NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DEVELOPING COMBAT BEHAVIOR THROUGH REINFORCEMENT LEARNING

by

Jonathan A. Boron

June 2020

Thesis Advisor:	Christian J. Darken
Second Reader:	Jonathan K. Alt

Research for this thesis was performed at the MOVES Institute.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
DEVELOPING COMBAT BEHAVIOR THROUGH REINFORCEMENT LEARNING

5. FUNDING NUMBERS

6. AUTHOR(S) Jonathan A. Boron

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

The application of reinforcement learning in recent academic and commercial research projects has produced robust systems capable of performing at or above human performance levels. The objective of this thesis was to determine whether agents trained through reinforcement learning were capable of achieving optimal performance in small combat scenarios. In a set of computational experiments, training was conducted in a simple aggregate-level, constructive simulation capable of implementing both deterministic and stochastic combat models, and neural network performance was validated against the tactical principles of mass and economy of force. Overall, neural networks were able to learn the ideal behaviors, with the combat model and reinforcement learning algorithm having the most significant impact on performance. Moreover, in scenarios where massing was the best tactic, the training duration and learning rate were determined to be the most important training hyper-parameters. However, when economy of force was ideal, the discount factor was the only hyper-parameter that had a significant effect. In summary, this thesis concluded that reinforcement learning offers a promising means to develop intelligent behavior in combat simulations, which could be applied to training or analytical domains. It is recommended that future research examine larger and more complex training scenarios in order to fully understand the capabilities and limitations of reinforcement learning.

14. SUBJECT TERMS
artificial intelligence, autonomous planning, neural networks, reinforcement learning, machine learning, combat model, wargaming, modeling and simulation

15. NUMBER OF PAGES
101

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DEVELOPING COMBAT BEHAVIOR THROUGH REINFORCEMENT LEARNING

Jonathan A. Boron
Captain, United States Marine Corps
BS, George Mason University, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION

from the

NAVAL POSTGRADUATE SCHOOL
June 2020

Approved by: Christian J. Darken
Advisor

Jonathan K. Alt
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The application of reinforcement learning in recent academic and commercial research projects has produced robust systems capable of performing at or above human performance levels. The objective of this thesis was to determine whether agents trained through reinforcement learning were capable of achieving optimal performance in small combat scenarios. In a set of computational experiments, training was conducted in a simple aggregate-level, constructive simulation capable of implementing both deterministic and stochastic combat models, and neural network performance was validated against the tactical principles of mass and economy of force. Overall, neural networks were able to learn the ideal behaviors, with the combat model and reinforcement learning algorithm having the most significant impact on performance. Moreover, in scenarios where massing was the best tactic, the training duration and learning rate were determined to be the most important training hyper-parameters. However, when economy of force was ideal, the discount factor was the only hyper-parameter that had a significant effect. In summary, this thesis concluded that reinforcement learning offers a promising means to develop intelligent behavior in combat simulations, which could be applied to training or analytical domains. It is recommended that future research examine larger and more complex training scenarios in order to fully understand the capabilities and limitations of reinforcement learning.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
DoD	Department of Defense
Dota2	Defense of the Ancients 2
LSTM	Long Short-Term Memory
MCDP	Marine Corps Doctrinal Publication
MOBA	Multiplayer Online Battle Arena
OPFOR	Opposing Forces
PPO	Proximal Policy Optimization
RTS	Real-time Strategy
TRPO	Trust Region Policy Optimization
TTPs	Tactics, techniques, and procedures
VPG	Vanilla Policy Gradient

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Artificial intelligence (AI) has steadily progressed in capability over the past several decades, defeating world-champion human players in the games of checkers, chess, and Go. However, recent advances in deep reinforcement learning have now allowed researchers to attain professional human-level performance in popular real-time strategy (RTS) and multiplayer online battle arena (MOBA) games, such as StarCraft II and Defense of the Ancients 2. Different from traditional board games, RTS and MOBA games provide players an incomplete depiction of the environment, requiring decisions to be made on estimates and imperfect information. This presents an opportunity to the military domain, as high-performing AI agents in constructive simulations have the potential to improve professional military education, support course of action analysis, and validate doctrinal strategies.

The research performed in this thesis examined whether reinforcement learning was capable of producing optimal offensive AI opposing force (OPFOR) behavior in small tactical engagements. In this regard, optimal behavior and tactical behavior are separate concepts; however, engagements were designed such that optimal behavior aligned with two tactical principles of war – mass and economy of force. AI agents were trained in a turn-based, aggregate-level constructive simulation, in which entities represented either company- or platoon-sized units. The training environment where these entities maneuvered was a ten-by-ten featureless plain divided into hexagonal spaces. As such, entities were capable of discretely moving in one of six possible directions each turn, a method commonly used in wargames. Using an incremental approach, feed-forward neural networks were trained to fight increasingly complex engagements, which consisted of two-versus-one, two-versus-two, and three-versus-two scenarios. In each circumstance, neural networks controlled two or more red offensive OPFOR entities, while static blue friendly entities remained in a fixed defense. Several different reinforcement learning algorithms and numerous neural network training hyper-parameter configurations were tested to identify the prime factors for performance.

The first portion of this thesis examined whether agents could learn ideal behaviors in a two-versus-one scenario by testing several different combat models and reinforcement learning algorithms. In this situation, agent performance was validated against the principle of war known as mass, as the best course of action was for agents to simultaneously consolidate forces on the opposing unit. Overall, agents learned the desired behavior, although those trained in deterministic combat environments generally outperformed their stochastic counterparts. In particular, agents trained in deterministic models were more synchronized in their maneuvers, taking less time to converge both AI-controlled entities on the opposing unit. The next test configuration was a two-versus-two scenario. For this case as well, performance was validated against the principle of mass, as the best strategy was for both red entities to attack and defeat each blue defensive unit in a sequential manner. This task was also met with success, as trained agents coordinated and destroyed opponents in a logical order. Again, it was observed that agents trained in deterministic combat models outperformed stochastic ones, demonstrating faster force consolidation against opponents.

The second portion of this thesis validated AI performance against the tactical concept of economy of force. In a three-versus-two scenario, three attacking AI-controlled companies were set against a static platoon and company arranged in a defensive line. In this situation, two different behaviors emerged. The first strictly employed the principle of mass, with all three AI-controlled entities coordinating successive attacks on the opposing units. However, the second displayed a different principle of war, economy of force, as the three AI-controlled entities separated into two groups. In this case, one group of two entities attacked the company-sized unit, while a single entity simultaneously engaged the platoon. The difference in performance between these behaviors was found to be related to the discount factor. A lower discount factor produced agents focusing on speed and economy of force, whereas a large discount factor, closer to one, trained agents to mass on their targets. Moreover, the size and number of hidden layers appeared to impact the overall quality of performance, but this was not found to be statistically significant.

The last part of this thesis compared the performance of three reinforcement learning algorithms, which included Vanilla Policy Gradient (VPG), Proximal Policy

Optimization (PPO), and Trust Region Policy Optimization (TRPO), in a two-versus-one scenario implementing a deterministic combat model. In this regard, there was no performance distinction between VPG and PPO, although TRPO outperformed both PPO and VPG by exhibiting higher levels of teaming and coordination than its PPO and VPG counterparts.

Overall, this thesis shows that reinforcement learning techniques are capable of attaining optimal performance in small combat scenarios. It was found that the type of combat model and algorithm implemented in the training environment significantly impacted agent performance, with deterministic combat models mostly producing superior results. Moreover, in a scenario in which the agent has the option of pursuing two different strategies, the hyper-parameters selected, particularly the discount factor, will influence the final behavior. While this research has made significant progress in applying reinforcement learning to the military domain, more complex and diverse scenarios need to be studied to fully understand the capabilities and limitations of reinforcement learning.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My love and gratitude go to my wonderful wife, Miho, for her patience and endurance during the long days and the many times I brought my studies home. To Dr. Chris Darken and the entire MOVES staff, thank you for the unyielding support and dedication over the past two years. To my peers and fellow students, thank you for serving as a sounding board and source of morale. Finally, to the professionals at SoarTech, I am grateful for your recommendations and access to research projects.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. DOCTRINE AND SIMULATION

The Marine Corps' primary method of warfighting, as outlined in Marine Corps Doctrinal Publication (MCDP) 1, *Warfighting* [1], is through the employment of maneuver warfare, which relies on gaining a situational advantage over the enemy rather than fighting him directly. Maneuver warfare, a theoretical concept, is attained through sound application of the tactical tenets and principles of war. The tactical tenets, of which there are six, address more abstract leadership and decision-making properties, while the nine principles of war are aids to consider while conducting mission planning. Of note, use of the tactical tenets and principles of war transcends all three levels of war: tactical, operational, and strategic.

MCDP 1-3, *Tactics* [2], lists the tactical tenets as achieving a decision, gaining an advantage, being faster, adapting, cooperating, and exploiting success and finishing. As is often described as being part of the "art and science" of warfare, these tenets require both technical knowledge of available resources and creativity in their employment. Consequently, the "science" of warfare is understanding how different weapon systems, personnel, and force structures generate unique effects, whereas, the "art" of warfare is being able to combine these effects to achieve a particular tenet. While the tenets may be described in high-level detail, there is no specific formula for producing them. Moreover, these tenets are not conducted in isolation, but rather are combined to attain some particular greater effect. As situations change, the importance of each tenet may also shift, leaving the responsibility to the decision-maker as to when and how to employ each one.

Complementing the tactical tenets are the principles of war, which include mass, objective, offensive, security, economy of force, maneuver, unity of command, surprise, and simplicity [3]. As opposed to the tactical tenets, the principles of war provide a more specific guide for planners to consider while evaluating the mission, enemy forces, terrain, and friendly forces available. However, like the tactical tenets, they do not represent a stringent process and are not always equally weighted. It may not always be practical or

feasible to implement all principles of war in an operation. Therefore, it is left to the discretion of the commander as to which principles to prioritize or exclude.

As alluded to previously, there is no specific definition or process for applying the tactical tenets or principles of war. Individual units, of all various functions and mission sets, have the tendency to develop and interpret doctrine, resulting in what are known as tactics, techniques, and procedures (TTPs). Far from being harmful, TTPs provide simple and efficient steps to subordinate units and Marines that have been generally proven to be effective through combat or training. However, TTPs may be limited in scope or quickly become obsolete as enemy tactics and equipment change. In general, the impetus for a change in TTPs results from casualties or failure on the battlefield. In this regard, combat simulations provide an efficient means to compare, identify, or predict more effective TTPs prior to a live-training exercise or contact with the enemy.

Although many combat simulations exist in all branches and sectors of the Department of Defense (DoD), their primary purpose can be effectively classified in one or more of three categories: training and education, course of action analysis, and doctrine and equipment development. Simulations have proven to be a useful tool for leaders to train their subordinates prior to live execution, offering an environment that can reproduce dangerous or uncommon situations with little physical risk to the trainees. Moreover, simulations have long been used to build, compare, and contrast various courses of action. What would otherwise take an extensive amount of planning and resources for live rehearsals can be built and executed in virtual environments in a matter of minutes or hours. Lastly, simulations are commonly used to guide high-level decision making, such as equipment acquisition and force structure [4]. Wargaming simulations provide senior leaders a forum to examine the impacts of changing equipment capability or unit task organization.

B. PROBLEM FRAMING

Simulated adversarial behavior, commonly known as red forces or opposing forces (OPFOR), relies on either hard-coded behaviors or external human support. Although both methods can sufficiently function as solutions, there are inherent drawbacks in their use

[5]. Hard-coded adversarial behavior is logistically simple to implement once programmed and has properties that can be scaled to increase difficulty. However, any changes to the simulation will also typically require non-trivial updates to the OPFOR behavior model. Moreover, human opponents are often able to find glitches or weaknesses within the coded behaviors that can be unrealistically or repeatably exploited to defeat the simulated adversary [5]. Additionally, the behavior of this simulated entity becomes predictable over time, providing the human player another unfair advantage. On the contrary, the negative attributes of hard-coded adversarial behaviors can be mitigated through human-controlled red forces. This prevents players from exploiting programming bugs and mitigates any predictability that computer-controlled adversaries may present. However, human support is logistically expensive and limited to the availability and work schedule of supporting personnel. Consequently, a simulation requiring human OPFOR support may require advance funding and scheduling in order to ensure resources are allocated.

Given drawbacks of current OPFOR employment options, such as those discussed above, reinforcement learning offers a promising solution to generate adversarial behavior in combat simulations that is intelligent and adaptive. Over the past 20 years, reinforcement learning agents have continued to progress in capability and resiliency, defeating world champions in games with large action spaces, such as chess, Go, and poker [6], [7]. Moreover, the fusion of deep learning and reinforcement learning algorithms has enabled artificial intelligence (AI) systems to perceive abstract features in environments and use that information to exhibit robust behavior. In this regard, commercial video games have proven to be a popular venue for researchers to train and test reinforcement learning agents. Even more so, real-time strategy (RTS) games have proven to be particularly difficult, forcing agents to make decisions with imperfect information in hopes of attaining long-term rewards. Despite these challenges, agents have achieved impressive results, defeating some of the best human players in the popular games of StarCraft II and Defense of the Ancients 2 (Dota2) [8], [9]. Thus, it appears feasible that deep reinforcement learning agents could serve as competitive and effective opponents in DoD programs of record simulations.

On the other hand, deep reinforcement learning agents may also provide profound insight to the DoD's doctrine and policies. While thousands of years of warfare have influenced the doctrinal publications and texts referenced above, there is still no absolute certainty that they represent the best means of warfighting. Consequently, if it can be demonstrated that deep reinforcement learning consistently discovers and employs optimal strategies, then these AI agents can be used to test and validate current doctrine. Moreover, this will also provide an avenue to test and explore courses of action against adversary warfighting techniques, independent of (possibly incorrect) beliefs toward adversary behavior, allowing the DoD to maintain a competitive edge. It should be noted, though, that tactical behavior and optimal behavior are independent and separate notions from each other. Optimality in one situation may not be comparable to any one tactical concept, and, likewise, application of a particular tactical principle may not represent the optimal solution. However, for the purposes of this project, all scenarios are designed such that optimal and tactical performance converge.

C. SCOPE OF THESIS

This thesis seeks to explore the ability of neural networks, trained through reinforcement learning, to demonstrate optimal, tactical behaviors in a constructive combat simulation. Included in this task is the optimization of state representation and action space, neural network configurations, and reinforcement learning parameters. Program of record simulations currently in the DoD could be used as training and testing platforms; however, anomalies identified in the simulation during agent training and evaluation may require contractor or developer support to correct. Additionally, complex environments are not necessary for the small tactical scenarios that will be employed. Therefore, a simple combat model coded in Python will be the primary environment to train and evaluate trained agents. This will also allow ease of interoperability with a large pre-existing library of machine learning algorithms and other reinforcement learning tools.

For the purposes of this thesis, the training environment will be built as an aggregate-level constructive simulation; that is, all personnel and equipment will be simulated and combat will be resolved at the platoon or company level. Because the focus

is on developing and comparing simple behaviors, terrain and weather effects will not be incorporated into the simulation. Moreover, logistical considerations, such as fuel and ammunition, will not be included, as round expenditure and number of movements will not be tracked nor penalized. Lastly, inter-unit communication will not be explicitly modeled and it will be assumed that perfect command and control exists within the training environment.

D. BENEFITS OF THESIS

Numerous research projects have examined deep reinforcement learning performance in RTS and commercial video games, although very few have studied the topic in the military domain. Thus, this thesis will expand on this field and provide a fundamental understanding as to how deep reinforcement learning can produce intelligent OPFOR behavior in constructive simulations. Although various aspects of combat are abstracted or omitted, the objective is to attain optimum performance in a simple environment that can be incrementally expanded upon in more difficult scenarios. Consequently, the agents trained during research will not be directly transferrable to other simulations for immediate use, although, the techniques applied to generate good performance can be. Therefore, the methods and parameters that consistently produce good results will be more valuable to the progress of AI in the military domain than the trained neural networks themselves.

Ultimately, the greatest benefit of this thesis is in its assessment of whether reinforcement learning is capable of achieving optimal performance. If confidence can be attained that AI agents discover and execute superior tactics and strategies, then DoD simulations and wargames will significantly benefit from their incorporation as OPFOR. As opposed to relying on teams of contractors to program specific behaviors or manually control entities, neural networks trained through deep reinforcement learning will serve as adaptable opponents for varying purposes and scenarios. These applications will vary from supporting large staff training exercises to validating future force design structures. Because these AI agents will be software based, they can be deployed to any number of commands and organizations, allowing for a broad audience of users. Moreover, modifying

the training parameters of an agent can lead to a number of different effects, scaling the difficulty of an AI opponent or shifting its overall strategy. Accordingly, service members of all backgrounds and experience levels will benefit from this work.

II. BACKGROUND

This chapter will introduce the high-level concepts behind reinforcement learning and explore some recent research projects that relate to this thesis. The projects discussed mostly apply to novel forms of reinforcement learning algorithms or different neural network architectures. However, each of these studies evaluates the performance of trained agents in a game environment or simulation, often comparing their tactical or strategic behavior to human performance.

A. INTRODUCTION TO REINFORCEMENT LEARNING

Science fiction and human imagination have long speculated the idea of highly intelligent machines and the possible impact they may have on society. The advent of modern computing and micro-processing has produced commonplace devices that have made some of those visions a reality [10]. However, despite the ever-increasing processing power and memory made available, certain tasks still remain difficult for machines to complete. Consider, for example, a large ship such as a Navy destroyer. Navigating the vessel may seem to be a difficult task that requires a considerable amount of intelligence and training. While human operators do undergo hours of training and evaluation before being qualified to maneuver the boat, a path-finding algorithm linked to GPS coordinates could be used to replace a human operator at a relatively low computational cost. Although, even if the ship could navigate itself, there would still be a need for humans on board to manage weapon systems and respond to mechanical failures. In this regard, it is not the intelligence nor problem-solving capabilities of humans, but rather the adaptability and responsiveness that is valuable [10]. Humans are often able to take unexpected, unstructured environments and develop some pragmatic solution. Even if resolution is beyond individual abilities, there is still a practical solution to call for help and obtain additional resources. It is this common-sense intelligence that has been found to be very difficult to program into computers [10]. Nonetheless, the field of deep reinforcement learning has generated systems and agents capable of performing complex tasks at or above human level.

