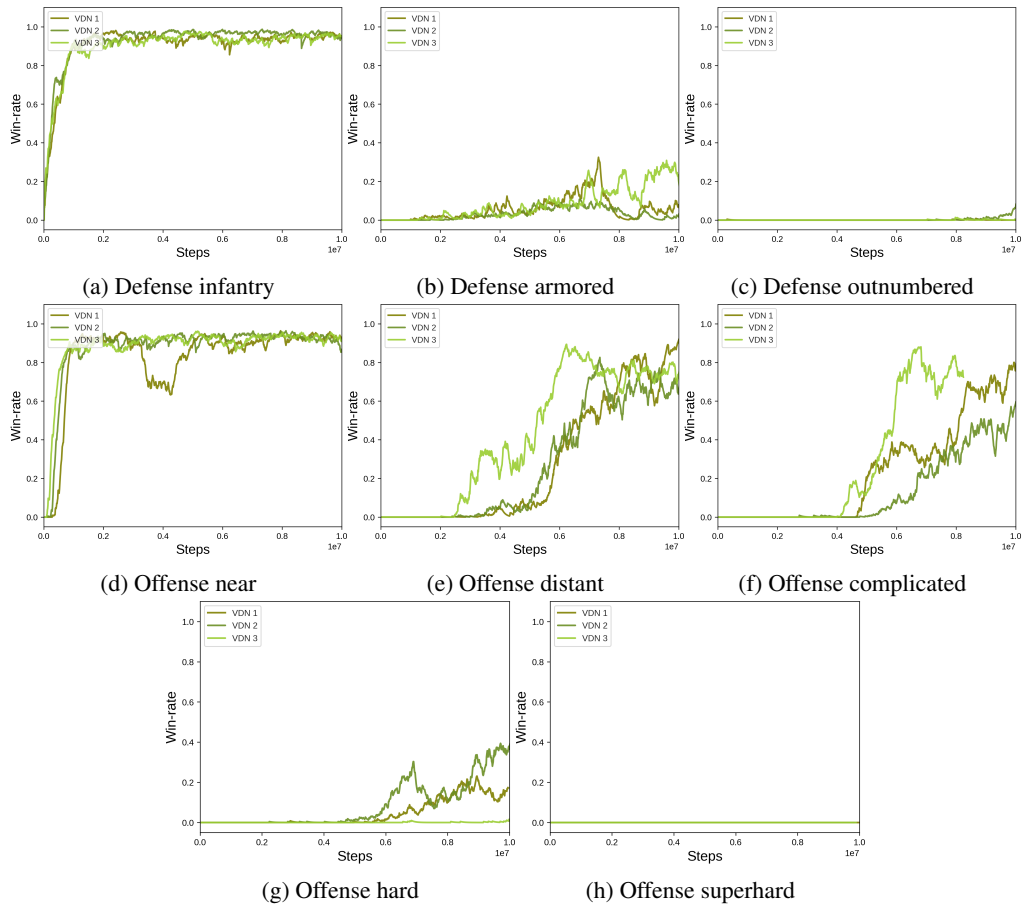
Figure D.22: VDN trained on the parallel episodic buffer

