Summer 2021

MAIDRL: Semi-centralized Multi-Agent Reinforcement Learning using Agent Influence

Anthony Lee Harris
Missouri State University, Anthony999@live.missouristate.edu

MAIDRL: SEMI-CENTRALIZED MULTI-AGENT INFLUENCE DENSE REINFORCEMENT LEARNING

A Master's Thesis

Presented to

The Graduate College of

Missouri State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science, Computer Science

By

Anthony Lee Harris

July 2021

MAIDRL: SEMI-CENTRALIZED MULTI-AGENT INFLUENCE DENSE REINFORCEMENT LEARNING

Computer Science

Missouri State University, July 2021

Master of Science

Anthony Lee Harris

ABSTRACT

In recent years, reinforcement learning algorithms, a subset of machine learning that focuses on solving problems through trial-and-error learning, have been used in the field of multi-agent systems to help the agents with interactions and cooperation on a variety of tasks. Given the enormous success of reinforcement learning in single-agent systems like Chess, Shogi, and Go, it is natural for the next step to be the expansion into multi-agent systems. However, controlling multiple agents simultaneously is extremely challenging, as the complexity increases tremendously with the number of agents in the system. Existing approaches in this regard use a wide range of centralized, decentralized, semi-centralized, and even hybrid centralized-decentralized state representation methodologies. In this thesis, I propose a novel semi-centralized deep reinforcement learning algorithm, MAIDRL, for multi-agent interaction in mixed cooperative and competitive multi-agent environments. Specifically, I design a robust DenseNet-style actor-critic structured deep neural network for controlling multiple agents based upon the combination of local observations and abstracted global information to compete with opponent agents. I extract common knowledge through influence maps considering both enemy and friendly agents for fine-grained unit positioning and decision-making in combat. Compared to the centralized method, my design promotes a thorough understanding of the potential influence that a unit has without the need for a complete view of the global state. In addition, this design enables multi-agent understanding of a subset of the global information relative to common goals, unlike fully decentralized methods. The proposed method has been evaluated on StarCraft Multi-Agent Challenge scenarios in a real-time strategy game, StarCraft II, and the results show that, statistically, the agents controlled by MAIDRL perform better than or as competitive as those controlled by centralized and decentralized methods.

KEYWORDS: deep reinforcement learning, A2C, DenseNet, multi-agent system, influence map, StarCraft II, SMAC, MAIRL, MAIDRL

MAIDRL: SEMI-CENTRALIZED MULTI-AGENT INFLUENCE

DENSE REINFORCEMENT LEARNING

By

Anthony Lee Harris

A Master's Thesis
Submitted to the Graduate College
Of Missouri State University
In Partial Fulfillment of the Requirements
For the Degree of Master of Science, Computer Science

July 2021

Approved:

Siming Liu, Ph.D., Thesis Committee Chair

Razib Iqbal, Ph.D., Committee Member

Jamil Saquer, Ph.D., Committee Member

Anthony Clark, Ph.D., Committee Member

Julie Masterson, Ph.D., Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to thank Dr. Razib Iqbal for setting me on the path of graduate research, and Dr. Siming Liu for his tremendous guidance and insights throughout my graduate studies. Their rigor and dedication to their crafts have been an inspiration that has pushed me to strive for nothing short of excellence in this work. Additionally, I extend my thanks to Dr. Jamil Saquer and Dr. Anthony Clark for serving on my thesis committee in addition to Dr. Liu and Dr. Iqbal. These four incredible professors have taught me a tremendous amount in my time at Missouri State University, and I am immensely grateful.

I would also like to thank my father, Rodger Harris, and grandparents, Mae Lynd and Jackie Morgan, whose unending auxiliary support has made all my studies possible. Their willingness to go out of their way to support me in whatever ways they could is something I will never forget. Finally, it is with utmost gratitude that I thank my wife and best friend, Staci Lunn, for her infinite patience and overwhelming support during the entirety of my studies. Words cannot express how proud and happy I am to have her by my side. I am humbled and grateful to have had such a phenomenal group of individuals to depend upon, and I could not have made it this far without each of them.

This thesis is dedicated to my sons Aiden and Alexander Harris. I am so immensely proud of you both, and I hope that this serves as a reminder that you can achieve *anything* in this life. I look forward to many more years of laughter and games with you and mom, and I love you both more than I can possibly express.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

Abbreviation	Definition
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
AC	Actor-Critic
AI	Artificial Intelligence
AIM	Agent Influence Map
ALE	Arcade Learning Environment
BC	Behavioral Cloning
BRNN	Bidirectional Recurrent Neural Network
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DP	Dynamic Programming
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ELU	Exponential Linear Unit
GA	Genetic Algorithm
GCLA	Global Critic - Local Actor
IM	Influence Map
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
PS-MAGDS	Parameter-Sharing Multi-Agent Gradient Decent Sarsa
MAIDRL	Multi-Agent Influence Dense Reinforcement Learning
MAIM	Multi-Agent Influence Map
MAIRL	Multi-Agent Influence Reinforcement Learning

MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent System
MDP	Markov Decision Process
ML	Machine Learning
MOBA	Multiplayer Online Battle Arena
NN	Neural Network
NP	NumPy
PF	Potential Field
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RRL	Relational Reinforcement Learning
SAC	Stochastic Actor-Critic
SARSA	State, Action, Reward, State, Action
SC2	StarCraft II
SC2LE	StarCraft II Learning Environment
SCC	StarCraft Commander
SCDDPG	Semi-Centralized Deep Deterministic Policy Gradient
SD	Standard Deviation
SL	Supervised Learning
SMAC	StarCraft Multi-Agent Challenge
TCLA	Total Critic - Local Actor
TD	Temporal Difference
TF	Tensorflow
TRPO	Trust Region Policy Optimization
UL	Unsupervised Learning

INTRODUCTION

Artificial Intelligence (AI) has made enormous progress in many aspects of our world in recent decades. The rapid evolution in AI has enabled the high performance of a variety of tasks including robotics, autonomous driving, and game playing. Particularly in game playing, AI has reached human-level performance or even outperformed the best human experts [1]-[5]. However, the majority of the achievements of AI have been in single-agent circumstances, where collaboration and competition among agents are unnecessary [2], [4], [6], [7]. Meanwhile, there are a large number of applications that involve interaction between multiple agents that are naturally modeled as cooperative or competitive multi-agent systems (MAS). For example, coordination of self-driving vehicles, multi-robot control, and multiplayer games all operate in a multi-agent domain. Among many AI techniques, reinforcement learning (RL) has been considered one of the most promising methods in the past several years [8]. In this thesis, I am interested in investigating how to expand RL effectively to learn interactions among agents in MAS.

The recent development of deep neural networks (DNN) combined with improved computational capability has led to considerable progress in RL, with many RL methods being expanded into deep RL (DRL). One traditional RL method that has experienced significant success is Q-Learning. A major limitation of Q-Learning is the inherent need to build a table of mappings from state-action pairs to calculated action values [9]. This poses a significant scalability issue in that, as the complexity of the state space grows, the number of possible state-action pairs grows exponentially, leading to the requirement of exponentially larger amounts of memory. The introduction of DNNs into this method spawned a new learning method called

Deep Q-Networks (DQN), which effectively resolved the curse of dimensionality with respect to memory consumption by instead using a DNN as a function approximator to approximate the expected action value function [6], [7].

Another contributor to the rapid growth of DRL in both single-agent and multi-agent RL (MARL) scenarios has been the development of widely available simulation environments such as the Arcade Learning Environment (ALE) introduced by Bellemare et al. in [10], which uses various Atari games as a platform for learning, Malmo by Johnson et al. in [11], which similarly uses Minecraft as a platform, the StarCraft II Learning Environment (SC2LE) by Vinyals et al. in [12], and the StarCraft Multi-Agent Challenge (SMAC) by Samvelyan et al. in [13], each of whom are deployed in StarCraft II (SC2) environments. A sample of screenshots from the ALE can be seen in Figure 1, where the games Pitfall!, Space Invaders, Freeway, and SeaQuest are shown from left to right. Video games, particularly those with research-oriented tools and environments like those aforementioned, allow researchers to perform hundreds or even thousands of experiments in parallel. This is particularly valuable as RL is generally a learn through experience methodology. Unfortunately, solving MAS problems with traditional RL is non-trivial. Extending RL to enable cooperation and competition among agents is critical to building artificially intelligent systems in multi-agent environments.

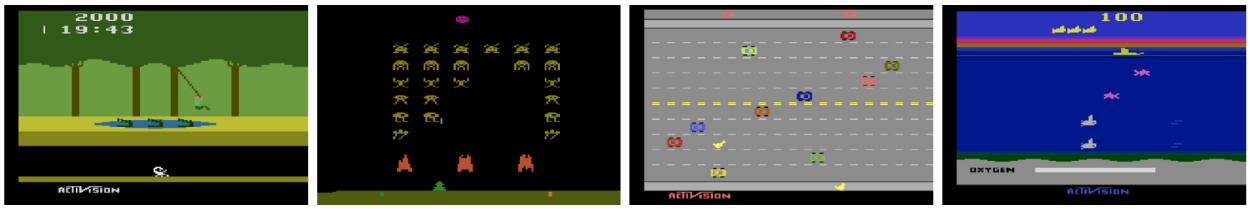

Figure 1. Sample Game Screenshots from the Arcade Learning Environment [10].

There are many challenges in scaling up RL into MAS. One of the primary challenges of MARL is that the traditional RL techniques like Q-Learning and policy gradient do not generalize well to MAS. This is evident when considering the state of the MAS from the perspective of an individual agent. The actions of this agent can be perceived to have non-stationary impacts in the environment as a direct result of the actions of other agents. This leads to significant stability issues, as well as the limitation of applicable stability enhancing techniques such as experience replay. Furthermore, MAS themselves introduce additional layers of complexities. For example, in cooperative MAS, agents must be able to act as a coordinated unit which requires an understanding of the agents with whom they are cooperating. This challenge in particular has been the focus of much research. A common starting point is to utilize complete state information to train agents. This technique, commonly referred to as centralized learning, provides each agent with perfect environmental information regarding both the agent's local observations and the state observations on a global scale.

Many MAS, however, try to avoid the use of such perfect information as it may not be available during execution. Such work has led to decentralized learning, which only provides an agent with its local observations. This decentralized method, while more widely applicable in practice due to the lack of dependence upon global information, generally struggles to understand global objectives, leading to lackluster performance in many cases. This has been improved somewhat by the introduction of the hybrid centralized learning, decentralized execution as presented by Foerster et al. and Lowe et al. in [14], [15]. In response to the aforementioned challenges, there has been much work regarding an intermediate technique called semi-centralized learning [8]. Semi-centralized learning can be applied in various ways, but it largely revolves around the representation or provision of global information in an

imperfect way that is more generalizable to scenarios where perfect information may be otherwise unavailable. An example of a semi-centralized MARL technique is the Semi-Centralized Deep Deterministic Policy Gradient (SCDDPG), which creates a 2-level hierarchical structure of AC networks, one of which is provided only with local agent observations while the other is provided with global observations, and both of which are then used to provide varying levels of information granularity to the agents [8]. How to use global information with macro-level control has been arguably solved in the context of SC2 by Vinyals et al. with AlphaStar in [3] and Wang et al. with StarCraft Commander in [5], however, the question of how to use global information for fine-grained decision making is still open.

My response to the above question is the novel definition and application of Agent Influence Maps (AIM), aggregated into a global Multi-Agent Influence Map (MAIM), which is used in addition to local agent observations for fine-grained decision-making. By abstracting a subset of the perfect global information into an incomplete but descriptive representation, I demonstrate a significant improvement over centralized, decentralized, and hybridized methods. I evaluate the applicability of MAIMs by defining a simple artificial Neural Network (NN) that contains only a single hidden layer and comparing the results of MAIM state representation to centralized and decentralized alternatives, with no additional changes. This use of semi-centralized MAIM state representation is defined as a new learning paradigm that I coin as Multi-Agent Influence Reinforcement Learning (MAIRL), which is a preliminary contribution of my work. This novel MAIRL method is then applied with a DenseNet-style model architecture to improve the robustness and capability of the system. The DenseNet-style model architecture was chosen because it has been shown to handle the exploding and vanishing gradient problems of DNNs in a more memory-efficient manner than the comparable ResNet structure [16], [17].

The specifics of my DenseNet-style architecture implementation can be found in the Simple versus Dense A2C section on page 34. To my knowledge, the application of DenseNet-style architectures has only ever been done in the context of Convolutional Neural Networks (CNNs) in image classification [16], [18], [19] optical flow prediction [20], or even in keyword identification [21], but my work currently benefits from this architecture using simple Dense layers, making this application somewhat novel as well. I define this combined use of MAIRL and DenseNet-style model architectures as the logical extension of the aforementioned learning paradigm, which I present as Multi-Agent Influence Dense Reinforcement Learning (MAIDRL), and it is the primary contribution of my work. MAIRL and MAIDRL are evaluated on StarCraft Multi-Agent Challenge (SMAC) [22] scenarios in a real-time strategy game, SC2. The results show that, statistically, the agents controlled by MAIDRL perform better than or as well as those controlled by centralized and decentralized methods in SMAC scenarios of varying complexity.

The remainder of this thesis is organized as follows: the Background section introduces various concepts regarding machine learning, with emphasis on RL, while the Related Work section introduces some of the relevant RL techniques that have been shown to perform well in RL and MARL scenarios. The Simulation Environment section provides a detailed description of the environment that was used in my experimentation, and the Methodology section provides detailed information regarding said experimentation. The Results and Discussion section subsequently presents the results from my experimentation, as well as the observed behaviors that were learned by the various considered methodologies, and the Conclusion And Future Work section draws the conclusions and suggests future works.

BACKGROUND

Machine learning (ML) is a subset of AI and is defined to be the use of computational methods to better predict future information based upon historical data [23]. The primary components of ML methods can be categorized into any one of three broad categories, with many advanced ML methods using some combination of the three: supervised learning (SL), unsupervised learning (UL) and reinforcement learning (RL) [24], each of whom are discussed in the following subsections.

Supervised Learning

SL is an incredibly powerful methodology and is defined as the process of identifying some function f that maps some set of inputs X to a corresponding set of target outputs Y, such that a minimum amount of cost or risk is accrued [23]. The cost or risk is often problem-dependent, and it determines the efficacy of the resulting feature mapping. As the amount of available labeled data grows, so too does the power of SL, because the data, as well as the learned function mapping inputs to outputs, grows more representative of the complete distribution of all possible input-output pairs, which in turn leads to better generalizability to pairs that have not been encountered before. SL can also be supplementary to RL by being used as a warm-start solution to provide insights more quickly and efficiently to RL agents via supervised means. This method is referred to as behavioral cloning, or imitation learning [25]. SL is the most commonly used methodology in classification, regression, and ranking problems [23], but it has a major limitation: data dependency. SL, while preferrable if possible, is only possible when labeled data is readily available, and there are many cases when it is not.

Unsupervised Learning

When there exists some feature set of inputs, but no known corresponding labeled output set, UL is used in lieu of SL. UL is the process of using an unlabeled set of feature data to draw conclusions about the entire feature space, and is most often used in clustering, which is the UL equivalent of categorization [23]. Clustering is the attempt to identify commonalities amongst subsets of the inputs such that the inputs within a cluster are logically similar to those within the same cluster, while also remaining distinct and separable from those in other clusters. Due to the uncertainty of the determined clusters relative to the true categories, however, it is challenging to truly define the efficacy of UL methods, though some metrics such as the Silhouette score [26] have been defined to provide some insight to the quality of the determined clusters. Like SL, however, UL still maintains dependency upon the availability of feature data, albeit without the dependence upon the corresponding labels. As such, UL is only viable in scenarios where this feature data is available.

Reinforcement Learning

RL, unlike SL and UL, is wholly non-reliant upon the availability of data. Rather, RL is a goal-directed learning methodology that learns through interaction in an environment, instead of pre-existing data [9]. This definition then shifts the dependency from data availability to interactability in an environment, which can be considered as the process of generating the data or experience that is then used to make insights about future interactions. An important distinction between RL and other types of ML is that RL learns through interaction evaluation based upon the results of the interaction, rather than by considering a “correct” action [9]. This process is similar to the way in which we might consider living creatures learn in reality. For

example, when a child is first learning how to walk, they might stumble and fall hundreds of times simply exploring what they *can* do with the tools at their disposal in the environment around them, rather than learning from what is necessarily *correct*. Given enough time though, a child eventually identifies patterns in their actions that ultimately lead to the ability to walk. Admittedly, there numerous other factors at play in this example, such as the child potentially observing intelligent behavior from other individuals in the environment, but the core concept remains.