Deep reinforcement learning is effectively a combination of two fields: deep learning and reinforcement learning. When dealing with unstructured environments, deep learning models excel, as they are able to make connections between features in large sets of data [10]. However, this is limited to recognition and other passive activities, such as image classification, and does not support decision-making or behavior. On the other hand, reinforcement learning is the mathematical formalization of a decision-making problem that correlates to behavior [10]. An agent makes decisions that result in actions that change the state of the environment. From the environment, the agent observes the consequences of the actions, possibly receiving a reward. Standard reinforcement learning requires the designer to identify key features of the environment, which is likely to be highly specific to the current task and not transferable to other problems. Thus, combining deep learning and reinforcement learning allows for an automated learning process, in which the system itself identifies environmental features and acts upon the observations.

The principles and notions behind deep reinforcement learning are derived from biological and natural learning processes [10]. Model-free reinforcement learning refers to a system where the agent doesn't understand how the world works; it simply attempts to maximize the reward function. An example might be a dog learning to sit or lie down on command. The dog does not understand the purpose behind its actions, only that there is a likelihood of receiving a treat, or reward, upon completion. On the other hand, in model-based reinforcement learning, the agent first tries to learn how the world works, then uses that knowledge to predict what will happen when certain actions are taken. This might be similar to a wolf learning to hunt, where the wolf maximizes its reward based on its understanding of the environment and its prey's pattern of life. However, developing an optimal behavior policy from rewards is not a trivial problem. Humans and animals may infer rewards from only a few experiences, as there are many things in life that do not occur often or only a handful of times. Thus, learning can occur through means other than direct experience. Another common method of learning is through instructor demonstration, in which a student directly copies the teacher or predicts rewards from observed behavior. Moreover, learning can also occur in unsupervised environments through general observations, such that the agent is able to anticipate rewards from future actions based on

its understanding of the world. Lastly, learning may be transferred from other tasks, such as previous experiences that appear similar to the current task. To some extent or another, machine learning algorithms have attempted to represent these different learning methods.

As it pertains to reinforcement learning and other methods of machine learning, there exists a model of behavior, often termed a policy and labeled as π_θ [10]. The overarching goal is to find a good parameter vector, that is θ, which causes the policy to produce the desired actions given a particular state. This parameter vector θ may be manifested in different ways. However, in systems where a deep neural network represents the policy, θ specifically consists of the weights in that neural network. In order to optimize θ, reward functions are implemented such that the sum of rewards are maximized over all time steps. Figure 1 below exhibits the relationship of the policy to the world state (s), action space (a), and parameter vector.

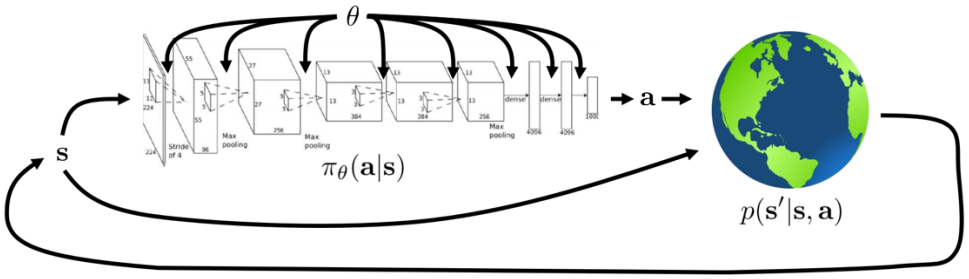

Figure 1. Graphical Illustration of Reinforcement Learning. Source: [10].

As displayed above, the policy is provided the world state, which may consist of complete information or only a partial observation. From this state, the policy executes some action that causes a transition to a new state with certain probability. The state, action, reward, and transition probabilities of moving to a new state consist of a Markov decision process. That is, the probability of moving to a particular new state (s_{+1}) is independent of the state the system was in (s_{-1}) before the current state (s).

There are numerous reinforcement learning algorithms that strive to develop the ideal policy, each with their own assumptions, advantages, and trade-offs. However, each

algorithm will follow the same general structure, which consists of sample generation, model fitting or return estimation, and policy improvement [10]. Figure 2 below presents the relationship of these components.

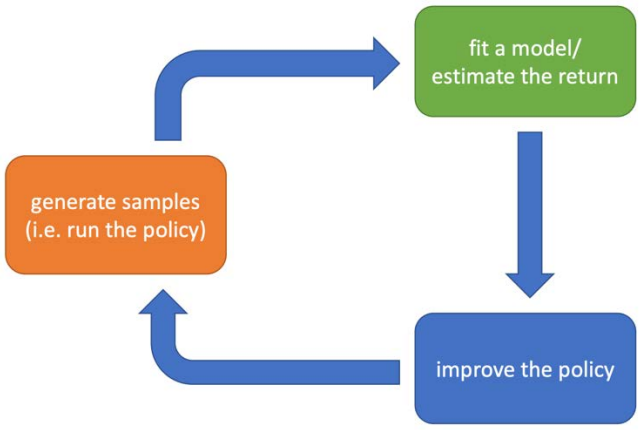

Figure 2. Structure of a Reinforcement Learning Algorithm. Source: [10].

For all reinforcement learning algorithms, there is some component that interacts with the world to implement the policy. Depending on how the environment is defined for the learner, this can comprise of a software agent operating in a game or a robot manipulating parts on an assembly line. Regardless, samples from interactions are generated and used to provide feedback as to the effectiveness of the policy [10]. This policy evaluation may be accomplished through several different means, such as measuring total rewards or assessing the model fit. Nonetheless, the data subsequently gained through evaluation is then used to improve the policy. This process repeats itself such that the probability of executing actions and trajectories that result in favorable behavior increase, while the probability of low performance behavior is decreased [10]. Overall, there is no particular facet of this cycle that is inherently more computationally expensive than the others, as the environment and type of algorithm used affect the cost of each component.

Predicting future rewards is an important aspect of reinforcement learning, as an agent must understand what action will cause the best result for the state that it's in. For this purpose, reinforcement learning algorithms often use quality functions (Q^{π}) or value

functions (V^{π}) [10]. Essentially, a quality function estimates the total reward from taking an action while in a particular state ($Q^{\pi}(s,a)$), while the value function is a total reward estimate from being in a state without consideration of a specific action ($V^{\pi}(s)$). If the quality function is known, then taking the maximum return value of all possible actions while in a state can be used to shape and improve the policy. Additionally, since the value function is simply an average estimation of reward from a state, a quality function value for a specific action that is higher than the return of the value function indicates that an action is better than average. Most reinforcement learning algorithms use these functions in some capacity or another to estimate the performance of a policy.

Generally speaking, reinforcement learning algorithms are classified as being either off-policy or on-policy [10]. An off-policy algorithm is able to use samples generated from other policies to improve its policy, while an on-policy algorithm must produce new data every time its policy changes. Therefore, off-policy algorithms tend to be more efficient as they do not require as many samples for learning. However, this does not necessarily mean that off-policy algorithms are faster. In the event that data can be produced in a negligible amount of time, model fitting and policy improvement may slow down the learning process. Consequently, despite being less efficient, on-policy algorithms can often perform faster than off-policy ones. Nevertheless, even within these two categories, various types of reinforcement learning algorithms exist, as efficiency, stability, ease of use, and underlying assumptions are all important considerations. Additionally, the complexity of the model, i.e., the environment, and the policy being learned will impact which specific algorithm is most suitable.

B. MULTI-AGENT LEARNING

A single agent attempting to learn an optimal policy can be a challenging issue, as numerous research projects have demonstrated. However, a single agent operating in isolation is rarely an accurate representation of the real world, where potentially hundreds or thousands of other entities may influence the state space. This is particularly true in the military domain, in that a service member, except under unusual circumstances, will always be working in at least a team of two. When this idea is applied to reinforcement learning,

in which multiple agents are required to cooperate or compete in the same environment, the learning process becomes much more complex, since the state transitions and rewards of an agent are now impacted by the actions of all other agents. Thus, Tampuu et al. [11] provide insight to the performance of agents trained both in multiplayer and single player environments, using the Atari game Pong as a training and testing venue.

For this particular study, agents were individually controlled through respective deep Q networks, with convolutional neural networks approximating the Q function. Agents were provided state information through raw visual sensory data; although, state updates were only provided once every four frames. The training environment, Pong, was selected because it supports two-player mode and previous work demonstrated successful training of deep Q networks in the game. The action space for the game consisted of only four possible activities: move paddle down, move paddle up, keep paddle still, or serve the ball. Agents initially had a fractional exploration rate of 1.0, but this rate decreased and leveled off at 0.05 over the first one-million time-steps. In training the agents, the authors attempted to demonstrate differing behaviors based on the rewards provided, that is, competition, cooperation, or some mix between the two. As such, an agent was always penalized a score of -1 when it was scored against, but the reward for scoring on the opponent was varied between +1 and -1 between trials. In addition to the final game scores, the average number of paddle-bounces, wall-bounces, and serving times were also tracked. Overall, agents were trained for a total of 50 training cycles consisting of 250,000 time-steps each.

The results show that networks trained in a multi-agent environment display a wide range of dynamic behaviors, which, predictably, are heavily influenced by the reward system used [11]. When a score of +1 was provided to the agents for scoring against an opponent, the authors observed that agents present highly competitive behavior. As such, the agents tended to manage the ball such that the number of paddle-bounces were minimized but wall-bounces maximized. Moreover, agents tended to serve the ball relatively quickly after scoring; a reasonable reaction given that the only way to receive a reward was through active gameplay. On the other hand, when a score of -1 was provided, regardless of which entity scored, agents exhibited strong cooperation [11]. In this regard,

the agents learned to minimize wall-bounces and maximize paddle-bounces. Additionally, agents tended to position themselves at the edge of playing field and return the ball as close to horizontal as possible, making it easier for their counterpart to catch and return. Since active gameplay would only result in negative rewards, agents were reluctant to serve the ball, only serving when selected through random action. When the reward for scoring was varied between +1 and -1, behaviors tended to favor a collaborative strategy, even with a reward as high as +0.75.

In addition to observing the behaviors and strategies that emerged from reward variation, agents that were trained in a multiplayer setting also competed against agents trained in single player mode. In head-to-head games, Tampuu et al. [11] noted the performance of multiplayer-trained agents was overwhelmingly superior to that of single player-trained, with an average final score 21-2. This was in part attributed to the fact that an agent trained against a hard-coded algorithm (single player) is exposed to a limited set of behaviors, and the agents often learn to capitalize on design weaknesses and bugs [11]. On the contrary, agents trained in a multiplayer environment are introduced to an opponent with robust behavior that changes as learning progresses. Consequently, agents that train against each other, as opposed to a hard-coded algorithm or static opponent, have the potential to perform better against unknown or unfamiliar adversaries.

C. FIRST-PERSON SHOOTER GAMES

The systems trained to and proficient in Atari 2600 games constitute major milestones in deep reinforcement learning, especially considering that raw visual data was the primary source of state information [12]. However, despite the super-human performance some of these systems attained, the situations they reflect are not representative of real-life conditions. Namely, these games are two-dimensional environments with a third-person view of the player. Consequently, Kempka et al. [13] developed and tested a software interface that would specifically support reinforcement learning in a three-dimensional, first-person environment, using the first-person shooter game Doom as their test platform. Of the many first-person shooter games available as a test platform, the authors chose Doom because it has an active community of users, open-

source code, multi-player mode, intuitive scenario builder, and customizable resolution options. Additionally, according to the study's authors, it has relatively low system processing and disk space requirements, lending itself well for multiple instances of the game to be run on the same computer. While previous work had been conducted on first-person shooters, these previous studies provided high-level information to the agent that would otherwise be unavailable to a human player, such as the global locations of allies, enemies, resources, and structures [13].

As opposed to presenting agents high-level information, as previous work had done, Kempka et al. [13] decided to exclusively use visual data. Thus, the reinforcement learning agents interact with the game environment through an application programming interface (API) the authors label as VizDoom. This interface has four primary features that make it flexible to use for the purposes of machine learning. First, it provides four different possible agent control modes: synchronous player, asynchronous player, synchronous spectator, and asynchronous spectator [13]. In the asynchronous modes, gameplay proceeds continuously at 35 frames per second. However, in synchronous mode, the game waits for the human or agent to make a decision before proceeding to the next frame. Moreover, in player modes, the agent is the decisionmaker, whereas in spectator modes the agent observes a human player. Secondly, VizDoom is capable of running user-defined scenarios, which may include custom maps and environments, game-play mechanics, or victory conditions [13]. This feature is primarily based on open-source editing suites and programs developed by the community of Doom users. Third, the agent is able to access a renderer's depth buffer, which may provide additional context to the visual information an agent receives [13]. Lastly, VizDoom supports off-screen rendering and frame skipping. Off-screen rendering eliminates the need for a graphical user interface, allowing for experiments to be conducted on servers, and frame skipping omits a certain number of frames, such that not all frames are rendered and presented to the agent.

Having developed the API, Kempka et al. [13] performed two experiments to test its viability. In both experiments, the agents were trained through Q-learning, with a convolutional neural network approximating the Q-function. The first experiment consisted of a simple scenario where the agent's objective was to kill an enemy monster.

The agent received a reward of 101 for killing the monster, but was penalized -5 for missing a shot and -1 for every action taken. The second experiment was more complex in that the agent was tasked with navigating through a maze with an acid surface. The agent continuously lost health and needed to search for and acquire health kits in order to stay alive. Making the situation more complex, vials of poison were also dispersed through the environment, which reduced the agent's health further if picked up. The agent was rewarded 1 for every time step, but penalized -100 for dying.

Overall, in both experiments, the study's authors found the agents learned optimal behaviors. For the first experiment, all agents trained were able to attain high performance, with the number of frames skipped primarily affecting the learning rate [13]. In the second experiment, agents displayed human like performance, navigating toward health kits and avoiding poison vials, although, hesitation was often observed when there were multiple paths for the agent to take [13]. Nonetheless, the authors demonstrated the utility of the VizDoom API and the potential for future reinforcement learning to be conducted using raw visual information flow as state information in three-dimensional, first-person environments.

D. RTS GAMES

A challenging aspect of deep reinforcement learning is teaching agents in domains where rewards may be initially scarce or non-existent. RTS and other related games, such as StarCraft or Defense of the Ancients, require players to develop a strategy and build economic and combat power before directly confronting opponents [9]. An effective policy may include early actions that appear to an agent to be either non-beneficial or detrimental to the current state, but will ultimately profit in the long-term. This is unlike a first-person shooter or arcade environment, in which an agent is likely to learn which actions are effective early in the scenario. StarCraft II and Dota2 are two strategy games that have recently drawn a significant amount of interest and attention in deep reinforcement learning research [9]. Both games have massive state-action spaces, on the order of hundreds of thousands, and limit the ability of players to observe the entire environment, introducing a high level of uncertainty. Moreover, these games have a suite of different maps and

terrains, several multiplayer configurations, and multiple races or nations to play as. Perhaps most importantly, these games also have large and enthusiastic communities of players, with highly competitive tournaments and ranking systems. These factors considered, OpenAI [9] was able to train a multi-agent team in Dota 2, ultimately beating top ranked players and teams in both one-on-one and team matches.

OpenAI's approach to training agents included using a form of proximal policy optimization (PPO), which alternates between sampling data in the environment and updating an objective function through stochastic gradient descent [9]. An exponential decay factor between 0.998 and 0.9997 was used, correlating to a reward decay half-life ranging from 46 seconds to about five minutes [9]. Moreover, a single layer, 1024-unit long short-term memory (LSTM) network controlled each of the agents. In this regard, LSTM is a form of recurrent neural network, which is different from traditional neural networks in that there are loops in their structure that allow information to persist. Each agent trained for an equivalent of 180 years every day, playing against itself without any external human data or replays. In order to support this training regimen, OpenAI utilized 128,000 pre-emptible CPU cores and 256 GPUs, which supported synchronized gradient descent. Unlike some of the previously discussed research conducted on Atari 2600 games and Doom, the OpenAI algorithms did not obtain state information strictly from the raw visual output. Rather, they interfaced directly with the game through a Bot API, which provided them the same information a human player would see. However, despite the fact that there was no explicit advantage in information availability, the agents had the ability to access it nearly instantaneously, whereas human players were forced to search for it on the screen [14]. Furthermore, the trained agents were limited in performance, as they were only able to use 18 of over 100 possible "hero" player options.

In competitions against human players, the OpenAI agents performed exceptionally well, winning two out of three matches against a top-ranked professional team and consistently beating top-ranked players individually in one-on-one matches [15]. Additionally, in an effort to identify exploits and weaknesses, the agents were made available for public matches, allowing anyone to play against them. In these public games, the OpenAI agents were able to beat 99.4% of its opponents over the course of nearly

430,000 matches. Although not totally unbeatable, these trained agents achieved the ability to perform at professional levels in an environment that is highly uncertain and chaotic. Even human players have begun to adopt the novel strategies that the agents have introduced. Overall, the results OpenAI attained demonstrate that deep reinforcement learning systems are capable of learning complex policies in a long-term, strategic context.

Further refining agent constraints, DeepMind [8] in its AlphaStar project was able to train and employ systems capable of defeating elite human players in StarCraft II. However, unlike the agents OpenAI produced, these agents did not have instantaneous access to all available data and were limited to a certain number of actions per minute. As such, agents were only capable of conducting 22 non-duplicate actions during the course of a five-second window. This meant that agents had to prioritize which actions to take, which included searching the map for more information. Moreover, agents were not restricted to any particular player configuration, able to play as any one of three available races and against any opponent. AlphaStar's only significant limitation was the number of game environments it was able to play in, which was constricted to four of seven available one-on-one maps. Overall, to ensure AlphaStar's gameplay abilities were fair, professional StarCraft II players were consulted during the research process and approved the final system constraints.