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

(d) Offense near   (e) Offense distant   (f) Offense complicated

(g) Offense hard   (h) Offense superhard
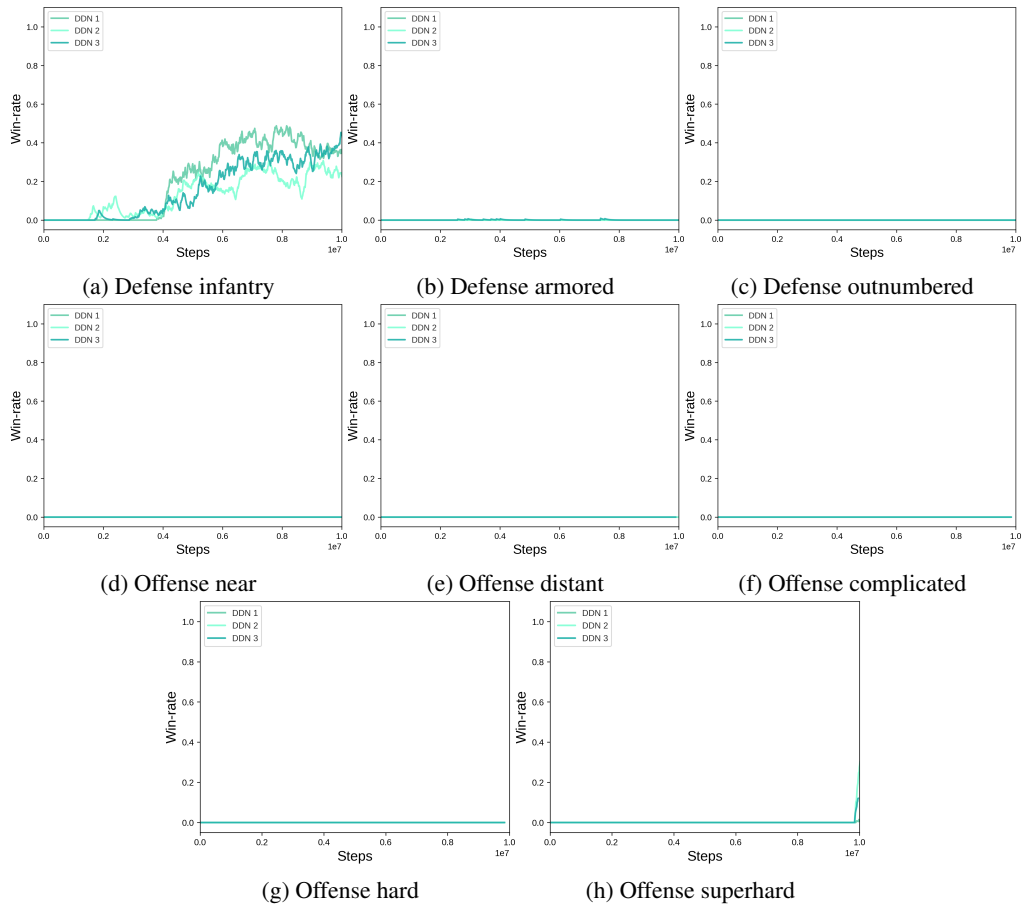
Figure D.23: DDN trained on the parallel episodic buffer

(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard
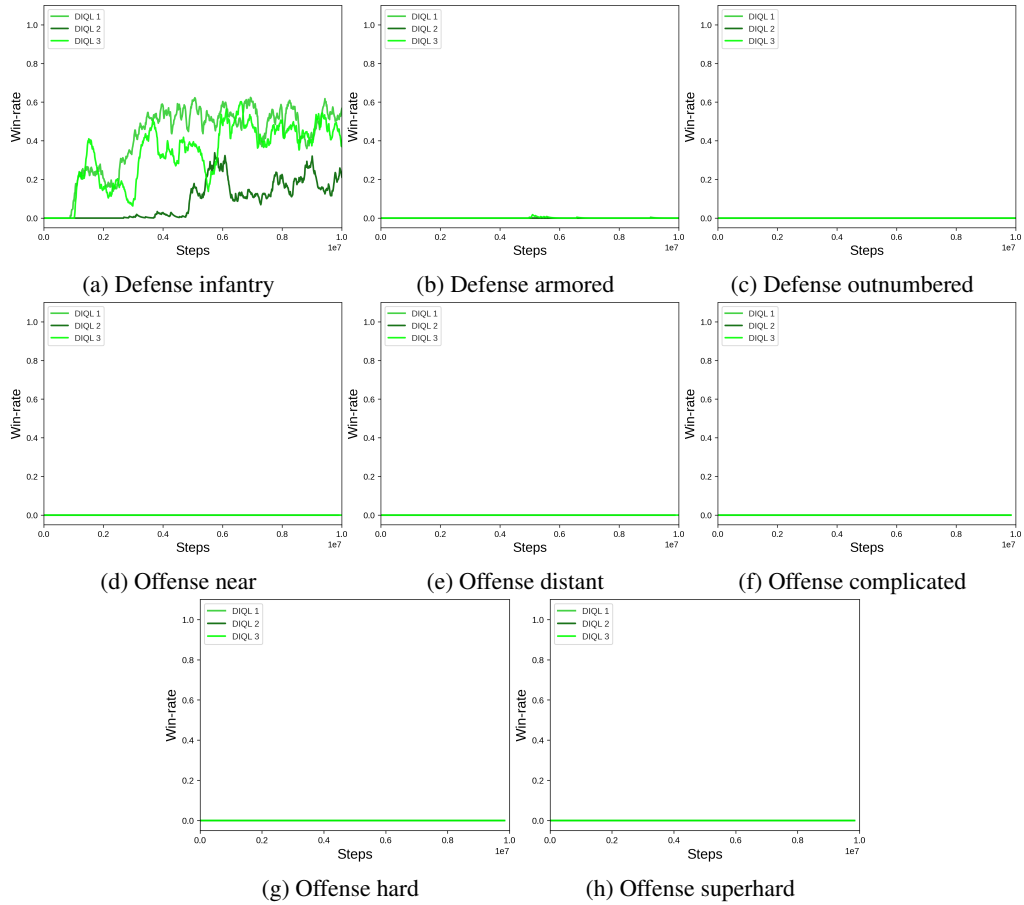
Figure D.24: DIQL trained on the parallel episodic buffer