Foundations of Reinforcement Learning

As my work is in the field of RL, I will now introduce various components and methodologies of traditional RL in detail. Figure 2 provides a high-level overview of the agent-environment interactions in RL. To contextualize this figure with the preceding example, one would consider the child to be the agent in the environment, which would be the world around said child. The state then is simply the observed condition of the world around the child at a given time step, and the action taken by the child is simply how the child interacted with the world given the state of the environment at the given time step. Finally, the reward could be considered to be distance traveled via bipedal means.

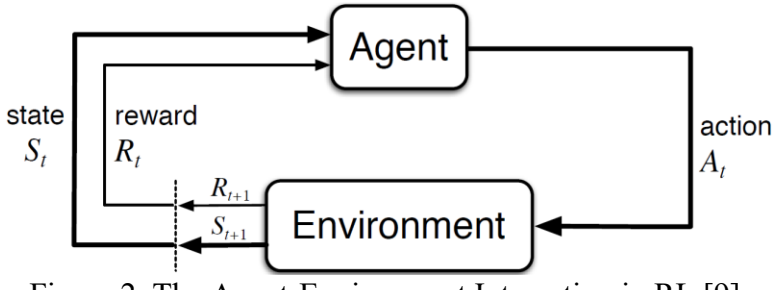

Figure 2. The Agent-Environment Interaction in RL [9].

Formally, the learner or decision-maker is called the agent, while everything outside the agent is referred to as the environment. Scenarios such as these are most commonly modeled to Markov Decision Processes (MDP) [9], [27], [28]. MDPs are non-deterministic search problems where the outcome of an action at a state is independent from preceding states and results in some numerical reward r and new state s' and can be represented as a 4-tuple $(state,\ action,\ reward,\ state')$, or, more concisely, $(s,\ a,\ r,\ s')$ [28]. For every given state s, the agent acts according to a policy π which is a mapping from state to probabilities of selecting each possible action, denoted as $\pi(a|s)$. RL methods define how this policy is updated based upon the agent's experience, and the goal overall is simply to maximize the expected return over some number of steps [9]. The utility or expected value of a state is defined to be sum of the discounted reward for each time step as shown in Equation 1, where γ is the discount rate which fulfills the condition $0 \leq \gamma \leq 1$, R is the reward received in the time steps following the current time step t, k is the number of time steps into the future being considered, and T is the terminal time step. Simply put, the discount rate determines the current value of future rewards, where the value of the future reward is only γ^{k-1} times what it would be immediately worth [9]. This process occurs at each time step until the MDP reaches a problem-specific terminal state, and the sequence of transitions from the initial state to the terminal state is referred to as an episode.

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \tag{1}$$

The large majority of RL methodologies depend upon value functions, defined to be functions of states or state-action pairs, to determine the expected utility or potential usefulness of being in a state or taking an action in a given state. These functions are defined using the

agent's policy, as well as the discounted reward calculation G_t shown in Equation 1. Equation 2 and Equation 3 show the state value and action value functions for policy π, respectively [9]. I note that the state value function is often referred to as the V value function, whereas the action value function is similarly referred to as the Q value function.

$$V^{\pi}(s) = E[G_t|S_t = s] \tag{2}$$

$$Q^{\pi}(s, a) = E[G_t|S_t = s, A_t = a] \tag{3}$$

The Q and V value functions may be approximated through agent experience. In the case of Monte-Carlo methods, an agent may keep an average received reward for each action in each visited state, which would result in Q converging to the true action-value for each action as the number of times that a state is visited approaches infinity. Similarly, V converges to the true state-value under the same conditions, without considering individual actions. Equation 4 provides a simple example of an every-visit Monte-Carlo state value function update, where α is the step-size parameter, often referred to in modern ML methods as the learning rate. As the number of states grows large though, Q and V are more efficiently defined as parameterized functions, where the parameters are tuned to reflect the actual values more closely in each case. The inherent exponential increase in the number of possible states relative to the number of state variables is referred to as the "curse of dimensionality" [29], and it is a problem that is solved to a large degree with the parameterized variants of the Q and V value functions. The introduction of parameterized Q and V value functions also brings to bear the introduction of a means of more efficiently guiding the parameters to the optimal state: experience replay. Experience replay is simply a methodology that stores the experiences that an agent obtains while interacting in the

environment in a data structure that is often called a replay memory. The experiences stored therein are then revisited periodically to improve the long-term stability of the agent and reduce the likelihood of an agent needing to revisit previously explored experiences in real-time. This helps prevent what is often referred to as catastrophic forgetting, which is the result of parameters being tuned in a way that is poorly representative of the experiences gained in earlier episodes due to the prevalence of potentially contrary more recent experiences [30].

$$V(S_t) = V(S_t) + \alpha[G_t - V(S_t)] \tag{4}$$

Some RL methodologies use a fourth component in addition to the states, actions, and rewards. This additional component is called the model of the environment, and it allows for inferences to be made regarding the ways in which the environment is expected to behave. In most model-based RL methodologies, the model comprises of an explicit transition and reward function, denoted as $T(s, a, s')$ and $R(s, a, s')$, respectively. The transition function T is logically equivalent to the probability of the state s' occurring, given the state-action pair (s, a) [28]. Methodologies such as these are generally referred to as dynamic programming (DP) methodologies, and they are unique from many other approaches in that they bootstrap their updates. That is, the updates to the value functions are determined in part by the current approximations of the value function in a future time step. This update process is shown in Equation 5 by using the Bellman equation [27] for V as an update rule, and it is called iterative policy evaluation [9]. In iterative policy evaluation, the value of a given state S_t is determined by the largest expected value at the state S_{t+1}, assuming that the optimal action is taken. Simply put, DP methods simulate the entire episode using the environmental model and specified reward

discount rate to identify the best course of action to take. Model-based methods like DP are not as widely used as their model-free counterparts, however, because of their dependence upon a perfect model of the environment, as well as their tremendous computational expense as the complexity of the environment grows large.

$$V(S_t) = \max_a E[R_{t+1} + \gamma V(S_{t+1}) | S_t = s, A_t = a] \quad (5)$$

One of the most powerful and novel foundational methodologies available in RL is temporal-difference (TD) learning, which combines the ideas of learning from raw experience as in Monte-Carlo methods and updating estimates in part on other estimates as in DP methods [9]. TD methods have the advantage over DP methods in that they are not dependent upon perfect understanding of the environment, and they have the advantage over Monte-Carlo methods in that they are able to update in an online manner without the need to wait for the full completion of an episode. Equation 6 demonstrates the state value function update method used in a simple single-step TD method, known as TD(0) [9]. In short, the most significant difference between the TD update method and the Monte-Carlo update method in Equation 4 is that the targets are $R_{t+1} + \gamma V(S_{t+1})$ and G_t, respectively.

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (6)$$

One of the challenges present in RL that is not present in other forms of ML is the need for a balance between exploration and exploitation. That is, RL agents must use the experiences that they have gained in their interactions within the environment to maximize the reward

received, but in order to exploit in this way they must first discover the actions that yield desirable rewards through exploration. There are two significant algorithms that are used in this regard: ε-greedy and softmax. These algorithms are often referred to as behavior policies because they are distinct from the agent target policy π, but they are used to define the way in which an agent behaves [9]. These behavioral policies may be used in both on- and off-policy RL algorithms. I note here that the difference between on- and off-policy algorithms is that on-policy algorithms update the action values by considering the action value of the next state given by the current policy, while off-policy algorithms update using a greedy approach to the considered action in the next state. In ε-greedy, the variable ε, which represents the probability of taking a uniformly selected random action in the environment, is initialized to a value between 0 and 1 and is then decayed over time to some specified minimum value. As is suggested by the "greedy" portion of the name, when the agent does not take a random action, it instead takes what is currently defined to be the optimal action as determined by the policy. The softmax behavioral policy, however, is more exploitative in nature in that it takes a random action where the likelihood of an action being taken is the proportion of the action value relative to the sum of the values of all other valid actions, though this still allows for exploration.

Traditional Reinforcement Learning Algorithms

With an understanding of the foundation of RL, it is now possible to introduce and discuss some of the most iconic traditional RL algorithms: Q-Learning [31], Sarsa [32], and Policy Gradients [33]. Each of these algorithms have been instrumental in the advancement of RL and are the inspiration or foundation of many currently state-of-the-art RL algorithms [14], [6], [4]. I note though that each of these algorithms in isolation scale poorly to MAS, due to the

inherent assumption of action value stationarity, which is the assumption that an action taken in the environment will yield a stationary reward.

Q-Learning. Q-Learning was introduced in 1989 by Watkins and Dayan, and is considered to be one of the most important breakthroughs in RL [31], [9]. Q-Learning is an off-policy TD control algorithm that effectively removed the explicit reliance upon the policy in the training process. The policy still determines the way in which the state-action pairs are visited, but the analysis and proof of convergence process is made significantly simpler without the explicit consideration of the policy itself. All that is required for convergence is the continued updates of the state-action pairs. Equation 7 shows the action value function update logic, and I note that it is remarkably similar to that of the original TD learning state value function update logic [9].

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{7}$$

Sarsa. Sarsa, short for State, Action, Reward, State, Action [32], [9], is a slightly modified version of Q-Learning. Sarsa is also one of few RL methodologies that use a 5-tuple in lieu of a 4-tuple, as it also considers the action in time step $t + 1$. Simply put, Sarsa is the on-policy equivalent of Q-Learning, which can be seen in Equation 8. Note that the only difference in the update logic is the target. Where Q-Learning uses the greedy action determined by $\max_a Q(S_{t+1}, a)$ in action selection, Sarsa chooses an action based upon the current policy.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{8}$$

Policy Gradients. Unlike Q-Learning and Sarsa, policy gradient methods are strictly associated with the parameterized state and action value functions, though I note again that both Q-Learning and Sarsa are capable of being implemented with such parameterization, with variants like DQN [6], [7] and deep Sarsa [34]. I note here that policy gradients as they are known today are improvements made upon the general REINFORCE category of RL algorithms introduced in [35]. The appeal of policy gradients is that they provide an inherent generalizability that was not offered in either Q-Learning or Sarsa. Both Q-Learning and Sarsa are tabular, meaning that they store state and/or action values given some state-action pair in a table, and then update the values for the respective pair in each visit. This is undesirable, as it is more challenging to make intelligent decisions on states that have not been visited before. As such, researchers explored the concept of state and action function approximators in the form of parameterized state and action value functions, which promotes generalizability. Often, policy gradient methods such as these simply treat the weights of a NN as the parameters being tuned by the parameterized state and action value functions. The gradient of the achieved rewards relative to these parameters is then calculated such that the changes made to the parameters result in the most rapid improvement in a policy's performance. With respect to the core logic of policy gradient methods, the only major change in the underlying traditional RL method is the way in which the policies are updated. Unlike the very simple update rules shown in Equation 7 and Equation 8, Equation 9 shows the more complicated calculation for the gradient of the parameters θ where ρ is a specified reward assignment calculation and d is the stationary distribution of states under the policy π, which is assumed to be independent of the initial state of all policies [33].

$$\frac{\partial \rho}{\partial \theta} = \sum_{s} d^{\pi}(s) \sum_{a} \frac{\partial \pi(s, a, \theta)}{\partial \theta} Q^{\pi}(s, a, \theta) \tag{9}$$

Extending Traditional Reinforcement Learning

Given the aforementioned foundations of RL, as well as the core algorithms that have been built upon them, it is important to consider how the boundaries of RL have been expanded by my predecessors in the RL research community, such as the expansion of Q-Learning into the deep learning space with DQN. The following chapter fulfils this role, as it considers the novel ways in which these foundational algorithms have been expanded upon in both single- and multi-agent systems.

RELATED WORK

Extensive studies have been performed on applying different variants of RL algorithms to controlling agents in both single-agent systems and cooperative and competitive MAS. The DQN method that improved upon Q-Learning has been further improved in recent years, with improvements such as stabilized memory replay with prioritized replay memory sampling which more efficiently tunes the network parameters by retraining on state-action pairs that are deemed valuable to the overall performance of the agent as shown by Schaul et al. in [36]. Concurrently, Nair et al. used parallel learning with a distributed network and replay memory to spread the burden of learning across multiple instances of a simulation, effectively increasing the rate at which agents learn [37]. Policy over-estimation reduction was introduced by Van Hasselt, Guez, and Silver with double DQN which strove to create more intelligent value estimations by decomposing the action selection into action selection and action evaluation [38], while dueling DQNs were introduced by Wang et al. to improve state value estimation via creative use of network architectures to allow for simultaneous updates of both state and action value function parameters [39]. Policy gradients, on the other hand, have been improved with higher sampling efficiency techniques introduced by Dong et al. in [40], leading to a decrease in training times by combining the benefits of model-free and model-based RL with a Gaussian process that creates a system model capable of generating supplementary off-policy samples to enrich the on-policy experience pool.

Konda et al. introduced the Actor-Critic (AC) algorithm that combines value-based and policy-based learning methods by utilizing the best features of both Q-Learning and policy gradients [41]. In this algorithm, the actor network is used to estimate the best action(s) to take

by approximating the action value function, whereas the critic network is used to approximate the state value function. Figure 3 shows this architecture graphically and illustrates that the critic is trained using TD and linear approximation and is then used to provide an approximate gradient to the actor during training. This method is often referred to as Stochastic Actor-Critic (SAC), as the policy is represented as a parametric probability distribution, where the parameters refer to the network weights [41].

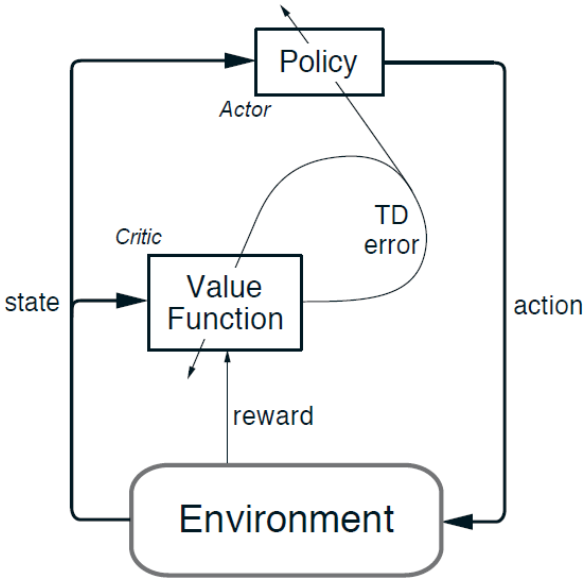

Figure 3. The Actor-Critic Architecture shown in [9].

This method has been adopted and improved widely in the DRL community, with numerous variants such as deep deterministic policy gradient (DDPG), which applied target network to expand the AC learning process into deterministic methodologies by utilizing a distinct stochastic behavior policy to approximate a true deterministic policy [42]. This is an important success in the field of RL, as a deterministic policy gradient was previously considered to be non-existent, at least without the use of the transition probability distribution, which is not considered in model-free algorithms such as AC [43]. DDPG and its variants have been shown to

yield considerable success in real-time strategy scenarios such as StarCraft [44]. Xie and Zhong expanded the DDPG algorithm into semi-centralized DDPG (SCDDPG) which utilized two-level AC structures to process local observations and global information to help the agents with interactions and cooperation in StarCraft combat [8]. Lowe et al. utilized DDPG in MARL scenarios and introduced Multi-Agent DDPG (MADDPG) [15] which showed promising results by developing cooperation and competition amongst agents via inter-agent policy approximation in the Grounded Communication environment [45].