The training regimen for AlphaStar consisted of two main parts; an initial supervised learning phase and a multi-agent, self-play reinforcement learning phase [8]. The primary purpose of supervised learning was to initialize agents and maintain diverse exploration that produced a robust policy. This was performed with 971,000 replays from the top 22 percent of human players. From these replays, a statistic *z* was calculated that represents the build order of each player's first 20 units and buildings. This *z* statistic was used in both the supervised and reinforcement learning portions of training. Moreover, cumulative statistics were also produced from the units, buildings, upgrades, and special effects purchased during the game. The agent parameters were then updated during learning such that the Kullback-Leibler divergence between agent and human actions was optimized. In total, three separate groups were produced from supervised learning, and,

once complete, training continued with a form of multi-agent reinforcement learning termed as league training [8].

League training implemented three distinct groups of agents, which primarily differed in how they selected opponents [8]. As training progressed, these agents produced copies of themselves, with parameters frozen at a specific point, and added those copies to the league. The opponents for the first group, main agents, consisted primarily of themselves and all previous players, which contributed to developing a robust policy. Opponents for main agents were selected such that games were not wasted against weak agents, providing main agents the opportunity to overcome the most difficult opponents. The second group of agents, main exploiters, were only matched with the current generation of main agents, as their primary function was to isolate and exploit weaknesses in the main agents. Lastly, the third group of agents consisted of league exploiters, who only play against past players. Their purpose, according to the study's authors, was to identify and exploit weaknesses in the entire league. In total, agents completed 44 days of league training. Moreover, the reinforcement learning algorithm itself was a policy gradient algorithm similar to advantage actor-critic, with updates applied asynchronously on replayed experiences [8]. This was an off-policy learning approach that updated the most current policy with experiences generated from previous ones.

Three different sets of agents, each set containing three agents trained in the three respective races, were compared against human performance in live games. The first set only completed supervised training, the second set 27 days of league training, and the third set the full 44 days of league training. Overall, all three sets performed well against human opponents, with the weakest set of agents, having only completed supervised training, ranked in the top 16 percent of human players. On the other hand, both sets of agents that completed some or all of league training were ranked in the top half-percent of players. However, the set of agents that trained for the full 44 days of league training were each ranked in the top 0.2 percent, correlating to a grandmaster status. These impressive results, for both Dota2 and StarCraft II, demonstrate the robust decision-making capabilities that deep reinforcement learning agents are capable of attaining in a partially observable environment.

E. MILITARY APPLICATION

Commercial RTS games most closely match the constructive simulations that the DoD uses to train large unit staffs, that is, the player must balance intelligence, logistics, and combat power. However, despite these similarities, the strategies and tactics employed in RTS games rarely match real world doctrine and are tailored to the unique unit capabilities and control functions that distinguish each game. Moreover, large military simulations are much more complex to learn and operate, with potentially hundreds to thousands of different units and force configurations, robust command and control options, and extensive logistical considerations. Consequently, Lian [16] in his master's thesis explored the possibility of training agents to execute simple tactical actions that real operational units would perform. As such, he took a gradual approach to training, focusing on achieving fundamental behaviors such as enemy recognition and force consolidation.

Because most military simulations do not have APIs that allow reinforcement learning algorithms to connect to the underlying program, Lian [16] constructed his own simple combat simulation using the Unity3d game engine. The underlying combat model was based on probability of kill, with each entity firing a shot every two to three seconds over a normal distribution. Scenarios consisted of one-on-one, two-on-one, or two-on-two, in which friendly forces would respawn until certain terminal conditions were met. These conditions included the destruction of all enemy entities, the simultaneous destruction of all friendly entities (before respawn could occur), or after a certain period of time in which no reward had been attained. Entities were controlled through individual convolutional neural networks, with hidden unit sizes of 20, 40, or 60. Moreover, Q-learning with an experience replay buffer was used to facilitate training, employing learning rates between 0.01 and 0.1 and discount factors from 0.9 to 0.99.

Lian [16] found large success in his initial objective: training agents to locate and close with the enemy. His state representation included using sector rings divided into eight separate sections, with each portion representing one of the eight cardinal and inter-cardinal directions [16]. Using these sector rings, an agent had a general idea of where other friendly and enemy entities were and could move in any of the eight directions or stay in place. Moreover, two reward function were tested. The first provided a reward to the agent at

every step, which was a value equal to the inverse of the distance between the agent and the closest enemy. Consequently, the reward would gradually increase as the agent approached an enemy unit. On the contrary, the second function returned a reward of +1 after each action as long as the agent moved closer to an enemy entity. Otherwise, no reward was provided. Ultimately, the reward function proved to be the most critical factor in learning, as the second reward function easily produced the desired behavior.

Efforts to train agents to consolidate in a synchronized manner during an attack proved more difficult. In addition to the sector rings, range rings were added to the state representation to provide agents an estimate of the distance between them and friendly and enemy entities [16]. As with the previous task, the action space included moving in one of eight directions or waiting in place. Once an agent was within a particular distance of an enemy unit, combat would begin automatically. Again, two separate reward functions were examined for performance. The first function returned a reward of +1 if an agent destroyed an enemy unit. The second function, accounting for friendly casualties, consisted of the total number of enemy units killed (scaled by a constant factor) minus the total number of friendly units lost (scaled by a separate constant). Overall, neither reward function produced favorable performance, although certain neural network configurations and training hyper-parameters produced better results. In summary, Lian [16] concluded that longer training durations, lower learning rates, and higher discount factors resulted in better performance. He also noted that the reward function and training configuration had a significant impact as well.

F. CONCLUSION

As the research projects above have demonstrated, deep neural networks trained through reinforcement learning have progressively improved their ability to perform in complex environments. This has been particularly exemplified in the systems that OpenAI and DeepMind have developed, which have successfully defeated some of the most capable human players in two of the most popular online games. Although military constructive simulations are considerably more complex than commercial games, these studies support the notion that deep reinforcement learning is capable of producing adaptive AI agents that

can serve as formidable OPFOR. Therefore, this thesis will take a similar approach as Lian [16], focusing on whether agents trained through reinforcement learning are capable of demonstrating optimal behavior and validating performance through tactical scenarios designed to elicit mass and economy of force. However, in his research, Lian implemented a single combat model and distributed control of agents to respective neural networks. This thesis will differ in that multiple combat models will be tested and a single neural network will control all AI agents.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY

This chapter will present in detail the environment and methods through which the neural networks were trained. As with many complex reinforcement learning projects, this research took an incremental approach to learning, beginning with simple tactical scenarios and gradually scaling up in complexity. These scenarios began by focusing on developing basic behaviors in agents, such as simultaneous two-on-one teaming, and progressed to more complex concepts, to include economic distribution of forces. Additionally, multiple combat models, reward functions, and reinforcement learning algorithms were implemented throughout this project to determine their impact on agent performance. However, the training environment remained constant, as all agents were trained in a featureless, 10-by-10 area. Moreover, feed-forward neural networks were used to train and control all agents, and all state information was pushed to the neural networks in a flattened data form. Throughout training, the neural networks controlled the red OPFOR entities, as the ultimate objective of this research was to generate intelligent adversary behavior.

A. TRAINING ENVIRONMENT

Game engines such as Unity and Unreal offer robust suites that simplify development of high-quality, three-dimensional games and simulations [17]. However, the ability for these environments to integrate with reinforcement learning libraries, such as Tensor Flow or PyTorch, remains arduous and non-intuitive. Considering that a graphically rich environment is not necessary to learn basic maneuver concepts, a simple two-dimensional training environment was built and coded in Python, using Pyglet to display fundamental state information to the human viewer. Python was specifically chosen as the programming language for its capacity to connect to two open-source reinforcement learning tools that OpenAI has developed, Spinning Up and Gym. These resources will each be described in greater detail later, although, the main benefit of their use is the seamless ability to connect to and conduct experiments with various reinforcement learning algorithms and machine learning libraries.

The training environment itself consists of a 10-by-10 hexagonal grid, similar in principle to those of many wargames. There are no terrain or feature considerations, as the map can be thought of as a flat plain. AI-controlled entities do not move simultaneously, but rather in a sequential turn-based manner. For example, if the neural network controls three units, it would take three steps for all entities to execute movement. If any entity moves outside the bounds of the environment, that particular episode will be immediately ended. Additionally, there is no human-in-the-loop feature that would allow a viewer to manipulate or control entities during an episode.

1. Graphical Representation

As displayed in Figure 3, the graphical interface is broken down into five widgets, with each one visually depicting some component of state information.

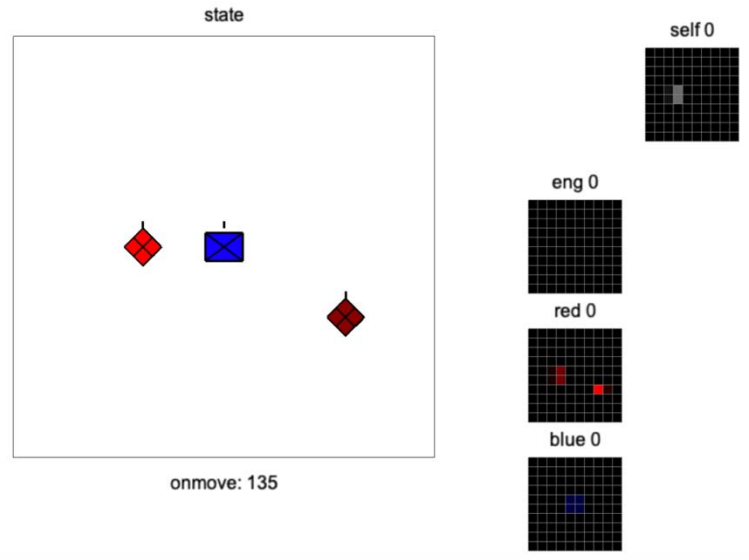

Figure 3. Graphical Display of State Information

The large, primary widget on the left summarily presents the positions, allegiances, and sizes of all active entities. The “onmove” label beneath this widget indicates which entity is currently in an action step, referring specifically to its unique entity identification number. All entities are depicted in accordance with Marine Corps Reference Publication 5-12A, *Operational Terms and Graphics* [18], with all friendly units represented by blue

rectangles and OPFOR represented by red diamonds. Additionally, indicators above each unit graphic designate the relative size of the entity. A single vertical bar above each unit graphic denotes a company sized unit; approximately 150-personnel in strength. Moreover, three small, solid circles represent a platoon-sized element – approximately 50 personnel. Lastly, a single small, solid circle signifies of a squad, or about 15 people. Once an entity has been destroyed, that is, its size has been reduced to zero, it is removed from display on the widget.

On the other hand, the four small, black widgets on the right side of the graphical interface display the position of the unit currently in-action, the position of any engaged units, the position of all red units, and the position of all blue units, respectively. These widgets also represent the state information that is provided to the neural network at each update. Of note, the data presented in these widgets is much less detailed and often indistinct, as the location of each entity is weighed and distributed across a grid-representation of the environment. For example, if an agent is positioned on or near the border between two grid squares, both squares will be proportionally highlighted.

2. Combat Models

Several combat models, each with its own unique reward system, can be implemented in the training environment. The simplest of these models consists of a referee system, providing agents a higher reward for quicker massing and consolidation of forces. When this reward model is implemented, units do not actually execute combat, as the scenario ends once both entities have engaged an opponent or the time limit is reached. Accordingly, Equation (1) below displays how the reward is calculated.

$$\text{Reward} = \text{Max}\left(0.0,\ 1.0 - \frac{\text{time\_between\_engagements} - 1.0}{\text{reward\_window}}\right) \tag{1}$$

Because there is a minimum of at least one step between entity engagements, a time_between_engagements value of 1.0 indicates that two entities converged on an opposing unit in the optimal interval, thus yielding a maximum reward of 1.0. Additionally, the reward_window term can be scaled to increase or decrease the time frame in which an entity receives any reward for massing, with the returned reward decreasing to 0.0 as

time_between_engagements increases. Although, since this reward function tracks the engagement times of two AI-controlled entities and ends once they both engage the same opponent, it was only used in this research for scenarios involving a single opponent.

The second combat model implemented was a probability of kill model. As the name suggests, this model resolves combat through generating a pseudo-random float uniformly distributed on [0,1) and comparing it to a pre-established kill value. Once one or more AI entities engage with an opposing unit, each entity has the opportunity to target and potentially destroy an opposing entity for every time-step it is in combat, regardless of whether or not it is in an action step. The basic algorithm for calculating probability of kill follows the corresponding logic:

```
if pKill ≥ random_float:
entity.target.alive = false
entity.target = none
return reward
```

The reward given to the agent for a successful hit can be any number; however, for this thesis a reward of +1.0 was returned for the destruction of an opposing entity and -0.6 for the destruction of an AI-controlled entity. After an entity is destroyed, it is removed from the environment and no longer has any influence on the scenario. Because this combat model does not rely on tracking times between entity engagements, it can be used in any combination of force ratios.

The final two combat models examined in this thesis are similar in principle, as they are both applications of Lanchester's equations for modern combat. The underlying assumptions for this combat model are that forces are homogenous in nature, able to exert perfect command and control, and capable of massing fires [19]. Considering all entities are treated as infantry rifle companies or platoons, operating in the open with no obstacles or cover, these assumptions are perfectly reasonable. Two variations of Lanchester's modern combat equations were implemented, a deterministic version and a stochastic version. Equation (2) illustrates how attrition is calculated for the deterministic model.

$$\text{opposing\_force\_attrited} = \text{combat\_efficiency} \times \text{force\_size} \tag{2}$$

Furthermore, Equation (3) describes attrition for the stochastic model, where random_float is uniformly distributed on the interval [0,1).

$$\text{opposing\_force\_attrited} = \text{combat\_efficiency} \times \text{force\_size} \times \text{random\_float} \quad (3)$$

Unlike the referee and probability of kill models, Lanchester equations require an entity to have a running force size in order to generate combat power. Depending on the scenario, entities in this study were assigned force sizes of either 150 or 50, which correlate to a company or platoon, respectively. Moreover, the combat efficiency represents the total opposing force attrited per friendly entity per unit time. Combat efficiencies can theoretically be any positive number, but most reasonable values are well below 1.0. For this thesis, a value of 0.05 was chosen for both AI- and non-AI-controlled entities. This value was low enough to observe effects on targets and provide sufficient time for entities to converge on opposing units, but high enough to keep episode lengths reasonably short.

Regardless of which Lanchester equation is used, the total offensive effect of each entity engaged in combat is calculated before it is applied. That is to say, attrition is not immediately applied to the targeted entity, as this would artificially reduce the combat power of any targeted entities whose offensive force is calculated thereafter. Rather, for each entity in combat, the attrition inflicted is first determined; the list of engaged entities then is reiterated, with offensive effects applied to respective targets. Once an entity's force size reaches zero, it is removed from the scenario and unable to influence future combat. The reward provided to the agent for destroying an opposing entity is based upon the ratio of remaining to original force size and scaled with the size of the entity destroyed. Equation (4) describes how this reward is calculated.

$$\text{reward} = \frac{\text{current\_size}}{\text{original\_size}} \times \frac{\text{target\_original\_size}}{150} \quad (4)$$

Consequently, an entity receives a larger reward if it is able to conserve more of its force size in combat. It also receives a larger reward if it destroys a larger unit, as the reward generated from destroying a platoon will be one-third the size of one from destroying a company. On the contrary, the penalty for the destruction of an AI-controlled entity was kept constant at –0.05. This particular value was selected for its performance in pilot

experiments, as algorithms that scaled the penalty with friendly losses did not produce satisfactory behaviors.

3. Reinforcement Learning Tools

The main reason for developing this training environment in Python was for its ease in connecting to external reinforcement learning resources. In particular, two resources were used throughout this research, Spinning Up and Gym. OpenAI develops and maintains both projects, as they are open-source tools designed to assist researchers in deep reinforcement learning. Although the two resources are formally separate, there are some interdependencies, as Spinning Up implements a reinforcement learning algorithm architecture designed to connect to Gym.

Spinning Up is described as an educational resource to introduce novice researchers to deep reinforcement learning [20]. Aside from the software package, it contains extensive documentation on the background of deep reinforcement learning, an explanation of popular algorithms, and references to key research papers and articles. The software itself contains the framework to connect environments with reinforcement learning algorithms and neural network libraries, such as Tensor Flow and PyTorch. Spinning Up primarily functions on Unix based operating systems, such as Ubuntu or MacOS, but solutions are available to run it in on Windows as well. Moreover, OpenAI recommends using a Python library called Anaconda, which comes with an environment manager that streamlines package and environment management. In addition, running Spinning Up requires Python3, Gym, and OpenMPI to be installed as well.

Complementing Spinning Up is Gym, a toolkit that allows researchers to compare reinforcement learning algorithms [21]. Gym comes with a standard library of test environments, including many classic reinforcement learning problems, but also allows users to construct their own. Agents employing Gym do not require any particular internal structure or format, as the primary purpose of Gym is to provide a common interface between environments and algorithms. Consequently, for each step that occurs in an environment, Gym returns four pieces of information: an observation specific to the environment, a reward, a Boolean indicating if an episode is complete, and diagnostic

information that does not impact agent learning. Additionally, action and observation spaces are standardized as well, supporting both discrete and continuous environments.

Once Spinning Up and the supporting software have been downloaded, individual trials can be run directly from the command terminal. In this case, the environment, learning hyper-parameters, and algorithm are all entered directly as arguments. After the trial is initiated, progress is printed directly to the terminal, summarizing basic metrics for each epoch of training completed. Upon ending, the trained neural network parameters, hyper-parameter configurations, and epoch metric data is saved to a respective folder. From this data, the performance of the trained agent and a plot of the results can be viewed. This method of running individual trials is not efficient for large-scale experimentation, but Spinning Up also includes a Python script that allows for full-factorial experiments to be designed and executed. In this regard, using the script to design experiments is simple, as desired values for each hyper-parameter are entered and saved. If a specific value is not entered for a particular factor, Spinning Up will resort to a default. Moreover, much like individual trial runs, the experiment can be initiated and progress tracked through the command terminal. The results for each experiment design point, to include replications for all seeds, are saved in respective folders.