(a) Defense infantry

(b) Defense armored

(c) Defense outnumbered

(d) Offense near

(e) Offense distant

(f) Offense complicated

(g) Offense hard

(h) Offense superhard
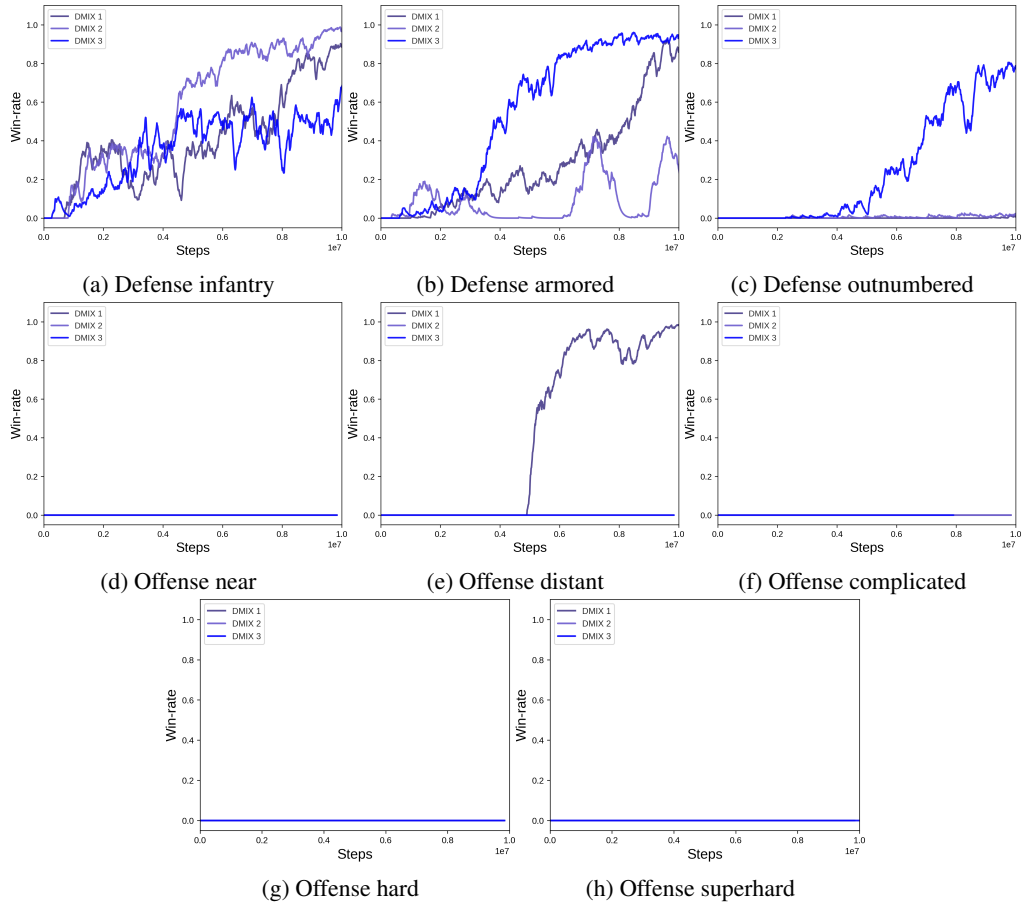
Figure D.25: DMIX trained on the parallel episodic buffer