In addition to the deterministic AC variants, Mnih et al. built upon SAC and proposed Advantage Actor-Critic (A2C), which improves the learning curve of SAC by considering the advantage of taking an action over another simply by considering the difference in the estimated reward calculated by the critic and the actual reward received at any given timestep [46]. In the same work, Mnih et al. also introduce Asynchronous Advantage Actor-Critic (A3C), which builds upon the benefits of A2C by running multiple instances of the simulation in parallel and then averaging the network weights across all instances to reduce the overall training time and increase the learning efficiency. Both of these variants have shown significant improvements over the standard SAC and many other traditional RL methods [46]. The AC family of algorithms has also been combined with Monte-Carlo Tree Search by Silver et al. showing remarkable success in AlphaGo [2], which was able to defeat 18-time Go world champion Lee Sedol, 4 games to 1, a feat which was previously believed to be at least a decade away [47]. Silver et al. then further built upon this success with AlphaZero, which learned only from self-play in lieu of human play, and outperformed AlphaGo in a matter of days [4]. AC algorithms have also been explored in tandem with centralized learning, decentralized execution techniques by Foerster et al. in StarCraft [14]. My work differs from Foerster et al.'s work in that I focus

primarily on representing the global information in an abstracted way and using robust network architectures to improve performance as opposed to providing the critic with direct global feature input, though I do compare the results of MAIDRL to that of a centralized learning, decentralized execution method. Furthermore, one of the primary contributions of Foerster et al.'s work is their definition of a counterfactual multi-agent baseline that aims to improve upon the way in which A2C learns by computing agent-specific advantages that consider the advantage of taking an action over a given default action [14].

In the MARL spectrum specifically, algorithms such as the Bidirectionally Coordinated Network (BiCNet), introduced by Peng et al. and shown in Figure 4, have shown that using a vectorized extension of the AC family can perform well with arbitrary numbers of agents being considered [48]. This work is interesting in that they use a centralized Bi-directional Recurrent Neural Network (BRNN) [49] in their AC architecture, which is then used to represent communication between the agents in the environment. Schulman et al. improved the way in which DRL models learn with the introduction of Trust Region Policy Optimization (TRPO) [50] in 2015 and Proximal Policy Optimization (PPO) [51] in 2017, and both of these methods have been shown to effectively reduce the variance in the results output by DRL models. PPO was then used by Berner et al. to train an agent capable of defeating top human players in the popular Multiplayer Online Battle Arena (MOBA) game Dota 2 [1] and by Wang and Vinyals to achieve GrandMaster rank in SC2 with the StarCraft Commander (SCC) [5] and AlphaStar [3] models, respectively. The research around the use and improvement of PPO is distinct from my work as PPO's focus is on improving the way in which models learn by clipping the gradient, while my focus is on state representation. Sarsa has been expanded into MAS with Parameter-Sharing Multi-Agent Gradient Descent Sarsa (PS-MAGDS) by Shao, Zhu, and Zhao in [52]. PS-

MAGDS utilized a centralized state representation and updated the network weights using parameterized Q value functions in lieu of the traditional Q value functions shown in Equation 8. This work, while focused on solving a similar micromanagement problem to my own, is considerably different in terms of state representation, reward calculation, and training methodology.

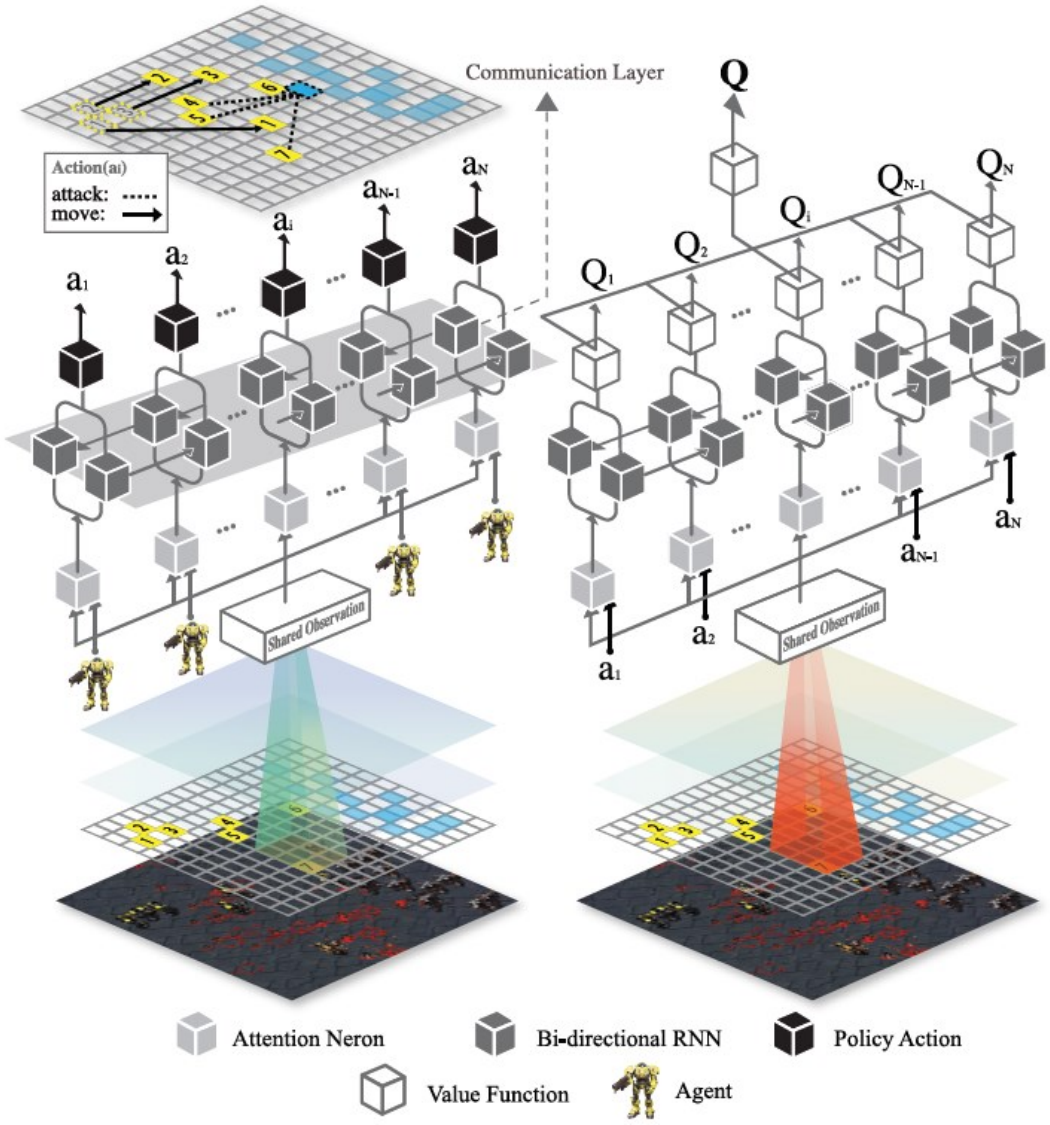

Figure 4. BiCNet Architecture Visualization [48].

There has also been research into several other RL methods that are distinct from DRL but can be supplementary. For example, Bain and Sammut introduced Behavioral Cloning (BC), also called Imitation Learning, which is the process of using supervised learning to learn a policy based upon a dataset of state-action pairs from expert replays [25]. BC has been shown to improve the performance of DRL methods when used as a warm-start in various complex environments such as Minecraft [53], [54] and SC2 [3], [5]. Reward Shaping, introduced by Ng, Harada, and Russell, allows agents to achieve intermediate rewards for accomplishing incremental steps towards goals that may not be inherently clear at a high level [13]. This method of rewarding agents for intermediate actions is widely used to encourage more rapid understanding of the objectives by agents and has been shown to be effective in various scenarios [14], [1]. Though there have been some incredible successes, Reward Shaping can often result in agents over-optimizing policies based upon the shaped reward in lieu of the sparse reward. This prompted Huong and Ontañòn to introduce an alternative solution called Action Guidance that handles sparse rewards without losing the benefit of sample efficiency that comes with reward shaping by using a main agent, whose objective is to learn the sparse reward function with the assistance of some auxiliary agent that learns from the shaped reward [55]. Relational Reinforcement Learning (RRL), inspired by Muggleton and De Raedt [56] and defined by Zambaldi et al. [57], is an interesting and unique approach to RL that aims to represent states, actions, and policies using a relational language, which improves the generalizability of said features. This method may become more widely utilized as the desire for more generally intelligent agents rises in game AI, though it is not currently used as widely as the other supplementary methods mentioned here.

Heuristic search can also be used in RL, with algorithms such as Monte-Carlo Tree Search, shown in Figure 5, and Portfolio Greedy Search showing promising results in various scenarios such as Silver et al.'s work in Go [2], [4], Churchill et al. and Liu, Louis, and Nicolescu's analysis in SC2 [58], [59], [60], and Churchill and Buro's large-scale combat in SC2 [61]. Sun et al. [62] and Pang et al. [63] demonstrate that a hierarchical structuring of simple tasks that make up a more complex task can yield positive results in highly complex environments by each defeating cheater-level built-in game AI in SC2. Skrynnik et al. [53] further demonstrate the capability of hierarchical structuring by earning the first-place position in the MineRL competition in 2020 [64]. A comparable, yet notably distinct, method was introduced by Lee et al., which described a modular architecture of models that allows for various subsets of the environmental complexities to be learned by dedicated models that convey their information to an overarching system has also shown promise in SC2 scenarios [65].

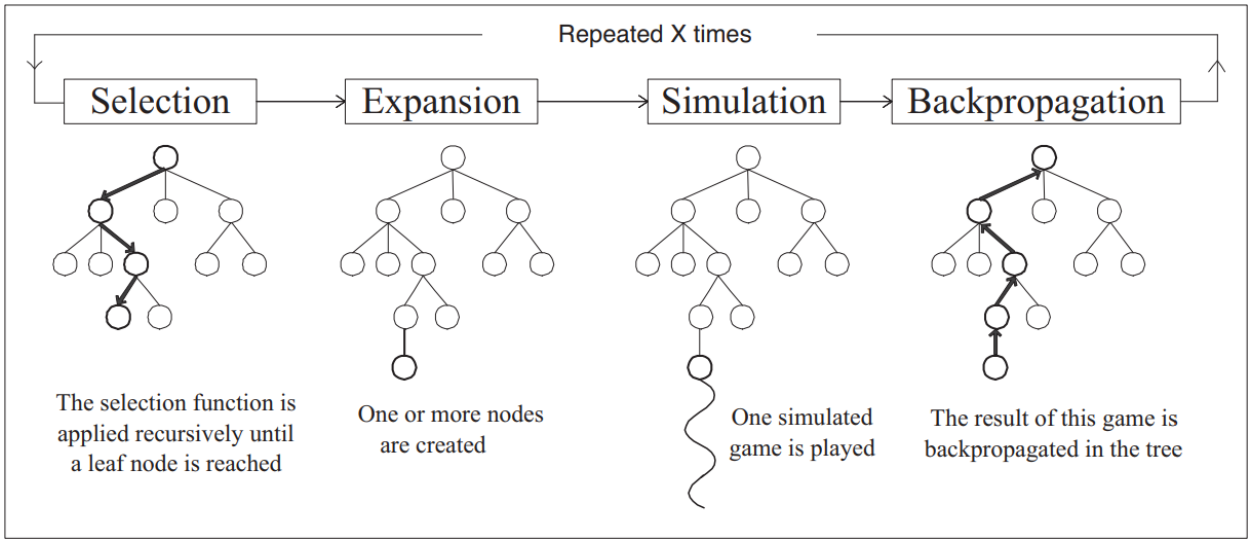

Figure 5. Outline of a Monte-Carlo Tree Search [66].

Of all of the work that I am aware of, the work that is most similar to mine with respect to my interpretation of agent influence maps is the work done by Liu, Louis, and Nicolescu in 2013 [59], [60]. In these works, Potential Fields (PF) and Influence Maps (IM) are defined using various unit parameters and are used to represent unit influence in the environment. A sample IM from their work is shown in Figure 6. Liu, Louis, and Nicolscu use PF to determine how the agents will navigate toward areas of interest that are identified by the IM. These works, however, focus on the use of genetic algorithms (GA) to determine the optimal IM and PF parameters to use, and the comparison of said GAs to heuristic search algorithms, whereas my work focuses solely on the use of DRL with a predefined IM definition based upon my own understanding of the environment.

Figure 6. Sample Influence Map Representation from Liu, Louis, and Nicolescu's Work [60].

SIMULATION ENVIRONMENT

The StarCraft Multi-Agent Challenge (SMAC) [22] environment was used as the MARL simulation environment. SMAC is built upon the StarCraft II (SC2) Learning Environment [12], and provides a set of multi-agent micromanagement scenarios where the objective is to defeat your opponents with the given units. Micromanagement is defined in this context as fine-grained control of individual units, whereas the logical inverse is macromanagement, which is defined as the high-level strategic considerations, such as resource management and control of multiple units simultaneously [22]. This difference is an important distinction to make, as my work focuses solely on the micromanagement of individual units who share a common goal, rather than the macromanagement of the army as a whole. That is, my work treats each agent in the environment as a unique entity whose actions are not bound to the actions determined by a group command.

Some sample SMAC scenarios can be seen in Figure 7, each of which are heterogeneous in nature. Scenarios such as those provided by SMAC can be represented with a Markov game, which is a multi-agent extension of Markov Decision Processes (MDPs) [48], [15], [14]. A Markov game with N agents comprises a set of states S that describe the properties of the agents and the environment, and a set of actions $A_1, A_2, \ldots, A_N$ and observations $O_1, O_2, \ldots, O_N$ for each of the N agents. Each of the agents treat the surrounding units as a part of their local information and perform an action in the environment based upon its own observation. $S \times A_1 \times \ldots \times A_N \rightarrow S'$ denotes the state transition from S to S' where each agent performs an action following a policy π in each environmental step.

Figure 7. Sample SMAC Scenarios [22].

There are two types of observation spaces in SMAC that I consider: local and global. The local observation space represents a fully decentralized perspective, where each agent has access to the information that it can observe in the local vicinity, while the global observation space is the fully centralized equivalent. The local observation is defined as the following attributes in the form of a one-dimensional vector for both allied and enemy units within sight range, which SMAC sets to 9: distance, relative x, relative y, health, and unit type. All aforementioned features are provided by the SMAC environment, where the distance is the Euclidean distance between the observing and the observed agents, the relative x and y are the directional difference with respect to the same pair of agents, and the health and unit type are of the observed agent. The global observation contains information regarding all units on the map with the relative positions to the center of the map along with all other features from the local observations, albeit from a global perspective. The global observation additionally contains the cooldown and previous action for each unit where the cooldown is representative of how long a unit must wait after attacking to be able to attack again. All features are normalized by their maximum values in both the local and global observation spaces. I do not simulate noise in the SMAC environment, and assume that the features provided are the true feature values. The actions that living units are allowed to take are a discretized subset of the full action set that is available in SC2. They are

move North, South, East, or West, attack an enemy by identifier, and stop. Units that have been defeated are limited to only the no-op action and are invisible to all surviving units. I note here that units are restricted on their attack action to agents that are within their shooting range, where each unit in SMAC has a shooting range set to 6 [22]. The SMAC definition of the sight and shooting range is shown in Figure 8, with the cyan and red circles representing sight and shooting, respectively.

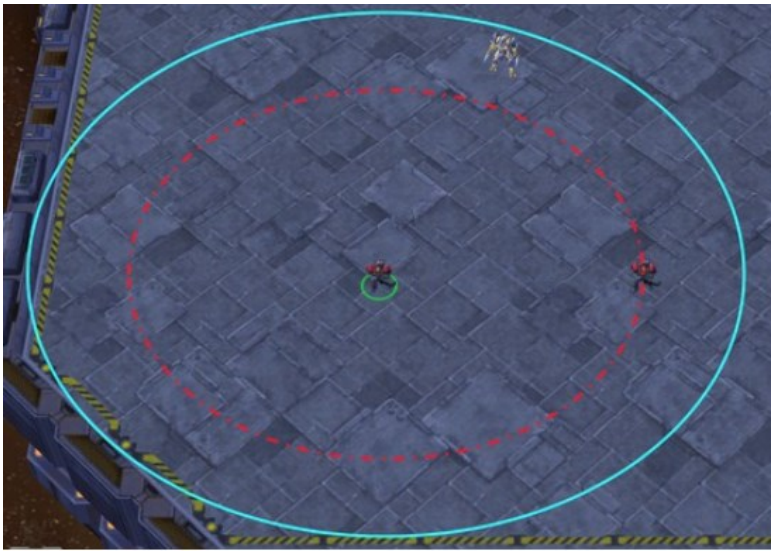

Figure 8. The SMAC Sight and Shooting Range, shown by the Cyan and Red Circles [22].