B. NEURAL NETWORK

This research implemented feed-forward neural networks to train and control agents. Because Spinning Up's architecture is not well suited for using raw visual feed as state input, convolutional neural networks were not considered for this thesis. Spinning Up is specifically capable of implementing two machine learning libraries, TensorFlow and PyTorch, both of which are open source. PyTorch was chosen for use, although for the purposes of this project neither library is particularly superior to the other. Facebook AI Research laboratory is the primary developer of PyTorch, but it also has a large community of active contributors as well [22].

Vanilla Policy Gradient (VPG) and PPO were the main reinforcement learning algorithms applied during training; however, Trust Region Policy Optimization (TRPO) was also tested for utility as well. Both VPG and PPO are on-policy algorithms, albeit with

different approaches to learning. The basic premise behind VPG is to increase the probability of taking actions that lead to a higher reward while decreasing the likelihood of low return actions through small incremental steps [20]. Stochastic gradient ascent is applied to the policy (i.e. neural network) parameters until an optimal policy has been reached. On the other hand, PPO attempts to improve a policy through taking the largest steps possible [20]. In order to prevent over-stepping and collapsing performance, the change between new and old policies is limited with respect to the difference between probability distributions, this limiting factor is called the clip ratio. Both algorithms start with a high rate of exploration but gradually move to exploitation as the training regimen progresses.

1. State Space

The neural networks were provided four inputs for the state space. These included the absolute locations of itself, all red entities, all blue entities, and any engaged entities. To represent these positions, the map is discretized into a 10-by-10 grid, with a sensor located in the middle of each grid space. Figure 4 shows an example of how sensors are arranged and interrelated.

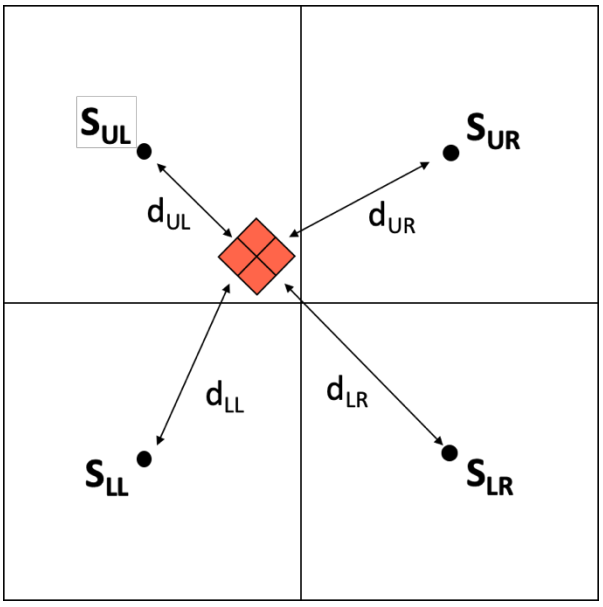

Figure 4. Illustration of Grid Space Sensor Arrangement

Using the upper left sensor (S_{UL}) as an example, a grid space is weighted based on the distance (d_{UL}) from an entity to its respective sensor. Equation (5) presents how these weights are calculated, where W_{UL} is the respective weight of S_{UL}.

$$W_{UL} = \text{Max}(1.0 - d_{UL}, 0.0) \tag{5}$$

Once the weights of all grid sensors have been calculated, they are then collectively normalized, as described in Equation (6). In this particular case, N_{UL} represents the normalized weight of S_{UL}.

$$N_{UL} = \frac{W_{UL}}{W_{UL} + W_{LL} + W_{UR} + W_{LR}} \tag{6}$$

The normalized weights are used to convey the global position of an entity, as grid spaces with a non-zero weight are applied a gradient shading of color relative to the magnitude of their weight. For example, if the normalized weight for the upper left sensor (N_{UL}) is 1.0, then it can be inferred that the entity is positioned in the center of the grid. In this case, that single square would be darkly shaded. On the other hand, if all four sensors are weighted equally ($N_{UL} = N_{LL} = N_{UR} = N_{LR} = 0.25$), then the entity is located on the intersection of four grid squares, and all four spaces would be lightly shaded.

The color of shading applied depends on the nature of the state input being represented. As such, an entity's own position will be grey, but the positions of all red entities and blue entities will be shaded red and blue, respectively. The following figures demonstrate how various force layouts would be represented to the neural network for each state input.

Figure 5 displays how an active entity's position is conveyed to the neural network. In this situation, only the single active entity will be considered in weighing grid squares, which is indicated with grey shading. Moreover, for the convenience of a human viewer observing neural network performance, the active entity is indicated on the main widget by a lighter shade of red.

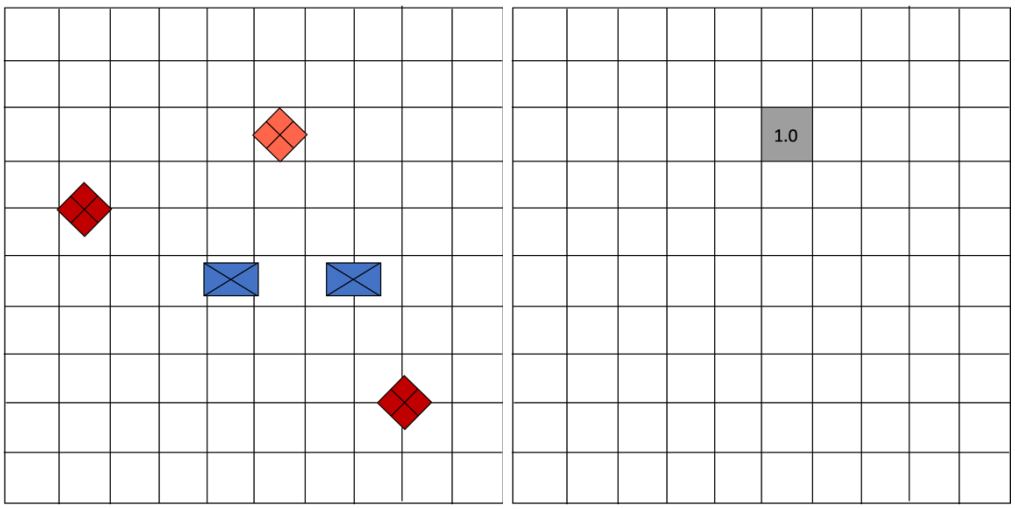

Figure 5. State Representation of Active Entity's Own Position

As depicted above, the active entity is toward the top, middle section of the environment. Because the entity is in the middle of the grid space, the total weight given to that space is 1.0 and only the respective single square is shaded.

Figure 6 below shows how the same force layout would be represented with respect to the positions of all red entities. As might be implied, only red entities are considered for calculating the weight of grid spaces. Moreover, red entities that are in close proximity to each other can add to the weights of the grid spaces they influence.

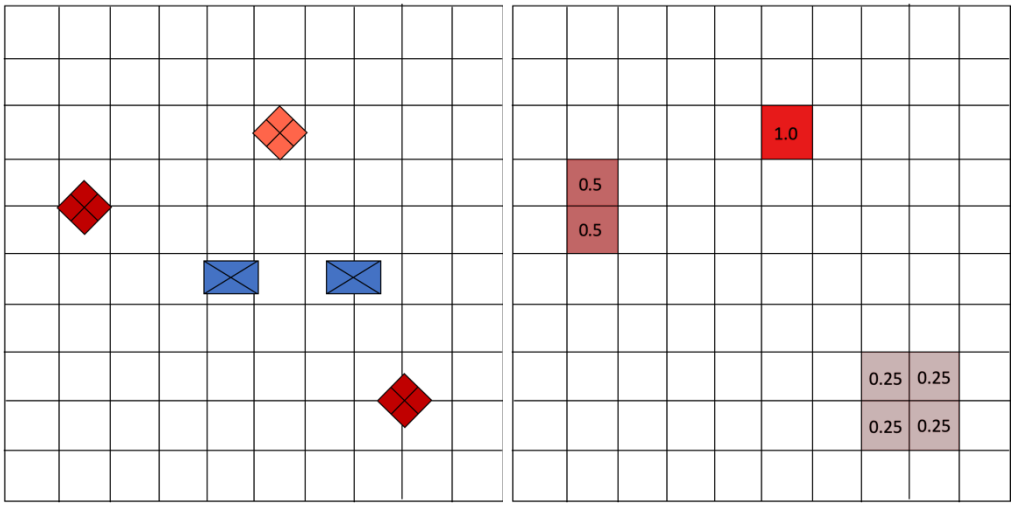

Figure 6. State Representation of Red Entity Positions

As portrayed above, there are three red entities active in the environment, to include the entity currently in its action step. Each entity demonstrates how it's position on or near a border impacts the weights of surrounding grid spaces. The red entity to the far left sits on the horizontal border of two grid spaces, thus distributing weight equally between them. Meanwhile, the entity to the bottom right rests on four corners, influencing each of the four grid spaces, respectively.

Keeping with the same force layout, Figure 7 shows how blue entities would be represented for state input. This is similar to the previous example, except that only blue entities are considered.

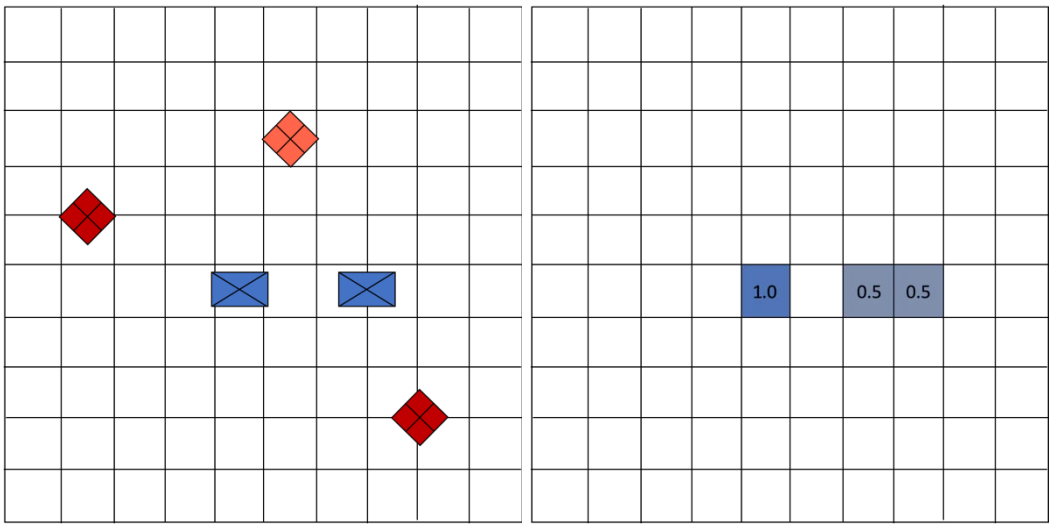

Figure 7. State Representation of Blue Entity Positions

Depicted above are two blue entities active in the environment. Similar to previous cases, their positions relative to adjacent grid spaces influence the weights and shading of surrounding squares.

Figure 8 portrays a different scenario from the ones previously discussed above, as three red entities and one blue entity are engaged in combat. In representing the state input for engaged entities, all entities that currently have targets are including in weighing grid spaces, regardless of allegiance.

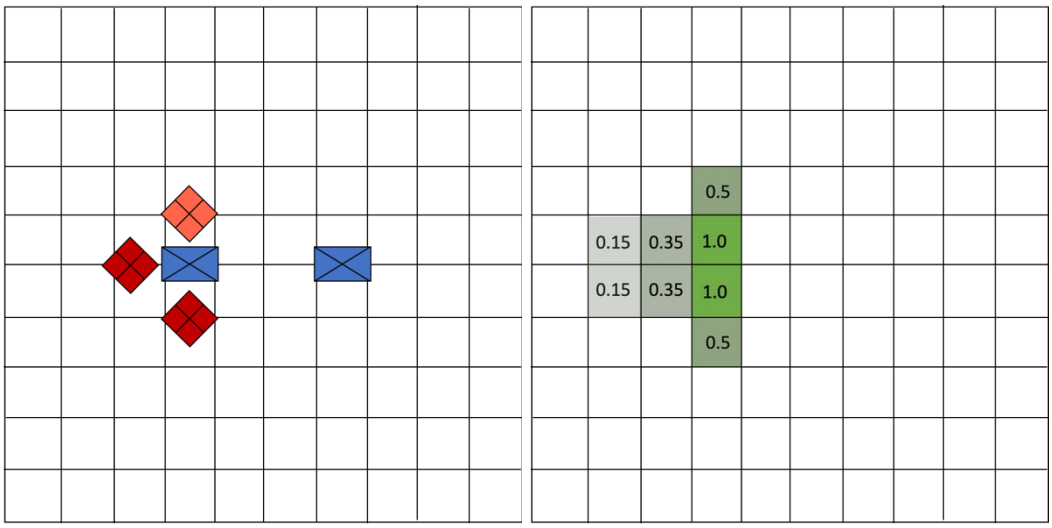

Figure 8. State Representation of Engaged Entities

The above figure shows how units that are crowded together can collectively influence the cumulative weight of individual grid spaces. The single engaged blue entity lies on the horizontal border between two grids, distributing its weight equally between them. However, the two red units to its north and south also contribute half of their weights to the two squares. As a result, the weights in those respective grid spaces now sum to 1.0. In total, this creates a sort of heat map, in which the darker color represents a higher density of entity presence.

2. Action Space

Provided the inputs discussed in the previous section, neural networks have seven options in the action space. Agents have the ability to move in six possible directions, similar to how units might move in a hexagonal board game. Starting from a northerly direction (0 degrees), these six directions consist of the azimuths observed while moving in 60-degree increments around a compass. In addition to these six directions, waiting, or no action, is also a possible choice as well. Considering the combat models implemented make no provisions for logistics and agents are not penalized for their number of moves, there is no explicit benefit for an agent to wait. Moreover, there is no action to engage in combat, as combat begins automatically once two or more opposing units move within range of each other.

3. Training Parameters

The training hyper-parameters varied between the different reinforcement learning algorithms used; however, there were several common factors adjusted. These included the learning rate, training duration, and discount factor. Five values were tested for the learning rate, which included 0.01, 0.005, 0.001, 0.0005, and 0.0001. Moreover, neural networks were trained for durations of 150, 500, or 1000 epochs. For most trials the discount factor was kept at 0.99, although it was modified between 0.965 and 0.995 for certain experimental designs. Furthermore, for experiments implementing PPO, the clip ratio was modified based on whether the combat model was deterministic or stochastic. For deterministic experiments, the clip ratio was increased to 0.3 in order to facilitate faster learning. On the other hand, for stochastic experiments, the clip ratio was limited to 0.1 in order to prevent negative experiences from disproportionately impacting otherwise good behavior. With respect to the neural network configuration, the activation function was kept constant through all experiments, as rectified linear units were used as opposed to sigmoid or hyperbolic tangential functions. The number of hidden units were adjusted to implement one or two layers of 32, 64, or 96 hidden units.

C. TRAINING SCENARIOS

A total of three different training scenarios were examined in this research, with each scenario capable of applying different combat models. The scenarios consisted of two red companies versus one blue company, two red companies versus two blue companies, and three red companies versus one blue company and one blue platoon. During these scenarios, the starting position of the blue entities remained constant, although the initial positions of the red entities could be held fixed in a particular formation or uniformly distributed at varying distances across the six previously discussed directions.

1. Two Red Companies versus One Blue Company

The first scenario consisted of a two-versus-one configuration, in which the primary objective was to train the two red companies to consolidate forces in a synchronized manner against the single blue company. Multiple force laydowns were used in training for

this scenario, as Figure 9 displays the initial force layout with the red companies in a column formation to the north.

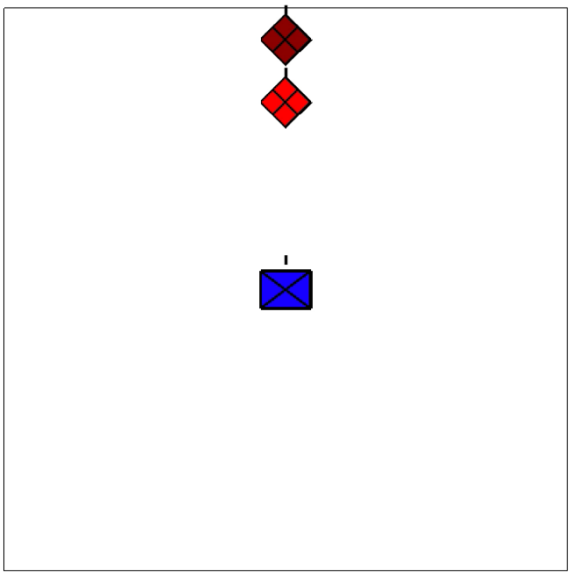

Figure 9. Two Red Companies versus One Blue Company

The column formation served as a challenging initial force layout for the red companies, as the agent must learn to deconflict unit movements and avoid blocking counterparts during approach. The simplest initial force layout for the red companies consisted of one red company to the north and another red company to the south east, equally distant from the blue opponent. On the contrary, the most difficult layout randomly placed the red companies across the map for each iteration. For this scenario, a single episode would end when either a single red or blue company was destroyed or a red entity moved outside of the environment bounds.

2. Two Red Companies versus Two Blue Companies

Expanding on the first scenario, the second scenario comprised of a two-versus-two configuration. In this case, the optimal behavior for the red companies was to organize into a single effort and sequentially defeat the two opposing blue companies. Again, different

initial force layouts were implemented during training, with Figure 10 presenting an example of how the red companies might initialize in a randomly generated formation.