(a) Defense infantry   (b) Defense armored   (c) Defense outnumbered

(d) Offense near   (e) Offense distant   (f) Offense complicated

(g) Offense hard   (h) Offense superhard
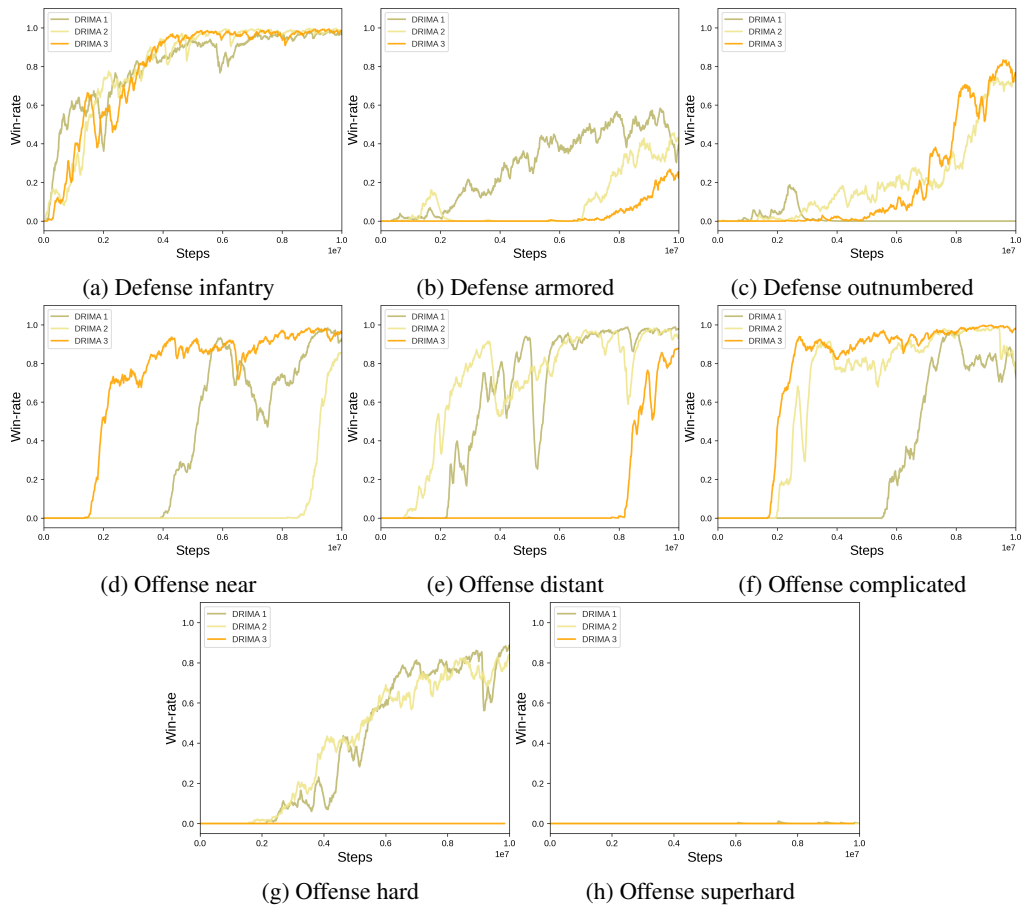
Figure D.26: DRIMA trained on the parallel episodic buffer