The reward achieved in each episode is scaled to be between 0 and 20, inclusive, as defined in SMAC [22]. The reward is a shaped reward that awards points for damage dealt to the enemy, bonus points on a kill, and a substantial bonus for victory. Rewards are not received by individual agents, rather, the rewards are accumulated by the allied forces as a whole and are aggregated into a shared pool as shown in Equation 10.

$$R = 20 \times \frac{\sum_{t=1}^{T}(\sum_{n=1}^{N}(D_n) \times 10k) + 200w}{R_{max}} \tag{10}$$

Equation 10 shows the reward calculation for a full episode of a game where t is the current environmental step, T is the terminal step, n is the agent identifier of an agent, N is the maximum number of agents, D_n is the damage dealt to enemy units by agent n in step t, k is the number of enemies that have been defeated in step t, R_{max} is the maximum possible reward amount, and w is 1 on a win and 0 on a loss. The total reward of an episode is calculated when the terminal state has been reached, and I then calculate the discounted rewards with a decay rate of 0.9, normalize the discounted values, and use the resulting reward per environmental step to train my models. I note here that, by default, each unit in my experimentation has 45 health points, which are translated to rewards when an allied agent deals damage to enemy agents.

SMAC provides scenarios that utilize units of each of the three races in StarCraft II. These races are Terran, Zerg, and Protoss. The Terran race is a human race that boasts emphasis on a balance between technological and biological units. The Zerg and Protoss races represent the extremes of biological and technological races, respectively, with the Zerg being an exclusively biological race with emphasis on evolution and metamorphosis, while the Protoss are very heavily technologically oriented, with minimal dependence upon biological units. My work focuses purely on the use of Marines, which are medium-ranged Terran infantry units. This focus on homogenous armies is largely due to the foundational nature of my work, and I note here that the introduction of heterogeneous armies is wholly supported by my methodologies, though this may require additional parameter tuning.

The SMAC combat scenarios that were used in training and evaluation are $3m$, $8m$, $25m$, and $8m\_vs\_9m$, shown in Figure 9, Figure 10, Figure 11, and Figure 12, respectively. I used the $8m$ scenario in SMAC as the baseline experimental scenario where each team controls eight marines to fight against each other. This scenario serves as a good foundation for my

research, as it has neither very large nor very small numbers of units, and it is representative of a medium-scale MARL scenario, while the other selected scenarios demonstrate the generalizability of the evaluated methods. Each of these scenarios comprises only Marines, with the number of Marines per team given in the name of the scenario. Scenarios with a single number are symmetric and the rest are asymmetric in favor of the built-in SC2 game AI, whose difficulty is set to 7 in SMAC, which is equivalent to Elite in SC2, which in turn poses a significant challenge to players that are unaware of the AI that they are combating, due to its aggressive, but predictable with context, nature. The objective of the agents in each scenario is to defeat the built-in SC2 game AI by eliminating all the enemies without losing all the allied troops. This task, while conceptually simple, requires a detailed understanding of the behaviors and capabilities of the agents on each team. Cooperation amongst allied agents is critical to the success of the team as a whole. Allied agents must work together and focus their fire on enemies to efficiently reduce the amount of damage received while maintaining damage dealt over time by minimizing sustained casualties.

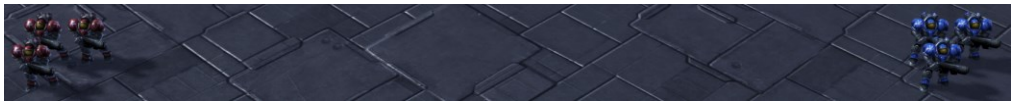

Figure 9. SMAC's 3*m* Scenario, with 3 Marines on Each Team.

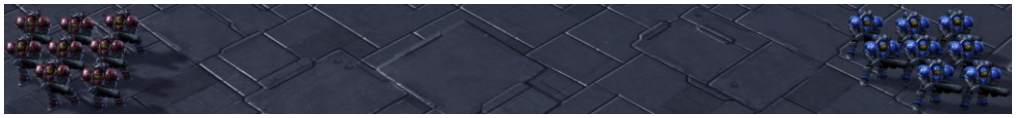

Figure 10. SMAC's 8*m* Scenario with 8 Marines on Each Team.

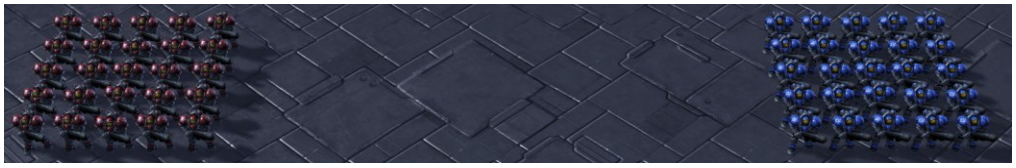

Figure 11. SMAC's 25*m* Scenario with 25 Marines on Each Team.

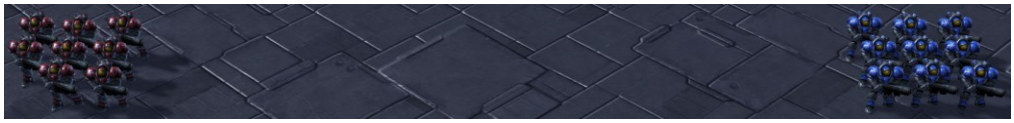

Figure 12. SMAC's 8*m*_vs_9*m* Scenario with 9 Marines Controlled by the Built-in SC2 Elite AI.

METHODOLOGY

All of my experiments were performed using Python 3.7.3 with NumPy (NP) 1.19.5 [67] and Tensorflow (TF) 2.4.0 [68] as the framework used to create and train all of my NNs. I used SMAC 1.0.0 [22] as my MARL environment, with SC2 version 4.10. To ensure that my results could be reliably reproduced, I refrained from using GPU acceleration and constrained the models to CPUs, seeding all random generators with values ranging from 1-32, corresponding to the experiment number relative to the specific methodology used in said experiment. While this did result in a moderate increase in the experimental completion time, particularly of experiments with NNs of larger parameter counts, it was deemed a worthwhile cost to ensure experimental reproducibility.

General Experimental Features

Each explored MARL method was executed in 32 independent instances with corresponding random seeds for TF, NP, and SMAC. All experiments are executed for a duration of 3,000 episodes in their respective combat scenarios. My experiments promote early exploration followed by incrementally increasing exploitation with a hybrid ε-greedy, softmax approach called ε-soft that initializes ε to 1.0 and diminishes it to 0.0001 over the course of 30,000 environmental steps. My implementation is considered ε-soft because when the greedy action would be taken in traditional ε-greedy, I instead select an action in accordance with the softmax behavioral policy which randomly selects an action, where the likelihood of selecting an action is the proportion of the action's estimated value relative to the sum of the estimated values for all actions. I chose this particular methodology because it promotes continued intelligent

exploration of the state space, even when ε itself becomes arbitrarily small. The value of ε at environmental step t can be calculated as shown in Equation 11 below, where ε_0 is 1.0, ε_{min} is 0.0001, and γ is 30,000. Table 1 shows some sample ε values at various environmental steps, to better demonstrate the values at various points.

$$\varepsilon_t = max\left(\varepsilon_0 - \frac{t \times \varepsilon_0}{\gamma}, \varepsilon_{min}\right) \tag{11}$$

Table 1. Sample ε Values at Various Environmental Steps.

Environmental Step (t)	Probability of Random Action (ε_t)
0	1.0000
1,000	0.9667
5,000	0.8333
10,000	0.6667
25,000	0.1667
30,000+	0.0001

In addition to ε-soft, I utilized A2C as the core learning algorithm in all of my experiments, with separate NNs for the actor and critic. The size of the inputs to the actor and critic varies between explored MARL methods and combat scenarios. The number of output neurons in the actors also depend upon the combat scenario, while the number of output neurons in the critics is always one. The activation function used in all layers except for the output layers is the Exponential Linear Unit (ELU) with $\alpha = 1.0$ [69]. The output layer of the actors used softmax, and the output layer of the critics used a simple linear activation. The actor and critic were compiled with losses categorical cross-entropy and mean squared error, respectively, and both use the *adam* optimizer with a learning rate of 0.00001 [70]. The use of batch normalization

and dropout layers was considered, though they were not used in the reported results due to a lack of sufficient improvement. Additionally, preliminary experimentation with Proximal Policy Optimization (PPO) [51] and experience replay both yielded a considerable decrease in overall performance compared to the standard A2C algorithm and are thus excluded from the final results.

I note here that I used TF's saved model format to save the actor and the critic so that they may be loaded back into memory and evaluated after the initial training process. The actor and critic are only saved when the running average reward achieved at the end of an episode is greater than the previous maximum, so the saved networks are representative of the best performers that were found for each combination of seed, method, and scenario.

Advantage Actor Critic (A2C)

In the A2C RL algorithm, there are two intuitive components: the actor and the critic. I utilized two separate NNs, one for each component, and each with the same hidden layer architecture. The hidden layer architecture used in my MAIRL and MAIDRL experiments are described in the Simple versus Dense A2C section. My A2C implementation represents a single controller that controls each agent individually, and the resulting shared parameters allow for more rapid and intelligent learning because the parameters are updated based upon the experiences of each agent.

The critic can be viewed as a value-based, model-free approximation of the state value function shown in Equation 2, which represents the expected reward to be returned by a given state based upon the policy. Let $V^*(s)$ be the optimal state value function, i.e., the state value function that returns the maximum reward. The goal of the critic is to make $V^{\pi}(s;\theta_c) \approx V^*(s)$

where θ_C represents the parameters of the critic. The critic parameters θ_C are updated using standard gradient ascent at the end of an episode, where the discounted reward values are used to perform a supervised update [46]. Because the critic is necessary to calculate the advantage to be used when training the actor, the critic update occurs after the advantage has been calculated.

The actor can be viewed as a value-based, model-free approximation of action value function shown in Equation 3, which represents the expected reward for action a given state s and following policy $\pi(a|s)$. Let $Q^*(s,a)$ be the optimal action value function, i.e., the action value function that returns the maximum reward. The goal of the actor is to make $Q^\pi(s,a;\theta_A) \approx Q^*(s,a)$ where θ_A represents the parameters of the actor. I note that, in accordance with the A2C algorithm, $Q^\pi(a_t,s_t;\theta_A) - V^\pi(s_t;\theta_C)$ is the target network representative of the advantage of taking an action over another, and it is determined in part by both the action value estimation determined by the actor and the state value estimation determined by the critic. At the end of an episode, the parameters θ_A of the actor are updated in the direction determined by the gradient ascent in Equation 12 [46].

$$\nabla_{\theta_A} log\, \pi(a_t|s_t)(Q^\pi(a_t,s_t;\theta_A) - V^\pi(s_t;\theta_C)) \tag{12}$$

Simple versus Dense A2C

Before considering the inclusion of the DenseNet-style model architecture, I explored the effectiveness of using a MAIM state representation with a NN that contained only a single hidden 1024-neuron layer. I used the $8m$ scenario in this experiment, noting that $8m$ was chosen as the baseline scenario due to its intermediate scale and difficulty. I improve this simple A2C by defining a DenseNet-style model architecture for both actor and critic [16].

Figure 13 demonstrates an example of a DenseNet-style grouping of layers which is defined to be an arbitrary number of n layers such that the output of every layer is input to each of the following layers in the group via concatenation. My network architectures contain two of such groups, each with five 128-neuron dense layers. The only additional layer in my DenseNet architecture is a 256-neuron dense layer that precedes the first DenseNet-style group of layers, immediately after the input layer.

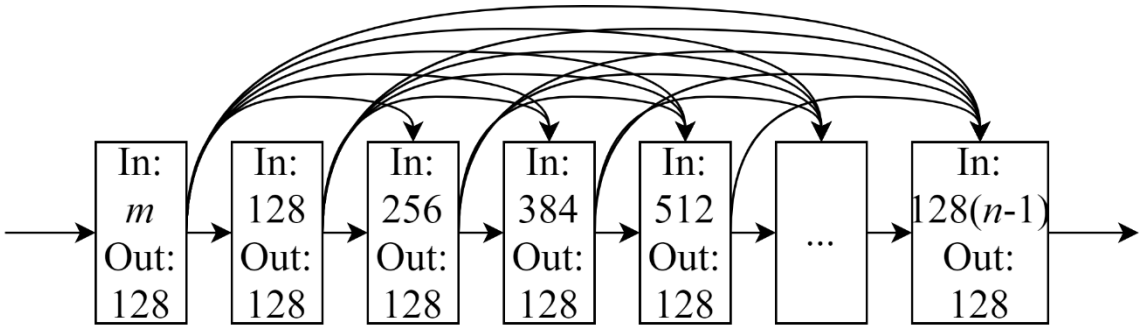

Figure 13. Example of a DenseNet-style grouping of n 128-neuron layers.

Centralized versus Decentralized

I apply A2C with centralized, decentralized, and hybrid centralized learning decentralized execution state representations to investigate how to effectively use global information for fine-grained decision-making in multi-agent environments. First, I apply A2C in with centralized state representations where each agent utilizes local and global observations to train the actor and critic networks for learning interactions among agents. Second, I similarly considered fully decentralized state representations, where agents only observed their local observations. These two methods are further used as the baseline for comparison with other algorithms. Third, I explored the option of a global critic with a local actor (GCLA), where the critic received the global information while the actor received the local observations of each agent. This particular

method is unique from the other methods in that the critic is arguably centralized but not with respect to each agent, while the actor remains entirely decentralized. Finally, I explored the option of total critic and a local actor (TCLA), where the critic is centralized, while the actor is decentralized. This method was inspired by the use of centralized training with decentralized execution as in the case of MADDPG [15].

Semi-centralized with Multi-Agent Influence Maps (MAIM)

Algorithm 1: Create and Update Agent Influence Map.

Result: Square agent influence matrix based on I_0, λ_I, and d_I, centered on the agent.

if *AIM not defined* **then**
| $N = 2 \times d_I + 1$
| $AIM = N \times N$ matrix of 0's
end
for $cell \in AIM$ **do**
| **if** $dist(cell, center) \leq d_I$ **then**
| | $AIM[cell] = \lambda_I \times I_0$
| **end**
end

In addition to the centralized, decentralized, and hybridized experiments, I propose a novel method to abstract the global features in such a way that is representative of how a human might interpret them. I define an Agent Influence Map (AIM) that is determined by three values: the source influence I_0, the influence decay rate λ_I, and the range of influence d_I for each unit in the environment. Algorithm 1 demonstrates the creation and update process of each AIM. Part of the appeal of this method is that it enables dynamic information representation with relative ease, simply by adjusting the values of I_0, λ_I, and d_I. For my experiments, I set d_I equal to the range of the agent as defined by SC2, I_0 equal to the current relative health of the agent as defined by

SMAC, and λ_I equal to the inverse of the distance from the agent, where a distance of 0 was assigned the value of I_0. To allow for a distinction between allied units and enemy units, I set I_0 of the allied units to the negative of the relative health provided by SMAC. Figure 14 demonstrates a sample 13×13 AIM in the form of a heatmap with the cell values shown to illustrate the rate at which agent influence decays.

0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.15	0.16	0.16	0.17	0.16	0.16	0.15	0.00	0.00	0.00
0.00	0.00	0.15	0.17	0.18	0.20	0.20	0.20	0.18	0.17	0.15	0.00	0.00
0.00	0.15	0.17	0.19	0.22	0.24	0.25	0.24	0.22	0.19	0.17	0.15	0.00
0.00	0.16	0.18	0.22	0.26	0.31	0.33	0.31	0.26	0.22	0.18	0.16	0.00
0.00	0.16	0.20	0.24	0.31	0.41	0.50	0.41	0.31	0.24	0.20	0.16	0.00
0.14	0.17	0.20	0.25	0.33	0.50	1.00	0.50	0.33	0.25	0.20	0.17	0.14
0.00	0.16	0.20	0.24	0.31	0.41	0.50	0.41	0.31	0.24	0.20	0.16	0.00
0.00	0.16	0.18	0.22	0.26	0.31	0.33	0.31	0.26	0.22	0.18	0.16	0.00
0.00	0.15	0.17	0.19	0.22	0.24	0.25	0.24	0.22	0.19	0.17	0.15	0.00
0.00	0.00	0.15	0.17	0.18	0.20	0.20	0.20	0.18	0.17	0.15	0.00	0.00
0.00	0.00	0.00	0.15	0.16	0.16	0.17	0.16	0.16	0.15	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00

Figure 14. Sample AIM Heatmap with Cell Values.