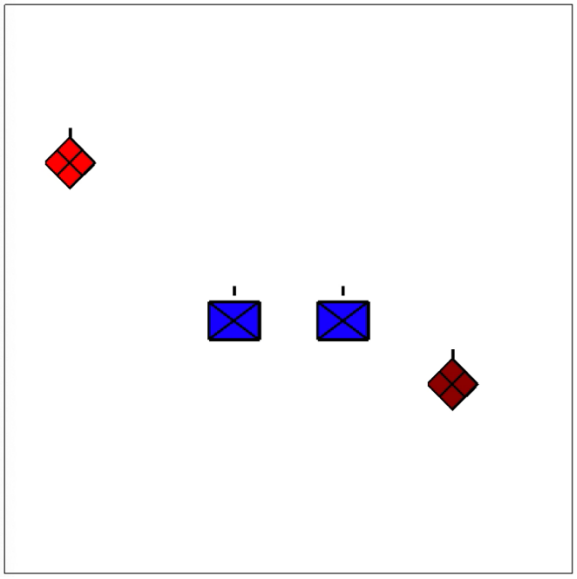

Figure 10. Two Red Companies versus Two Blue Companies

The random force layout displayed above demonstrates a particularly difficult situation, since the two red companies are split between the blue formation. In this case, one red unit must learn to move across the training environment to engage the same opponent as its counterpart, as opposed to simply engaging the closest blue company. Moreover, it is also possible for two blue companies to simultaneously attack a single red company if it takes a poor approach. The other force layout tested positioned two red companies in a column to the north at the beginning of each engagement, although this still requires the agent to learn to deconflict movements. Overall, in this scenario, an episode would terminate once two blue companies or one red company were destroyed or a red unit moved out of bounds.

3. Three Red Companies versus One Blue Company and Platoon

The final scenario entailed a three-versus-two force configuration, in which three red companies opposed a single blue company and platoon. Similar to previous scenarios,

the most optimal course of action was for all three red companies to sequentially destroy the blue company then platoon. However, the reward return between this strategy and other inferior ones was not significantly different. Additionally, depending on the discount factor, it may be more beneficial for the three red companies to separate into two groups in order to simultaneously destroy both the opposing company and platoon. Thus, this scenario was used to discern and validate the difference in behaviors between mass and economy of force. Figure 11 below shows the standard initial force layout for this scenario.

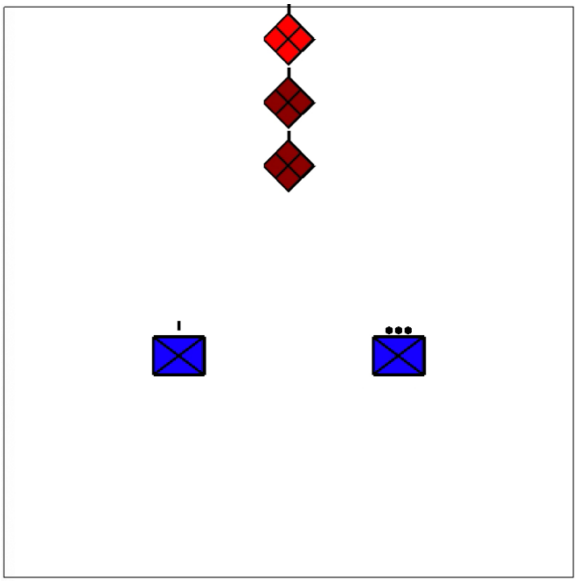

Figure 11. Three Red Companies versus One Blue Company and Platoon

The initial force layout shown above was used for all engagements in this scenario, as the distance between the two blue units limited the ability to randomly place the red companies. Nevertheless, the intent of this scenario was to observe the differences in employment of the red companies, as the two blue units are far enough away that two distinct engagements must be fought. Consequently, the agent has the option of defeating each blue unit sequentially, elongating the total battle; or breaking its forces into two groups and defeating the blue units at the same time, shortening the overall length of battle. For this scenario, an episode would end if both blue units were destroyed, two red companies were destroyed, or a red company moved outside of the map boundaries.

D. PERFORMANCE EVALUATION

The overall objective of this thesis was to evaluate trained agents in their ability to exhibit optimal behaviors in tactical scenarios, where ideal performance was consistent with the principles of mass and economy of force. The average discounted and undiscounted rewards were used to compare and assess the performance of neural networks; however, these statistics only provided partial understanding of the final behavior learned. Consequently, other metrics were also analyzed in order to determine the quality of an agent's performance. These included the mean time between entity engagements and the mean episode length, with each supporting a particular tactical principle.

The average reward and mean time between entity engagements were most helpful in assessing an agent's ability to display synchronized teamwork and mass. The average undiscounted reward indicated which hyper-parameter sets performed best and could be compared to the theoretical values of perfect performance. On the other hand, the mean time between engagements specified the time gap between which red entities were engaging blue opponents. Thus, a mean time between engagements of 1.0 would indicate that an agent had learned perfect massing behavior. However, it should be noted that perfect performance was strictly evaluated with respect to synchronization and timing. Tactical units converging on an enemy would realistically target an offset angle of approximately ninety degrees, ensuring their fires were deconflicted to prevent fratricide. Since there is no representation of fratricide in the combat models used, approach angles did not influence rewards or performance.

In evaluating an agent's behavior where economy of force is appropriate, the average discounted reward and mean episode length were the primary metrics examined. Because exhibiting economy of force may not result in the largest possible un-discounted reward, the neural networks achieving the greatest average rewards may not be the highest performing. Accordingly, the discounted reward was compared to the theoretical values of ideal economy of force execution. Moreover, the mean episode length indicated how quickly an agent was engaging and destroying opponents. An agent that had learned to divide its forces and separately attack opponents would produce shorter episode lengths

than an agent that consolidated combat power on each opponent sequentially. However, mean episode length must also be carefully considered alongside average reward, since a poorly performing network may also have short episode lengths.

IV. RESULTS

This chapter will discuss in detail the performance and behaviors of the trained neural networks. The first section explores the two force-consolidation scenarios and the learning and neural network parameters that generated the best performance. Next, in the second section, the economy of force scenario is reviewed and the optimal training factors considered. Lastly, the final section assesses the performance of the various reinforcement learning algorithms tested. Overall, all objectives were met with success, as agents demonstrated both tactical traits of mass and economy of force. This is displayed in the numerical results as well as the observed performance of trained neural networks.

Two statistical methods were used to analyze results and detect significant trends within and between data sets. For each unique set of data belonging to a particular combat model, reinforcement learning algorithm, and force scenario, a standard least squares regression model was fit to the data and an effect test conducted to identify important factors. These were completed using JMP Pro 15 statistical software [23]. Moreover, two-sample t-tests were conducted to compare the performance of algorithms implementing the same combat model and force scenario. For these cases, the alpha level was uniformly set to 0.05 and compared to the two-tailed p-value. Additionally, in this chapter, all mean point values are presented with their respective standard deviations.

A. MASS

The first method of assessing the ability of reinforcement learning to attain optimal performance was through the principle of mass. This was achieved in two distinct scenarios, using a combination of combat models and reinforcement learning methods. The first scenario examined consisted of two red companies versus one blue company.

1. Two Red Companies versus One Blue Company

For the two-versus-one configuration, a full factorial experiment was conducted with 480 design points. Each design point was replicated five times, for a total of 2400 trials. Table 1 displays the factors and their corresponding levels.

Table 1. Factors and Levels for Two-versus-One Mass Experiment

Factor	Levels
Learning Rate	0.01, 0.005, 0.001, 0.0005, 0.0001
Hidden Units	32, 2 x 32, 64, 2 x 64, 96, 2 x 96
Training Duration	500, 1000
Learning Algorithm	PPO, VPG
Combat Model	PKill, Referee, Deterministic and Stochastic Lanchester

For this experiment, trained neural networks were evaluated with respect to their undiscounted average reward and the percentage of engagements in which perfect teaming was observed. Because every combat model implemented its own unique reward system and method of combat resolution, each was evaluated separately. Moreover, trained neural networks were also broken-out based on reinforcement learning algorithm, as hyper-parameter sets will produce different results between algorithms. However, in both cases comparisons could be made as to the rate at which perfect teaming was observed. Figure 12 shows an example of a trained agent executing perfect massing behavior.

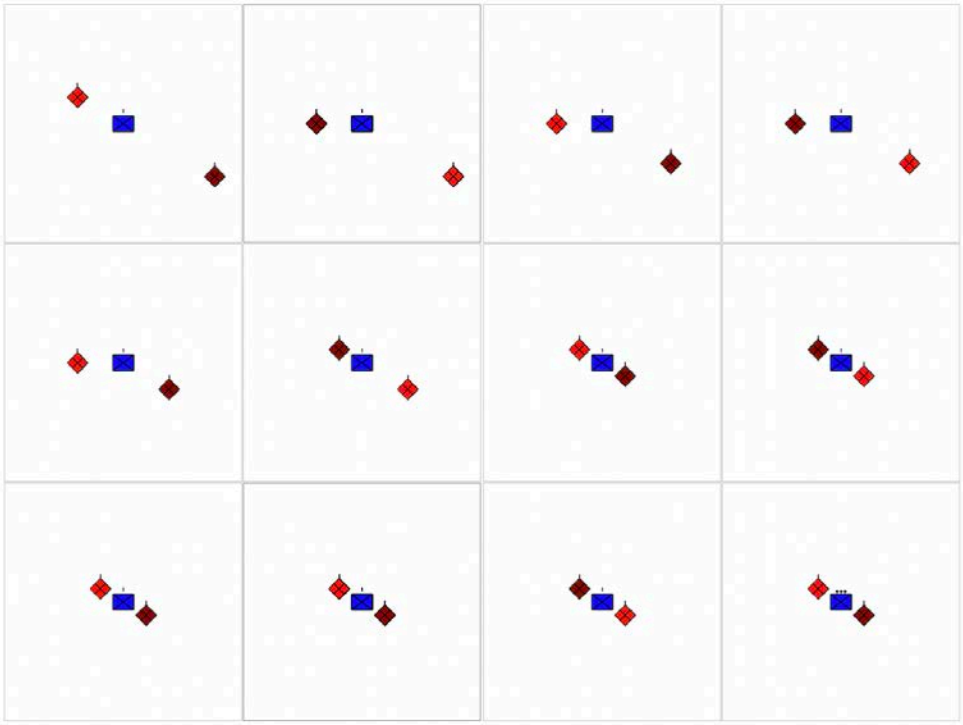

Figure 12. Red Entities Demonstrating Perfect Mass (Two-versus-One)

As can be observed, the red companies engaged the blue company in sequence, with the time difference between engagements equal to one time-step. Also, of note, the red company to the west waited in place several time-steps until its counterpart to the southeast was within engagement range.

a. Probability of Kill Combat Model

The first combat model used to train neural networks was probability of kill. The average results and standard deviations of the top five performing networks trained through PPO and VPG are presented in Table 2 and Table 3, respectively

Table 2. Probability of Kill Model Top Performing Neural Networks (PPO)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	2 x 64	0.0005	1000	0.435 ± 0.038	67.8 ± 2.3
2	96	0.005	1000	0.422 ± 0.017	72.6 ± 1.7
3	32	0.01	1000	0.429 ± 0.029	71.6 ± 3.1
4	2 x 96	0.01	1000	0.424 ± 0.032	68.8 ± 4.9
5	32	0.0005	1000	0.411 ± 0.027	67.8 ± 1.7

For agents trained through PPO, the most obvious trend in the data was the impact of training duration, as longer training produced better performing neural networks. Additionally, the learning rate was also found to have a significant impact, as larger learning rates resulted in better performance. In regard to the size and number of hidden layers, there were no statistically significant trends.

Table 3. Probability of Kill Model Top Performing Neural Networks (VPG)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	64	0.005	1000	0.436 ± 0.027	65.1 ± 3.6
2	32	0.01	1000	0.418 ± 0.022	65.8 ± 2.9
3	96	0.005	1000	0.405 ± 0.027	63.8 ± 6.1
4	64	0.01	1000	0.402 ± 0.024	62.9 ± 3.8
5	96	0.01	1000	0.400 ± 0.040	62.2 ± 3.9

Similar to the networks trained through PPO, agents trained with VPG significantly benefited from higher learning rates. Further, longer training durations appeared to have a positive influence on performance, although it was not found to be statistically significant. Moreover, single layer networks dominate the top five parameter configurations, but, ultimately, no statistical difference could be identified with respect to the size and number of hidden layers.

Between the two reinforcement learning algorithms, the common factor was the learning rate, as higher learning rates supported better performance. In order to assess the difference between algorithms in terms of average reward and percent perfect teaming observed, two-sample t-tests were performed at an alpha value of 0.05, with the null hypothesis that the difference between algorithms for both metrics would be zero. On the contrary, the alternate hypothesis was that there would be a non-zero difference between metrics. With respect to average undiscounted reward, the two algorithms were indistinguishable, as a two-sample t-test assuming equal variances produced a two-tailed p-value of 0.1570. On the other hand, when considering the average percentage of perfect teaming observed, a two-sample t-test assuming equal variances resulted in a two-tailed p-value of 0.0014. In this case, the null hypothesis was rejected as the perfect teaming percentage for neural networks trained with PPO were higher than those trained with VPG.

b. Referee Combat Model

The next combat model examined was the referee model. The top performing hyper-parameter mean results and standard deviations for VPG are displayed in Table 4.

Table 4. Referee Model Top Performing Neural Networks (VPG)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	96	0.01	500	0.973 ± 0.023	97.1 ± 2.4
2	96	0.01	1000	0.968 ± 0.032	96.6 ± 3.2
3	64	0.01	1000	0.965 ± 0.029	96.5 ± 2.9
4	2 x 96	0.01	1000	0.961 ± 0.019	95.6 ± 1.9
5	2 x 96	0.01	500	0.958 ± 0.022	95.4 ± 2.5

For neural networks trained with VPG, the only statistically significant factor was the learning rate, as larger learning rates contributed to better performance. Otherwise, the training duration and the size and number of hidden layers had no discernible impact on performance.

Of note, there is no table displaying the top results for neural networks trained with PPO. This is because 48 out of the 60 configurations tested were able to learn the perfect policy; that is, these trained agents demonstrated perfect teaming and attained the maximum reward 100 percent of the time. Consequently, it was difficult to further rate and assess the performance of these perfect performing neural networks in order to provide them a numerical rank. Moreover, an analysis of the high-performing agents did not reveal any statistically significant trends in regards to training duration, learning rate, or neural network configuration. Likewise, this also applied to the ten configurations that did not achieve perfect performance, as there were no obvious indications for the inferior results. This suggests that each neural network configuration was capable of attaining the ideal policy for this combat model.

Overall, for the referee model, PPO substantially outperformed VPG. This was confirmed statistically in two-sample t-tests assuming equal variance that compared both average reward and perfect teaming percentages. The alpha level for these tests was set 0.05, with the two null hypotheses stating that the difference in respective metrics between training algorithms would be zero. On the other hand, the two alternative hypotheses were that there would be a non-zero difference in metrics. Ultimately, both tests rejected their null hypothesis, as two-tailed p-values of 2.42×10^{-6} and 6.5×10^{-7}, respectively, were produced for the comparisons between perfect teaming percentage and reward.

c. Deterministic Lanchester Combat Model

Subsequently, neural networks were trained with a deterministic Lanchester combat model. Table 5 and Table 6 contain the training configurations and average results of the top performing neural networks trained with PPO and VPG, respectively.

Table 5. Deterministic Lanchester Combat Model Top Performing Neural Networks (PPO)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	2 x 96	0.0005	1000	0.810 ± 0.002	91.5 ± 2.0
2	96	0.001	1000	0.809 ± 0.003	91.5 ± 3.2
3	2 x 32	0.001	1000	0.809 ± 0.004	91.2 ± 3.9
4	2 x 64	0.0005	1000	0.809 ± 0.001	91.1 ± 1.6
5	32	0.005	1000	0.809 ± 0.002	90.9 ± 2.5

The main trend detected for agents trained with PPO was the training duration, as longer training resulted in better performance. Moreover, neural networks with one layer of 64 hidden units also generated a statistically significant negative influence. Otherwise, the learning rate was not found to have an impact.

Table 6. Deterministic Lanchester Combat Model Top Performing Neural Networks (VPG)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	32	0.01	1000	0.810 ± 0.002	93.0 ± 2.4
2	2 x 32	0.01	1000	0.809 ± 0.002	92.2 ± 2.0
3	2 x 32	0.01	500	0.809 ± 0.003	91.0 ± 2.7
4	2 x 32	0.005	1000	0.809 ± 0.002	90.9 ± 1.5
5	2 x 96	0.01	1000	0.801 ± 0.013	88.7 ± 4.5

For networks trained with VPG, the learning rate and training duration were assessed to be statistically. significant, as longer training and higher learning rates produced better performance. Despite the fact that most of the top five performing agents had two layers of 32 hidden units, no correlation was found in regards to number and size of hidden layers.

Overall, neural networks trained with both PPO and VPG in the deterministic Lanchester combat model performed well, as perfect teaming percentages were in the low 90s. A two-sample t-test assuming equal variances was conducted to compare the difference in average reward and perfect teaming percentages between algorithms. For these tests, the alpha value was set to 0.05. The two null hypotheses were that the difference in the two metrics, percent perfect teaming and average reward, between PPO and VPG

would be zero. The two alternative hypotheses were that there would be a difference. In both cases the null hypothesis was retained, as two-tailed p-values of 0.912 and 0.424, respectively, were generated for percent perfect teaming and average reward. Consequently, it was concluded that there was no difference in training performance with respect to the two reinforcement learning algorithms implemented.

d. Stochastic Lanchester Combat Model

The last combat model implemented for the two-versus-one configuration was the stochastic Lanchester combat model. The top five performing neural networks for PPO and VPG are presented in Table 7 and Table 8, respectively.

Table 7. Stochastic Lanchester Combat Model Top Performing Neural Networks (PPO)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	2 x 96	0.001	1000	0.837 ± 0.001	96.1 ± 1.4
2	32	0.005	1000	0.837 ± 0.001	94.5 ± 1.5
3	2 x 64	0.001	1000	0.836 ± 0.002	95.9 ± 1.4
4	2 x 32	0.005	1000	0.835 ± 0.002	94.1 ± 1.2
5	64	0.01	1000	0.835 ± 0.005	93.8 ± 2.4

For agents trained with PPO, the training duration and learning rate impacted results, as larger learning rates and longer training improved performance. However, no trends were apparent with respect to the number and size of hidden layers.