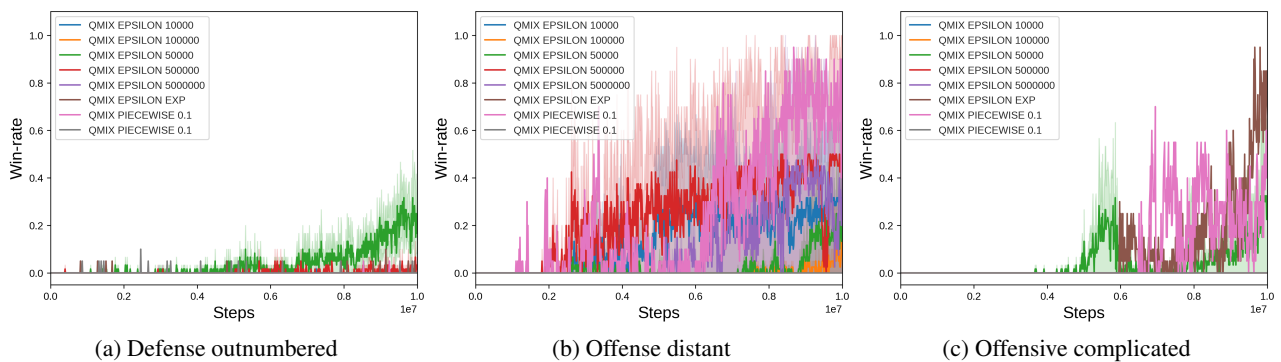(a) Defense outnumbered   (b) Offense distant   (c) Offensive complicated

Figure E.27: 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
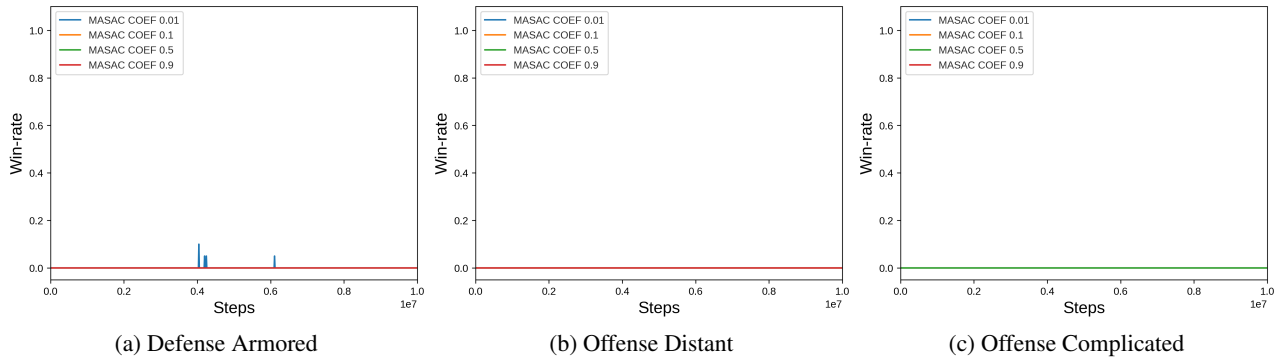
(a) Defense Armored   (b) Offense Distant   (c) Offense Complicated

Figure E.28: Sweeping out coefficient α of entropy term in loss function with MASAC algorithm. The numbers next to the COEF letter means the α. As the coefficient is close to the 0 it indicates less exploration, vice versa.