The AIM of each agent is aggregated into a Multi-Agent Influence Map (MAIM), where the AIM is simply added to the MAIM based upon the agents position in the environment. The AIM is positioned with the center over the agent position on the MAIM, and the overlapping cells are summed, ignoring any non-overlapping cells. The MAIM is a scaled representation of the units on the map in a SMAC scenario. For example, the $8m$ scenario has a map size of 32×32 units, but the MAIM dimensions can be any positive integers, and my implementation will scale the AIM to fit the MAIM's dimension while also maintaining the scale of the agents influence on $8m$, which maintains the environmental map aspect ratio relative to unit ranges. I

explored MAIMs of size 16×16, 32×32, and 64×64, each with the same AIM parameters, and each MAIM is flattened prior to propagation through the NNs.

Figure 15, Figure 16, and Figure 17 demonstrate three snapshots of MAIMs taken from a random agent versus the built-in SC2 AI at time steps 1, 6, and 17, respectively. Time step 1 was chosen to be displayed as it represents the first time step that the agents can take an action in the environment and it shows the uniform behavior of the built-in SC2 AI in contrast to the random AI. Time step 6 represents the first contact between the two armies, and time step 17 was arbitrarily chosen as it illustrates the change in the MAIM as units are defeated. Each of the figures show both the in-game state as well as the corresponding MAIM, with both the in-game state and corresponding MAIM cropped and scaled in the figures to allow for easier comparison, though I note that the MAIMs have not been adjusted horizontally and are representative of unit longitudinal positions. In both the in-game and MAIM representations, the built-in SC2 AI is represented by the red army, while the blue army is controlled by the random AI. The figures clearly show the built-in SC2 AI wholly overwhelms the random AI without losing a single agent. I note here that the built-in SC2 AI are scripted to move toward the spawn point of the blue army, so a conflict is almost guaranteed. This is reflected in the MAIM of each figure, as the blue influence region is shown to collectively move toward the red influence region's initial spawn point. An interesting phenomenon that is worth mentioning in this MAIM representation of the states is that there is a consistent visible conflict wall between competing influences, shown as the white line compressing otherwise circular regions of influence. This conflict wall may also be considered to represent the degree to which an influence source is overwhelming another, as this wall appears closer to the source of regions that are currently at a disadvantage,

as is shown when comparing the relative position of the wall in Figure 15, Figure 16, and Figure 17.

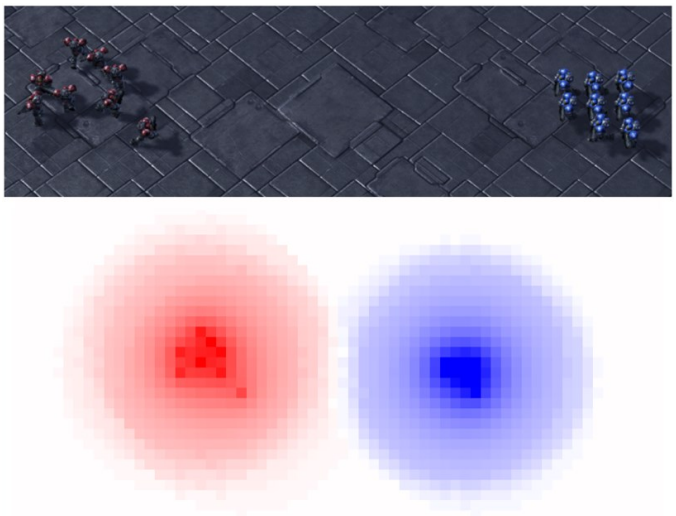

Figure 15. Sample MAIM and In-Game at Time Step 1.

Figure 16. Sample MAIM and In-Game at Time Step 6.

Figure 17. Sample MAIM and In-Game at Time Step 17.

RESULTS AND DISCUSSION

I analyze the results of the MARL methods used with three primary metrics: the average of the running average episode reward which I define to be the average reward of the most recent 50 episodes, the standard deviation of the running average episode reward, and the percentage of seeds that achieve a maximum running average reward with respect to various thresholds, all considering all 32 seeds. Henceforth, these metrics shall be referred to as overall performance, overall stability, and peak performance, respectively. I also perform more detailed evaluation regarding the peak performance per method across all seeds, considering the minimum, maximum, average, and standard deviation. That is, for each scenario and method, I consider the best performance of each seed, then find the minimum, maximum, average, and standard deviation with respect to the determined peak from each seed. The applicable tables contain the highest minimum, maximum, and average reward displayed in bold font, as well as the lowest standard deviation. I perform Welch's unequal variance t-test [71] on the average rewards achieved across all 32 seeds and compare MAIDRL to both the centralized and the decentralized methods in each of the considered scenarios. Finally, I discuss the learned behaviors that are observed for each MARL method when the trained actor of the best performing seed per MARL method is deployed in each scenario in a headed environment without any additional training.

Simple Architecture and MAIRL

Table 2, Figure 18, Figure 19, and Figure 20 display the results achieved by the use of the simple network architecture in the $8m$ scenario. It is evident that MAIRL considerably increased the overall and peak performance of the agents, as well as maintained comparable overall

stability to the centralized method, though the decentralized method was found to yield lower standard deviation in terms of peak performance across all seeds as shown in Table 2. Table 2 further shows that MAIRL wholly outperforms both the centralized and decentralized methods in terms of minimum, maximum, and average peak performance, with the 64×64 MAIRL variant achieving a 4.58 higher minimum, 5.03 higher maximum, and 6.18 higher average peak performance, all while maintaining a lower standard deviation than the centralized method. Figure 18 shows the overall performance of each method as well as the 90% confidence intervals based upon the standard deviations in Figure 19, and this illustrates that each of the MAIRL variants achieved an overall performance around or above 11, while the centralized and decentralized methods only achieved 8. MAIRL outperformed to such a degree that the upper bounds of the centralized and decentralized confidence intervals do not overlap with the lower of any of the MAIRL variants after episode 1300. The 64×64 MAIRL variant is shown in Figure 20 to be the only method to achieve running average rewards greater than 19, with over 15% of all seeds doing so. MAIRL is shown to be superior to both the centralized and decentralized methods, and I note that the performance of the decentralized method relative to the centralized is surprising, as it was expected to perform more poorly rather than almost identically.

Table 2. Peak Performance Across all Seeds with Simple Architecture with Best Result Per Scenario Bolded.

Scenario	Method	Min	Max	Avg	Std
$8m$	Centralized	5.31	14.55	9.95	2.79
	Decentralized	5.72	14.21	9.73	**2.15**
	64×64[1]	**10.30**	**19.58**	**16.13**	2.64
	32×32[1]	8.16	18.81	15.19	2.73
	16×16[1]	6.44	18.05	12.73	3.39

[1]MAIRL with given MAIM dimensions.

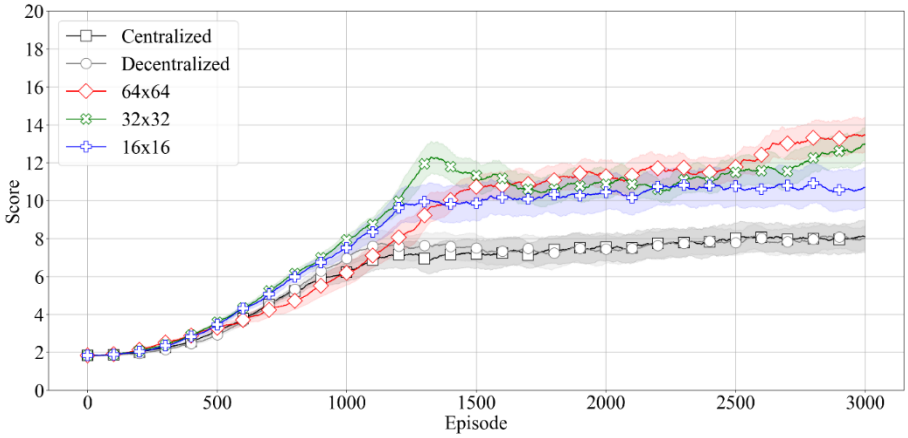

Figure 18. Overall Performance: Simple Architecture, $8m$.

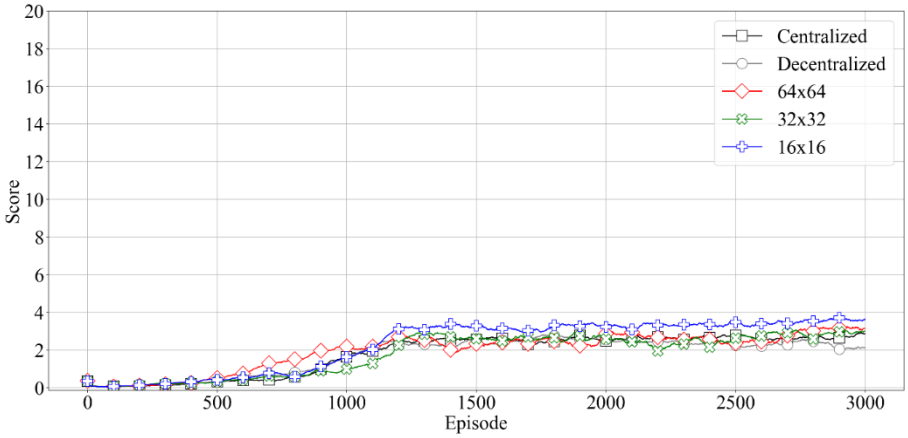

Figure 19. Overall Stability (SD): Simple Architecture, $8m$.

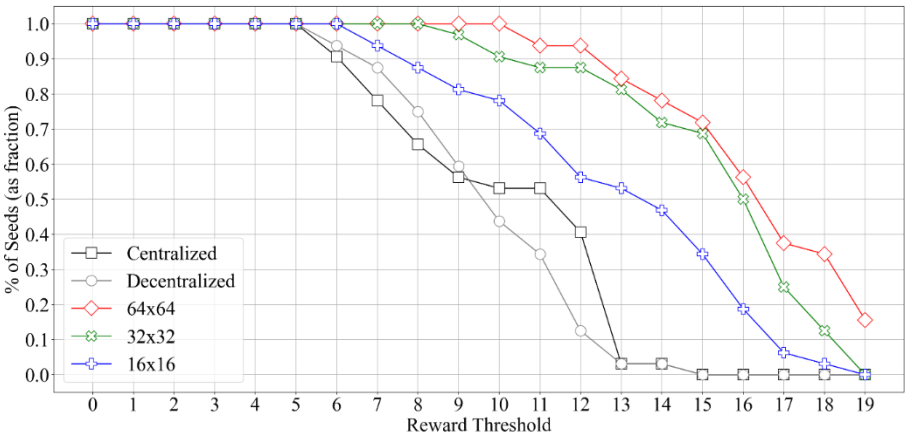

Figure 20. % of Seeds with Peak Performance ≥ Thresholds: Simple Architecture, $8m$.

DenseNet Architecture and MAIDRL

Due to the underwhelming performance of the centralized and decentralized methods relative to MAIRL with the simple network architecture, I considered ways to create more competitive baselines with which to form a basis of comparison for my proposed methods. I found that the introduction of a DenseNet-style model architecture considerably improved the performance of both the centralized and decentralized methods, while also demonstrating a notable improvement over the MAIRL early learning curve. Table 3, Figure 21, Figure 22, and Figure 23 present the peak performance across all seeds, overall performance across all seeds with 90% confidence intervals, standard deviation across all seeds, and percentage of seeds whose peak performance achieved various reward thresholds, respectively. In addition to the centralized, decentralized, and semi-centralized MAIDRL methods, they also contain the results of two hybridized methods, GCLA and TCLA. The following subsections present and analyze the results of each considered method with the DenseNet-style architecture.

Table 3. Peak Performance Across all Seeds: DenseNet Architecture – Initial Results with Best Result Per Scenario Bolded.

Scenario	Method	Min	Max	Avg	Std
$8m$	Centralized	3.24	**20.00**	14.20	4.22
	Decentralized	7.09	19.82	15.49	3.90
	GCLA	2.87	15.31	5.99	3.15
	TCLA	6.47	19.53	12.32	3.92
	64×64[1]	**10.35**	19.86	**16.84**	**2.99**
	32×32[1]	6.53	19.86	14.38	4.18
	16×16[1]	6.79	**20.00**	14.30	3.62

[1]MAIDRL with given MAIM dimensions.

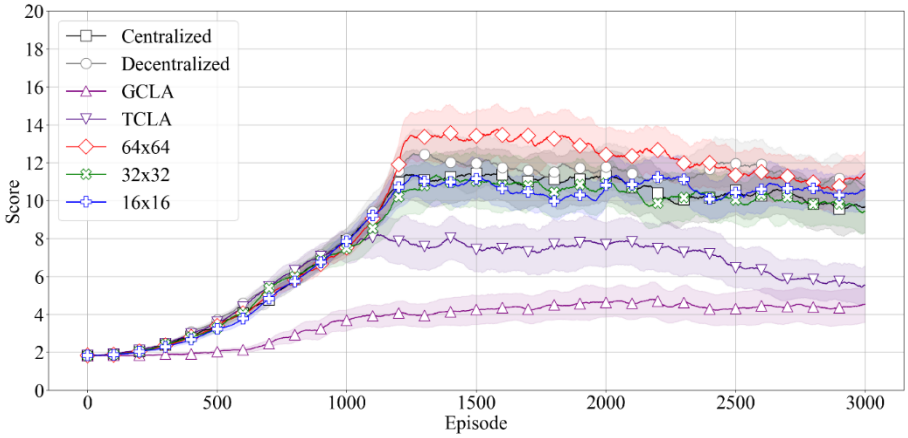

Figure 21. Overall Performance: DenseNet Architecture, $8m$.

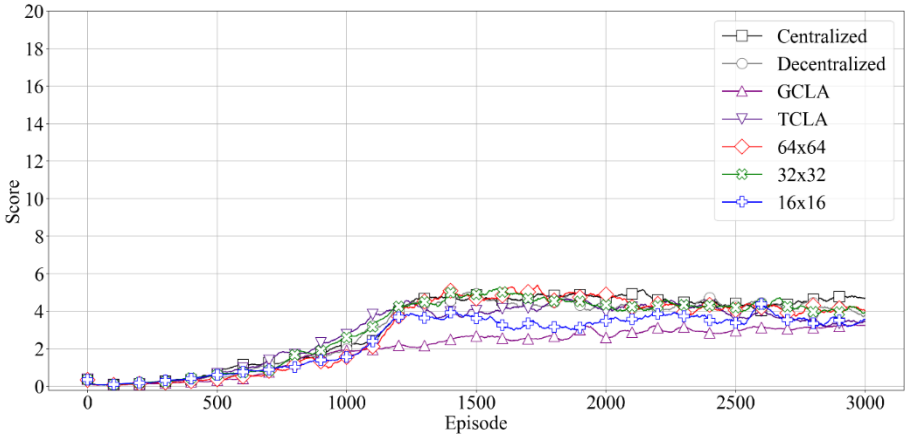

Figure 22. Overall Stability (SD): DenseNet Architecture, $8m$.

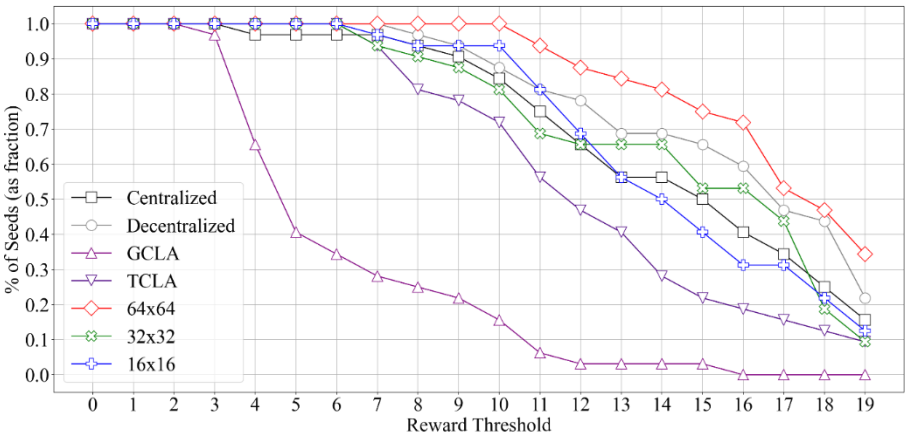

Figure 23. % of Seeds with Peak Performance ≥ Thresholds: DenseNet Architecture, $8m$.