Table 8. Stochastic Lanchester Combat Model Top Performing Neural Networks (VPG)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	2 x 64	0.005	1000	0.835 ± 0.002	86.0 ± 1.6
2	96	0.01	1000	0.833 ± 0.002	87.8 ± 2.6
3	96	0.005	1000	0.833 ± 0.002	83.8 ± 3.0
4	64	0.01	1000	0.832 ± 0.002	85.2 ± 3.6
5	2 x 96	0.005	1000	0.827 ± 0.011	82.6 ± 9.4

Similar to PPO, neural networks trained with VPG benefited from higher learning rates and longer training durations. No statistically significant trends were identified in regards to number and size of hidden layers.

Between PPO and VPG, there was a clear difference in performance. Neural networks trained with PPO performed very well, even better than their deterministic counterparts, with perfect teaming percentages in the upper-90s. On the contrary, neural networks trained with VPG, while still performing well, only reached perfect teaming percentages in the mid-80s. In order to confirm this finding, two-sample t-tests assuming equal variances were conducted to compare the difference in average reward and perfect teaming percentages. For both tests, the alpha level was set to 0.05. The two null hypotheses were that the difference between metrics would be zero, and the two alternative hypotheses were that there would be a non-zero difference. In both cases the null hypothesis was rejected, as two tailed p-values of 0.0167 and 1.07×10^{-5}, respectively, were produced for average reward and perfect teaming percentage. Consequently, it could be concluded that agents trained with PPO outperformed VPG in both metrics.

2. Two Red Companies versus Two Blue Companies

Once agents had successfully demonstrated optimal behaviors in a simple two-versus-one scenario, the next task was to achieve similar results in a two-versus-two configuration. For this scenario, 120 design points were replicated five times for a total of 600 trials. Table 9 contains the factors and their respective levels for the full-factorial experimental design.

Table 9. Factors and Levels for Two-versus-Two Mass Experiment

Factor	Levels
Learning Rate	0.01, 0.005, 0.001, 0.0005, 0.0001
Hidden Units	32, 2 x 32, 64, 2 x 64, 96, 2 x 96
Training Duration	500, 1000
Learning Algorithm	PPO
Combat Model	Stochastic Lanchester, Deterministic Lanchester

This experiment only implemented the stochastic and deterministic Lanchester models, as the referee model was only designed for scenarios with one blue opponent. Moreover, the probability of kill model was excluded for the inferior results it had produced in comparison to the Lanchester models in the two-versus-one scenario. Additionally, PPO was the sole reinforcement learning algorithm applied to train neural networks. Similar to before, trained agents were evaluated with respect to average award attained and percent perfect teaming observed. An example of an agent executing perfect massing behavior is displayed in Figure 13.

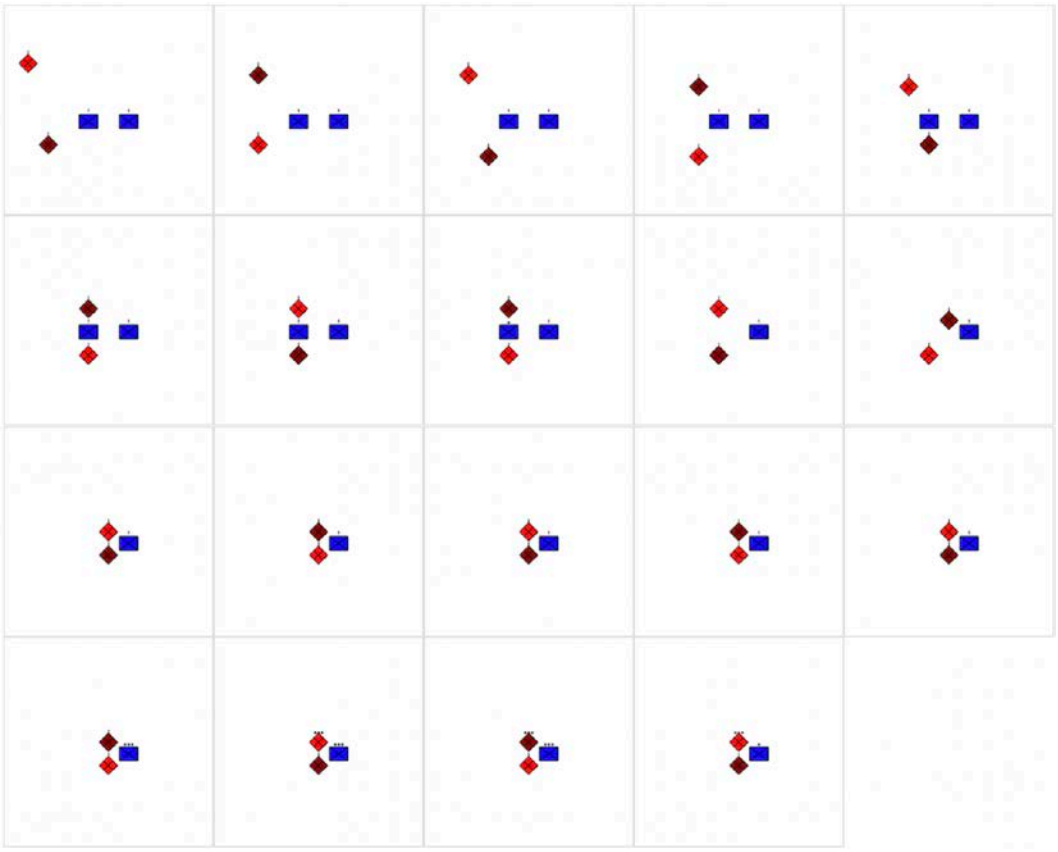

Figure 13. Red Entities Demonstrating Perfect Mass (Two-versus-Two)

As demonstrated above, the two red companies approached and engaged the first blue company to the west, with the red company to the southwest waiting in place for a

time-step until its counterpart could close in. After destroying the first blue company, they again sequentially converged on and destroyed the final blue company to the east.

For the two-versus-two scenario, neural networks were first trained with a deterministic Lanchester combat model. The mean results and standard deviations for the top five performing agents are presented in Table 10.

Table 10. Deterministic Lanchester Combat Model Top Performing Neural Networks (Two-versus-Two)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	96	0.001	500	1.404 ± 0.006	97.2 ± 0.8
2	96	0.001	1000	1.404 ± 0.009	96.9 ± 1.2
3	96	0.005	1000	1.402 ± 0.013	96.6 ± 2.2
4	2 x 64	0.005	1000	1.402 ± 0.008	96.1 ± 2.6
5	64	0.01	1000	1.398 ± 0.013	95.9 ± 3.2

The results above show that neural networks were generally able to learn the ideal behavior, as the perfect teaming percentages approach 100 percent. With respect to the training parameter configuration, the learning rate was the only factor found to be statistically significant, with smaller learning rates performing better.

After completing training with the deterministic Lanchester combat model, agents were subsequently trained with the stochastic version. Table 11 contains the mean results and standard deviations for the top five performing neural networks.

Table 11. Stochastic Lanchester Combat Model Top Performing Neural Networks (Two-versus-Two)

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Learning Rate	Duration		
1	32	0.005	1000	1.460 ± 0.014	88.5 ± 6.1
2	2 x 32	0.0005	1000	1.460 ± 0.007	87.3 ± 3.7
3	2 x 96	0.001	1000	1.460 ± 0.007	83.0 ± 4.2
4	2 x 32	0.001	1000	1.454 ± 0.006	88.2 ± 4.9
5	2 x 96	0.0005	1000	1.454 ± 0.006	83.8 ± 2.3

The data above reveals that agents learned the principle of mass reasonably well, as perfect teaming percentages approach 90 percent, but not as well as those trained in the deterministic environment. In regards to training and network parameters, training duration and learning rate were assessed to have a statistically significant impact on performance, with longer training and larger learning rates generally producing better behavior. In regards to number and size of hidden layers, no trends could be identified.

In total, neural networks were able to learn ideal massing behavior in both deterministic and stochastic Lanchester combat models, although the factors that impacted performance differed between them. In regards to the perfect teaming percentages, a two-sample t-Test assuming equal variances was conducted at an alpha value of 0.05 to compare the data. As such, the null hypothesis was that there was no difference in perfect teaming percentages between the two combat models. The alternate hypothesis was that there was a difference in perfect teaming percentages. The test produced a two-tailed p-value of 2.23×10^{-5}, therefore the null hypothesis was rejected and it could be confirmed that agents trained in the deterministic Lanchester combat model achieved higher perfect teaming percentages. However, while the agents trained in the stochastic combat model did not attain as high of perfect teaming percentages, their average rewards were higher. Similar to an effect that can also be observed in the two-versus-one scenario, this is a byproduct of the difference between the deterministic and stochastic attrition functions. Because of these fundamental differences, no statistical tests were performed to compare that particular metric.

For the consolidation of force scenarios, agents were effectively able to learn ideal teaming behavior. Although no trends could be consistently identified with respect to hyper-parameter configurations, the reinforcement learning algorithm and type of combat model applied to training were found to have a significant impact on performance. Between the two reinforcement learning algorithms, PPO generally produced better performance, although its advantage was mainly observed in stochastic combat models. On the other hand, agents trained in deterministic combat models, as a whole, performed superior to those trained in stochastic environments.

B. ECONOMY OF FORCE

After validating reinforcement learning performance in consolidation of force scenarios, the next experiment examined both the ability to learn economy of force and the factors that contributed to success. This entailed implementing a full-factorial experiment consisting of 210 design points, with the factors and levels listed in Table 12. All design points were replicated five times for a total of 1050 trials.

Table 12. Factors and Levels for Economy of Force Experiment

Factor	Levels
Learning Rate	0.01, 0.005, 0.001, 0.0005, 0.0001
Hidden Units	32, 2 x 32, 64, 2 x 64, 96, 2 x 96
Discount Factor	0.995, 0.99, 0.985, 0.98, 0.975, 0.97, 0.965

For this part of the research, the force configuration remained constant through all scenarios, with the red entities in a column formation and the blue entities in a line formation. Moreover, a single combat model was used – the deterministic Lanchester equation – which had consistently produced better performance than other combat models in previous experiments. This was decided because the focus was directed toward eliciting different behaviors from agents, as opposed to testing the robustness of learning in varying environments.

The behavior produced from this experiment took two different forms. The first form consisted of massing, in which all three red companies successively defeated the blue opponents. The second form demonstrated economy of force. In this case, two red companies synchronously attacked the blue company while, simultaneously, the third red company attacked the blue platoon. The main factor that separated these two different behaviors was the discount factor. Higher discount factors produced agents that favored mass and consolidation of force, whereas, lower discount factors trained agents that displayed economy of force. A stepwise depiction of how agents implemented mass and economy of force are illustrated in Figure 14 and Figure 15 below, respectively. Of note, both of these figures display the optimal application of their corresponding tactical principles.

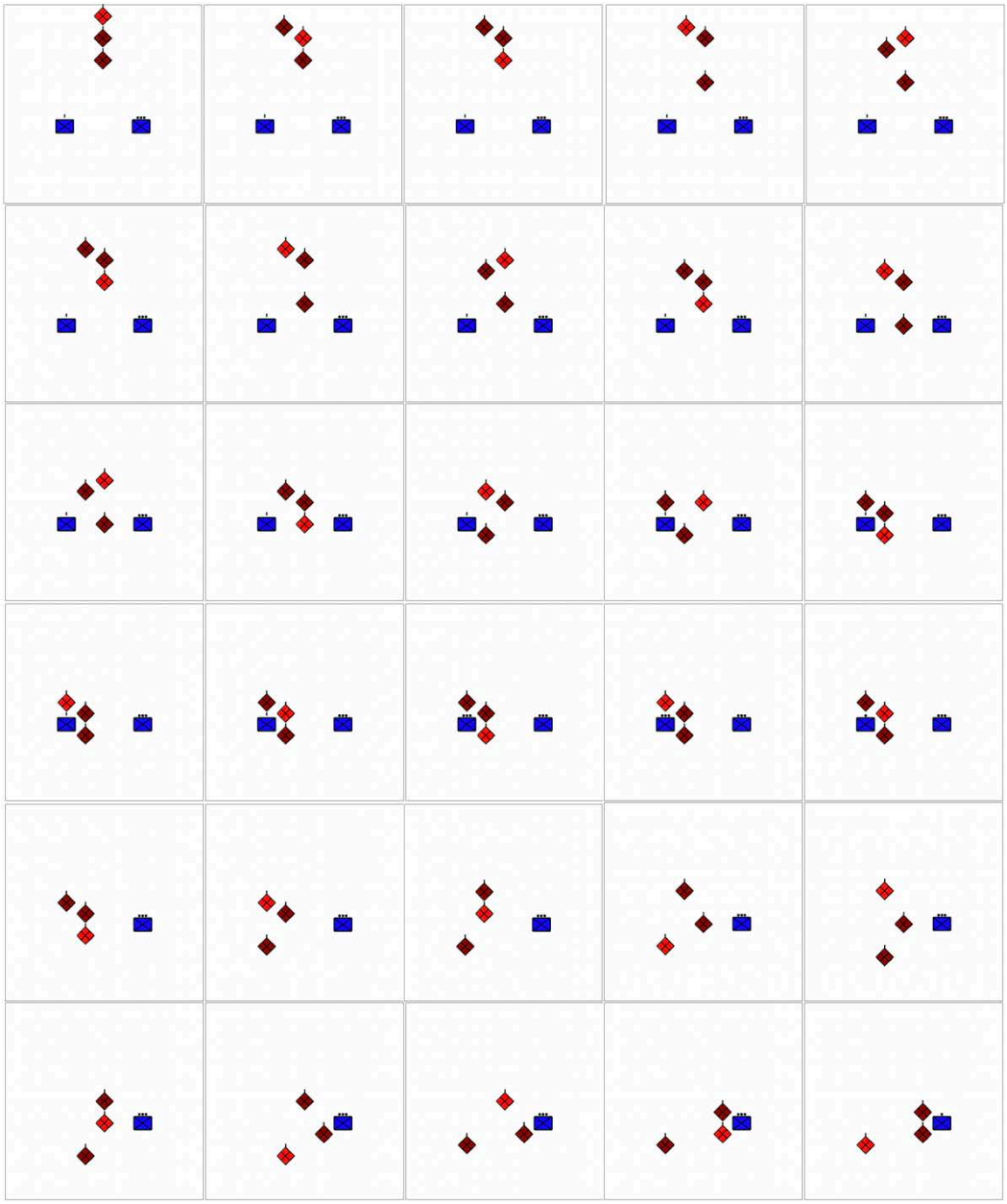

Figure 14. Red Entities Demonstrating Optimal Massing Strategy

In Figure 14, it took the three red companies 30 steps to defeat the two opposing blue units. The blue company is destroyed at step 20 and the blue platoon at step 30. Since this scenario implemented the deterministic Lanchester combat model, net rewards of

0.918 and 0.325 were yielded, summing to a total undiscounted reward of 1.243. Of particular interest, only two red entities engaged the final blue platoon, as the third red entity could not engage in time to effectively contribute.

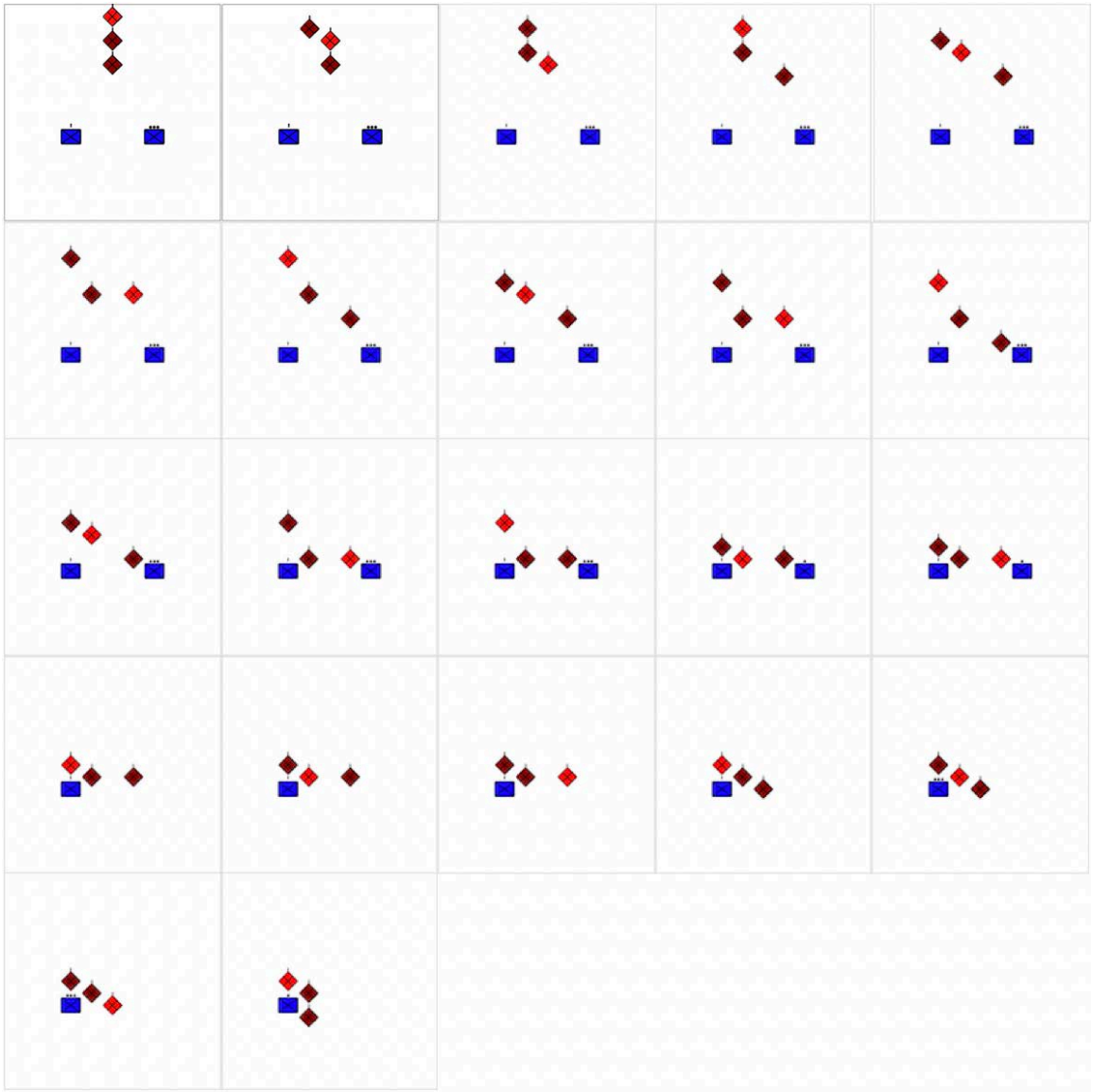

Figure 15. Red Entities Demonstrating Optimal Economy of Force

In Figure 15, the total engagement time was 22 steps. A single red company destroyed the blue platoon at step 15, and two red companies, assisted at the end by the

third red company, destroyed the blue company soon after, at step 22. This correlated to net rewards of 0.312 and 0.865, respectively, for a total undiscounted reward of 1.177.