(a) Defense Infantry   (b) Defense Armored   (c) Defense Outnumbered

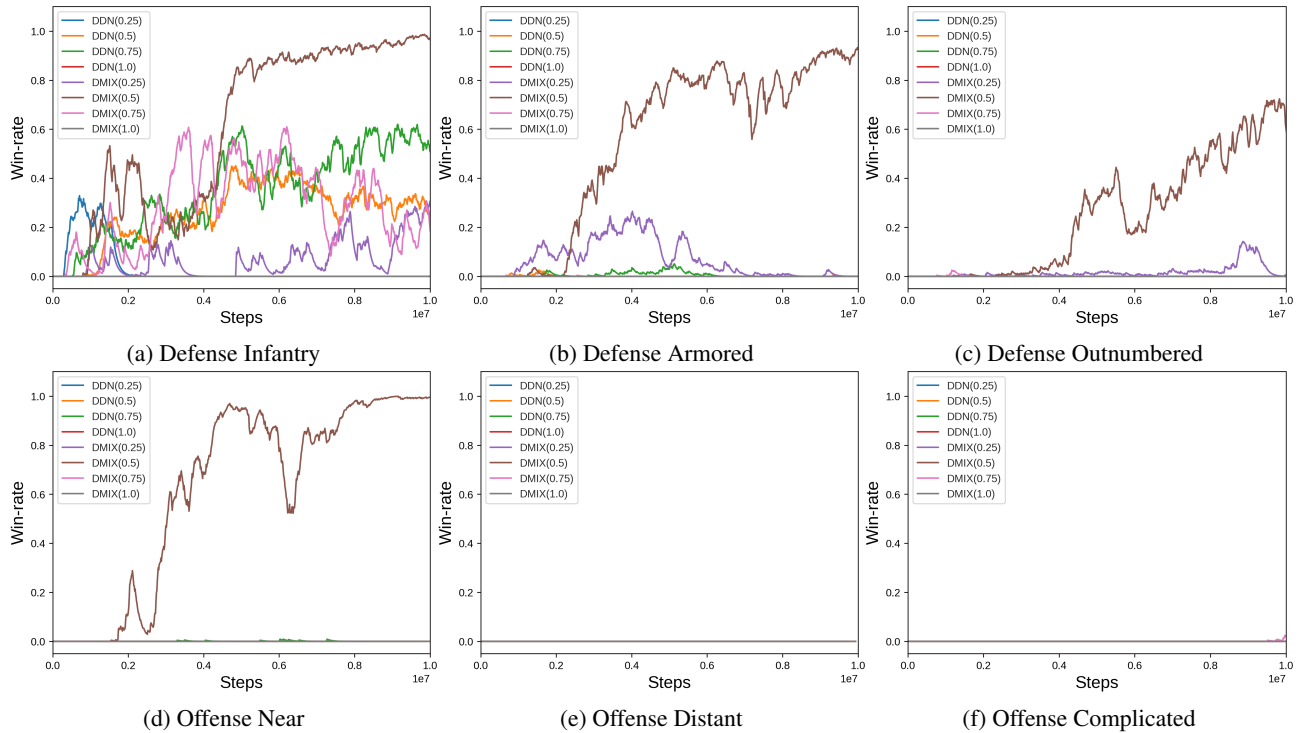(d) Offense Near   (e) Offense Distant   (f) Offense Complicated

Figure E.29: Risk-sensitive experiments with DFAC algorithm according to the sampling methods. DDN(0.25) and DDN(0.5) means sampling $\tau \sim \mathcal{U}([0, 0.25]), \mathcal{U}([0.25, 0.5])$ each using DDN algorithm.

(a) QMIX Training Curve

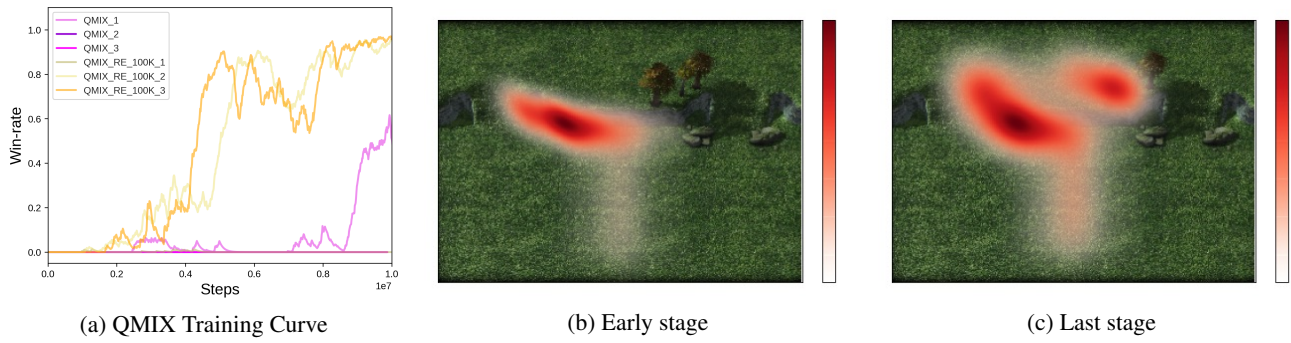
(b) Early stage

(c) Last stage

Figure E.30: (a) is the result of Reward engineering experiment. The line QMIX RE 100K indicate QMIX trained with reward engineering until 100k training time step. (b), (c) are heat-maps of all agents movements at 500k, 10m training time step each.