Centralized. Table 3 shows that the introduction of the DenseNet-style architecture raised the peak performance across seeds 5.45 points to a perfect maximum of 20, while simultaneously improving the average peak performance across seeds to 14.20, a 4.25-point improvement. Additionally, Table 3 shows that the centralized method with the DenseNet architecture can achieve a perfect running average score, however, this perfect running average does not appear frequently enough to bring the average across all seeds up to the level of the 64×64 MAIDRL variant. Table 3 also shows that the baseline yields the highest standard deviation in terms of peak performance compared to the other methods, albeit by a very small margin in the case of the 32×32 MAIDRL variant. The centralized method is shown in Figure 21 to perform respectably overall as a baseline, achieving a consistent running average 3 points higher than the simple network equivalent and balancing out around 11. Figure 22 shows that the overall standard deviation balances out around 5, which is notably somewhat higher than the simple network architecture equivalent, but consistent with the results of the other considered DenseNet methods. Figure 23 shows that this method maintains a high percentage of seeds that can achieve running averages that are close to the upper limit, with almost 16% of all seeds achieving a maximum running average greater than 19. The results of the DenseNet-style architecture with the centralized method suggest that the use of such a structure yields better results overall, with a moderate increase in standard deviation compared to the single-layer architecture, though I argue that the general improvements in the other metrics outweigh this detriment. As such, the centralized DenseNet architecture is considered a baseline for comparison for each of the considered methods, and said methods are applied in tandem with the same architecture.

Decentralized. The decentralized method with the DenseNet architecture once again exceeded expectations and performed nearly identically to the centralized method. Table 3 shows that, in terms of peak performance, the decentralized method actually outperformed the centralized method in every regard with the exception of the maximum peak performance, where it fell short by only 0.18. Figure 21 then illustrates that, in terms of overall performance, the decentralized method maintains a competitive level of performance with even the 64×64 MAIDRL variant, being the only method to overcome the overall performance at any point after episode 1300, though I note that this only occurs after the 64×64 MAIDRL variant diminishes in performance. The overall stability is shown to be on par with each of the other considered methods as shown in Figure 22. The surprising performance of this method is further emphasized in Figure 23, where it is shown that over 20% of all seeds managed to achieve running average rewards greater than 19. This result is second only to the 64×64 MAIDRL variant, which it trails behind by just over 10%. The results of the decentralized method with the DenseNet architecture are consistent with that of the centralized method, and further support the use of said architecture in my experimentation.

GCLA. The GCLA method is shown in Figure 21 and Figure 23 to perform considerably worse with regard to both overall and peak performance, only achieving an overall performance of 4 and with no seeds managing to exceed a peak performance of 16 at any point. To its credit, GCLA did demonstrate the lowest standard deviation when compared to the other MARL methods, staying close to 2-3 for the majority of the episodes, though this is most likely due to the generally lower scores. The maximum and average of the peak performance achieved in any seed was considerably lower than that of the compared methods, as shown in Table 3, only achieving 15.31 and 5.99, respectively. Because the actor learns from the advantage, which is

determined in part by the critic, the lack of understanding of the local state for each agent on the part of the critic unsurprisingly leads to a poor performance overall. This hypothesis is supported by the results of the TCLA method which performed much better than GCLA, with the only change being the provision of the local information in addition to the global. Due to the apparent inability of the GCLA method to perform on a competitive level, I do not consider it in any further experiments to reduce the level of noise present in the reported results.

TCLA. The TCLA method, inspired by [15], is shown in Figure 21 to perform notably more poorly overall than the MAIDRL and centralized methods, peaking at a score around 8 at episode 1100, then steadily declining from there. TCLA performed somewhat comparably to the other methods with respect to the percentage of seeds at various thresholds, though it is consistently below all other curves for the majority of said thresholds, except for GCLA. Table 3 further demonstrates the results shown in Figure 23. The maximum of the peak performance across all seeds is 19.53, which is only 0.29 lower than the next highest and is a 4.22 improvement over the GCLA method, but the average of the maximum running average reward is only 12.32, which suggests that such a peak is less than common. The results show that, while TCLA can perform well in the $8m$ scenario, it fails to do so consistently. This too was somewhat surprising, given the results that were shown in [15]. While TCLA did demonstrate that it is capable of performing well, the overall performance was poor enough that it also is not considered in future results.

MAIDRL. When a DenseNet-style model architecture was applied in tandem with MAIRL, I found that the improvements were marginally smaller than those of the centralized method but were statistically significant enough to warrant a fundamental addition to MAIRL, which in turn defines MAIDRL. My semi-centralized MAIDRL method produced varying levels

of results in the $8m$ scenario depending upon the size of the MAIM. The 16×16 and 32×32 MAIM MAIDRL variants both performed very similarly to the centralized baseline on each of the three metrics that I consider, though they both fall just below the baseline in terms of peak performance greater than 19 as shown in Figure 23. The 64×64 MAIM significantly outperformed each of the other MARL methods that I tested, notably without a significant decrease in overall stability, with Figure 22 demonstrating a consistent standard deviation just below 4. Figure 21 demonstrates that the learning curve for the 64×64 MAIM grew to the highest point around 14 in a comparable amount of time as the other methods. While the steady decline thereafter is somewhat worrisome, I note that there is a distinct improvement with respect to the peak performance above a near-perfect threshold of 19, with a total of just over 34%. Furthermore, Table 3 demonstrates that each MAIDRL variant outperformed the centralized and decentralized baselines in every regard when considering the peak performance across all seeds, except for the maximum, where the 16×16 MAIM variant matched the perfect score of the baseline, while the others fell short by 0.14 points. The results of the various MAIDRL variants in the $8m$ scenario indicates that my semi-centralized MAIDRL consistently performs at least as well or better than the centralized baseline per episode, while the peak performance across all seeds of the MAIDRL methods wholly outperform said baselines.

Generalizability of MAIDRL

To evaluate the generalizability of my approach, I further applied the DenseNet architecture and MAIDRL on three new SMAC scenarios: $3m$, $25m$, and $8m\_vs\_9m$. Table 4 contains the peak performance across all seeds for each of these scenarios, with the best minimum, maximum, average, and standard deviation with respect to each scenario being

highlighted in bold, as with Table 3. The following subsections detail the results per method per scenario, and I note that only the centralized, decentralized, and MAIDRL methods are considered in this expanded list of scenarios, as they each exhibited the most promise.

Table 4. Peak Performance Across all Seeds: DenseNet Architecture – Extended Results with Best Result Per Scenario Bolded.

Scenario	Method	Min	Max	Avg	Std
$3m$	Centralized	4.00	18.79	14.41	4.19
	Decentralized	1.64	11.88	9.25	1.98
	64×64[1]	3.74	**18.83**	11.48	5.04
	32×32[1]	5.89	18.80	14.86	4.10
	16×16[1]	**12.42**	18.76	**16.61**	**1.87**
$25m$	Centralized	8.18	13.27	10.81	1.35
	Decentralized	**8.25**	13.37	**11.12**	1.62
	64×64[1]	7.20	12.73	10.08	1.50
	32×32[1]	5.34	**13.53**	9.82	1.77
	16×16[1]	7.97	13.20	10.26	**1.25**
$8m\_vs\_9m$	Centralized	**7.69**	10.68	**9.59**	**0.76**
	Decentralized	4.03	10.23	7.42	2.22
	64×64[1]	6.25	**10.95**	9.21	1.20
	32×32[1]	7.05	10.67	9.33	1.98
	16×16[1]	6.55	10.12	9.00	0.99

[1]MAIDRL with given MAIM dimensions.

$3m$. $3m$ is a very similar scenario to $8m$, but the unit count on each team is reduced to three. I chose this scenario to explore the applicability of MAIDRL in combat scenarios of varying complexity. In theory, $3m$ should be simpler to learn than $8m$, though the results shown in Table 4, Figure 24, Figure 25, and Figure 26 suggest that the disproportionate reward

associated with winning the scenario relative to the availability of incremental rewards may have led to poorer performance than expected overall. A consistent trend that was found amongst the baseline and each MAIDRL variant is that none of them seemed to accomplish a peak performance greater than 19. I hypothesize that this is due to the somewhat different score distribution in $3m$ as a result of the lower number of units on the field, noting again the reward calculation in Equation 10 and the fact that the reward achieved is an aggregate sum of damage dealt to enemies with a large bonus for winning.

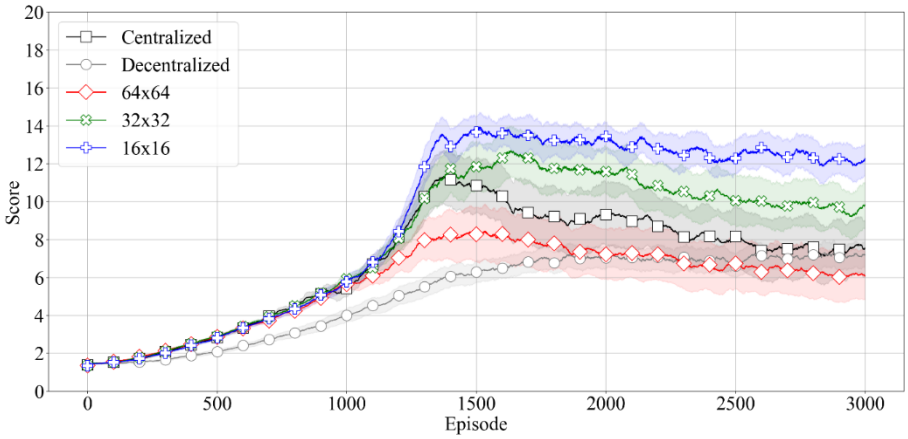

Figure 24. Overall Performance: DenseNet Architecture, $3m$.

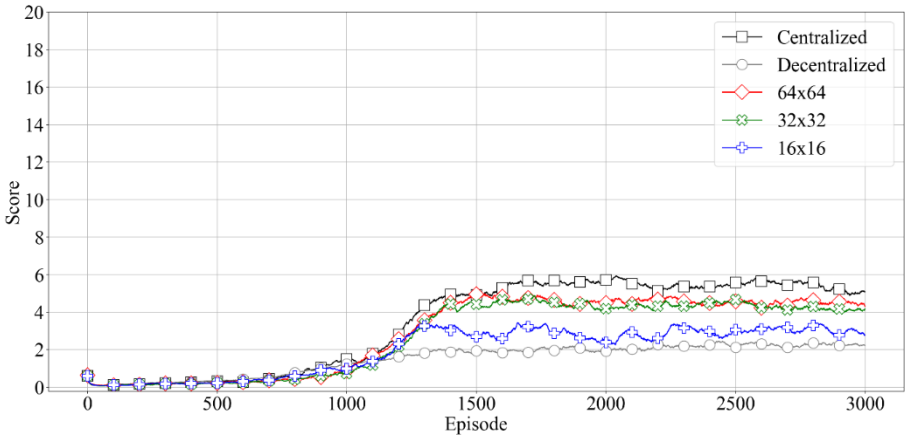

Figure 25. Overall Stability (SD): DenseNet Architecture, $3m$.

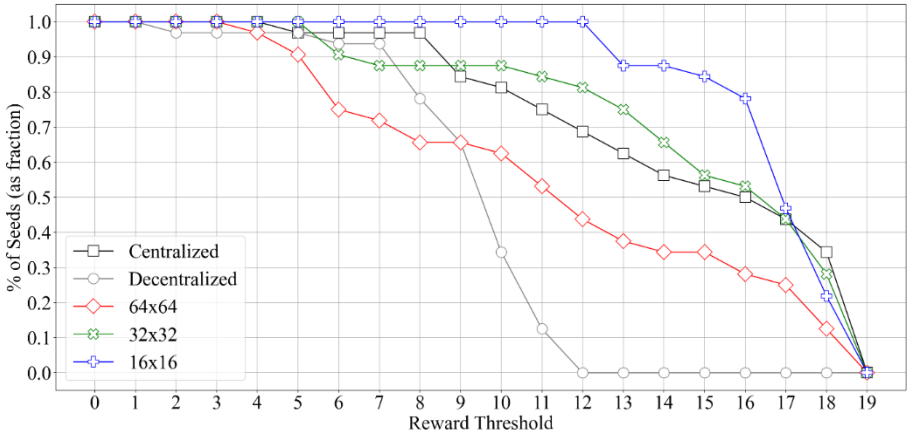

Figure 26. % of Seeds with Peak Performance ≥ Thresholds: DenseNet Architecture, $3m$.

Centralized. The centralized baseline performed almost exactly as well in $3m$ as it did in $8m$ with respect to the overall performance and stability, shown in Figure 24 and Figure 25, respectively. With a maximum overall performance around 11 and standard deviation leveling out near 6, the most concerning phenomenon that was recorded by the centralized baseline was the notable decrease in overall performance to 8 after reaching the initial maximum of 11. The percentage of seeds that achieved various peak performance thresholds also behaved similarly as in the $8m$ scenario, but with a notably more drastic decline at the highest threshold, decreasing from nearly 35% of seeds achieving peak performance above 18, to none above 19. Table 4 further illustrates that the centralized method performed consistently between $3m$ and $8m$, with the most significant difference being in the maximum of the peak performance across all seeds, with a reduction of 1.21 compared to the recorded $8m$ value.

Decentralized. The $3m$ scenario is the first scenario in which the decentralized method performed as poorly as I hypothesized it would throughout the duration of my experimentation. Table 4 shows an almost catastrophic failure of the decentralized method compared to both the results of this same method in the $8m$, as well as the other considered methods in the $3m$

scenario. Compared to its own $8m$ results, the decentralized method's average peak performance across all seeds decreased by 6.23, from a respectable and competitive 15.49 down to a meager 9.26, as shown in Table 3 and Table 4, respectively. This failure to compete is shown further in terms of overall and percentage of seeds achieving peak performance at various thresholds in Figure 24 and Figure 26, respectively, with the decentralized curves sagging far below the rest. In terms of overall stability, however, Figure 25 shows that the decentralized method maintained the lowest standard deviation of any of the other considered methods, though I note that this is most certainly due to the generally lower overall performance more than a significant show of improved stability.

MAIDRL. The MAIDRL variants performed much differently on $3m$ when compared to $8m$. In the $3m$ scenario, the 64×64 MAIM yielded the worst overall results, while it had previously shown to yield the best. Figure 25 shows that it maintained a standard deviation that was comparable to its own $8m$ performance, but the overall performance across seeds decreased considerably, as did the average peak performance per threshold, from 14 to 9 and 16.84 to 11.48, which are demonstrated in Figure 24 and Table 4, respectively. The 32×32 MAIM performed nearly identically to the $8m$ scenario, maintaining an overall performance around 12, then decaying over time to around 10. The most notable difference occurred in the 16×16 MAIM, which wholly outperformed the other variants, as well as the centralized method, achieving an overall performance of 14, with minimal decline thereafter. The lower bound of the 90% confidence interval consistently remained higher than the upper bound of the next highest confidence interval after episode 2500. It further improved with respect to the standard deviation, leveling out at only 2. This is markedly better than the other MAIDRL variants, and better still compared to the otherwise competitive centralized method, particularly when considering that it

simultaneously achieved the highest overall performance. Table 4 shows that the 16×16 variant also dominated in regard to the peak performance across all seeds, though it is noteworthy that Figure 26 suggests that this variant actually struggled more in terms of the very high thresholds greater than 18, while it maintained a solid lead up to that point. Interestingly, Table 4 also shows that the 64×64 variant achieved the highest peak overall, though less consistently than the other tested methods, which is reflected in the average of the peak performance across all seeds. This variance in results between MAIM resolutions and scenarios suggests that there may be room for future exploration into the matter, though it has little effect on my claim that MAIDRL as a methodology is statistically superior to the centralized and decentralized baselines.

$25m$. $25m$ represents one of the largest combat scenarios that SMAC offers, with 25 marines on either team. As such, it is a *considerably* more challenging scenario to master. I chose the $25m$ scenario because it is representative of large-scale combat. While $3m$ and $8m$ represent small- and medium-scale respectively, they are both significantly smaller and less complex than $25m$. The relative difficulty of $25m$ is illustrated throughout the results presented in Table 4, Figure 27, Figure 28, and Figure 29, though it is most clearly illustrated in Figure 29, where it is shown that none of the centralized, decentralized, nor the MAIDRL methods managed to achieve a peak performance greater than 14 in any of the 32 considered seeds. This is not particularly surprising, as the complexity of multi-agent scenarios increases considerably with more agents, though it does show that each method was at least able to make considerable progress toward solving this scenario.