In order to identify the exact cause for this difference in behavior, the discounted reward was calculated in both tactical cases for each discount factor value tested. That is, the actual reward the agent received when including time penalties. The baseline net reward for the two behaviors was calculated from the theoretical performance optima as defined by each principle, which were also exemplified in Figure 14 and Figure 15. These theoretical values were calculated by hand, considering the minimum time it takes for the entities to both converge on and defeat respective opponents. Movements were executed in the same discrete, hex-wise manner as the combat simulation, with one entity moving during each turn. Attrition and combat were resolved at the end of each turn, considering all of the entities that were presently engaged. Moreover, because a deterministic Lanchester combat model was implemented, there was no variance in combat outcomes. Entities employing the optimal tactic always won their engagements and obtained the maximum possible reward.

Based on undiscounted reward only, the most effective approach for the agent to take was a massing tactic, with a maximum attainable reward of 1.243 over 30 steps. On the contrary, applying economy of force resulted in a maximum reward of 1.177 in 22 steps. However, when taking into consideration the discount factor, this parameter influenced which behavior provided the agent a more valuable discounted reward. Consequently, for discount factors lower than 0.98, agents received higher discounted rewards for employing economy of force, as rewards quickly depreciated the longer it took to attain them. During later analyses, these theoretical discounted rewards were used to validate the performance of neural networks. In particular, for those exhibiting economy of force, as raw, undiscounted rewards were not a prime metric. Figure 16 demonstrates how manipulating the discount factor impacted the total theoretical discounted reward for both mass and economy of force.

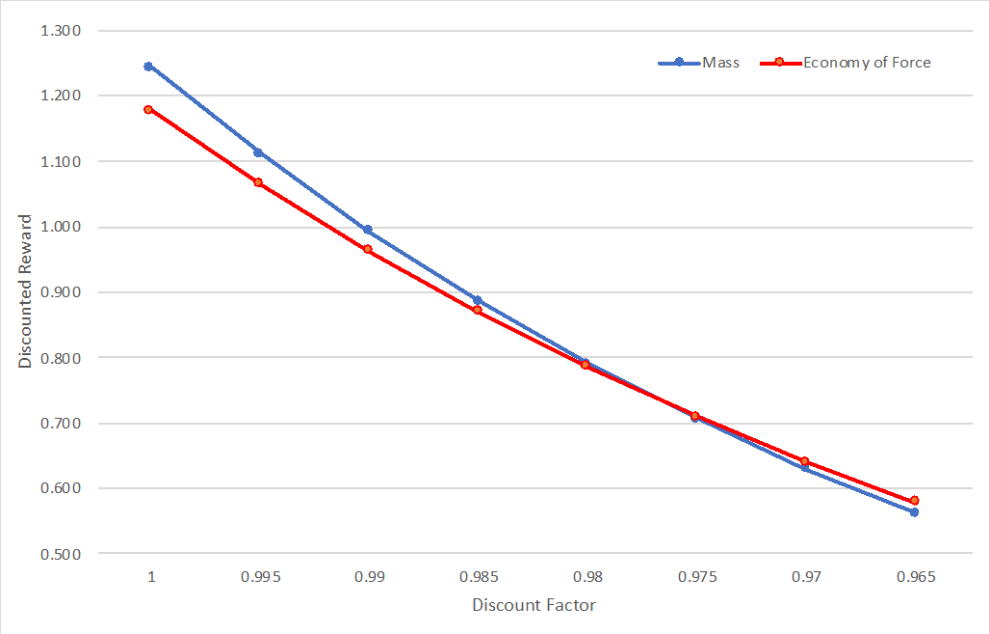

Figure 16. Impact of Discount Factor on Rewards

Despite resulting in a lower undiscounted reward, at discount factors of 0.975, 0.97, and 0.965, an agent receives higher discounted rewards for using economy of force. This can be attributed to the fact that the total engagement length is 8 steps shorter, resulting in quicker rewards.

Most network and training configurations performed relatively well for this task, although, it was more difficult to assess performance given average reward was not always the primary indication of good behavior. Nonetheless, the top five networks demonstrating mass and economy of force are listed in Table 13 and Table 14.

Table 13. Top Five Performing Neural Networks for Mass

Rank	Parameter			Average Reward	Average Episode Length
	Hidden Units	Learning Rate	Discount Factor		
1	32	0.0001	0.995	1.2415 ± 0.0002	30.2 ± 0.4
2	2 x 64	0.0001	0.99	1.2399 ± 0.0045	31.0 ± 0.0
3	2 x 96	0.005	0.99	1.2398 ± 0.0055	30.3 ± 0.6
4	2 x 96	0.0005	0.995	1.2393 ± 0.0058	30.6 ± 0.5
5	2 x 64	0.0005	0.99	1.2392 ± 0.0051	31.2 ± 0.4

With respect to demonstrating the principle of mass, the discount factor was the only significant parameter. The learning rate appeared to have an influence, with lower learning rates supporting stronger performance, but this was not found to be statistically significant. Additionally, larger neural networks appeared to perform better, but this trend could not be verified.

Table 14. Top Five Performing Neural Networks for Economy of Force

Rank	Parameter			Average Reward	Average Episode Length
	Hidden Units	Learning Rate	Discount Factor		
1	64	0.001	0.97	1.1813 ± 0.006	22.4 ± 0.5
2	64	0.0005	0.975	1.1834 ± 0.006	22.6 ± 0.5
3	2 x 96	0.0001	0.97	1.1834 ± 0.006	22.6 ± 0.5
4	32	0.001	0.965	1.1834 ± 0.006	22.6 ± 0.5
5	64	0.01	0.965	1.1867 ± 0.002	23.2 ± 0.4

As alluded to previously, it was more difficult to assess the performance of neural networks with respect to economy of force, as a low average undiscounted reward and low episode length did not automatically equate to good behavior. As a result, these networks were also evaluated in terms of average discounted reward, which were compared to the optimal discounted rewards presented in Figure 16. The comparison between actual, with standard deviations, and optimal theoretical values is illustrated in Table 15.

Table 15. Comparison of Discounted Reward Values

Rank	Parameter			Average Discounted Reward	Optimal Discounted Reward	Percent Error
	Hidden Units	Learning Rate	Discount Factor			
1	64	0.001	0.97	0.637 ± 0.004	0.640	0.50
2	64	0.0005	0.975	0.705 ± 0.003	0.709	0.54
3	2 x 96	0.0001	0.97	0.635 ± 0.004	0.640	0.74
4	32	0.001	0.965	0.572 ± 0.005	0.578	0.94
5	64	0.01	0.965	0.566 ± 0.007	0.578	2.12

Overall, trained agents performed very well, with a maximum percent error of 2.12%. Similar to the neural networks demonstrating mass, discount factor was the only significant parameter, except with lower discount factors yielding the desired economy of force behavior. On the other hand, there were no discernible trends in terms of learning rate or hidden units, although larger networks did again appear to have a weak positive influence on performance.

C. ALGORITHM COMPARISON

The last portion of this thesis directly compared the performance of different reinforcement learning algorithms. While VPG and PPO had been previously compared in the two-versus-one scenarios, an additional experiment was designed and executed with TRPO as the primary learning algorithm. This experiment used the two-versus-one scenario and implemented the deterministic Lanchester combat model, as both VPG and PPO had performed similarly well in this configuration. Table 16 depicts the experimental design with the respective factors and levels.

Table 16. Factors and Levels for TRPO Experiment

Factor	Levels
Delta	0.05, 0.01, 0.005, 0.001
Hidden Units	32, 2 x 32, 64, 2 x 64, 96, 2 x 96
Training Duration	500, 1000

There were a total of 48 design points for this experiment, which were replicated five times each for a total of 240 trials. While some of the factors remain the same from previous experiments, such as training duration and size and number of hidden units, a new term, delta, was introduced. This is because TRPO does not implement a policy learning rate in its algorithm. Comparable to PPO, TRPO seeks to improve the policy through taking large steps that are based on the difference between the previous and current policy [20]. Consequently, the delta term constrains the size of these steps in a similar fashion as PPO's clip factor.

These differences considered, Table 17 presents the mean results and standard deviations for the top performing neural networks trained with TRPO.

Table 17. Top Performing Neural TRPO Neural Networks

Rank	Parameter			Average Reward	Percent Perfect Teaming
	Hidden Units	Delta	Duration		
1	96	0.01	1000	0.811 ± 0.004	93.7 ± 3.9
2	2 x 64	0.05	1000	0.811 ± 0.001	93.4 ± 2.3
3	2 x 64	0.01	1000	0.811 ± 0.002	92.7 ± 3.3
4	2 x 64	0.05	500	0.810 ± 0.002	92.7 ± 2.3
5	96	0.05	1000	0.810 ± 0.002	92.7 ± 2.0

For agents trained with TRPO, training duration, delta value, and size and number of hidden layers were all found to have a statistically significant impact on performance, with longer training and larger deltas producing better results. Moreover, neural networks with two layers of 64 hidden units tended to perform better than other configurations.

Furthermore, Figure 17 shows the differences in perfect teaming percentages between reinforcement learning algorithms. A comparison of average reward was not presented, as values between the three algorithms were too close in value to depict graphically.

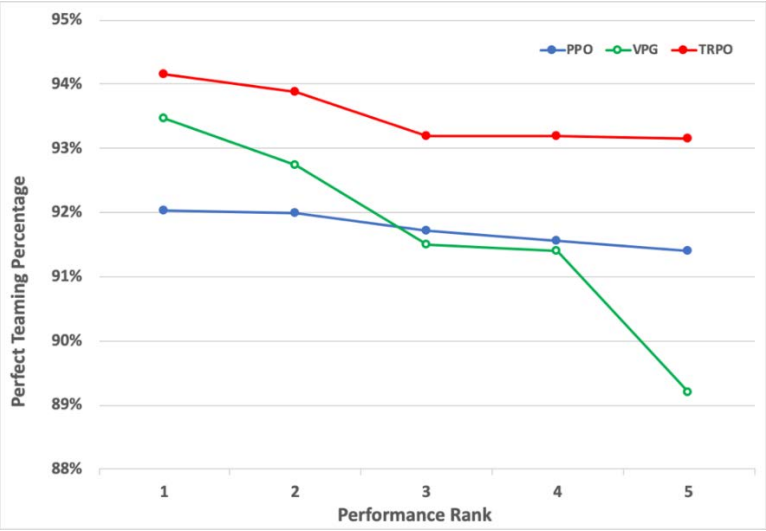

Figure 17. Perfect Teaming Percentage Comparison

The figure above appears to suggest that TRPO performed superior to both VPG and PPO with respect to perfect teaming percentages. In order to validate this claim, two two-sample t-tests assuming equal variances were conducted to compare TRPO to both PPO and VPG. For both tests the alpha level was set to 0.05 and the two null hypotheses that the difference between perfect teaming percentages would be zero. Contrarily, the two alternative hypotheses stated that there would be a difference between perfect teaming percentages. For both tests the null hypothesis was rejected, as the PPO-TRPO t-test yielded a two-tailed p-value of 8.3×10^{-5} and the VPG-TRPO t-test produced a two-tailed p-value of 0.039. Consequently, it can be confirmed that TRPO outperformed both PPO and VPG with respect to perfect teaming percentage.

Although a graphical comparison was impractical, two-sample t-tests assuming equal variance between PPO and TRPO and VPG and TRPO were still conducted to assess the difference in performance with respect to average reward. Similar to before, the alpha level was set to 0.05. The two null hypotheses were that there would be no difference in average rewards, and the two alternative hypotheses that there would be a non-zero difference. For the VPG-TRPO test, the null hypothesis was retained, as the t-test yielded a two-tailed p-value of 0.14. However, the PPO-TRPO t-test produced a two-tailed p-value of 0.0016, meaning the null hypothesis was rejected. Thus, no statistical difference was found between the average rewards of VPG and TRPO, but the average reward of TRPO was determined to be higher than that of PPO.

V. CONCLUSIONS

The overarching objectives of this thesis were to determine whether reinforcement learning was capable of eliciting optimal behavior in tactical engagements and which techniques produced superior results. In order to achieve this, neural networks were trained with different algorithms in simple combat scenarios that allowed for their performance to be validated. In particular, these scenarios were designed such that ideal neural network behaviors followed the tactical principles of mass and economy of force. In total, both objectives were met, as neural networks were able to learn the optimal behaviors. This chapter will present the overall findings, provide recommendations for possible improvement, and discuss future work that can capitalize on this research.

A. FINDINGS

1. Mass

The tactical concept of mass was the simplest principle of war to assess performance against, as the efficient consolidation of combat power reduces casualties and improves the probability of winning. Having implemented a reward system that incentivized kills and penalized death, performance was evident in the average reward that trained agents attained. Moreover, the rate at which trained neural networks executed perfect teaming was another metric to assess final performance. These evaluation measures combined allowed for a thorough analysis of training effectiveness.

In total, the factor with the largest impact on performance was the combat model, as deterministic models generally performed better than their stochastic counterparts. As presented previously in the results chapter, even the worst performing deterministic combat model attained perfect teaming percentages in the upper-80s, whereas, the worst performing stochastic combat model achieved perfect teaming percentages in the mid-60s. The reason for this difference in performance was likely due to the natural variance and probability inherent to stochastic models. An agent may have performed perfectly in one situation, yet by complete chance still lost the engagement. Contrarily, an agent may have executed a poor or reckless course of action, but still achieved victory. While repeating

each possible scenario enough times will theoretically allow the agent to learn the correct behavior and mitigate occasional outliers, unless the training duration is infinite, it is impossible to discount this effect as an insignificant influence. On the other hand, stochastic combat models also performed quite well in some instances. In the two-versus-one scenario, agents trained with PPO in the stochastic Lanchester combat model out-performed those trained in the deterministic Lanchester. While the stochastic Lanchester had additional components that made it more predictable than the probability of kill model, there was still an additional amount of variance in comparison to the deterministic version. Consequently, it cannot be stated conclusively that deterministic combat models will always produce superior results.

In addition to the combat model, the reinforcement learning algorithm implemented also had a considerable impact. While PPO and VPG performed similarly in several combat models, there were also distinctions identified in others as well. In particular, PPO was found to perform better in stochastic environments, as there was a statistically significant difference in performance between algorithms in the probability of kill and stochastic Lanchester combat models. It is difficult to pinpoint the exact cause for this observation, but is likely related to the manner in which the algorithms update policy weights. As previously described, PPO seeks to improve behavior in steps as large as possible, with the major constraint being the difference between new and old policies. On the other hand, VPG takes small incremental steps to improve the policy. Considering these differences, it is conceivable that VPG required longer training to match the performance of its PPO counterparts, especially since more samples were required to overcome the aforementioned variance in stochastic environments.

With respect to the training and neural network hyper-parameters, it was difficult to detect any overall trends and configurations that produced superior results. For agents trained with PPO, the most significant parameter was the training duration, with longer training generally contributing to stronger performance. Furthermore, the learning rate also often generated an influence, but with no inclination toward higher or lower rates. In a few cases the number and size of hidden layers was determined to be statistically significant, but with no configurations appearing dominant. On the other hand, for neural networks

trained with VPG, training duration and learning rate consistently had a significant effect on performance, with longer training and higher learning rates performing better. Although, unlike PPO, there were no instances in which hidden units had a detectable effect.

2. Economy of Force

Relative to evaluating mass, performance in demonstrating economy of force was more difficult to quantify and analyze. As opposed to looking directly at raw, undiscounted rewards, the average discounted rewards were calculated and compared to theoretically ideal performance values. This was because economy of force may not have produced the highest undiscounted reward with the reward function implemented. Consequently, this statistic, in combination with other metrics such as average length of engagement, allowed for the quality of behavior to be assessed.

The experiment for economy of force implemented a single combat model and learning algorithm, a decision informed from the results of the consolidation of force experiments. Consequently, the learning and neural network hyper-parameters were the only factors that were varied. In this regard, the discount factor had the largest impact on performance, as low discount factors produced behavior consistent with economy of force. This was to be expected, as decreasing the discount factor incentivized agents to attain rewards faster. On the contrary, high discount factors elicited traditional massing behavior, since rewards retained greater value in longer engagements. Aside from discount factor, no other hyper-parameters were found to have a statistically significant effect on performance. Larger neural networks of 64 or 96 hidden units appeared to support better performance, but this influence was too weak to verify.

3. Algorithm Comparison

A comparison of the three algorithms implemented in the two-versus-one scenario with the deterministic Lanchester combat model revealed that TRPO did outperform the other two algorithms with respect to perfect teaming percentages. Moreover, a three-way comparison of average reward showed no difference between TRPO and VPG, but there was a distinction between TRPO and PPO, with TRPO performing slightly better. It should

be noted that a multi-comparison statistical correction method, such as Bonferroni, was considered. However, because there were only three comparisons being made for a given variable, it was deemed unnecessary. Overall, it was difficult to generalize the performance of TRPO any further, as it was not examined in any other combat models, but certainly may prove to be another effective training mechanism.