(a) `Off_near` scenario

(b) `Off_distant` scenario

(c) `Off_complicated` scenario

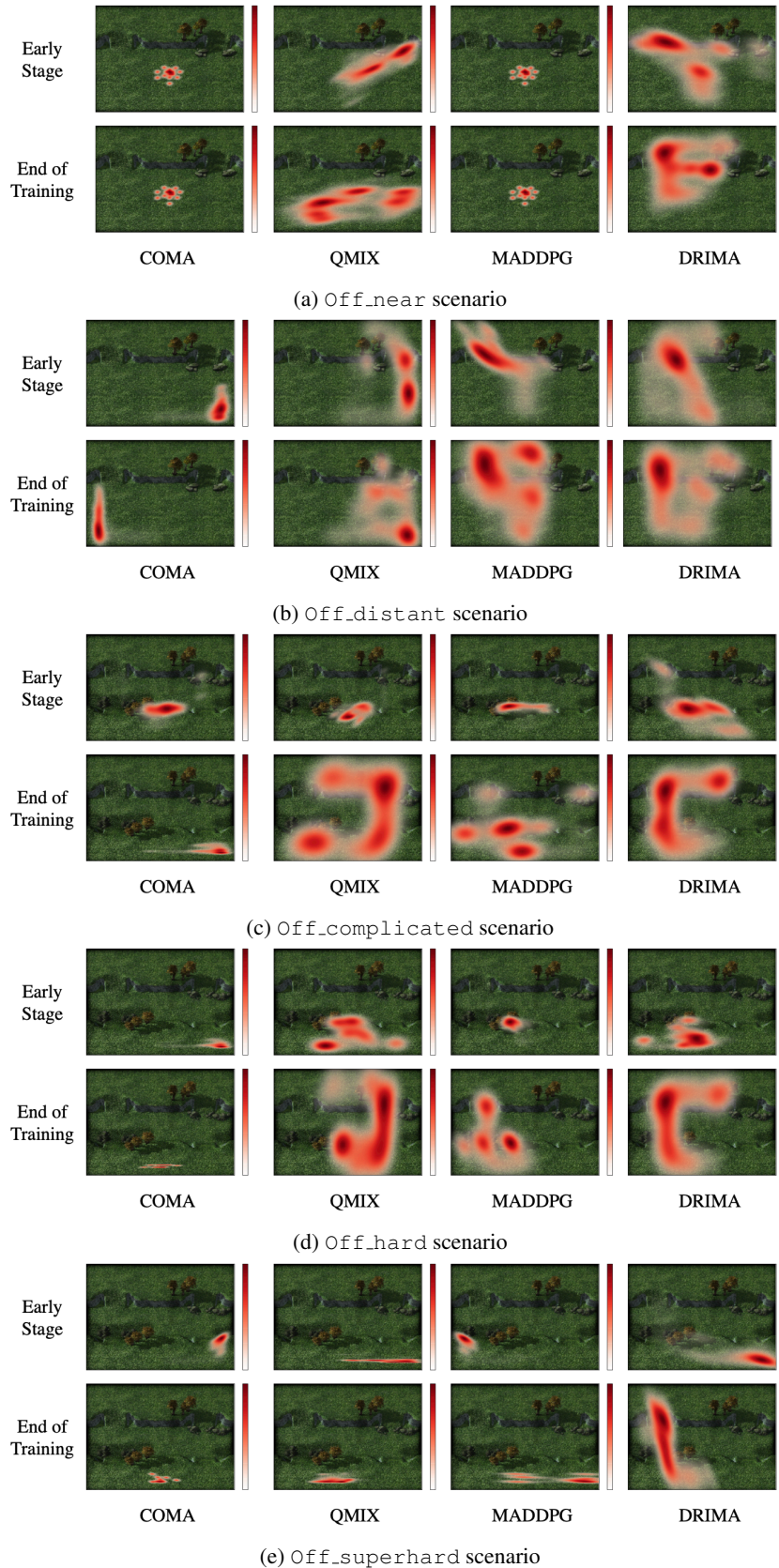(d) `Off_hard` scenario

(e) `Off_superhard` scenario

Figure E.31: Heat-maps of agents movement by COMA, QMIX, MADDPG and DRIMA in all offensive scenarios. All baselines are trained on the sequential episodic buffer. The result of early stage is captured at six hundred thousands steps while The end of training is 5 million cumulative steps.