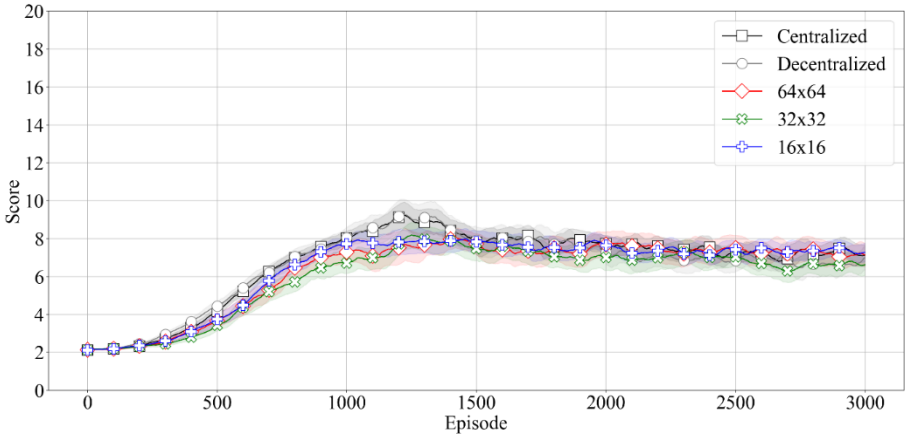

Figure 27. Overall Performance: DenseNet Architecture, $25m$.

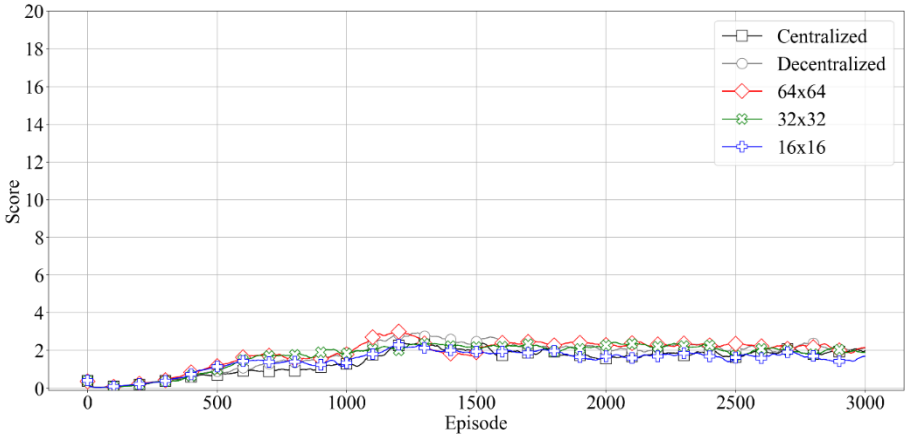

Figure 28. Overall Stability (SD): DenseNet Architecture, $25m$.

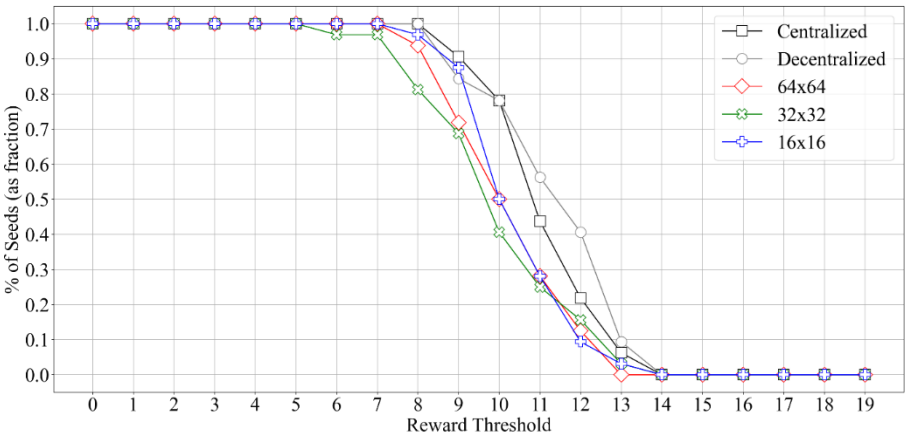

Figure 29. % of Seeds with Peak Performance ≥ Thresholds: DenseNet Architecture, $25m$.

Centralized. The centralized method again performed comparably to its performance in $8m$, though with a definite decrease in the maximum overall performance as shown in Figure 27. The standard deviation shown in Figure 28 demonstrated a notable improvement over the $8m$ and $3m$ scenarios, settling around 2 for the large majority of the episodes, though I note that this is most likely a direct result of the overall performance being consistently lower than the aforementioned scenarios. Interestingly, the centralized method does manage to set itself apart from my MAIDRL methods in Figure 29, where it maintained a consistently higher percentage of seeds capable of achieving the various considered reward thresholds.

Decentralized. As was the case with the $8m$ scenario, the decentralized method was surprisingly competitive in $25m$, achieving both the highest minimum and average peak performance as shown in Table 4 and falling only 0.16 short of the highest maximum. Figure 27 shows that the decentralized method manages to perform almost precisely like the centralized method, once again behaving contrary to my hypothesis regarding the expected performance. Furthermore, Figure 29 shows that the decentralized method actually outperforms the other considered methods in terms of percentage of seeds capable of achieving various reward thresholds, surpassing even the centralized method for thresholds between 11 and 14.

MAIDRL. The $25m$ scenario is the first scenario that I tested whose results are not immediately apparent with respect to whether my semi-centralized MAIDRL method is as performant or better than the centralized and decentralized baselines. Figure 27 and Figure 28 both seem to suggest that MAIDRL may be equivalent to said baselines, though Figure 29 shows that each MAIDRL variant fell below them in terms of peak performance. Table 4 shows that, when comparing the best performing seed of each method, MAIDRL variants still achieved the

highest maximum and lowest standard deviation values, though the decentralized baseline claimed the highest minimum and average.

$8m\_vs\_9m$. The $8m\_vs\_9m$ scenario was chosen because it presents an additional level of challenge to the $8m$ scenario. $8m\_vs\_9m$, unlike the symmetric scenarios I consider, gives a considerable advantage to the enemy forces, as they have an extra unit on their side. In order to win, a MARL system must demonstrate remarkable micro-management techniques and be able to wholly outperform the built-in SC2 game AI to an overwhelming degree. While success in this scenario is indicative of a superior methodology, failure is not necessarily indicative of a poor methodology. This scenario is used deliberately as an unfair challenge to the tested methodologies. This difficulty is reflected universally across each of the tested methods in Table 4, Figure 30, Figure 31, and Figure 32. Figure 32 clearly indicates that none of the centralized, decentralized, nor the MAIDRL methods managed to achieve a peak performance greater than 11, which is notably lower even than the $25m$ maximum of 14.

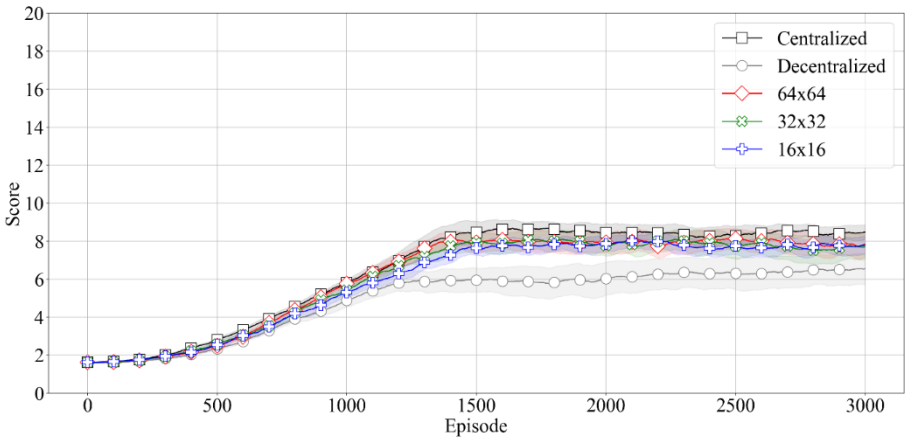

Figure 30. Overall Performance: DenseNet Architecture, $8m\_vs\_9m$.

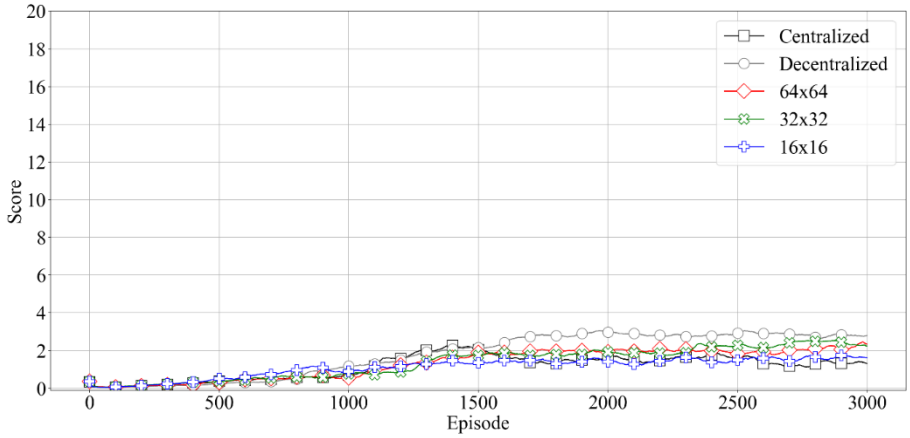

Figure 31. Overall Stability (SD): DenseNet Architecture, $8m\_vs\_9m$.

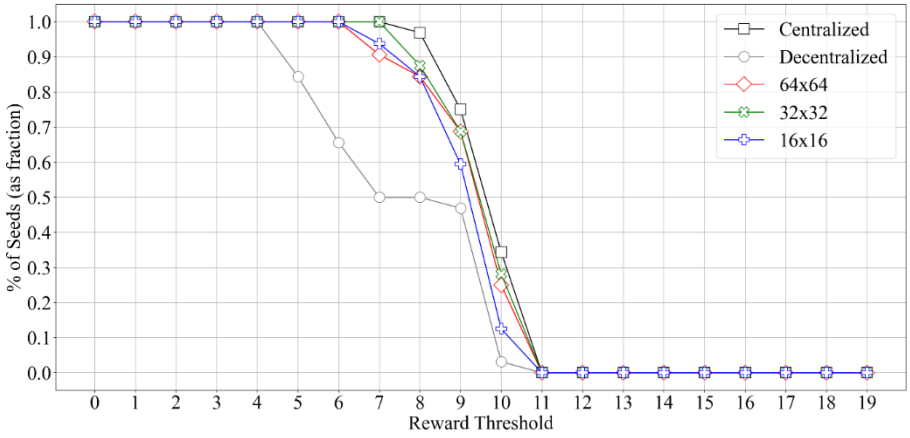

Figure 32. % of Seeds with Peak Performance ≥ Thresholds: DenseNet Architecture, $8m\_vs\_9m$.

Centralized. The centralized baseline performed comparably to the preceding experiments in the $8m\_vs\_9m$ scenario, though with definite decline in terms of overall performance, with the highest overall performance peaking near 9. As was the case in the $25m$ scenario, I note that the most significant decline in performance can be seen in regard to the peak performance per threshold in Figure 32. Table 4 shows that the centralized method was unable to achieve a running average reward greater than 11 in any of the seeds tested. This suggests that

the centralized method is incapable of wholly outperforming the built-in SC2 game AI, though it does perform admirably given the asymmetry of the scenario.

Decentralized. The decentralized baseline performed drastically worse than the other considered methods in every regard in this scenario, which is interesting as it suggests apparent volatility in performance across scenarios. It appears the decentralized method is as likely to perform comparably to the centralized method as it is to yield catastrophic failures. This is somewhat consistent with the hypothesized expectations of performance, though the potential of success appears to be greater than was previously understood, at least with respect to this specific environment and set of scenarios.

MAIDRL. While the performance of the MAIDRL variants also suffered as a whole in this scenario, they still performed very comparably to the centralized method, as shown in Figure 30, Figure 31, and Figure 32. In fact, each MAIDRL variant generally matches the centralized curve in each of the metrics considered. This suggests that, while MAIDRL was also incapable of wholly outperforming the built-in game AI, it did manage to maintain comparable performance to the centralized baseline. As with the $25m$ scenario though, it is not immediately apparent if MAIDRL can be considered equivalent or better than the centralized method in terms of overall performance without statistical analysis.

Statistical Analysis of the Mean Scores

Centralized versus MAIDRL. Table 5 contains the results of performing Welch's Unequal Variances t-test on the averages of the running average episode reward comparing the centralized baseline method to the MAIDRL variants, as well as the calculated statistical power of the test [71]. I use Welch's t-test in lieu of Student's because of the underlying assumption of

Student's t-test that the variances of the samples are equivalent [72]. I note here that the null hypothesis used in all of the t-tests in these comparisons is $H_0: \mu_{Centralized} \leq \mu_{MAIDRL}$, with alternative hypothesis $H_A: \mu_{Centralized} > \mu_{MAIDRL}$, and $\alpha = 0.05$ as the level of significance. The power of the test is representative of the probability that I would accept the alternative hypothesis if it were true.

Table 5. Welch's Unequal Variance t-test with Null Hypothesis $H_0: \mu_{Centralized} \leq \mu_{MAIDRL}$ where μ is the Average Episode Reward over all Episodes per Seed.

Scenario	Method	Test Statistic	P-Value	Power
$8m$	64×64[1]	-1.5045	0.9312	0.0001
	32×32[1]	0.2910	0.3860	0.1071
	16×16[1]	0.1013	0.4598	0.0662
$3m$	64×64[1]	1.6634	0.0507	0.7440
	32×32[1]	-1.9117	0.9697	0.0000
	16×16[1]	-4.4780	1.0000	0.0000
$25m$	64×64[1]	1.3098	0.0979	0.5663
	32×32[1]	2.5923	0.0071	0.9680
	16×16[1]	1.1923	0.1189	0.5018
$8m\_vs\_9m$	64×64[1]	1.2604	0.1062	0.5393
	32×32[1]	1.7636	0.0414	0.7865
	16×16[1]	2.4109	0.0095	0.9544

[1]MAIDRL with given MAIM dimensions.

From the table, it is clear that the overall performance of each MAIDRL variant is shown statistically to be greater than or equal to that of the centralized baseline in $8m$, with each p-value much larger than any common level of significance, albeit with small powers.

Furthermore, this trend continues in the $3m$ scenario, with even the poorly performing 64×64 MAIDRL variant demonstrating statistical significance. The $25m$ scenario shows that the 64×64 and 16×16 MAIDRL variants tested fulfil this same condition, while the 32×32 variant definitively does not come close to my selected level of significance, though I note that it is close to the commonly selected 0.01. Finally, the $8m\_vs\_9m$ scenario shows that the 64×64 MAIDRL variant fulfills the condition for statistical significance, the 32×32 variant comes close to my selected level of significance, and the 16×16 variant is very close to the common 0.01 level of significance. When considering this statistical analysis, I claim that my semi-centralized MAIDRL method is statistically shown to be capable of matching or outperforming the overall performance of a centralized alternative in MAS of varying complexities.

Decentralized versus MAIDRL. Similar to Table 5, Table 6 contains the results of performing Welch's Unequal Variances t-test on the averages of the running average episode reward, but Table 6 compares the decentralized baseline method to the MAIDRL variants [71]. I note that the null hypothesis in this case is $H_0: \mu_{Decentralized} \leq \mu_{MAIDRL}$, with alternative hypothesis $H_A: \mu_{Decentralized} > \mu_{MAIDRL}$, and $\alpha = 0.05$ as the level of significance to maintain consistency.

Table 6 shows that, even though the decentralized method did manage to perform much better than was expected, it did not perform well enough to reject the null hypothesis in almost all of the considered comparisons. The only case where the decentralized method is shown to be statistically superior to MAIDRL is in the $25m$ scenario. It is noteworthy that in every case where I do not reject the null hypothesis, the calculated p-values are at least twice as large as my selected level of significance, which further supports my claim that MAIDRL is as performant or better than a competitive decentralized baseline.

Table 6. Welch's Unequal Variance t-test with Null Hypothesis $H_0: \mu_{Decentralized} \leq \mu_{MAIDRL}$ where μ is the Average Episode Reward over all Episodes per Seed.