B. RECOMMENDATIONS

Having discussed the overall findings, there are several recommendations that may improve the performance of trained agents. These recommendations consist of changes to the experimental design, reinforcement learning methods, and combat simulation. With respect to the experiment itself, it would be worth expanding both the parameters and the respective levels tested. Given there was rarely a statistically significant difference detected between the varying sizes of hidden layers, this indicates that even the smallest networks were capable of effectively solving the model. Consequently, exploring the lower bound, that is using even smaller networks, may provide interesting insight. Additionally, as it pertains to the reinforcement learning algorithms, it would be beneficial to further analyze the impact of other parameters, such as discount factor. Moreover, since there are non-trivial distinctions between algorithms, examining and varying parameters unique to each one may also yield differences in trained agent performance.

While each of the reinforcement learning algorithms performed well, only rarely did they achieve perfect performance. Identifying the reason for this shortfall is difficult, as explaining the nuances behind the behaviors of neural networks is another emerging area of research. It is possible, though, that the neural networks were not provided enough experiences across the wide range of starting positions and configurations. In this case, it may be valuable to take a single experience and generalize it for all orientations on the map. For example, approaching an opponent from the southwest in a column formation is no different than advancing on an opponent in the same formation from the east. Therefore, replicating the same experience for different directions may mitigate this issue. Of course, it should also be noted that this method will only be effective if the terrain is rotated with the agent, as different types of terrain require different tactics and forms of movement.

Moreover, another possible solution is through changing, or adding to, the state input. This thesis provided neural networks a world-centric view of their position, although including an ego-centric perspective as well may also resolve this issue without the need to generate more experience.

In regards to the training environment and combat simulation, entities moved and executed combat in a stepwise approach commensurate with a hex-based wargame. This most certainly impacted training and performance, as enabling the neural network to move units simultaneously and in a continuous action space will likely impact final performance. For the purposes of this project, this would mean allowing entities to move in a continuous range of directions and distances. However, even conventional graphical simulations that apply continuous action spaces still function in a time-stepped manner. Consequently, building a discrete event simulation model may provide additional insight and produce unique results.

The last recommendation pertains to the reward system and its influence on agent behavior. The reward functions implemented in this thesis focused on attrition and combat outcomes, which was perfectly reasonable considering battles are often measured in casualties and material loss. However, this is not the only measure of combat effectiveness, as other aspects of combat may prove to be more important in certain circumstances. For example, time is a measurable element that may be a valuable metric when conducting a delaying operation or movement to contact. A unit tasked with impeding the approach of a much larger unit might measure success not in attrition, but in the number of days the opposing unit's advance was frustrated. On the other hand, other possible reward measures are more difficult to quantify, such as population influence, troop morale, and logistical throughput. While these factors may not have explicit units of measure, incorporating them into the combat model and reward system will affect the behavior of trained agents.

C. FUTURE WORK

There are a number of ways in which researchers can expand on the work completed in this thesis, studies which may consist of developing larger scenarios, implementing more complex environmental factors, or testing different combat models. Increasing the

complexity and size of the scenarios tested will be necessary in future work, as this project focused on relatively small unit engagements. While the obvious action in this regard is to increase the number and categories of units in-play, examining different mission sets and operation types is equally important. This ranges from programming scenarios representative of other simple tactical concepts, such as defensive operations, to more complex maneuvers that may include amphibious assaults or turning movements. Moreover, this thesis strictly employed static enemies that did not react to or counter the AI opponents. Future work will also need to teach neural networks to fight against dynamic opponents, starting with simple programmed behaviors and working up to human or opposing AI control.

In addition to the size and scope of the scenario, increasing the complexity of the environment is another area of future work. This can be manifested in a number of ways, as the effects of terrain, weather, and time of day can be incorporated to impact movement, logistics, and command and control. Each of these elements in itself is complex and difficult to accurately represent in a combat simulation, although, the importance of these factors cannot be disregarded. As such, the simplest example would be the addition of terrain to affect movement speed and trafficability, whereas, more complex environmental effects might consist of weather and temporal disruption of command and control capabilities.

Lastly, it will be valuable to for future researchers to duplicate results in different simulations and combat models. While the simple Python combat simulation developed for this thesis was sufficient given the size and scope of the scenarios, results will eventually need to be replicated in more robust simulations. Logistics, command and control, and intelligence are all functions that will need to be added to increase the fidelity and realism of combat. Additionally, changing the underlying combat model itself is a potential area of study, as a suite of methods for combat resolution, outside of Lanchester equations and probability of kill, are available for employment. One such example might consist of applying Wayne Hughes' salvo equations to a naval or amphibious environment.

D. CONCLUSION

This thesis has established the baseline capabilities for AI agents and validated the ability for reinforcement learning to produce optimal behaviors in small combat scenarios. While the performance of trained neural networks was consistent with the tactical principles mass and economy of force, these are only the fundamental building blocks to more advanced tactical and operational concepts. Much more work is necessary to train and deploy systems capable of providing training and analytical support to the operating forces. There is a plethora of options for expanding the scope and complexity of the work completed here, some of which have been discussed above; however, the end state should always be focused on improving the resources and capabilities of the warfighter.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. JMP DATA

This appendix contains the parameter estimate tables, generated by JMP, that were used to assess which factors impacted performance for each reinforcement learning algorithm, force configuration, and combat model. As such, orange text in the table represents a statistically significant result with a p-value less than 0.01. Furthermore, red text also indicates a factor was statistically significant, but with a p-value between 0.01 and 0.05.

Table 18. PPO – Probability of Kill JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.4994556	0.004231	118.05	<.0001*
lr	1.1597015	0.341684	3.39	0.0013*
hid[32]	-0.001009	0.002882	-0.35	0.7277
hid[32-32]	-0.000855	0.002882	-0.30	0.7679
hid[64]	-0.00483	0.002882	-1.68	0.0997
hid[64-64]	0.0022753	0.002882	0.79	0.4334
hid[96]	-0.000566	0.002882	-0.20	0.8450
duration	3.7038e-5	5.156e-6	7.18	<.0001*

For the combination of PPO and probability of kill model, the learning rate and training duration were identified to be statistically significant.

Table 19. VPG – Probability of Kill JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.3091048	0.040661	7.60	<.0001*
lr	14.955733	3.283857	4.55	<.0001*
hid[32]	-0.018517	0.027699	-0.67	0.5068
hid[32-32]	-0.004051	0.027699	-0.15	0.8843
hid[64]	-0.0006	0.027699	-0.02	0.9828
hid[64-64]	0.0072519	0.027699	0.26	0.7945
hid[96]	-0.001381	0.027699	-0.05	0.9604
duration	8.315e-5	4.955e-5	1.68	0.0993

For the combination of VPG and probability of kill model, the learning rate was the only factor found to be statistically significant.

Table 20. PPO – Referee Model JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.9927555	0.005488	180.88	<.0001*
lr	0.4883258	0.443244	1.10	0.2757
hid[32]	-0.006324	0.003739	-1.69	0.0967
hid[32-32]	0.0029395	0.003739	0.79	0.4353
hid[64]	-0.005199	0.003739	-1.39	0.1703
hid[64-64]	0.0041666	0.003739	1.11	0.2702
hid[96]	0.0042434	0.003739	1.13	0.2616
duration	1.8399e-6	6.688e-6	0.28	0.7843

For the combination of PPO and referee combat model, there were no factors identified to be statistically significant.

Table 21. VPG – Referee Model JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.4834929	0.12541	3.86	0.0003*
duration	0.0001734	0.000153	1.13	0.2617
lr	45.673774	10.12824	4.51	<.0001*
hid[32]	-0.082867	0.085431	-0.97	0.3365
hid[32-32]	-0.008387	0.085431	-0.10	0.9222
hid[64]	-0.026847	0.085431	-0.31	0.7546
hid[64-64]	0.0442862	0.085431	0.52	0.6064
hid[96]	-0.001485	0.085431	-0.02	0.9862

For the combination of VPG and referee combat model, learning rate was the only factor assessed to be statistically significant.

Table 22. PPO – Deterministic Lanchester Model Parameter JMP Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.8109723	0.00023	3526.9	<.0001*
duration	8.6864e-7	2.802e-7	3.10	0.0031*
lr	0.0062995	0.01857	0.34	0.7358
hid[32]	5.5783e-5	0.000157	0.36	0.7232
hid[32-32]	0.0001335	0.000157	0.85	0.3981
hid[64]	-0.000325	0.000157	-2.07	0.0431*
hid[64-64]	0.0001708	0.000157	1.09	0.2805
hid[96]	0.0001527	0.000157	0.97	0.3342

For the combination of PPO and deterministic Lanchester combat model, training duration and number and size of hidden units were assessed to be statistically significant, with neural network sizes of 64 having a slight negative influence.

Table 23. VPG – Deterministic Lanchester Model JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.6079822	0.054553	11.14	<.0001*
lr	13.104001	4.405792	2.97	0.0044*
hid[32]	-0.042137	0.037163	-1.13	0.2620
hid[32-32]	-0.013741	0.037163	-0.37	0.7131
hid[64]	-0.015041	0.037163	-0.40	0.6873
hid[64-64]	0.026195	0.037163	0.70	0.4840
hid[96]	0.0009863	0.037163	0.03	0.9789
duration	0.0001345	6.648e-5	2.02	0.0482*

For the combination of VPG and deterministic Lanchester combat model, learning rate and training duration were found to be statistically significant.

Table 24. TRPO – Deterministic Lanchester Model JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.8092877	0.000618	1308.5	<.0001*
duration	2.7974e-6	7.56e-7	3.70	0.0006*
delta	0.0216773	0.009641	2.25	0.0301*
hid[32]	-0.000347	0.000423	-0.82	0.4160
hid[32-32]	3.3012e-5	0.000423	0.08	0.9381
hid[64]	-0.000606	0.000423	-1.43	0.1594
hid[64-64]	0.0008889	0.000423	2.10	0.0418*
hid[96]	-0.000228	0.000423	-0.54	0.5926

For the combination of TRPO and deterministic Lanchester combat model, training duration, delta value, and number and size of hidden layers were determined to be statistically significant, as neural networks with two layers of 64 units had a slight positive effect.

Table 25. PPO – Stochastic Lanchester Model JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.8402218	0.000715	1174.4	<.0001*
lr	0.1299761	0.057782	2.25	0.0287*
hid[32]	-0.000415	0.000487	-0.85	0.3982
hid[32-32]	0.0002844	0.000487	0.58	0.5620
hid[64]	0.0001808	0.000487	0.37	0.7122
hid[64-64]	0.0004715	0.000487	0.97	0.3378
hid[96]	-0.000633	0.000487	-1.30	0.1997
duration	3.0404e-6	8.719e-7	3.49	0.0010*

For the combination of PPO and stochastic Lanchester combat model, training duration and learning rate were found to have a statistically significant influence.

Table 26. VPG – Stochastic Lanchester Model JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	0.6158839	0.060258	10.22	<.0001*
lr	14.108339	4.866489	2.90	0.0055*
hid[32]	-0.04237	0.041048	-1.03	0.3068
hid[32-32]	-0.014911	0.041048	-0.36	0.7179
hid[64]	-0.013377	0.041048	-0.33	0.7458
hid[64-64]	0.0227912	0.041048	0.56	0.5811
hid[96]	0.0017566	0.041048	0.04	0.9660
duration	0.0001485	7.343e-5	2.02	0.0483*

For the combination of VPG and stochastic Lanchester combat model, learning rate and training duration were assessed to be statistically significant.

Table 27. Two-versus-two Deterministic Lanchester JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1.3865913	0.011597	119.56	<.0001*
lr	-2.2396	0.936606	-2.39	0.0204*
hid[32]	0.0023228	0.0079	0.29	0.7699
hid[32-32]	-0.005932	0.0079	-0.75	0.4561
hid[64]	0.0128976	0.0079	1.63	0.1086
hid[64-64]	0.0084593	0.0079	1.07	0.2892
hid[96]	0.0114342	0.0079	1.45	0.1538
duration	0.0000186	1.413e-5	1.32	0.1938

For the two-versus-two force configuration implementing the deterministic Lanchester, the learning rate was the only factor assessed to be statistically significant.

Table 28. Two-versus-two Stochastic Lanchester JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	1.4588107	0.00387	376.91	<.0001*
lr	0.6317139	0.312581	2.02	0.0484*
hid[32]	0.0015258	0.002637	0.58	0.5653
hid[32-32]	-6.886e-5	0.002637	-0.03	0.9793
hid[64]	-0.001901	0.002637	-0.72	0.4741
hid[64-64]	0.0007455	0.002637	0.28	0.7785
hid[96]	-0.001061	0.002637	-0.40	0.6890
duration	2.6452e-5	4.716e-6	5.61	<.0001*

For the two-versus-two force configuration implementing the stochastic Lanchester, the learning rate and training duration were found to be statistically significant.

Table 29. Economy of Force JMP Parameter Analysis

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	-0.032042	0.106048	-0.30	0.7628
gam	1.260112	0.108203	11.65	<.0001*
lr	-0.41657	0.286841	-1.45	0.1480
hid[32]	0.0002956	0.002419	0.12	0.9029
hid[32-32]	-0.001915	0.002419	-0.79	0.4295
hid[64]	-0.001397	0.002419	-0.58	0.5643
hid[64-64]	0.0041574	0.002419	1.72	0.0873
hid[96]	-0.001833	0.002419	-0.76	0.4495

For the economy of force scenario, gamma, also known as the discount factor, was the only factor assessed to be statistically significant.

LIST OF REFERENCES

[1] *Warfighting*, Marine Corps Doctrinal Publication 1, Headquarters Marine Corps, Washington, DC, 1997. [Online]. Available: https://www.marines.mil/Portals/1/Publications/MCDP%201%20Warfighting.pdf

[2] *Tactics*, Marine Corps Doctrinal Publication 1–3, Headquarters Marine Corps, Washington, DC, 1997. [Online]. Available: https://www.marines.mil/Portals/1/Publications/MCDP%201-3%20Tactics.pdf

[3] *Operations*, Marine Corps Doctrinal Publication 1–0, Headquarters Marine Corps, Washington, DC, 2017. [Online]. Available: https://www.marines.mil/Portals/1/Publications/MCDP%201-0%20Marine%20Corps%20Operations.pdf

[4] Commandant of the Marine Corps, "38th Commandant's Planning Guidance," Washington, DC, USA, 2019. [Online]. Available: https://www.hqmc.marines.mil/Portals/142/Docs/%2038th%20Commandant%27s%20Planning%20Guidance_2019.pdf?ver=2019-07-16-200152-700

[5] S. Yildirim and S. Berg, "A survey on the need and use of AI in game agents," *Modeling Simulation and Optimization - Focus on Applications*, S. Cakaj, Ed. InTech, 2010.

[6] N. Brown and T. Sandholm, "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, Jan. 2018, doi: 10.1126/science.aao1733.

[7] D. Silver et al., "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *ArXiv171201815 Cs*, Dec. 2017, Accessed: Apr. 01, 2020. [Online]. Available: http://arxiv.org/abs/1712.01815.

[8] O. Vinyals et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019, doi: 10.1038/s41586-019-1724-z.

[9] OpenAI, "OpenAI Five," Jun. 25, 2018. [Online]. Available: https://openai.com/blog/openai-five/

[10] S. Levine, "CS285 deep reinforcement learning," Robotic AI & Learning Lab, Berkeley, CA, USA, 2019. [Online]. Available: http://rail.eecs.berkeley.edu/deeprlcourse/

[11] A. Tampuu et al., “Multiagent cooperation and competition with deep reinforcement learning,” *PLOS ONE*, vol. 12, no. 4, p. e0172395, Apr. 2017, doi: 10.1371/journal.pone.0172395.

[12] V. Mnih et al., “Playing atari with deep reinforcement learning,” *ArXiv13125602 Cs*, Dec. 2013, Accessed: Apr. 12, 2020. [Online]. Available: http://arxiv.org/abs/1312.5602.

[13] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “ViZDoom: A Doom-based AI research platform for visual reinforcement learning,” *ArXiv160502097 Cs*, May 2016, Accessed: Oct. 09, 2019. [Online]. Available: http://arxiv.org/abs/1605.02097.

[14] M. Gault, “OpenAI is beating humans at ‘Dota 2’ because it’s basically cheating,” Vice, Aug. 17, 2018. [Online]. Available: https://www.vice.com/en_us/article/gy3nvq/ai-beat-humans-at-dota-2

[15] VentureBeat, “OpenAI’s Dota 2 bot defeated 99.4% of players in public matches,” Apr. 22, 2019. [Online]. Available: https://venturebeat.com/2019/04/22/openais-dota-2-bot-defeated-99-4-of-players-in-public-matches/

[16] W. M. Lian, “Using neural networks to determine course of action for a land-based constructive simulation,” M.S. Thesis, Dept. of Comp. Sci., NPS, Monterey, CA, USA, 2019. [Online]. Available: http://hdl.handle.net/10945/63476

[17] Unity Technologies, “Unity real-time development platform.” Accessed Mar. 30, 2019. [Online]. Available: https://unity.com/

[15] *Operational Terms and Graphics*, Marine Corps Reference Publication 5–12A, Headquarters Marine Corps, Washington, DC, 2004. [Online]. Available: https://www.marines.mil/portals/1/mcrp%205-12a%20with%20ch.%201%20z.pdf

[19] “Introduction to Lanchester equations,” class notes for Introduction to Combat Modeling, Dept. of Ops. Res., Naval Postgraduate School, Monterey, CA, USA, fall 2019.

[20] OpenAI, “Welcome to Spinning Up in deep RL! — Spinning Up documentation.” Accessed Jan. 09, 2020. [Online]. Available: https://spinningup.openai.com/en/latest/

[21] OpenAI, “Gym: A toolkit for developing and comparing reinforcement learning algorithms.” Accessed Jan. 09, 2020. [Online]. Available: https://gym.openai.com

[22] PyTorch, “PyTorch: From research to production.” Accessed Mar. 25, 2020. [Online]. Available: https://www.pytorch.org

[23] SAS. Cary, NC, USA. 2019. JMP Pro 15. [Online]. Available: https://www.jmp.com/en_us/software/new-release/new-in-jmp-and-jmp-pro.html

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California