Scenario	Method	Test Statistic	P-Value	Power
$8m$	64×64[1]	-0.5555	0.7097	0.0079
	32×32[1]	1.2752	0.1035	0.5474
	16×16[1]	1.2173	0.1142	0.5156
$3m$	64×64[1]	-1.2629	0.8935	0.0003
	32×32[1]	-6.2435	1.0000	0.0000
	16×16[1]	-12.4153	1.0000	0.0000
$25m$	64×64[1]	0.8484	0.1998	0.3187
	32×32[1]	1.8915	0.0316	0.8342
	16×16[1]	0.5822	0.2814	0.2006
$8m\_vs\_9m$	64×64[1]	-3.4215	0.9994	0.0000
	32×32[1]	-3.2289	0.9989	0.0000
	16×16[1]	-2.9933	0.9978	0.0000

[1]MAIDRL with given MAIM dimensions.

Discussion of Learned Behavior

While the presented results so far are purely quantitative in nature, it is of interest to consider the qualitative perspective of the results as well. To that end, I identified the best performing network from each considered method in each scenario and simulated 10 episodes in a headed environment using only the trained controller. I found that there were two primary strategic components prevalent throughout the observed episodes: prioritized collaborative enemy targeting and unhindered unit positioning. Generally, controllers that prioritized agent targeting to focus fire on enemy units were victorious much more often than when agent targeting was not focused. Additionally, controllers that successfully positioned agents in such a

way that all the agents were able to start dealing damage as early as possible with minimal extraneous repositioning tended to be successful. The following subsections provide scenario-specific observations and observations of actors that behaved interestingly compared to others. I found that controllers that learned both prioritized collaborative enemy targeting and efficient agent placement tended to perform better than those that demonstrated one or neither.

$\underline{8m.}$ I considered both the simple network architecture controllers and the DenseNet architecture actors in the $8m$ scenario during this evaluation. Comparing the two architectures generally, I observed that the DenseNet architecture controllers tended to demonstrate a clearer understanding of the scenario and the agents behaved more cohesively overall. Another general observation was that the controllers that focused agent fire on a maximum of three enemy units at a time tended to be more successful. The decentralized controllers were observed to sometimes struggle to target enemy units when the number of enemies was near zero. This is surprising, as the surviving agents were close enough the remaining units to observe them with the local observations, but still took several time steps to target and eliminate them. GCLA, on the other hand, demonstrated an entirely unique strategy from any of the other methods. While all other methods moved the controlled army toward the opposing army at the start of each episode, GCLA moved the opposite direction and appeared to wait for the SC2 AI army to come within firing range. This strategy resulted in poorer agent positioning, however, and ultimately was unsuccessful a large majority of the time. The 64×64 MAIDRL variant demonstrated several episodes of overwhelming the SC2 AI, as shown in Figure 33, where there are 5 remaining controlled agents and only a single SC2 AI unit, which I note was defeated before any more MAIDRL agents were lost.

Figure 33. MAIDRL Wholly Overwhelming the Built-in SC2 AI on $8m$.

$\underline{3m.}$ The $3m$ observations provided much unexpected insight into the importance of unit positioning. While the controllers from each method generally demonstrated an ability to focus fire on an individual enemy unit, the most notable change in end-results was apparent when observing unit positioning. Due to the smaller number of units in either army, the feat of focusing fire had the potential to happen simply because of only a single agent at a time being within firing range. As a result, when the controllers managed to arrange the agents in such a way that it prompted the SC2 AI to spread their fire across multiple agents, victory was achieved almost every time. However, when the controllers did not manage to accomplish positioning the units in this way, the outcome of the skirmish was largely up to chance. I observed an instance where both armies were reduced to a single marine with 3 hit points going into the last time step of the episode, but the SC2 AI managed to deal damage before the controlled agent, resulting in a loss. This behavior helps explain the apparent lackluster performance presented in the quantitative results as well because near victories such as these are not awarded the winning bonus in the reward calculation. The only significantly different behavior occurred when observing the decentralized method's controller, which appeared to be as likely to move the controlled agents in apparently random directions as it was to cohesively move the agents toward

the enemy force at the start of an episode. Figure 34 illustrates the first actions taken in the environment by the controller trained using the decentralized method, and the agents are not being directed toward the enemy's initial position. If anything, this suggests a poor understanding of the environment, which is not unexpected given the fully decentralized nature of the relevant training.

Figure 34. Example of Apparently Random Agent Positioning using Decentralized Method.

$\underline{25m.}$ A general observation that was made in the $25m$ scenario is a general inability of the considered methods to position the controlled agents as efficiently as the SC2 AI. The centralized method did demonstrate the ability to adjust frontline agent positioning to allow agents behind them to get within firing range, but it did not do so consistently or efficiently enough to significantly change the course of the battle. An interesting behavior that was observed in all episodes regardless of the controller is the initial positioning of the units closest to the enemy army. In almost every episode, the 8-10 agents that spawned closest to the enemy army are positioned within the enemy firing range before they collectively start shooting. This results in the enemies having time to deal considerable damage to the RL-controlled forces before the controller issues an attack command. Another interesting observation was made with respect to the positioning of units on the skirmish line. Figure 35 shows the skirmish line in a sample episode with the currently repositioning units highlighted by a yellow box. This makes it

clear to see that the actor is less efficient than the SC2 AI at positioning units in such a way that they can contribute to the damage being dealt to the enemies. This was observed in every considered methodology and was notably more exaggerated when using a controller that demonstrated advanced agent movement cohesion, such as the 64 × 64 MAIDRL variant, which was the controller in Figure 35. Interestingly, this behavior is lessened in severity when using the decentralized controller, as the agents tend to move less as a cohesive unit and more individually to create a larger skirmish line.

Figure 35. Comparison of Traveling Units in $25m$.

$\underline{8m\ vs\ 9m.}$ None of the observed episodes from any method managed to defeat the SC2 AI with the unfair advantage of an extra unit, though the 32 × 32 MAIDRL variant did come very close in multiple episodes. This variant managed to reduce the enemy army down to a single unit with just under half health remaining, and it demonstrated impressive coordination in

both prioritized targeting and unit positioning, but it still did not quite manage to overcome the disadvantage. Figure 36 illustrates one such very close episode. After observing the $8m\_vs\_9m$ episodes, it seems to be the case that the strategy that must be learned is that of using higher hit point agents as a shield to protect low hit point agents to ensure that as much damage is being dealt to the enemy as possible for as much time as possible.

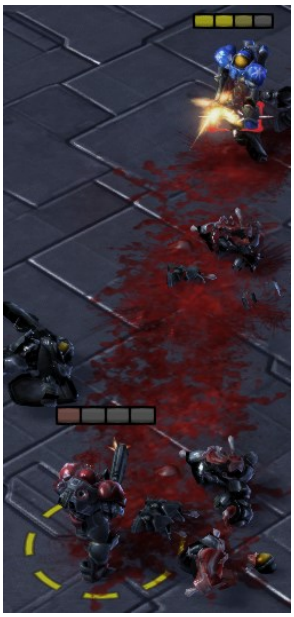

Figure 36. Nearest Achieved State to $8m\_vs\_9m$ Victory – 21 Remaining Enemy Health Points.

CONCLUSION AND FUTURE WORK

In this thesis, I introduced MAIRL and MAIDRL, novel semi-centralized methodologies to solve various MARL scenarios using abstracted feature information in the form of MAIMs and a robust model architecture inspired by DenseNet. MAIDRL was demonstrated in SMAC combat scenarios of varying complexity to be statistically capable of matching or bettering the overall performance of comparable centralized and decentralized baselines, even in scenarios with large or unfair numbers of agents. MAIDRL also demonstrated significant improvements in the overall performance, overall stability, and peak performance shown in some of said scenarios, with MAIDRL variants being the top performers by a large degree in both small- and medium-scale MARL scenarios.

While I did not demonstrate full comprehension of all scenarios by MAIDRL, there are several aspects can be improved in future work. First, a logical next step would be to incorporate the use of convolutional neural networks (CNN) in my network architectures. The MAIM representation of features lends itself very well to CNNs and would allow for a more intelligent interpretation of the MAIM by the networks. Second, the use of multiple MAIMs to represent various features would likely improve agent understanding of influence in the environment. I focused on the relative health of the unit as described in the Simple Architecture and MAIRL section, but there are other features that could be used as well, e.g., cooldown or possible damage per second. Building on these two potential improvements, the application of a 3D CNN that might learn the relationship between each of the feature MAIMs would also be interesting. Finally, there is need to investigate the generalizability of MAIDRL in heterogeneous scenarios.

I demonstrated generalizability in a range of homogeneous scenarios, but I acknowledge that the application of MAIDRL in heterogeneous scenarios is a necessary consideration for future work.

REFERENCES

[1] C. Berner *et al*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680,* 2019.

[2] D. Silver *et al*, "Mastering the game of Go with deep neural networks and tree search," *Nature,* vol. 529, p. 484–489, 2016.

[3] O. Vinyals *et al*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature,* vol. 575, p. 350–354, 2019.

[4] D. Silver *et al*, "A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play," *Science,* vol. 362, p. 1140–1144, 2018.

[5] X. Wang *et al*, "SCC: an efficient deep reinforcement learning agent mastering the game of StarCraft II," *arXiv preprint arXiv:2012.13169,* 2020.

[6] V. Mnih *et al*, "Human-level control through deep reinforcement learning," *Nature,* vol. 518, p. 529–533, 2015.

[7] V. Mnih *et al*, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602,* 2013.

[8] D. Xie and X. Zhong, "Semicentralized deep deterministic policy gradient in cooperative StarCraft games," *IEEE Transactions on Neural Networks and Learning Systems,* 2020.

[9] R. S. Sutton and A. G. Barto, Reinforcement Learning: an introduction, Cambridge, MA: MIT Press, 2011.

[10] M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling, "The Arcade Learning Environment: an evaluation platform for general agents," *Journal of Artificial Intelligence Research,* vol. 47, p. 253–279, 2013.

[11] M. Johnson, K. Hofmann, T. Hutton and D. Bignell, "The Malmo platform for artificial intelligence experimentation," in *IJCAI*, 2016.

[12] O. Vinyals *et al*, "StarCraft II: a new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782,* 2017.

[13] A. Y. Ng, D. Harada and S. Russell, "Policy invariance under reward transformations: theory and application to reward shaping," in *ICML*, 1999.

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

2018.

[15] R. Lowe *et al*, "Multi-agent Actor-Critic for mixed cooperative-competitive environments," *arXiv preprint arXiv:1706.02275,* 2017.

[16] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[17] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[18] K. Zhang, Y. Guo, X. Wang, J. Yuan and Q. Ding, "Multiple feature reweight DenseNet for image classification," *IEEE Access,* vol. 7, p. 9872–9880, 2019.

[19] F. Iandola *et al*, "Densenet: implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869,* 2014.

[20] Y. Zhu and S. Newsam, "DenseNet for dense flow," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017.

[21] M. Zeng and N. Xiao, "Effective combination of DenseNet and BiLSTM for keyword spotting," *IEEE Access,* vol. 7, p. 10767–10775, 2019.

[22] M. Samvelyan *et al*, "The StarCraft multi-agent challenge," *CoRR,* vol. abs/1902.04043, 2019.

[23] M. Mohri, A. Rostamizadeh and A. Talwalkar, Foundations of machine learning, MIT Press, 2018.

[24] M. I. Jordan and T. M. Mitchell, "Machine learning: trends, perspectives, and prospects," *Science,* vol. 349, p. 255–260, 2015.

[25] M. Bain and C. Sammut, "A framework for behavioural cloning," in *Machine Intelligence 15*, 1995.

[26] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics,* vol. 20, p. 53–65, 1987.

[27] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics,* vol. 6, p. 679–684, 1957.

[28] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

[29] R. Bellman, "Dynamic programming and Lagrange multipliers," *Proceedings of the*

National Academy of Sciences of the United States of America, vol. 42, p. 767, 1956.

[30] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences,* vol. 3, 1999.

[31] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning,* vol. 8, p. 279–292, 1992.

[32] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37, University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[33] S. SuttoRichard, D. A. McAllester, S. P. Singh and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Neural Information Processing Systems*, 1999.

[34] D. Zhao, H. Wang, K. Shao and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.

[35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning,* vol. 8, p. 229–256, 1992.

[36] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952,* 2015.

[37] A. Nair *et al*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296,* 2015.

[38] H. Van Hasselt, A. Guez and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[39] Z. Wang *et al*, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[40] D. Li, D. Zhao, Q. Zhang and C. Luo, "Policy gradient methods with Gaussian process modelling acceleration," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[41] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic algorithms," in *Advances in Neural Information Processing Systems*, 2000.

[42] D. Silver *et al*, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014.

[43] J. Peters and J. A. Bagnell, "Policy gradient methods," *Scholarpedia,* vol. 5, p. 3698, 2010.

[44] N. Usunier, G. Synnaeve, Z. Lin and S. Chintala, "Episodic exploration for deep deterministic policies: an application to StarCraft micromanagement tasks," *arXiv preprint arXiv:1609.02993,* 2016.

[45] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[46] V. Mnih *et al*, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[47] F.-Y. Wang *et al*, "Where does AlphaGo go: from church-turing thesis to AlphaGo thesis and beyond," *IEEE/CAA Journal of Automatica Sinica,* vol. 3, pp. 113-120, 2016.

[48] P. Peng *et al*, "Multiagent bidirectionally-coordinated nets: emergence of human-level coordination in learning to play StarCraft combat games," *arXiv preprint arXiv:1703.10069,* 2017.

[49] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing,* vol. 45, p. 2673–2681, 1997.

[50] J. Schulman, S. Levine, P. Abbeel, M. Jordan and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015.

[51] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347,* 2017.

[52] K. Shao, Y. Zhu and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Transactions on Emerging Topics in Computational Intelligence,* vol. 3, p. 73–84, 2018.

[53] A. Skrynnik *et al*, "Hierarchical deep Q-network from imperfect demonstrations in Minecraft," *Cognitive Systems Research,* vol. 65, p. 74–78, 2021.

[54] A. Amiranashvili, N. Dorka, W. Burgard, V. Koltun and T. Brox, "Scaling imitation learning in Minecraft," *arXiv preprint arXiv:2007.02701,* 2020.

[55] S. Huang and S. Ontañón, "Action guidance: getting the best of sparse rewards and shaped rewards for real-time strategy games," *arXiv preprint arXiv:2010.03956,* 2020.

[56] S. Muggleton and L. De Raedt, "Inductive logic programming: theory and methods," *The Journal of Logic Programming,* vol. 19, p. 629–679, 1994.

[57] V. Zambaldi *et al*, "Relational Deep Reinforcement Learning," *arXiv preprint*

arXiv:1806.01830, 2018.

[58] D. Churchill, Z. Lin and G. Synnaeve, "An analysis of model-based heuristic search techniques for StarCraft combat scenarios," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2017.

[59] S. Liu, S. J. Louis and M. Nicolescu, "Comparing heuristic search methods for finding effective group behaviors in rts game," in *2013 IEEE Congress on Evolutionary Computation*, 2013.

[60] S. Liu, S. J. Louis and M. Nicolescu, "Using CIGAR for finding effective group behaviors in rts game," in *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, 2013.

[61] D. Churchill and M. Buro, "Portfolio greedy search and simulation for large-scale combat in StarCraft," in *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, 2013.

[62] P. Sun *et al*, "TStarBots: defeating the cheating level builtin ai in StarCraft II in the full game," *arXiv preprint arXiv:1809.07193,* 2018.

[63] Z.-J. Pang *et al*, "On reinforcement learning for full-length game of StarCraft," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[64] W. H. Guss *et al*, "The MineRL 2020 competition on sample efficient reinforcement learning using human priors," *arXiv preprint arXiv:2101.11071,* 2021.

[65] D. Lee *et al*, "Modular architecture for StarCraft II with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018.

[66] G. Chaslot, S. Bakkes, I. Szita and P. Spronck, "Monte-Carlo tree search: a new framework for game AI," in *AIIDE*, 2008.

[67] C. R. Harris *et al* , "Array programming with NumPy," *Nature,* vol. 585, p. 357–362, 9 2020.

[68] M. Abadi *et al*, *TensorFlow: large-scale machine learning on heterogeneous systems,* 2015.

[69] D.-A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289,* 2015.

[70] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[71] B. L. Welch, "The generalization of Student's problem when several different population

variances are involved," *Biometrika,* vol. 34, p. 28–35, 1947.

[72] Student, "The probable error of a mean," *Biometrika,* p. 1–25, 1908.